CSEN 702: Microprocessors

Simplified 16-bit RISC Processor Memory Hierarchy Simulator

Ahmed Magdi	25-0455 T11
Ahmed Moataz	25-6033 T11
Mhoab Ghanim	25-8957 T10
Mohamed ElSaeed	25-7019 T10

Implementation

The project consists of several packages. The Simulator class in the simulator package has registers, memory and cache classes as attributes, it is the one responsible for connecting all pieces of the application together, loading and executing instructions from its instruction memory, and performing caching operation from a high level. Memory and Cache consists of blocks, where Block is a class consisting of words, Word itself can either be Data or Instruction. The instructions package contains instructions, each instruction is implemented as a class following the command pattern design, with a method 'execute' that is invoked by the Simulator class. All instructions are subclasses of the abstract class Instruction, and are created using the Instruction factory in the factories package. Likewise, all caches are subclasses of the abstract class Cache, and are created using the Cache factory. Each cache subclass is responsible for implementing methods that control caching logic depending on its type.

Sample Programs

```
data:

instructions:
     ADDI R2, R0, 1
     ADDI R3, R0, 5
     MUL R2, R2, R3
     ADDI R3, R3, -1
     BEQ R0, R3, 1
     JMP R0, -4
```

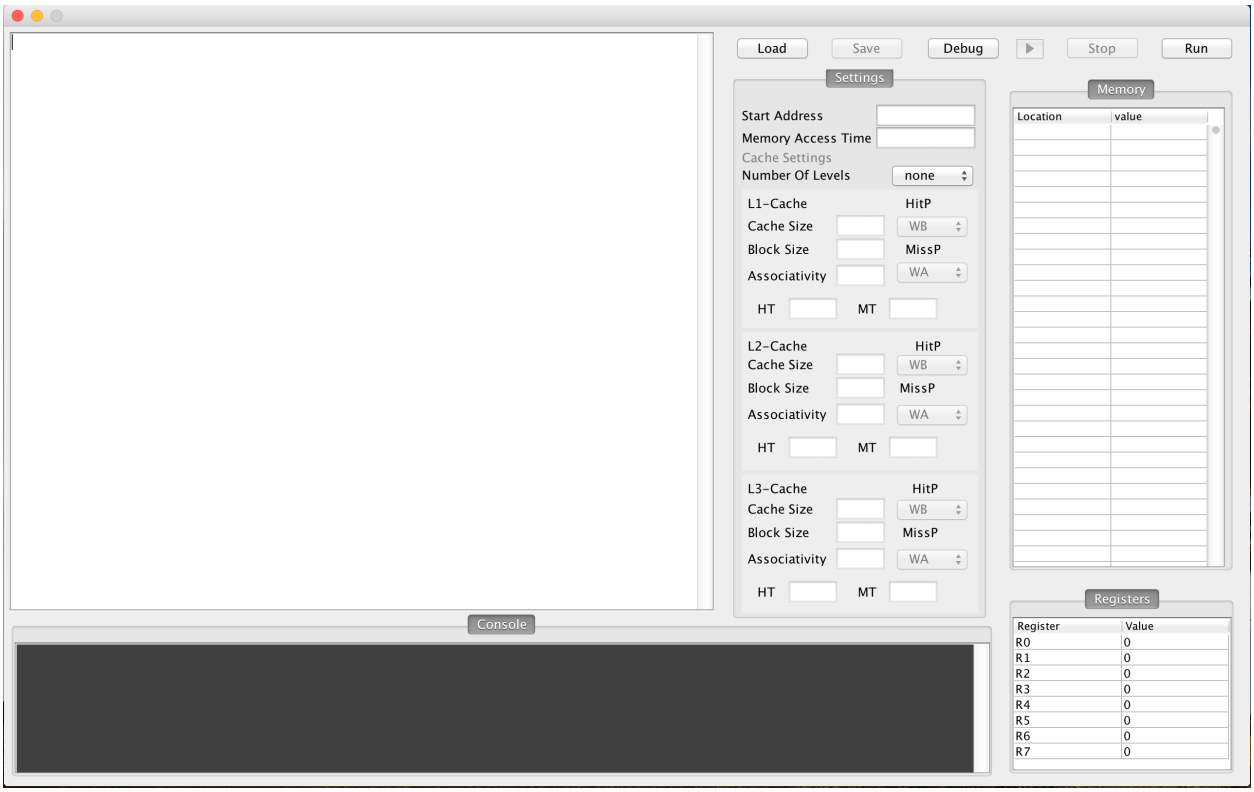
```
data:

instructions:
     ADDI R1, R0, 1
     ADD R2, R1, R1
     BEQ R2, R1, 1
     NAND R3, R2, R1
     JMP R0, 1
     MUL R3, R2, R1
     SW R3, R0, 2
     LW R4, R0, 2
     SW R2, R0, 3
     LW R5, R0, 3
```

```
data:

instructions:
     ADDI R1, R0, 1
     ADDI R2, R0, 10
     ADDI R3, R0, 0
     ADD R1, R1, R1
     ADDI R3, R3, 1
     BEQ R2, R3, 1
     JMP R0, -4
     SW R1, R0, 4
     LW R4, R0, 4
```

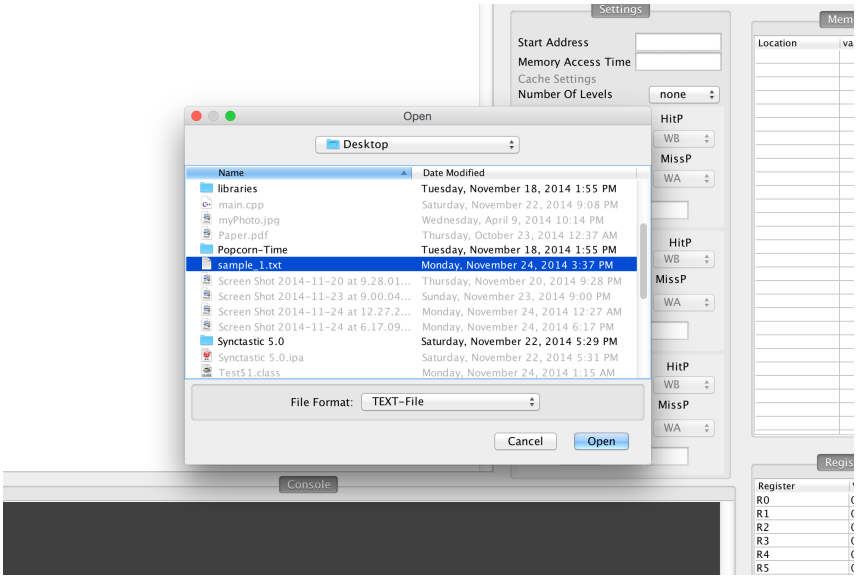
User Guide/Simulation

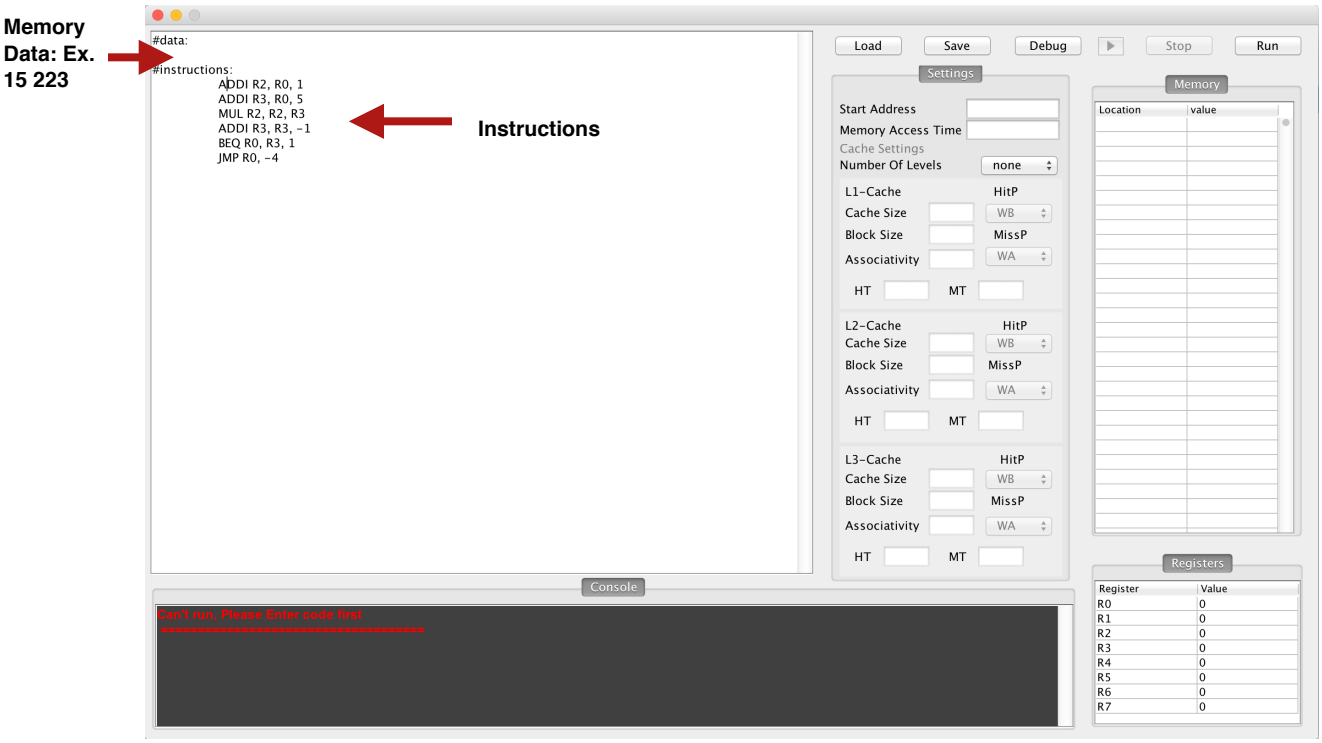


After Opening the program we will be promoted with this. We have two ways to input a code, either by loading a text file or writing the code then saving it.

1- We are going to load < sample_1.txt > by selecting the load button as shown.

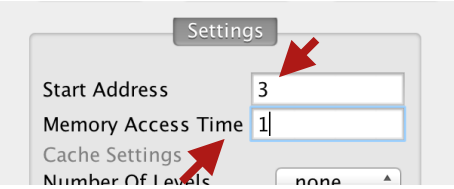
2- You will be promoted with a file browser. Choose the location of the file and select Open.





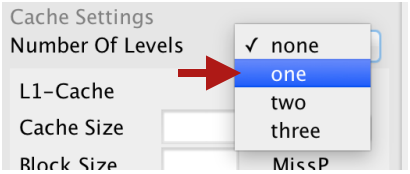
3- If there are any data you want add, in the editor after < #data: > enter the key and the the value with a space separated (ex. 15 223). Any the instructions should go after < #instructions: > .

4- Enter the starting address and the Memory Access Time is the Settings panel



Testing Level 1-cache (L1)

5- For choosing On Level Cache from the drop-down list of the Number Of Levels and select One.



6- Fill the Settings of the L1-Cache Level: Cache Size, Block Size, Associativity, the Hit/Miss policies and the Hit/Miss times.

Note that:

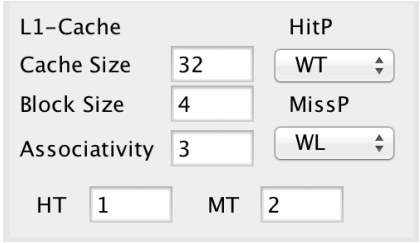
Hit Policies as following:

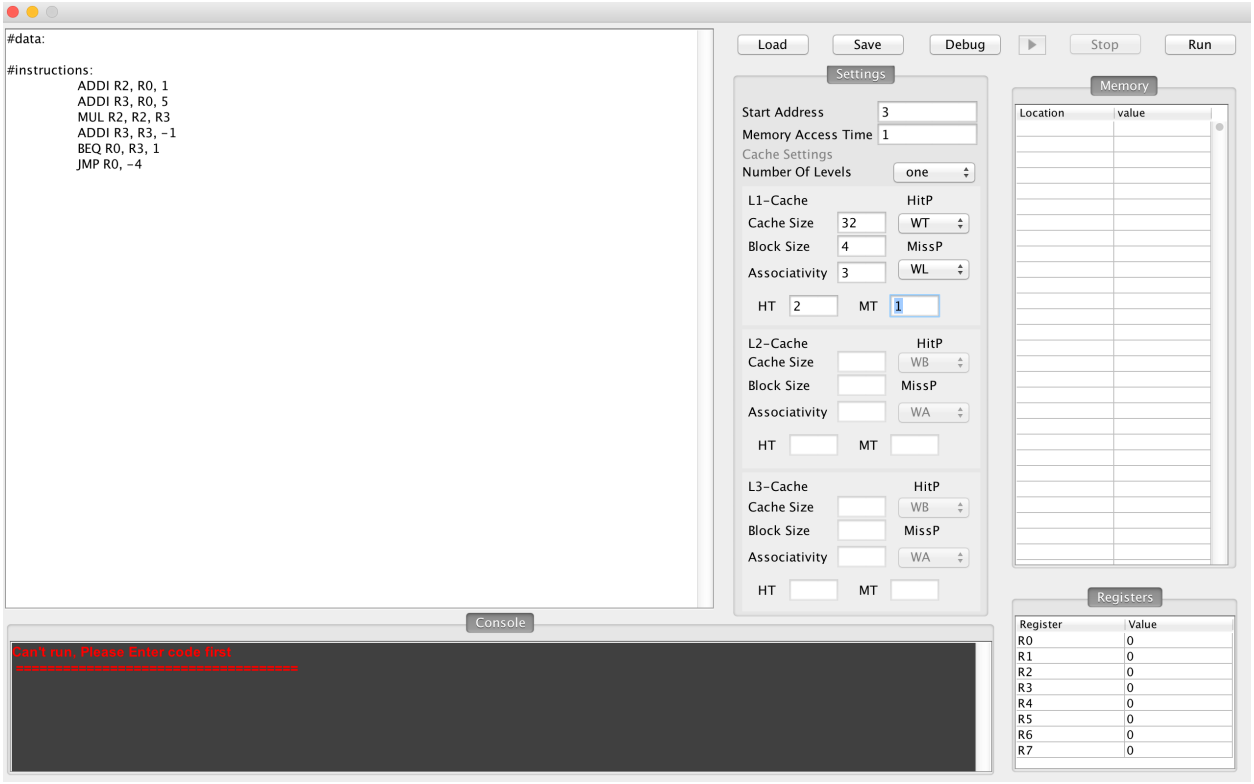
WB: Write Back, *WT*: Write Through

Miss Policies as following:

WA: Write Around, *WL*: Write Allocate

Associativity: ranges from 1 to 4

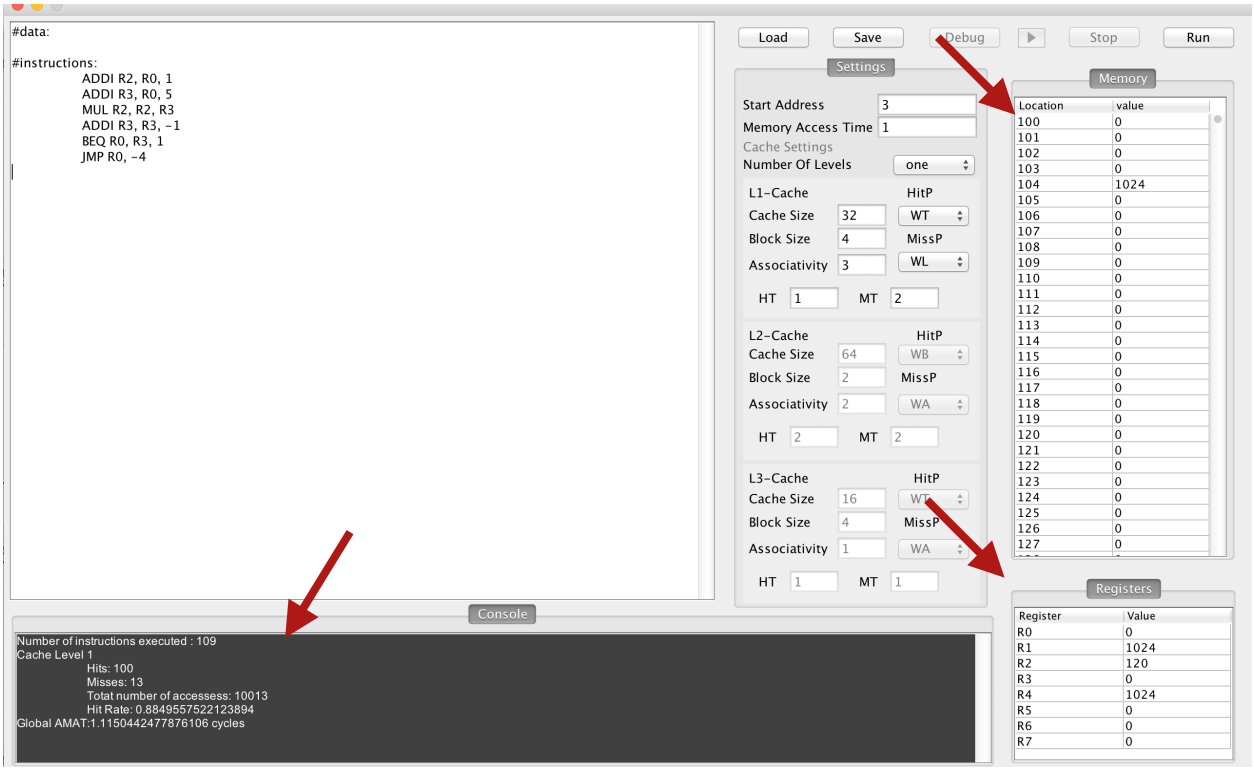




7- After all the settings have been set, we can now run the program.

Results of the Level 1-cache (L1)

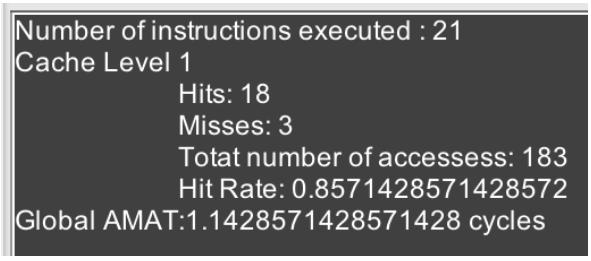
8- If every thing ran successfully you will promoted with the following, which contains the results in the console and the Registers/Memory data, but if you forgot anything you will be given errors of what is missing.



Analyzing the Results

9- The Console results will look like the following :

Cache Level
- Hits: #
- Misses: #
- Total Number Of accesses: #
- Hit Rate: #
- The AMAT: #



10- The registers Data will contain all the data from R0 to R7 that has been used in the program.

Registers

Register	Value
R0	0
R1	0
R2	120
R3	0
R4	0
R5	0
R6	0
R7	0

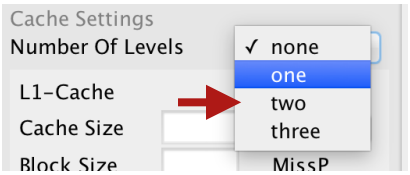
Memory

Location	value
100	0
101	0
102	0
103	0
104	0
105	0
106	0
107	0
108	0
109	0
110	0
111	0
112	0
113	0
114	0
115	0
116	0
117	0
118	0
119	0
120	0
121	0
122	0
123	0
124	0
125	0
126	0
127	0

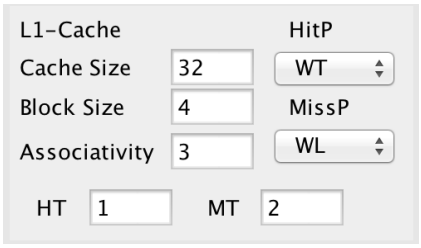
11- The Memory will contain all the data that is stored in the memory.

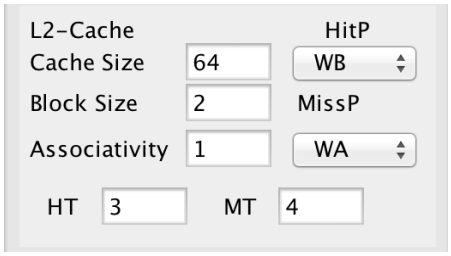
Testing Level 2-caches (L1- L2)

5- For choosing 2 level laches from the drop-down list of the Number Of Levels we select Two.



6- Assuming the same configuration for the L1-Cache, we configure the L2-cache with smiler settings.





7- After all the settings have been set, we can now run the program.

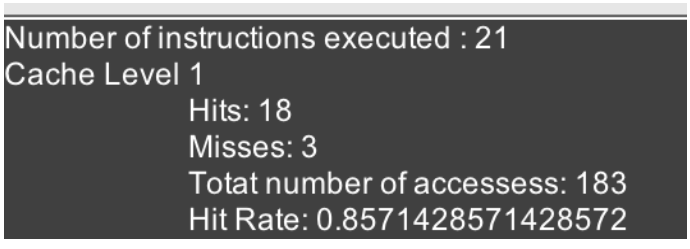
Results of The level 2-cache (L1-L2)

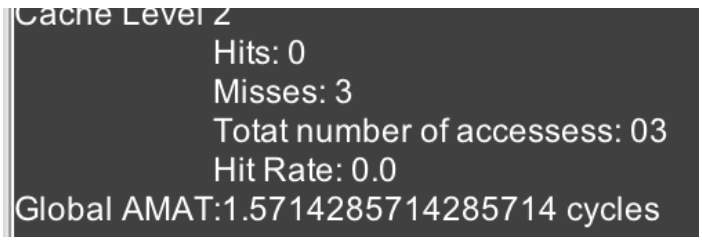
8- Same as the previous one.

Analyzing the Results

9- The Console results will look like the following :

Cache Level #
Hits: #
Misses: #
Total Number Of accesses: #
Hit Rate: #
The AMAT: #





10- The registers Data will contain all the data from R0 to R7 that has been used in the program.

Registers

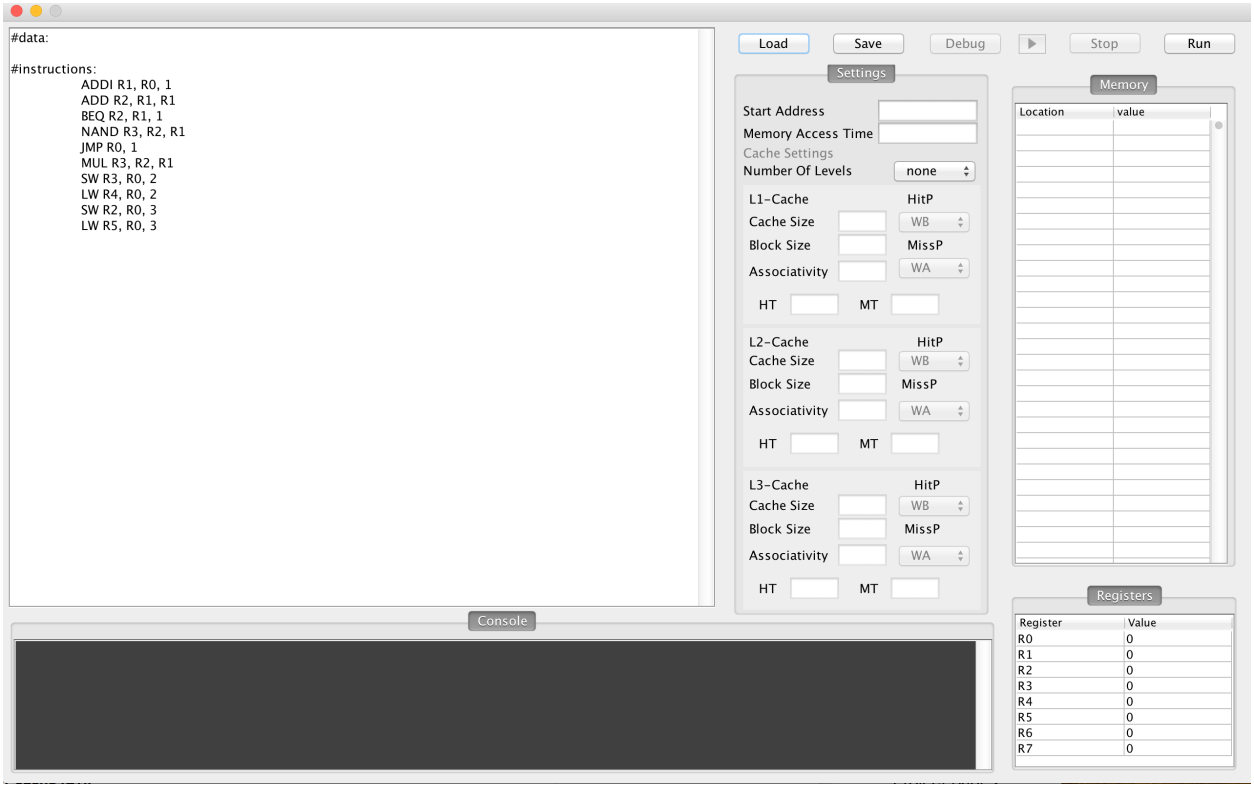
Register	Value
R0	0
R1	0
R2	120
R3	0
R4	0
R5	0
R6	0
R7	0

Memory

Location	value
100	0
101	0
102	0
103	0
104	0
105	0
106	0
107	0
108	0
109	0
110	0
111	0
112	0
113	0
114	0
115	0
116	0
117	0
118	0
119	0
120	0
121	0
122	0
123	0
124	0
125	0
126	0
127	0

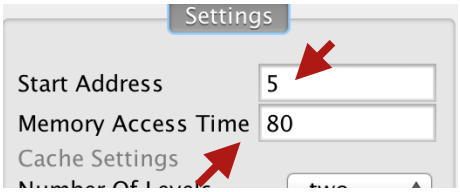
11- The Memory will contain all the data that is stored in the memory.

Loading Sample_2.txt



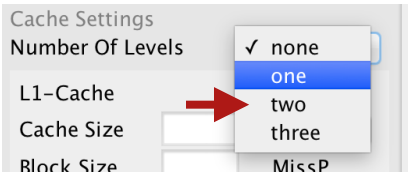
Steps 1,2 & 3 are the same as the previous example.

4- Enter the starting address and the Memory Access Time is the Settings panel

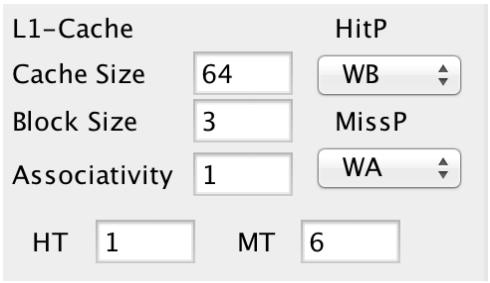


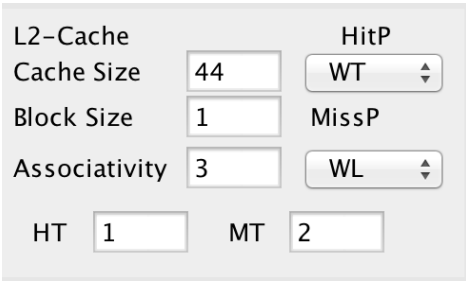
Testing Level 2-cache (L1-L2)

5- For choosing Two Levels of Caches from the dropDown List of the Number Of Levels we select Two .



6- Fill the Settings of the L1-Cache & L2- Cache with the required data.





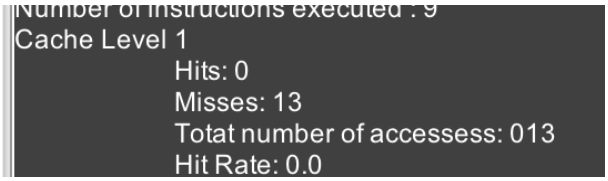
7- After all the setting has been set, we can now run the program.

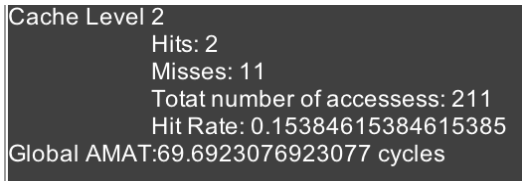
Results of The 2 levels 2-cache (L1-L2)

8- Same as the previous example.

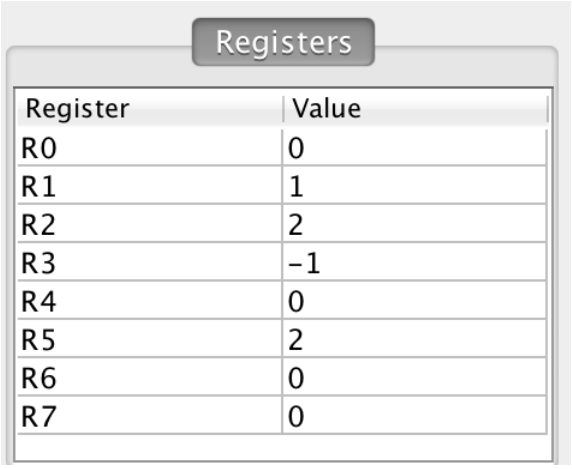
Analyzing the Results

9- The Console results will look like the following :



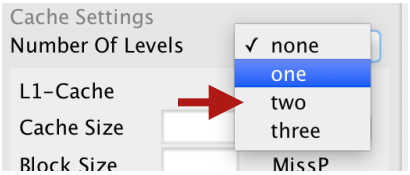


10- The registers Data will contain all the data from R0 to R7 that is been used in the program.

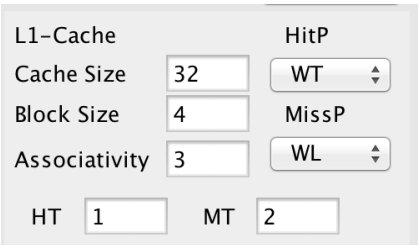


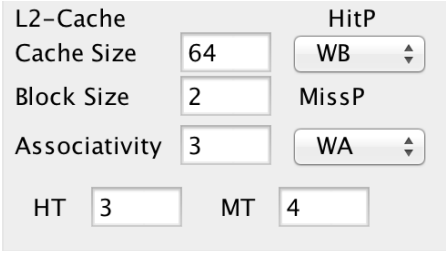
Testing Level 3-caches (L1-L2-L3)

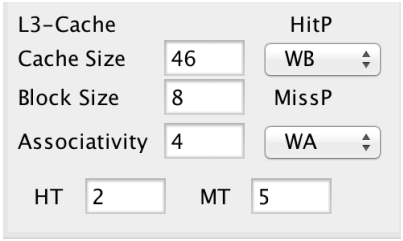
5- For choosing 3 Levels of Caches from the drop-down list of the Number Of Levels we select Three .



6- Assuming the same configuration for the L1-Cache, we configure the L2-cache with smiler settings.







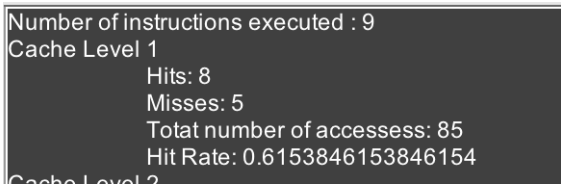
7- After all the setting has been set, we can now run the program.

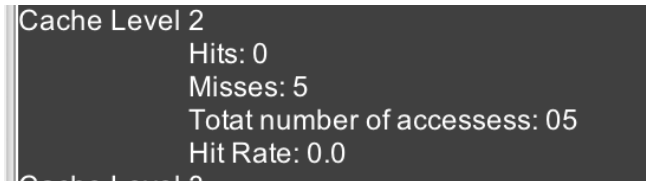
Results of The level 3-caches (L1-L2-L3)

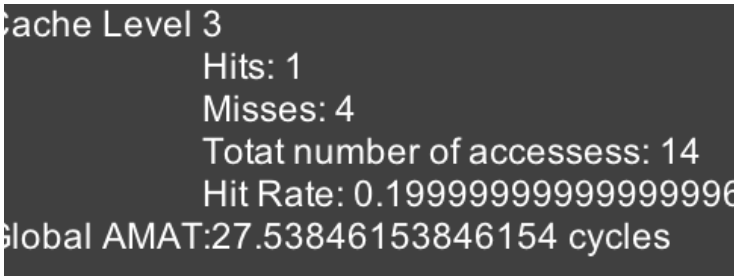
8- Same as the previous one.

Analyzing the Results

9- The Console results will look like the following :







10- The registers Data will contain all the data from R0 to R7 that is been used in the program.

Registers

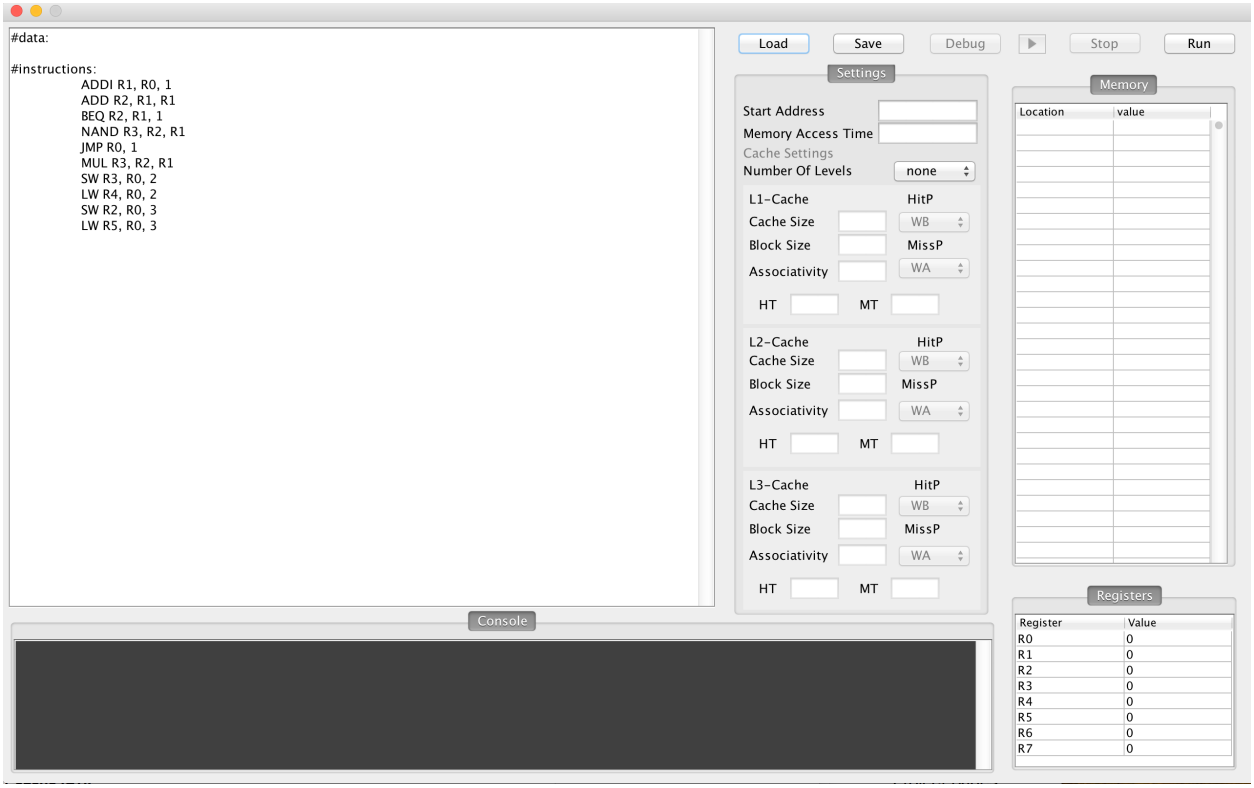
Register	Value
R0	0
R1	1
R2	2
R3	-1
R4	-1
R5	2
R6	0
R7	0

Memory

Location	value
100	0
101	0
102	-1
103	2
104	0
105	0
106	0
107	0
108	0
109	0
110	0
111	0
112	0
113	0
114	0
115	0
116	0
117	0
118	0
119	0
120	0
121	0
122	0
123	0
124	0
125	0
126	0
127	0

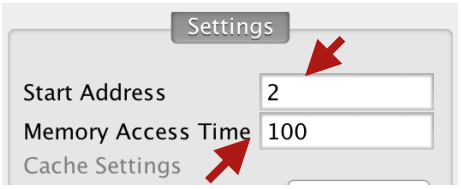
11- The Memory will contain all the data that is stored in the memory.

Loading Sample_3.txt



Steps 1,2 & 3 are the same as the previous example.

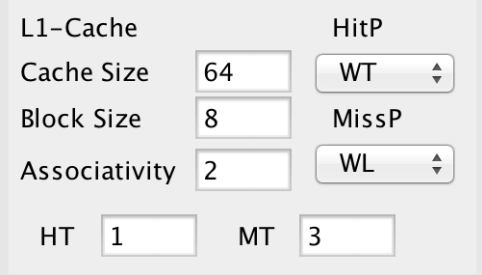
4- Enter the starting address and the Memory Access Time is the Settings panel

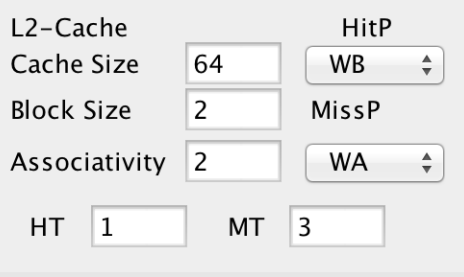


Testing level 2-cache (L1-L2)

5- For choosing Two Levels of Caches from the dropDown List of the Number of levels we select Two .

6- Fill the Settings of the L1-Cache & L2- Cache with the required data.





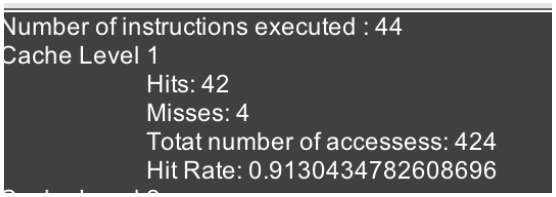
7- After all the setting has been set, we can now run the program.

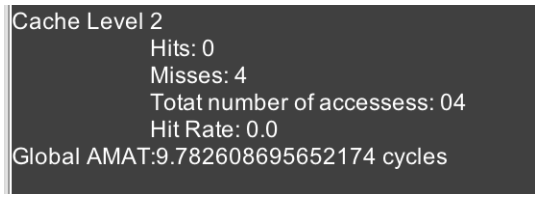
Results of The 2 levels 2-cache (L1-L2)

8- Same as the previous example.

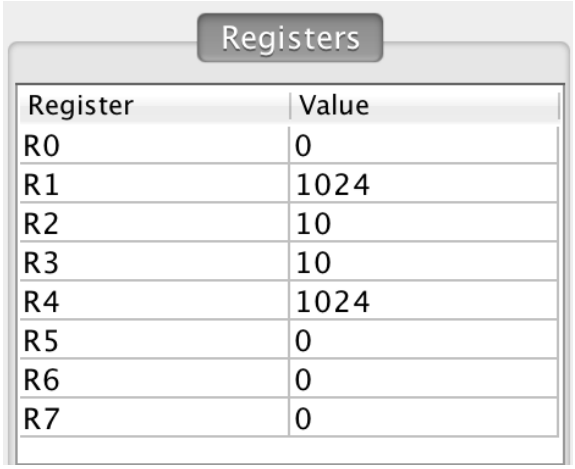
Analyzing the Results

9- The Console results will look like the following :



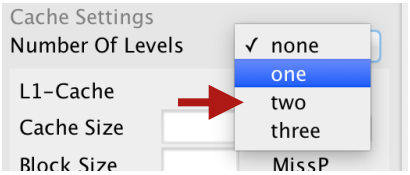


10- The registers Data will contain all the data from R0 to R7 that is been used in the program.

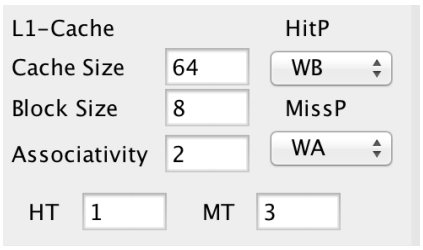


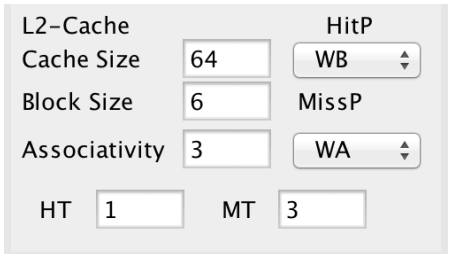
Testing level 3-caches (L1- L2-L3)

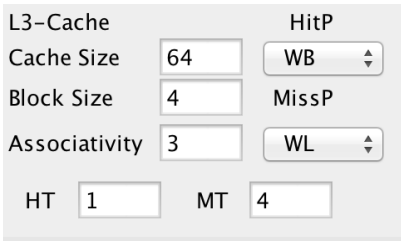
5- For choosing 3 Levels of Caches from the dropDown List of the Number of levels we select Three .



6- Assuming the smiler configuration for the L1-Cache, we configure the L2-cache with smiler settings.







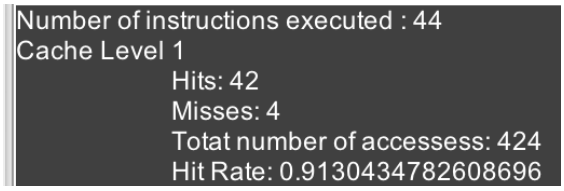
7- After all the setting has been set. now we can run the program.

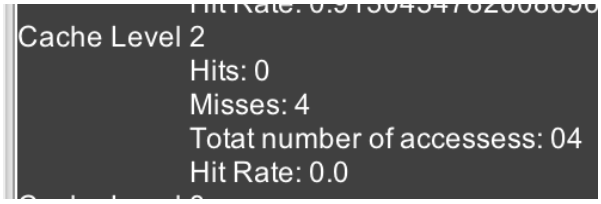
Results of The level 3-caches (L1-L2-L3)

8- Same as the previous one.

Analyzing the Results

9- The Console results will look like the following :





10- The registers Data will contain all the data from R0 to R7 that is been used in the program.

Registers

Register	Value
R0	0
R1	1024
R2	10
R3	10
R4	0
R5	0
R6	0
R7	0

Memory

Location	value
100	0
101	0
102	0
103	0
104	0
105	0
106	0
107	0
108	0
109	0
110	0
111	0
112	0
113	0
114	0
115	0
116	0
117	0
118	0
119	0
120	0
121	0
122	0
123	0
124	0
125	0
126	0
127	0

11- The Memory will contain all the data that is stored in the memory.

Discussion

The First Program: a program calculating the factorial of a number

1-Level Cache
The Hit rate for the first program was very high compared to the miss rate, The reason is that the program is small containing only 5 instructions, the number of executed instructions is 21 as there exists branch and jump instructions but thanks to spatial locality they are cached since the block size is 4, it caches 4 instructions at a time leading to only 3 misses which are cold misses.

2-Level Cache
The first program was a successful one with 2 caches as there were only 4 misses due to accessing the blocks for the first time, which requires fetching them from memory afterwards no need for accessing cache 2 or memory as all instructions are in cache 1 which explains why cache 2 has 0 misses.

The Second Program: a program doing logical or mathematical operations depending on a condition

2-Level Caches
The Second program performance was too bad as the program was a sequential one so most of the instructions were accessed for the first time resulting in cold misses and level cache 1 was direct-mapped which leads to many conflicts as instructions are mapped to one location, Cache level 2 had a block size of 1 which means it only stores one instruction which lead to many misses at it need to load each instruction and it doesn't gain any benefit from the spatial locality.

3-Level Cache
The performance didn't improve much by adding a third level cache because the middle cache was a direct-mapped with block size of 2 words which didn't help the first cache to get what it needs from it, it had to either go to the third level cache or memory which leads to a great penalty.

The Third Program

2-Level Caches

The third program is simply a loop adding a number from 1 to n, so the spatial locality principle is used and the misses are few as the instructions within the same block are cached.

3-Level Cache

Same as 2 Level caches scenario, the hit rate is very high indicating that the cache stores the instructions in the loop results in low miss rate as the body of the loop is cached in level cache 1.

Work Distribution

Mohamed ElSaeed:

- Instructions package: classes, constructors, and the execute/0 methods.
- Cache package: classes, constructors, and the methods.

Mohab Ghanim:

- Project design.
- Simulator package: classes, constructors, and the methods.

Ahmed Moataz:

- Factories: the cacheFactory and the instructionFactory
- searchData/1 and searchInstruction/1 methods for the different types of caches.
- The insertData/1 method in the Cache abstract class.
- The Report.

Ahmed Magdi:

- GUI.
- The Report.