*Tomasulo Superscaler Simulator*

Ahmed Nader Mohamed 28-5986

Ahmad Osama Ibrahim 28-4199

Mayada Nour 28-3789

Mina Zeidan 28-4237

Raghda Khaled Soltan 28-3256

**Implementation Description:**

**The structure of the program** is as follows:

-Main Memory object containing a 2d array of strings representing the instructions, where each row in the array represents a block in the memory.

-Cache object having an array of cacheLevels objects where each level has its own geometry and write policy in case of a hit and access time in terms of cycles. And also has 3 arrays for tag, validity bits and data.

-General purpose registers object which has an array of Registers each having a value and status.

-Reservation Stations object having an array of Stations, each station object has name, busy, operation, Vj, Vk, Qj, Qk, Destination, Address, step to show tomasulo’s stage eg. Issue and finally cycles left which shows the number of cycles needed in the execute stage for the type of instruction in the station.

-Reorder buffer object having an array of ROBentry objects each having type, destination, value, ready, branchtaken.

-InputOutput class which takes all the input from the user and initializes all the objects in the program.

-Simulation class which simulates the four stages of tomasulo : Issue, Execute, Write, Commit.

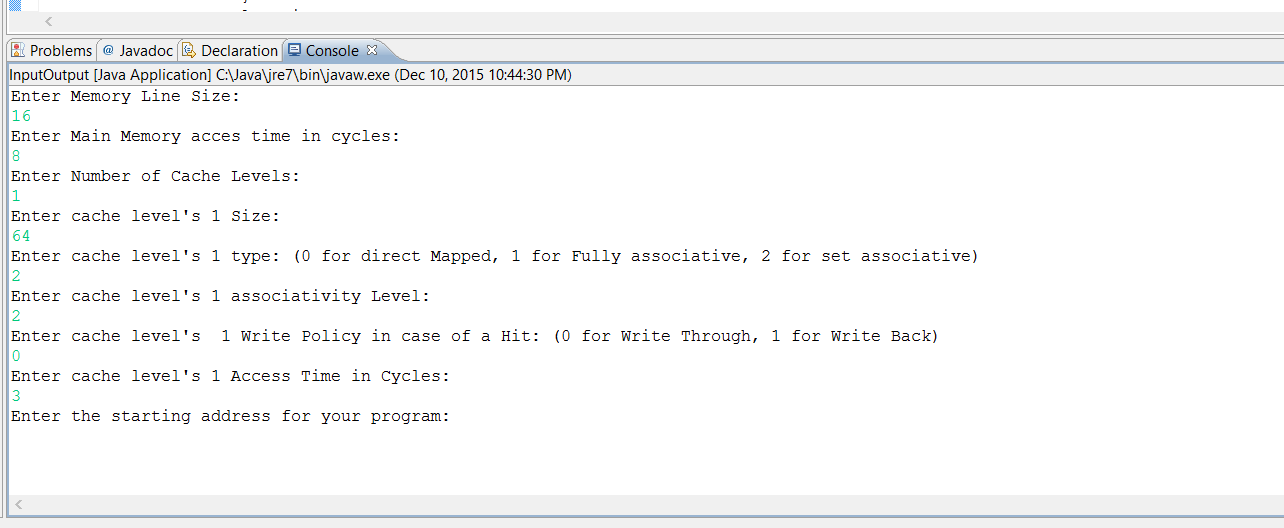
**Program flow:**

After running the program, the user at first enters all the info needed and an assembly program of arithmetic, load/ store and branching instructions. Then all the program objects are initialized with the input data and the program is at first stored as it is in Memory. Then each instruction is read from the Icache by looping through every cache level till getting a cache hit. In case of a cache miss then this instruction is fetched from the main Memory and then the whole block of this instruction is to be cached in all cache levels. Afterwards, all instructions are fetched at a time by placing them all in the instruction buffer, assuming that we ignore the cycle needed for fetching and start counting the program cycles in the Issue stage. In case of load and store instructions, address to be loaded from or stored into is to be stored in Dcache. Number of cycles needed for this is first calculated in Issue stage to identify the number of cycles needed for each of them in execute stage. Then instructions are issued, executed, written and committed according to Tomasulo’s algorithm.

**Dividing The work among the team members:**

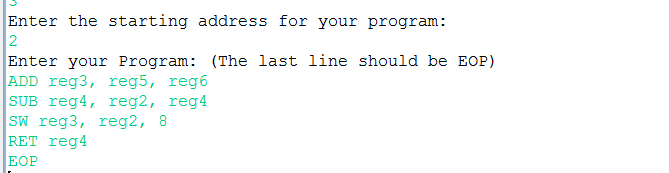
**User guide for the program:**

After running the program, the user should enter the line size in Bytes ex. 16. Then main memory access time in cycles, number of cache levels, size of each cache level, type of each level (0 for direct mapped , 1 for fully associative and 2 for set associative), then only in case of set associative, the user enters the associativity level. Then the write policy in case of a hit for each cache level (0 for write through, 1 for write back). And the access cycles for each level.

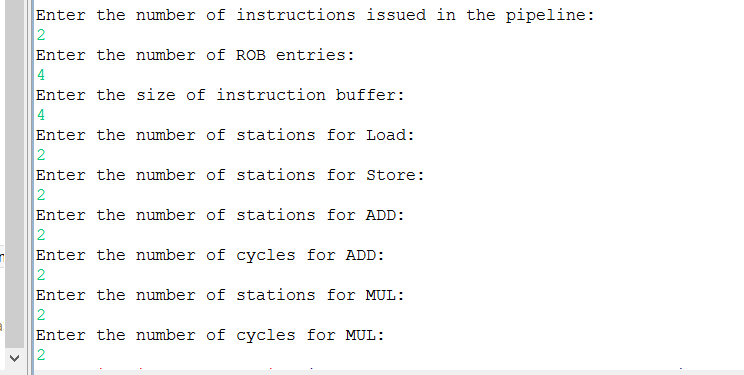


Then the user is asked to enter the start address for the program and the program itself.

The program instructions should be exactly like the given instructions in the project description, in terms of the instruction type and spaces left between each register, .. etc. The last line in the input program should be EOP.



Then the user is asked to enter the number of pipeline instructions, the number of rob and instruction buffer entries, which should be greater than or equal to the number of instructions written, and number of reservation stations for each Functional unit, and number of cycles needed to execute each instruction type except for load and store whose cycles are calculated through the program.



**How The work was Split:**

The work was split between us into parts: Memory Hierarchy including Caches and This part Ahmad Osama and Mayada Nour implemented it.

The Other part was taking the input from the user and the simulation part for Tomasulo and that was what Ahmed Nader, Mina Zeidan and Raghda Khaled did. Where Ahmed Nader implemented the structure of tomasulo and all the needed classes such as ROBentry, ROB, Station, Reservationstations and Simulation. And also implemented the issue stage and the helper methods related to it. Mina implemented the execute and write stages and Raghda implemented the commit stage and all the output data of the code as the total cycles, IPC, AMAT.

**Test Programs:**

1st program, With first used configuration:

Add reg1, reg2, reg1

mul reg3, reg3, reg4

SW reg3, reg2, 9

ADDI reg2, reg3, reg4

Nand reg2, reg3, reg5

Jmp reg3, 5

EOP

Used hardware configuration:

1 level of cache, line size is 32 bytes and cache size is 256 bytes, type of cache was Direct mapped.

4 ROB entries

OUTPUT:

Cycles: 11

Icache level 1's hit ratio is: 1.0

Dcache level 1's hit ratio is: 1.0

Icache's AMAT is: 3.0

Dcache's AMAT is: 3.0

Icache's IPC is: 1.0

Dcache's IPC is: 1.0

1st program, With second used configuration:

Add reg1, reg2, reg1

mul reg3, reg3, reg4

SW reg3, reg2, 9

ADDI reg2, reg3, reg4

Nand reg2, reg3, reg5

Jmp reg3, 5

EOP

Used hardware configuration:

3 level of cache, 1st is fully associative and 2nd and 3rd are direct mapped, line size is 16 bytes and cache size is 128 bytes, type of cache was Direct mapped.

Cycles: 21

Icache level 1's hit ratio is: 1.0

Dcache level 1's hit ratio is: 1.0

Icache level 2's hit ratio is: 1.0

Dcache level 2's hit ratio is: 1.0

Icache level 3's hit ratio is: 1.0

Dcache level 3's hit ratio is: 1.0

Icache's AMAT is: 3.0

Dcache's AMAT is: 3.0

Icache's IPC is: 1.0

Dcache's IPC is: 1.0

2nd Program:

LW reg2, reg3, 5

MUL reg2, reg2, reg5

SUB reg3, reg2, reg1

SW reg2, reg3, 4

EOP

Used hardware configuration:

1 level of cache, line size is 16 bytes and cache size is 256 bytes, type of cache was Direct mapped.

4 ROB entries.

OUTPUT:

Cycles: 26

Icache level 1's hit ratio is: 1.0

Dcache level 1's hit ratio is: 1.0

Icache's AMAT is: 3.0

Dcache's AMAT is: 3.0

Icache's IPC is: 1.0

Dcache's IPC is: 1.0

2nd Program:

LW reg2, reg3, 5

MUL reg2, reg2, reg5

SUB reg3, reg2, reg1

SW reg2, reg3, 4

EOP

Used hardware configuration:

2 level of cache, 1st is set associative with size 64 bytes, line size is 8 bytes and 2nd is direct mapped with size 256 bytes, line size is 16 bytes type of it was Direct mapped.

3 ROB entries.

OUTPUT:

Cycles: 39

Icache level 1's hit ratio is: 1.0

Dcache level 1's hit ratio is: 1.0

Icache level 2's hit ratio is: 1.0

Dcache level 2's hit ratio is: 1.0

Icache's AMAT is: 2.0

Dcache's AMAT is: 2.0

Icache's IPC is: 1.0

Dcache's IPC is: 1.0

3rd Program:

Lw reg3, reg5, 5

ADD reg2, reg2, reg5

Sw reg2, reg5, 5

Beq reg2, reg3, -3

ret reg3

EOP

* However this program had a loop so the output was not correct for that program.

Discussion for Results obtained:

1st program had only 11 cycles as the number of execute cycles for Add and MUL instructions were only 2 and 3 cycles. While the 2nd configuration for the first program had 21 cycles as Add and Mul instruction were 5 and 7 cycles.

Also both had hit ratio 100% as when we get a cache miss for the first instruction then all the block is cached and as the number of instructions in the program can all be included in 1 block so all the following instructions are cached.

2nd Program spanned 39 cycles as the Add instruction took 21 cycles to execute.