Project 2: femTomas

CSCE 3301 - Computer Architecture (Spring 2023)

The American University in Cairo

Amer Elsheikh	**900196010**
Gehad Ahmed	**900205068**

Dr. Cherif Salama

THE AMERICAN UNIVERSITY IN CAIRO

Implementation:

In our implementation, we have three main structures:

Instruction Class:

In this class, we handle reading the instruction and extracting all the necessary information from it like the different registers, and immediate. Also, it holds the clock cycle time values of the different stages of the algorithm for each instruction, so to be printed at the end of the program execution.
Also, we have an enum with the different instruction types we have to ensure that we work with human readable variables not run indices in the array.

ReservationStation Struct:

In this struct, we add all the details related to each functional unit in the reservation station.

Tomasulo Class:

In this class, we handle all the implementation of the Tomasulo simulation.
We have different functions to handle each stage (issue, execute, write back), each has its own logic with some special handling of some dependencies.

We have a function execution_logic that has the main logic for each instruction to be used in the execute stage.

We also have some reading functions (read_inst, read_hardware, and read_mem), to handle different file reading. Also in the read_hardware, it either creates a default hardware with the number specified in the project description or reads the hardware description provided by the user.

Bonus

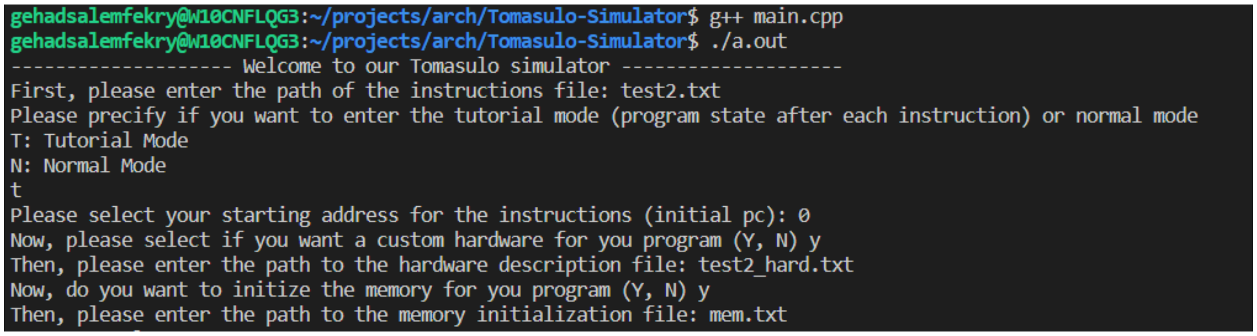
We supported a variable hardware organization. The user specifies the number of reservation stations for each class of instructions, and specifies the number of cycles needed by each functional unit type in the format (#units, execute_cycles, address_cycles_if_any) in a text file

User Guide:

Our simulator is implemented using C++, you can run it using any C++ compiler (IDE or from terminal).

Then, when the program start, you get these questions to answer:

- Path of the instruction file
- If the program to be run in tutorial mode or normal mode
- Starting address for the instructions
- If the program work with custom hardware or not, and path of this file
- If the program work with memory initialization or not and path of this file



```
gehadsalemfekry@W10CNFLQG3:~/projects/arch/Tomasulo-Simulator$ g++ main.cpp
gehadsalemfekry@W10CNFLQG3:~/projects/arch/Tomasulo-Simulator$ ./a.out
-------------------- Welcome to our Tomasulo simulator --------------------
First, please enter the path of the instructions file: test2.txt
Please precify if you want to enter the tutorial mode (program state after each instruction) or normal mode
T: Tutorial Mode
N: Normal Mode
t
Please select your starting address for the instructions (initial pc): 0
Now, please select if you want a custom hardware for you program (Y, N) y
Then, please enter the path to the hardware description file: test2_hard.txt
Now, do you want to initize the memory for you program (Y, N) y
Then, please enter the path to the memory initialization file: mem.txt
```

For the format of the input files, we need to make sure that the files are following these specifications:

Instruction file:

Each line has one instruction.

The program is not case sensitive, but the instruction type has to match one of the specified instructions in the project.

Hardware file:

It has 8 lines representing the 8 functional units in the following order: LOAD, STORE, BNE, JUMP, ADDITION, NEG, NAND, SLL.

For the first 2 lines, they have 3 numbers: NumberOfUnits, CyclesComputeAddress and CyclesOfExecution
For the rest 6 lines, they have 2 numbers: NumberOfUnits and CyclesOfExecution

Memory file:

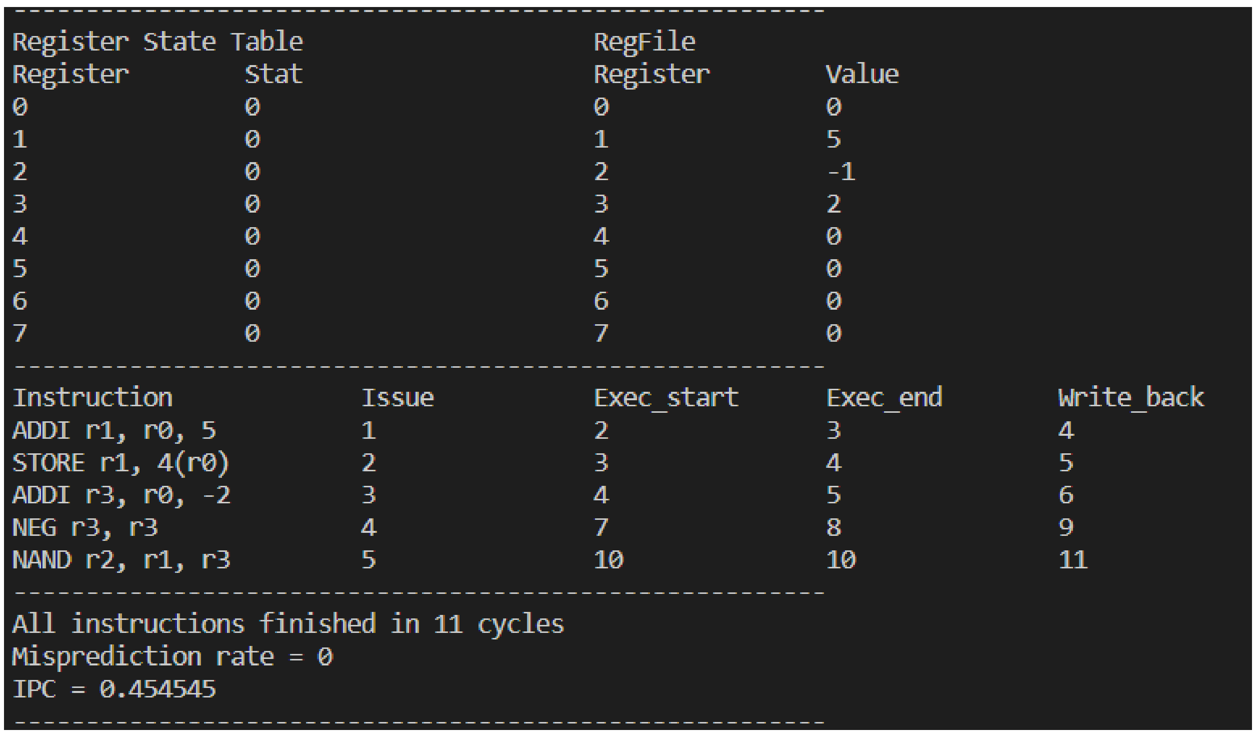
It is lines with 2 numbers in each line where the first number is the *memory address* and the second is the *memory value*.

Testing:

We created several tests to be able to run our Tomasulo simulation on all instructions with all scenarios and possible dependencies.

Test 1: Simple Test

In this test (test1.txt), we were testing that the program is functionable using a simple program, and the result was correct. The final register table along with the needed calculations are:



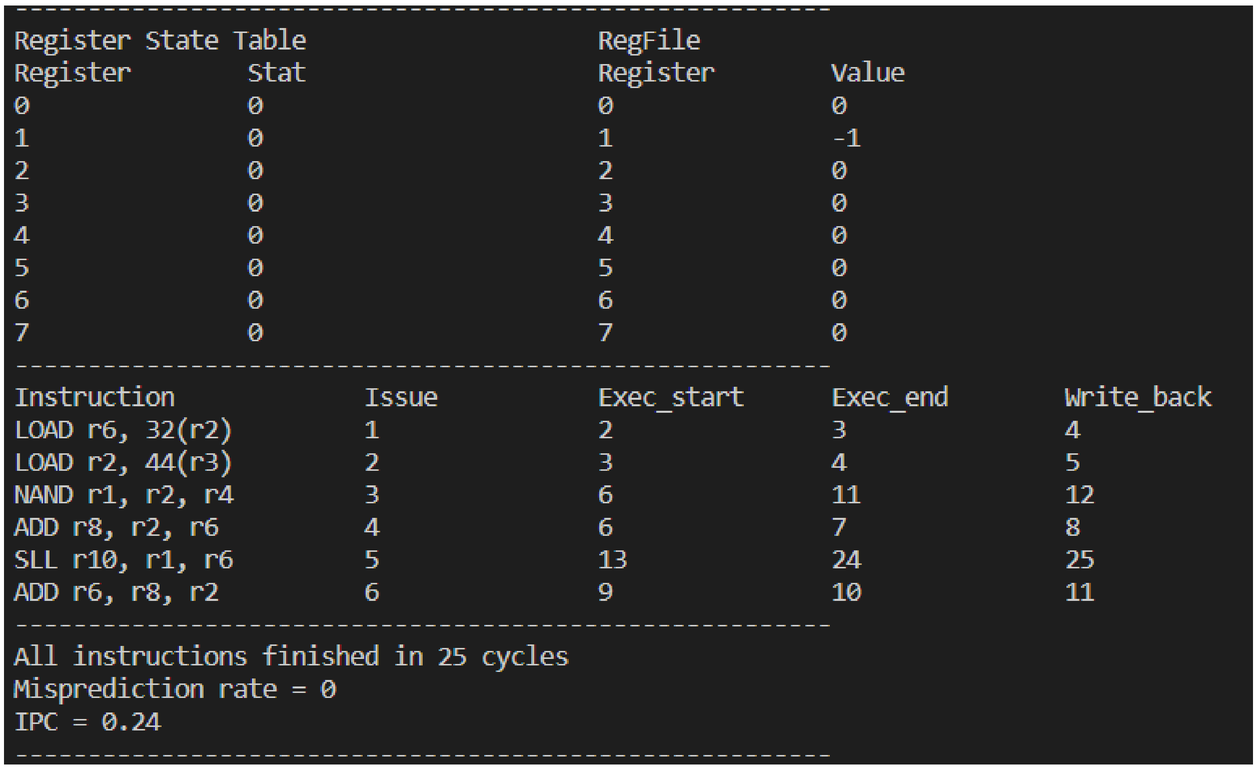
```
--------------------------------------------------------
Register State Table                RegFile
Register        Stat                Register        Value
0               0                   0               0
1               0                   1               5
2               0                   2               -1
3               0                   3               2
4               0                   4               0
5               0                   5               0
6               0                   6               0
7               0                   7               0
--------------------------------------------------------
Instruction             Issue           Exec_start      Exec_end        Write_back
ADDI r1, r0, 5          1               2               3               4
STORE r1, 4(r0)         2               3               4               5
ADDI r3, r0, -2         3               4               5               6
NEG r3, r3              4               7               8               9
NAND r2, r1, r3         5               10              10              11
--------------------------------------------------------
All instructions finished in 11 cycles
Misprediction rate = 0
IPC = 0.454545
--------------------------------------------------------
```

As seen, the final instructions (NAND) waited for NEG to write back before starting execution as there is a RAW dependency. Also, at the end the value of r2 was -1 as expected.

Test 2: Hardware

In this test (test2.txt, test2_hard.txt), we tested the algorithm using a user defined hardware description and see if it is working properly with the new assigned numbers of reservation stations or not and it worked.

Also this test was using the hardware specified (different instructions but same hardware needs) in one example in the course slides, so we compared our results with the slides and they are correct.



```
--------------------------------------------------------
Register State Table                    RegFile
Register        Stat                    Register        Value
0               0                       0               0
1               0                       1               -1
2               0                       2               0
3               0                       3               0
4               0                       4               0
5               0                       5               0
6               0                       6               0
7               0                       7               0
--------------------------------------------------------
Instruction             Issue           Exec_start      Exec_end        Write_back
LOAD r6, 32(r2)         1               2               3               4
LOAD r2, 44(r3)         2               3               4               5
NAND r1, r2, r4         3               6               11              12
ADD r8, r2, r6          4               6               7               8
SLL r10, r1, r6         5               13              24              25
ADD r6, r8, r2          6               9               10              11
--------------------------------------------------------
All instructions finished in 25 cycles
Misprediction rate = 0
IPC = 0.24
--------------------------------------------------------
```

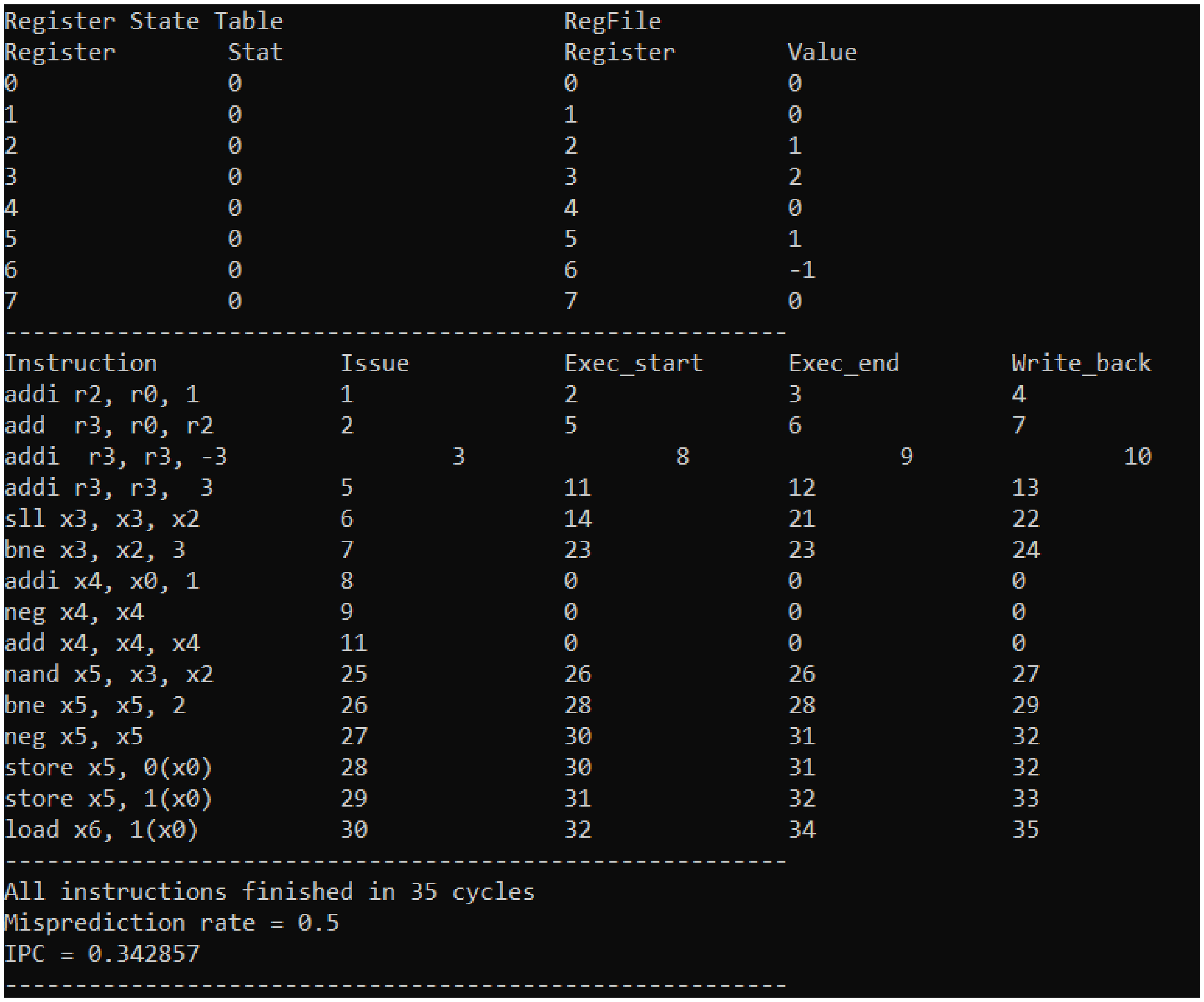
In this test, the hardware had the following specifications:

- 2 load units that need 1 cycle to compute the address and 1 cycle to read memory.
- 1 store unit that needs 1 cycle to compute address and 1 cycle to write to memory.
- 1 BNE unit that takes 2 cycles
- 1 JAL/RET unit that takes 1 cycle.
- 3 ADD/ADDI unit take 2 cycles
- 1 NEG unit take 1 cycle
- 1 NAND unit takes 6 cycles.
- 1 SLL unit takes 12 cycles.

<u>The table is the same as the one in the slides.</u>

Test 3: BNE

In this test (bne_test.txt), we made two BNE statements where the first one should be taken but the second should not.
As seen below, the misprediction rate is half as it should be.



```
Register State Table                       RegFile
Register        Stat                       Register        Value
0               0                          0               0
1               0                          1               0
2               0                          2               1
3               0                          3               2
4               0                          4               0
5               0                          5               1
6               0                          6               -1
7               0                          7               0
--------------------------------------------------------
Instruction             Issue           Exec_start      Exec_end        Write_back
addi r2, r0, 1          1               2               3               4
add  r3, r0, r2         2               5               6               7
addi  r3, r3, -3                3               8               9               10
addi r3, r3,  3         5               11              12              13
sll x3, x3, x2          6               14              21              22
bne x3, x2, 3           7               23              23              24
addi x4, x0, 1          8               0               0               0
neg x4, x4              9               0               0               0
add x4, x4, x4          11              0               0               0
nand x5, x3, x2         25              26              26              27
bne x5, x5, 2           26              28              28              29
neg x5, x5              27              30              31              32
store x5, 0(x0)         28              30              31              32
store x5, 1(x0)         29              31              32              33
load x6, 1(x0)          30              32              34              35
--------------------------------------------------------
All instructions finished in 35 cycles
Misprediction rate = 0.5
IPC = 0.342857
--------------------------------------------------------
```

Moreover, the two instructions after the first BNE (the one that is taken) were not executed (their exec_start, exec_end, and write_back are zeros), but they were issued as they should, and then they were flushed (we left the issued cycle printed to verify that).

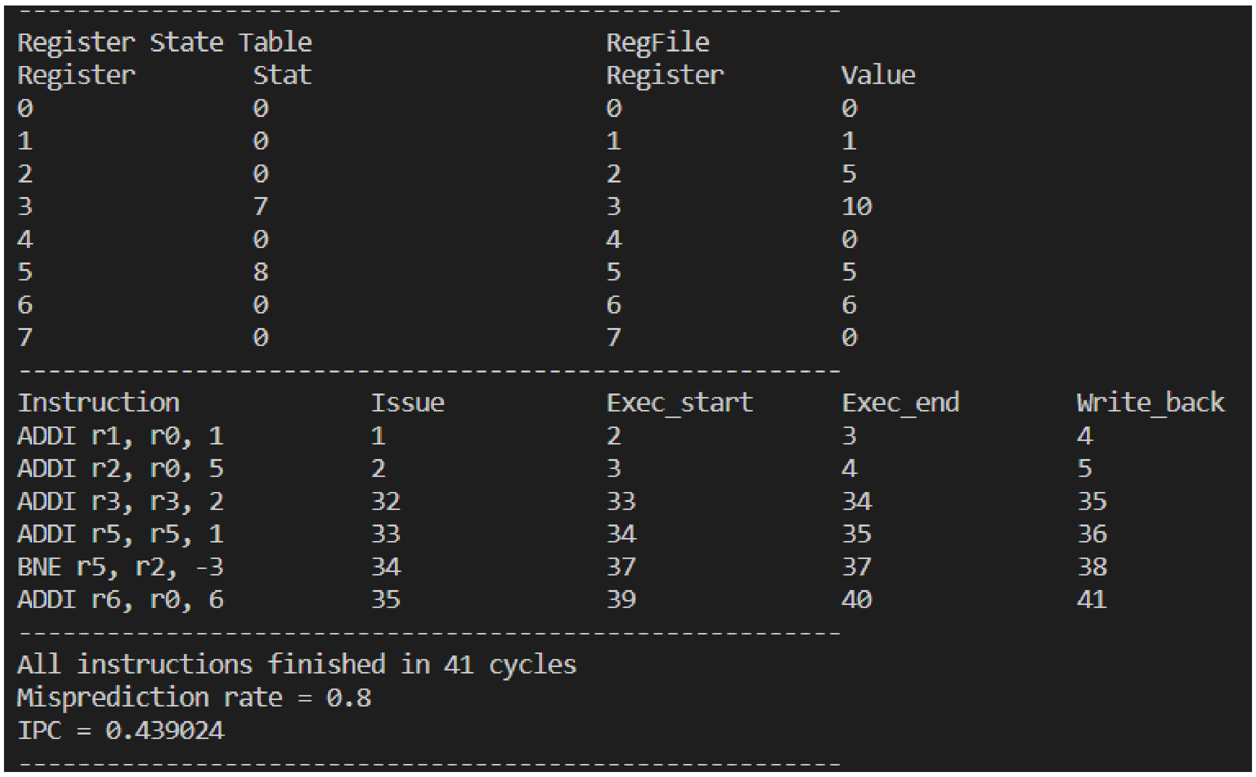
However, for the nand instruction (that we branch to), it was issued at cycle 25 (1 cycle after the BNE wrote back) which is correct as it was first issued at cycle 12 but it was then flushed and issued correctly at cycle 25.

Notice that for the second BNE, the issuing happened normally without flushing since it was not taken. However, the execution for NEG (the first one after the second BNE) waited until the BNE to write back.

Test 4: Loop

In this test (test_loop.txt), we made a loop that made 5 iterations and added 2 to register 3 in each iteration.

We can see in that r3 has the value 10 as expected



```
--------------------------------------------------------
Register State Table            RegFile
Register        Stat            Register        Value
0               0               0               0
1               0               1               1
2               0               2               5
3               7               3               10
4               0               4               0
5               8               5               5
6               0               6               6
7               0               7               0
--------------------------------------------------------
Instruction             Issue           Exec_start      Exec_end        Write_back
ADDI r1, r0, 1          1               2               3               4
ADDI r2, r0, 5          2               3               4               5
ADDI r3, r3, 2          32              33              34              35
ADDI r5, r5, 1          33              34              35              36
BNE r5, r2, -3          34              37              37              38
ADDI r6, r0, 6          35              39              40              41
--------------------------------------------------------
All instructions finished in 41 cycles
Misprediction rate = 0.8
IPC = 0.439024
--------------------------------------------------------
```

One of our assumptions is that we display the last issue time for each instruction, so in the loop, the instructions in the loop body get issued 5 times and only the last one is shown.

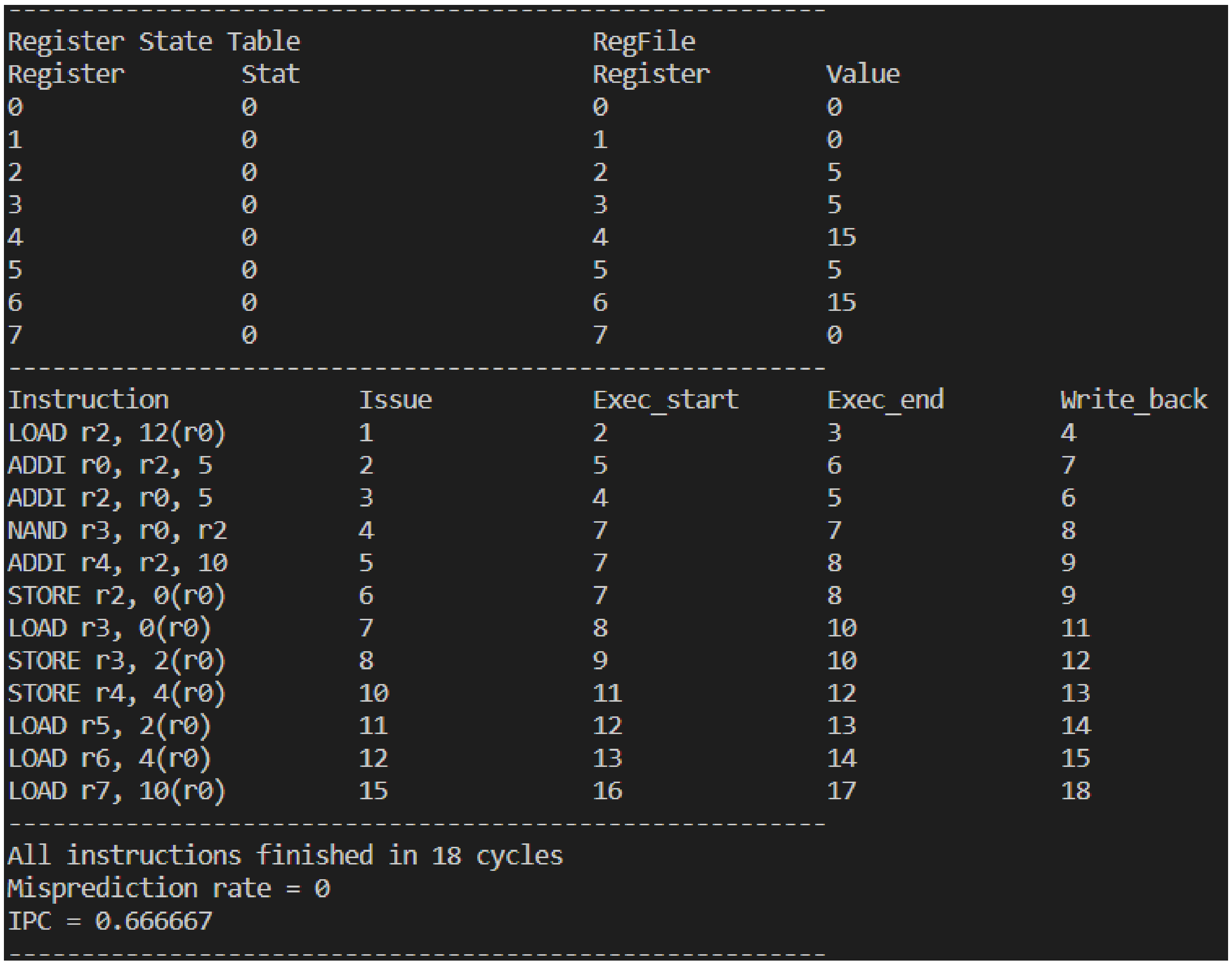
Also, in this program the BNE statement got executed 5 times where 4 of them was taken and the last one was not taken, and this is shown in the misprediction rate which is ⅘ = 0.8 as shown in the screenshot.

Test 5: Memory

In this test (test_mem.txt, mem.txt), we wanted to test reading data from memory, load and store dependencies and the correctness of loading and storing to the memory.

We have stored some values and then loaded them back in other registers and the result is correct..

We can see that r5 and r6 has the same value as r3 and r4, which was done after storing and then loading the data from the memory.



```
-------------------------------------------------------
Register State Table                   RegFile
Register        Stat                   Register        Value
0               0                      0               0
1               0                      1               0
2               0                      2               5
3               0                      3               5
4               0                      4               15
5               0                      5               5
6               0                      6               15
7               0                      7               0
-------------------------------------------------------
Instruction             Issue          Exec_start      Exec_end        Write_back
LOAD r2, 12(r0)         1              2               3               4
ADDI r0, r2, 5          2              5               6               7
ADDI r2, r0, 5          3              4               5               6
NAND r3, r0, r2         4              7               7               8
ADDI r4, r2, 10         5              7               8               9
STORE r2, 0(r0)         6              7               8               9
LOAD r3, 0(r0)          7              8               10              11
STORE r3, 2(r0)         8              9               10              12
STORE r4, 4(r0)         10             11              12              13
LOAD r5, 2(r0)          11             12              13              14
LOAD r6, 4(r0)          12             13              14              15
LOAD r7, 10(r0)         15             16              17              18
-------------------------------------------------------
All instructions finished in 18 cycles
Misprediction rate = 0
IPC = 0.666667
-------------------------------------------------------
```

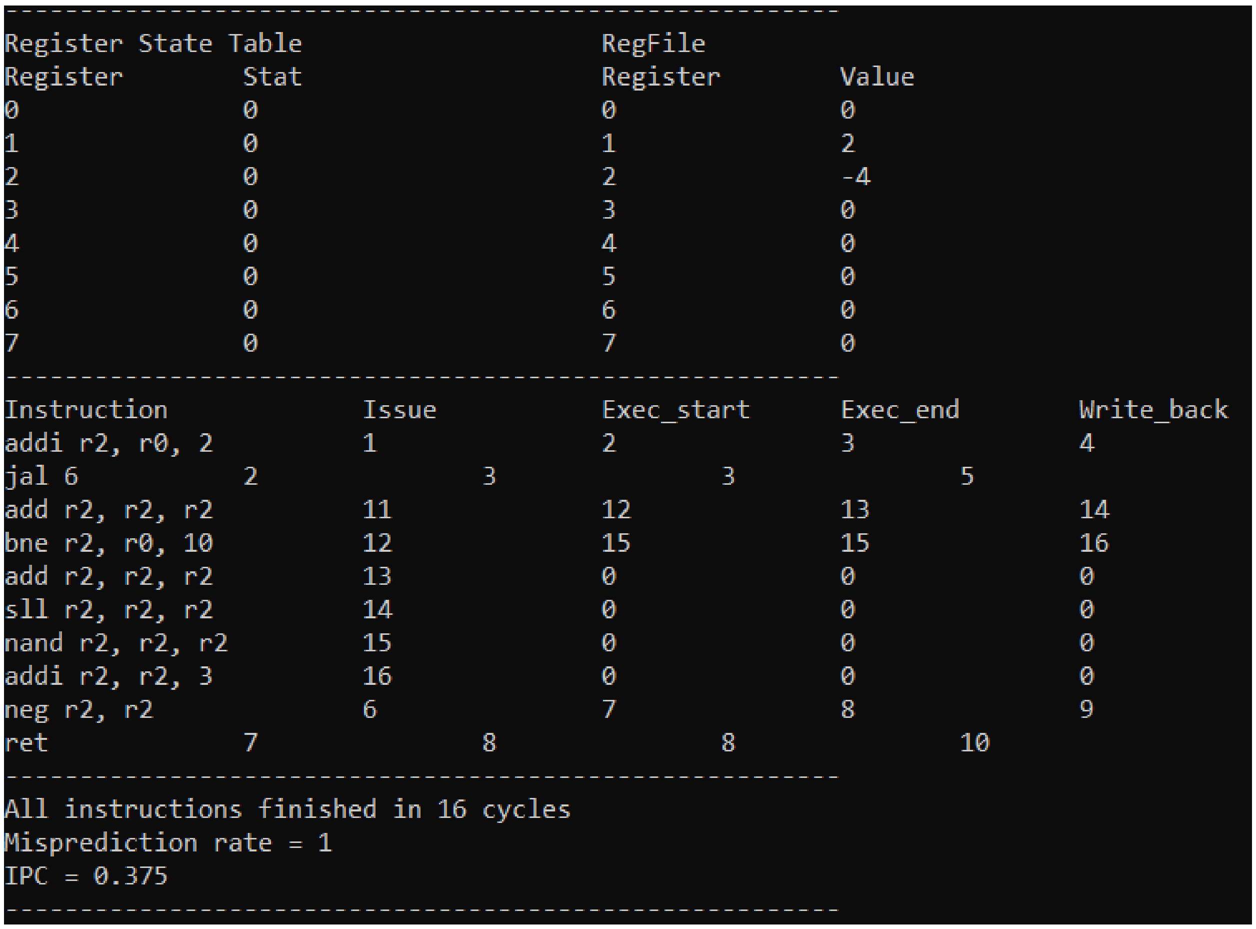
As the Tomasulo algorithm doesn't handle WAR, RAW, WAW dependencies in the memory itself, we handled them separately in our code, and ensured to test them as seen in the screenshot.

We can see in the “STORE r2, 0(r0)” and “LOAD r3, 0(r0)” as an example for RAW dependency, the execution of the load took 3 cycles (1 for address calculation + 2 for reading memory) instead of normal 2 cycles. That happened because its reading from the memory stalled for 1 cycle until the previous store finished writing. So, it only began reading at cycle 10, and it stalled at cycle 9, and it calculated the address at cycle 8.

Test 6: Jump and Ret

In this test (jump_test.txt), we tested the functionality of JAL and RET works. As seen below, we initialize r2 to 2 and then jump to the function that negates it, then r2 is -2. Then, we return back to double r2, so it becomes -4 and then the branch jumps outside the program to terminate it.

The program ran correctly, and r2 became -4 at the end. Moreover, the value of r1 was set to 2 which is the correct location of the instruction directly after jal.



```
--------------------------------------------------------
Register State Table                RegFile
Register        Stat                Register        Value
0               0                   0               0
1               0                   1               2
2               0                   2               -4
3               0                   3               0
4               0                   4               0
5               0                   5               0
6               0                   6               0
7               0                   7               0
--------------------------------------------------------
Instruction             Issue           Exec_start      Exec_end        Write_back
addi r2, r0, 2          1               2               3               4
jal 6           2               3               3               5
add r2, r2, r2          11              12              13              14
bne r2, r0, 10          12              15              15              16
add r2, r2, r2          13              0               0               0
sll r2, r2, r2          14              0               0               0
nand r2, r2, r2         15              0               0               0
addi r2, r2, 3          16              0               0               0
neg r2, r2              6               7               8               9
ret             7               8               8               10
--------------------------------------------------------
All instructions finished in 16 cycles
Misprediction rate = 1
IPC = 0.375
--------------------------------------------------------
```

Notice that the four instructions after BNE were flushed and not executed.

Another thing to note in the test is that the jal instruction didn't write back in cycle 4. Instead, it waited until cycle 5 since the Common Data Path was already used in cycle 4 by addi.

What Works

All instructions work along with support for user specified hardware

What Does not Work

Nothing