Computer architecture - Fall 2020
Project 2 - Tomasulu simulator
Team Members:
- Mina Ashraf Gamil - 900182973
- Sherif Gabr - 900183120

Bonus features implemented:

1. RISC-V Assembly parser with instructions fed from a text file named “instructions.txt” that can be changed to include any amount or type of supported instructions.
2. Supported a variable hardware organization where the user can enter the number of reservation stations as well as the number of cycles each of them take to finish. **(counted as two bonuses)**
3. Supported the educational idea without making it web-based. The user can step in the program and see what happens each cycle, but in the terminal not in a browser.

Brief description of the project:

The project is built in an object-oriented fashion where each component of the system has its own class of attributes and methods.

Classes in our project:

- **Tomasulu class**: The main driver of the simulation and the container of all other classes.
- **Instruction class**: A class that contained all the information for each instruction, such as: issue time, execution start and end, write back time, operation type and which reservation station is responsible for handling this instruction.
- **Reservation station class**: A class that represents each reservation station and the data needed for its successful operation, such as: busy flag, Vj, Vk, Qj, Qk, and the result.
- **Register status class**: a class that marks which reservation station is currently producing the corresponding register, and whether this register is ready or not.

Data and Logic flow of the simulation:

- The simulation starts by loading the assembly instructions into our instruction memory. Then, our parser extracts the data from each instruction based on its type so that the destination register and the two source registers are initialized successfully inside the instruction class. It also handles instructions that do not have any destination or sources that are replaced with immediates.

- Then the actual simulation starts by repeatedly calling the three main functions: Issue, Execute, and Write Back until we are done. The stopping condition is when all reservation stations are empty as well as we are below a certain amount of cycles that is dynamically determined by the amount of instructions in the instruction memory.

- **Issue()**
    - Before issuing an instruction, we check if there is an available reservation station that can be assigned to it. We also need to make sure that we can issue any instructions. The only two conditions that can prevent the issuing of new instructions are when we have already issued a JALR or RET instruction since they are unconditional jumps.
    - If a previous instruction was a BEQ instruction, we store all subsequent instructions' reservation stations in a buffer. Since our default branch prediction is that the branch is not taken, we flush these reservation stations only if we found that the branch was taken. Otherwise, there is no need to change anything.
    - We mark the destination register as busy and being produced by our reservation station on the Register Status vector. Then, we fill all the data: Vj, Vk, Qj, Qk, and other needed values in the reservation station.
- **Execute()**
    - Now, we are ready to execute the instructions that have been issued, but before doing so, we check if some conditions are met successfully.
    - The conditions are:
        - The reservation station should be already busy
        - The instruction in the reservation station should have been issued in a previous cycle as we cannot execute an instruction in the same cycle it was issued in.
        - We need to check that we have not already finished executing this instruction by checking if the end time is now larger than the issue time plus the time needed for this functional unit to finish execution.
        - Finally, we need to make sure that we are not executing any instruction after we have met a branch that we don't know whether it is going to be taken or not. Remember, we never execute based on a prediction.

    - If these conditions are met, we keep counting cycles for each reservation station until the last cycle where we finally execute it.
    - For Branch instructions, we can now check whether the branch was to be taken or not, and we can consequently decide whether to flush the used reservation stations or not.
    - Once we finish executing the instructions, we mark the result as ready so that we can forward it to other reservation stations if needed.

- **WriteBack()**
    - Now we check for all reservation stations if the result is ready and that execution was already finished or not. If that was the case, we can write back.
    - We free the reservation station and decrement the counter of used reservation stations.
    - We also set the register status to indicate that the value is no longer benign produced by a reservation station.
    - Now wel loop over all other reservation stations, and if there is a reservation station waiting for our destination, we send the values to it and check if we can execute it or there are other dependencies.
    - For store instructions, we can now write their value in the memory.

That’s it.

User Guide

1. To execute program:
   a. Windows: `g++ -I . main.cpp Instruction.cpp ReservationStation.cpp Tomasulu.cpp Rstat.cpp -o main.exe`
   b. MAC/Linux: `g++ -I . main.cpp Instruction.cpp ReservationStation.cpp Tomasulu.cpp Rstat.cpp -o main`

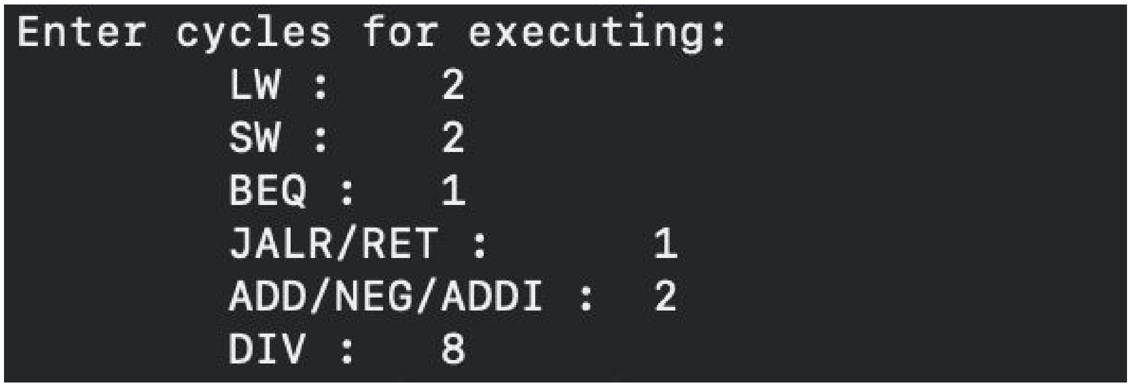
   The previous command is to be run in the terminal.

2. The program takes the assembly instructions as input in file “instructions.txt” that must be located in the same directory as the executable. Only the following set operands are compatible with the program and they must be uppercase:

   LW SW BEQ JALR RET ADD ADDI NEG DIV

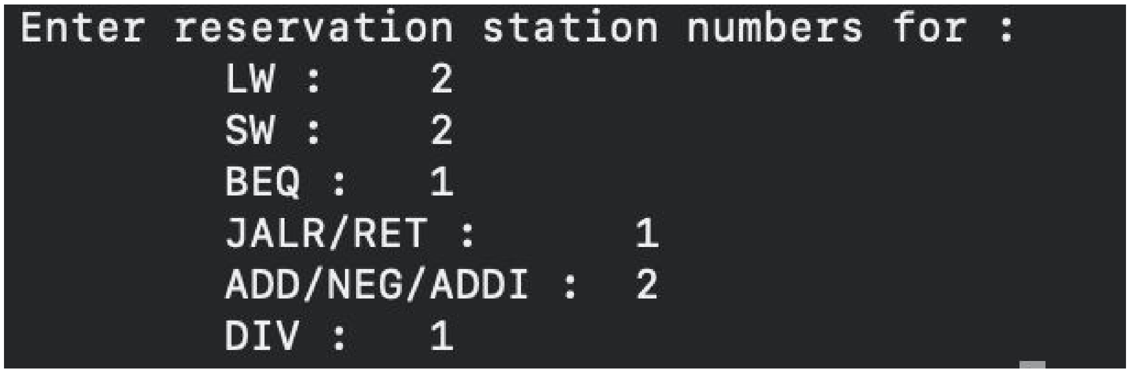
3. The user is prompted with the message to enter the first instruction’s address, which is the address from which the program is loaded to memory.

4. The user is asked to enter the number of cycles it takes each set of instructions to complete execution. Snapshot from program:



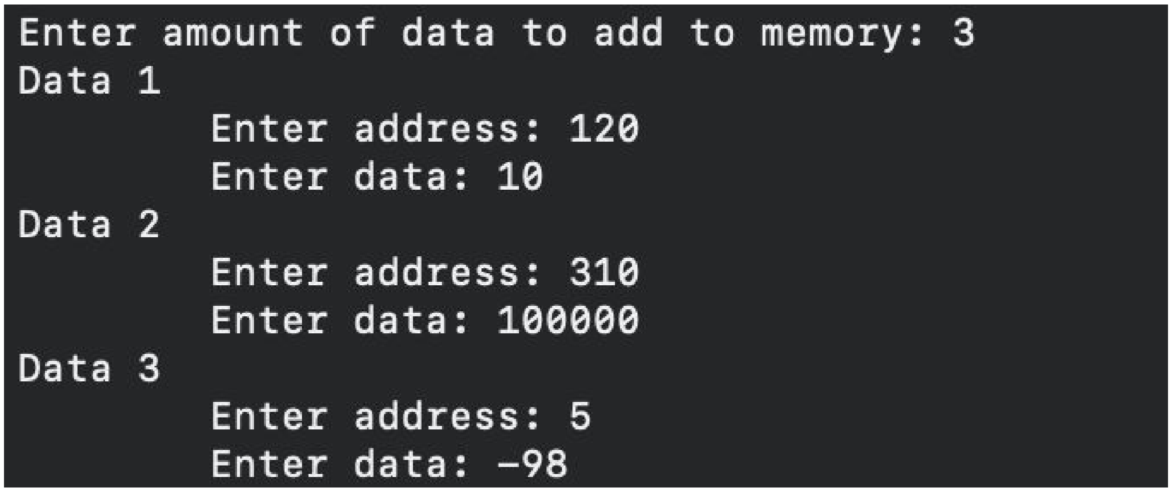
```
Enter cycles for executing:
        LW :    2
        SW :    2
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  2
        DIV :   8
```

5. The user is asked to enter the number of reservation stations for each set of instructions. Snapshot from program:



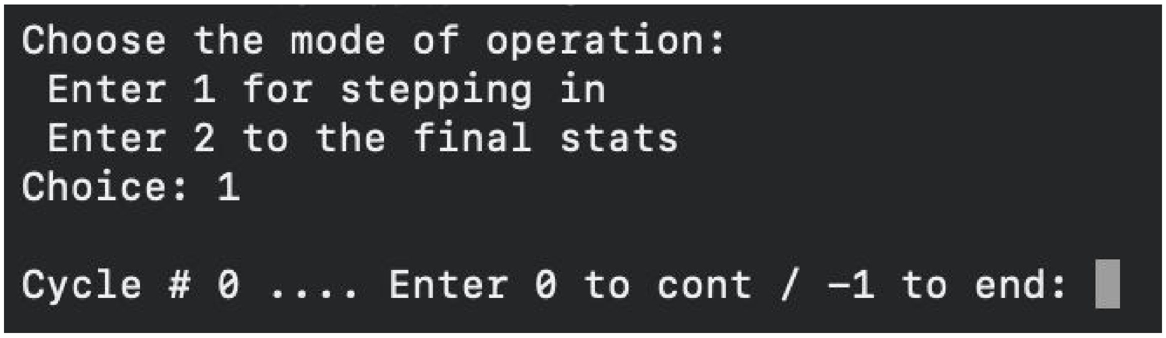
```
Enter reservation station numbers for :
        LW :    2
        SW :    2
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  2
        DIV :   1
```

6. The user is then prompted to enter the number of values the user wishes to initialize the memory with.
7. For every value to memory, the user must enter the address to memory, with the maximum being 320, and the value to insert into memory.



```
Enter amount of data to add to memory: 3
Data 1
        Enter address: 120
        Enter data: 10
Data 2
        Enter address: 310
        Enter data: 100000
Data 3
        Enter address: 5
        Enter data: -98
```

8. There are 2 modes of operation from which the user can pick. The “stepping in” mode and the “final” mode. In the stepping in mode, the user must enter 1 when prompted with type of mode and in this mode the user can see instructions’ and registers’ status at every clock cycle until completion, by pressing 0 at every cycle. Also upon entering this mode, the user can switch to the final mode whenever they choose by entering -1. In the final mode, the final stats of the program is displayed which includes cycle times it took instruction to finish the issue, execute and writeback stage, along with the final values of the register file with the register status. Other stats include the total execution time of the assembly program, the instruction per cycle (IPC) of the program, and the branch misprediction percentage.



```
Choose the mode of operation:
 Enter 1 for stepping in
 Enter 2 to the final stats
Choice: 1

Cycle # 0 .... Enter 0 to cont / -1 to end:
```

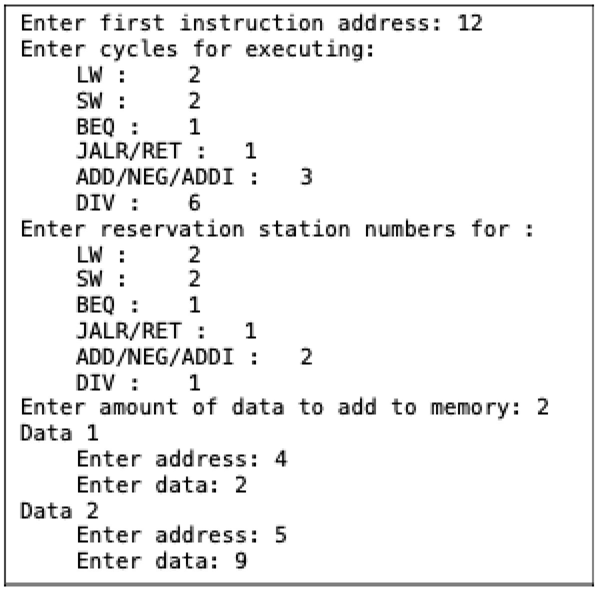
Tests

- Test case 1 (Looping)

Assembly Program:

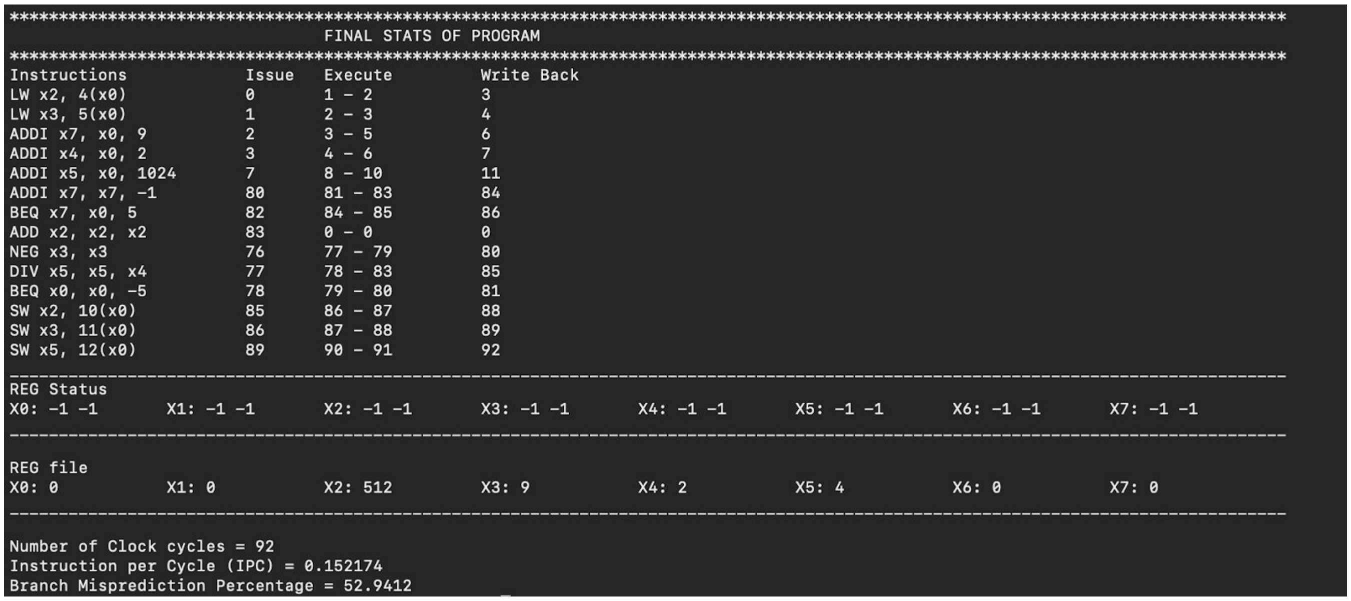
```
LW x2, 4(x0)
LW x3, 5(x0)
ADDI x7, x0, 9
ADDI x4, x0, 2
ADDI x5, x0, 1024
ADDI x7, x7, -1
BEQ x7, x0, 5
ADD x2, x2, x2
NEG x3, x3
DIV x5, x5, x4
BEQ x0, x0, -5
SW x2, 10(x0)
SW x3, 11(x0)
SW x5, 12(x0)
```

Input:



```
Enter first instruction address: 12
Enter cycles for executing:
    LW :    2
    SW :    2
    BEQ :   1
    JALR/RET :   1
    ADD/NEG/ADDI :   3
    DIV :   6
Enter reservation station numbers for :
    LW :    2
    SW :    2
    BEQ :   1
    JALR/RET :   1
    ADD/NEG/ADDI :   2
    DIV :   1
Enter amount of data to add to memory: 2
Data 1
    Enter address: 4
    Enter data: 2
Data 2
    Enter address: 5
    Enter data: 9
```

Output:



```
********************************************************************************************************************************
                                FINAL STATS OF PROGRAM
********************************************************************************************************************************
Instructions            Issue   Execute         Write Back
LW x2, 4(x0)            0       1 - 2           3
LW x3, 5(x0)            1       2 - 3           4
ADDI x7, x0, 9          2       3 - 5           6
ADDI x4, x0, 2          3       4 - 6           7
ADDI x5, x0, 1024       7       8 - 10          11
ADDI x7, x7, -1         80      81 - 83         84
BEQ x7, x0, 5           82      84 - 85         86
ADD x2, x2, x2          83      0 - 0           0
NEG x3, x3              76      77 - 79         80
DIV x5, x5, x4          77      78 - 83         85
BEQ x0, x0, -5          78      79 - 80         81
SW x2, 10(x0)           85      86 - 87         88
SW x3, 11(x0)           86      87 - 88         89
SW x5, 12(x0)           89      90 - 91         92
--------------------------------------------------------------------------------------------------------------------------------
REG Status
X0: -1 -1       X1: -1 -1       X2: -1 -1       X3: -1 -1       X4: -1 -1       X5: -1 -1       X6: -1 -1       X7: -1 -1
--------------------------------------------------------------------------------------------------------------------------------

REG file
X0: 0           X1: 0           X2: 512         X3: 9           X4: 2           X5: 4           X6: 0           X7: 0
--------------------------------------------------------------------------------------------------------------------------------

Number of Clock cycles = 92
Instruction per Cycle (IPC) = 0.152174
Branch Misprediction Percentage = 52.9412
```

This assembly program is a simple loop that iterates 9 times, each time computing the addition, division, and negation.

Observations:

1. We can see that the instructions continue looping therefore issuing until the base case is met (counter reaches 0).

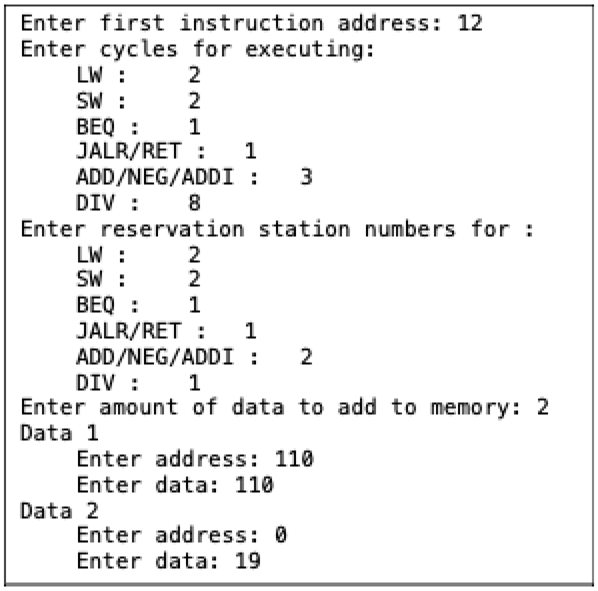
2. Since all instructions finishes writing back, all register are free (-1 -1)
3. The instruction after the first branch, in the last iteration, is issued, however, it is not executed since the branch is taken.
4. The correct values of the registers are written in the register file at the end
5. The total execution time, IPC, and branch misprediction are displayed

- Test Case 2 (Jumping)

Assembly Program:

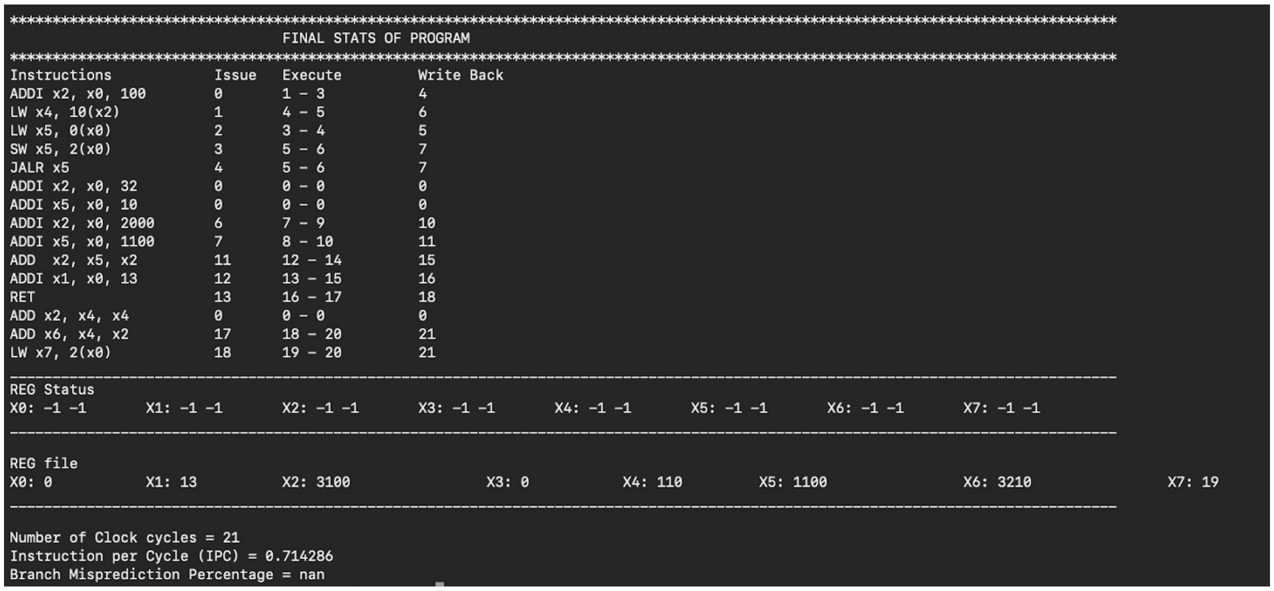
```
ADDI x2, x0, 100
LW x4, 10(x2)
LW x5, 0(x0)
SW x5, 2(x0)
JALR x5
ADDI x2, x0, 32
ADDI x5, x0, 10
ADDI x2, x0, 2000
ADDI x5, x0, 1100
ADD  x2, x5, x2
ADDI x1, x0, 13
RET
ADD x2, x4, x4
ADD x6, x4, x2
LW x7, 2(x0)
```

Input:



```
Enter first instruction address: 12
Enter cycles for executing:
    LW :    2
    SW :    2
    BEQ :   1
    JALR/RET :  1
    ADD/NEG/ADDI :  3
    DIV :   8
Enter reservation station numbers for :
    LW :    2
    SW :    2
    BEQ :   1
    JALR/RET :  1
    ADD/NEG/ADDI :  2
    DIV :   1
Enter amount of data to add to memory: 2
Data 1
    Enter address: 110
    Enter data: 110
Data 2
    Enter address: 0
    Enter data: 19
```

Output:



```
****************************************************************************************************************************
                                FINAL STATS OF PROGRAM
****************************************************************************************************************************
Instructions            Issue   Execute         Write Back
ADDI x2, x0, 100        0       1 - 3           4
LW x4, 10(x2)           1       4 - 5           6
LW x5, 0(x0)            2       3 - 4           5
SW x5, 2(x0)            3       5 - 6           7
JALR x5                 4       5 - 6           7
ADDI x2, x0, 32         0       0 - 0           0
ADDI x5, x0, 10         0       0 - 0           0
ADDI x2, x0, 2000       6       7 - 9           10
ADDI x5, x0, 1100       7       8 - 10          11
ADD  x2, x5, x2         11      12 - 14         15
ADDI x1, x0, 13         12      13 - 15         16
RET                     13      16 - 17         18
ADD x2, x4, x4          0       0 - 0           0
ADD x6, x4, x2          17      18 - 20         21
LW x7, 2(x0)            18      19 - 20         21
----------------------------------------------------------------------------------------------------------------------------
REG Status
X0: -1 -1       X1: -1 -1       X2: -1 -1       X3: -1 -1       X4: -1 -1       X5: -1 -1       X6: -1 -1       X7: -1 -1
----------------------------------------------------------------------------------------------------------------------------

REG file
X0: 0           X1: 13          X2: 3100                X3: 0           X4: 110         X5: 1100                X6: 3210                X7: 19
----------------------------------------------------------------------------------------------------------------------------

Number of Clock cycles = 21
Instruction per Cycle (IPC) = 0.714286
Branch Misprediction Percentage = nan
```

Observations:

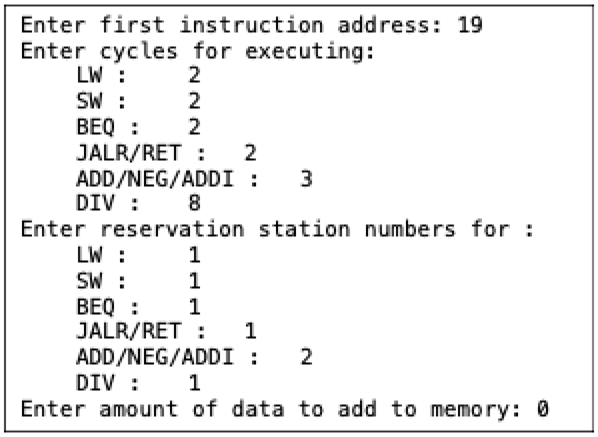
1. The value in x5, before the jalr instruction, is 19. The first instruction is loaded into memory at address 12 therefore, it will jump to the 8th instruction (19-12+1) which is the ADDI x2, x0, 2000
2. We can see that the instructions after the JALR did not issue. Likewise, the instruction after RET did not issue since it jumped over it.
3. The RET instruction jumps to the address stored in x1 (13) in that case to the instruction ADD x6, x4, x2. We can also see that the RET instruction did not begin execution until the previous instruction, which is writing to x1 finishes writing back.
4. The correct values of the registers are written to the register file at the end.

- Test Case 3: (Dependencies)

Assembly Program:

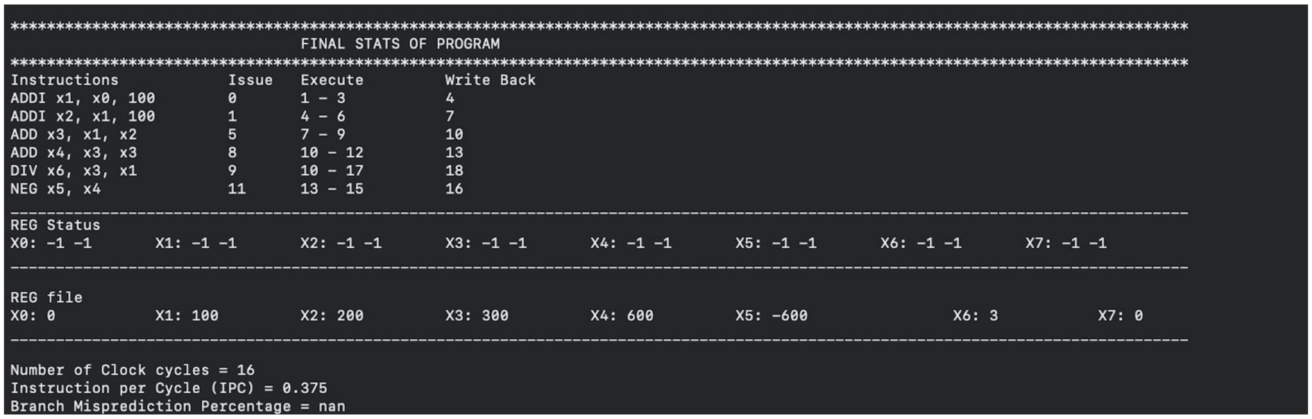
ADDI x1, x0, 100
ADDI x2, x1, 100
ADD x3, x1, x2
ADD x4, x3, x3
DIV x6, x3, x1
NEG x5, x4

Input:



```
Enter first instruction address: 19
Enter cycles for executing:
    LW :     2
    SW :     2
    BEQ :    2
    JALR/RET :   2
    ADD/NEG/ADDI :   3
    DIV :    8
Enter reservation station numbers for :
    LW :     1
    SW :     1
    BEQ :    1
    JALR/RET :   1
    ADD/NEG/ADDI :   2
    DIV :    1
Enter amount of data to add to memory: 0
```

Output:



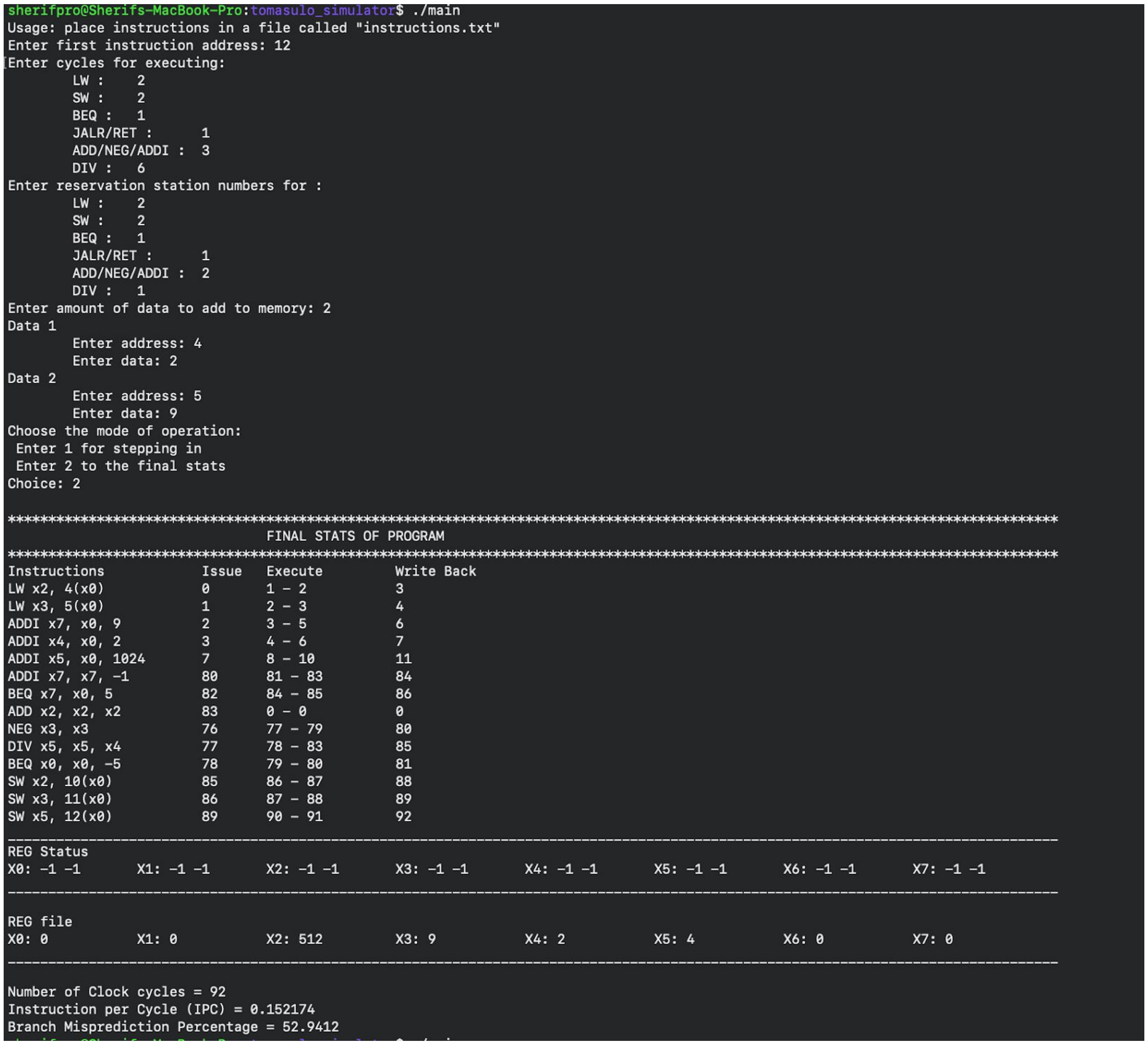
```
*****************************************************************************************************************
                                FINAL STATS OF PROGRAM
*****************************************************************************************************************
Instructions            Issue   Execute         Write Back
ADDI x1, x0, 100        0       1 - 3           4
ADDI x2, x1, 100        1       4 - 6           7
ADD x3, x1, x2          5       7 - 9           10
ADD x4, x3, x3          8       10 - 12         13
DIV x6, x3, x1          9       10 - 17         18
NEG x5, x4              11      13 - 15         16
_________________________________________________________________________________________________________________
REG Status
X0: -1 -1       X1: -1 -1       X2: -1 -1       X3: -1 -1       X4: -1 -1       X5: -1 -1       X6: -1 -1       X7: -1 -1
_________________________________________________________________________________________________________________

REG file
X0: 0           X1: 100         X2: 200         X3: 300         X4: 600         X5: -600                X6: 3           X7: 0
_________________________________________________________________________________________________________________

Number of Clock cycles = 16
Instruction per Cycle (IPC) = 0.375
Branch Misprediction Percentage = nan
```

Observations:

1. The 2nd instruction waits for the 1st instruction to finish writing back to reg x1 before beginning execution.
2. Since there are only 2 reservation stations for add/neg/addi, the 3rd instruction ADD x3, x1, x2 must wait for an empty reservation station, therefore it waits for an instruction to finish writing back in order to be issued.
3. The 4th instruction waits for the 3rd instruction to finish writing back to get x3 before beginning execution.
4. The correct values of the registers are written to the register file at the end.

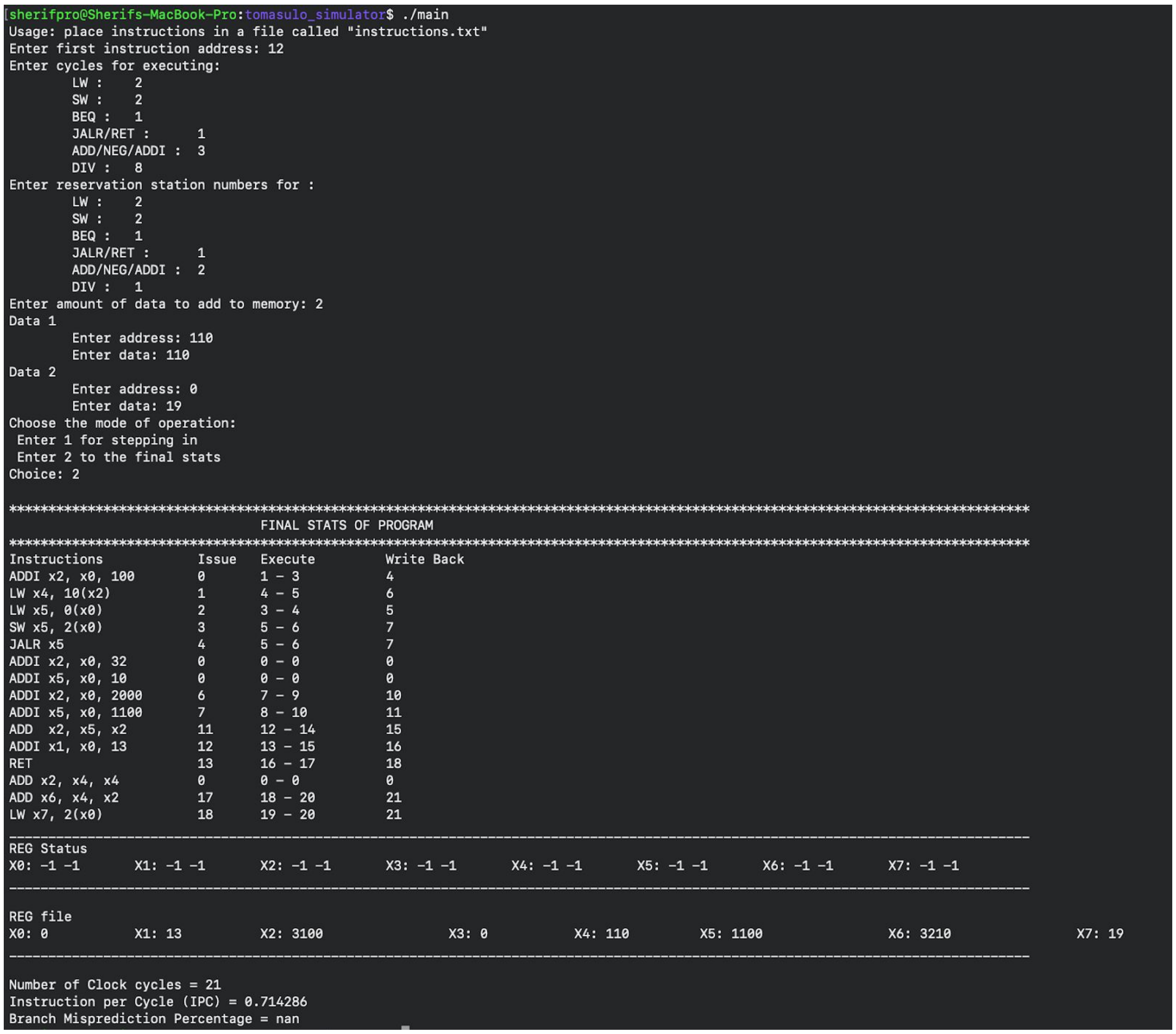
Complete Screenshots of Tests:

Test1:



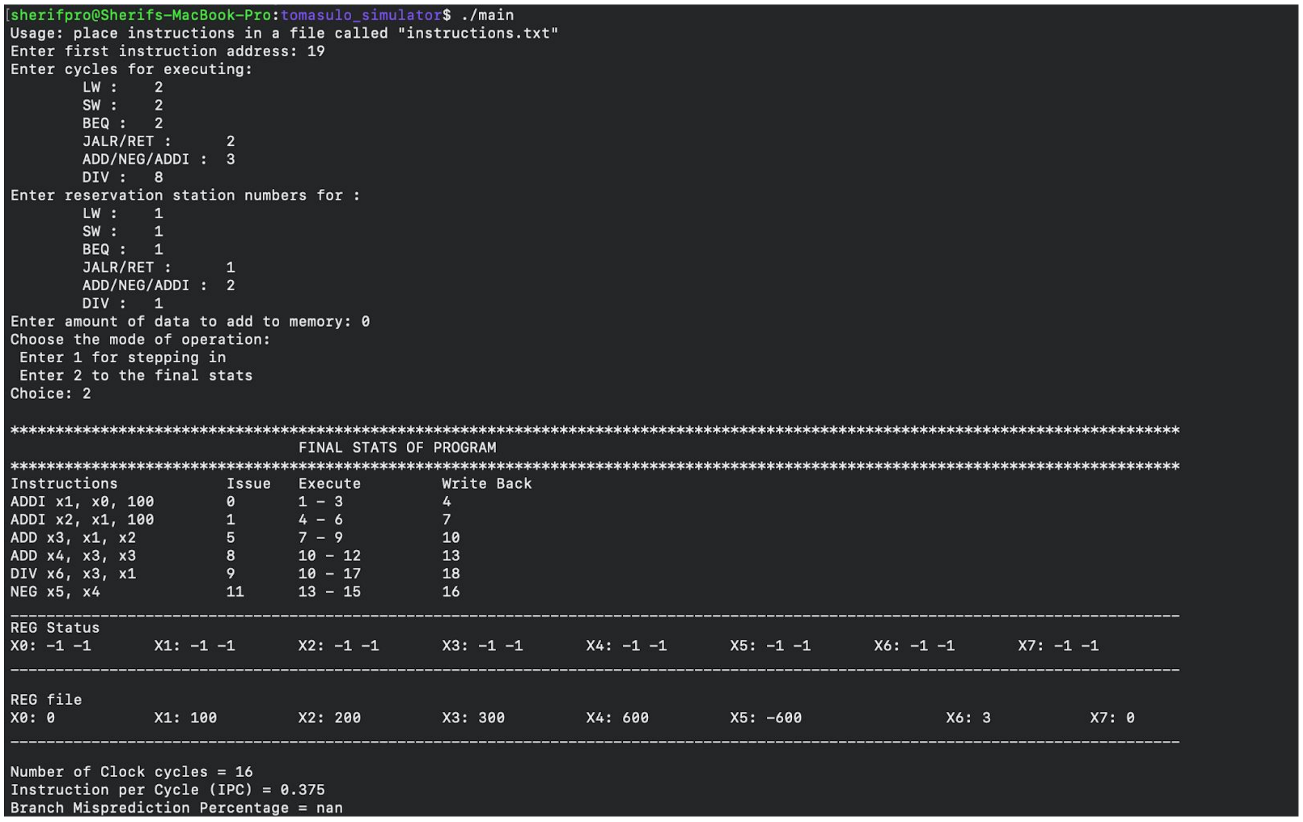
```
sherifpro@Sherifs-MacBook-Pro:tomasulo_simulator$ ./main
Usage: place instructions in a file called "instructions.txt"
Enter first instruction address: 12
Enter cycles for executing:
        LW :    2
        SW :    2
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  3
        DIV :   6
Enter reservation station numbers for :
        LW :    2
        SW :    2
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  2
        DIV :   1
Enter amount of data to add to memory: 2
Data 1
        Enter address: 4
        Enter data: 2
Data 2
        Enter address: 5
        Enter data: 9
Choose the mode of operation:
 Enter 1 for stepping in
 Enter 2 to the final stats
Choice: 2

*************************************************************************************************************************************
                                FINAL STATS OF PROGRAM
*************************************************************************************************************************************
Instructions            Issue   Execute         Write Back
LW x2, 4(x0)            0       1 - 2           3
LW x3, 5(x0)            1       2 - 3           4
ADDI x7, x0, 9          2       3 - 5           6
ADDI x4, x0, 2          3       4 - 6           7
ADDI x5, x0, 1024       7       8 - 10          11
ADDI x7, x7, -1         80      81 - 83         84
BEQ x7, x0, 5           82      84 - 85         86
ADD x2, x2, x2          83      0 - 0           0
NEG x3, x3              76      77 - 79         80
DIV x5, x5, x4          77      78 - 83         85
BEQ x0, x0, -5          78      79 - 80         81
SW x2, 10(x0)           85      86 - 87         88
SW x3, 11(x0)           86      87 - 88         89
SW x5, 12(x0)           89      90 - 91         92
-------------------------------------------------------------------------------------------------------------------------------------
REG Status
X0: -1 -1       X1: -1 -1       X2: -1 -1       X3: -1 -1       X4: -1 -1       X5: -1 -1       X6: -1 -1       X7: -1 -1
-------------------------------------------------------------------------------------------------------------------------------------

REG file
X0: 0           X1: 0           X2: 512         X3: 9           X4: 2           X5: 4           X6: 0           X7: 0
-------------------------------------------------------------------------------------------------------------------------------------

Number of Clock cycles = 92
Instruction per Cycle (IPC) = 0.152174
Branch Misprediction Percentage = 52.9412
```

Test2:



```
[sherifpro@Sherifs-MacBook-Pro:tomasulo_simulator$ ./main
Usage: place instructions in a file called "instructions.txt"
Enter first instruction address: 12
Enter cycles for executing:
        LW :    2
        SW :    2
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  3
        DIV :   8
Enter reservation station numbers for :
        LW :    2
        SW :    2
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  2
        DIV :   1
Enter amount of data to add to memory: 2
Data 1
        Enter address: 110
        Enter data: 110
Data 2
        Enter address: 0
        Enter data: 19
Choose the mode of operation:
 Enter 1 for stepping in
 Enter 2 to the final stats
Choice: 2

**********************************************************************************************************************************
                                FINAL STATS OF PROGRAM
**********************************************************************************************************************************
Instructions            Issue   Execute         Write Back
ADDI x2, x0, 100        0       1 - 3           4
LW x4, 10(x2)           1       4 - 5           6
LW x5, 0(x0)            2       3 - 4           5
SW x5, 2(x0)            3       5 - 6           7
JALR x5                 4       5 - 6           7
ADDI x2, x0, 32         0       0 - 0           0
ADDI x5, x0, 10         0       0 - 0           0
ADDI x2, x0, 2000       6       7 - 9           10
ADDI x5, x0, 1100       7       8 - 10          11
ADD  x2, x5, x2         11      12 - 14         15
ADDI x1, x0, 13         12      13 - 15         16
RET                     13      16 - 17         18
ADD x2, x4, x4          0       0 - 0           0
ADD x6, x4, x2          17      18 - 20         21
LW x7, 2(x0)            18      19 - 20         21
----------------------------------------------------------------------------------------------------------------------------------
REG Status
X0: -1 -1       X1: -1 -1       X2: -1 -1       X3: -1 -1       X4: -1 -1       X5: -1 -1       X6: -1 -1       X7: -1 -1
----------------------------------------------------------------------------------------------------------------------------------

REG file
X0: 0           X1: 13          X2: 3100                X3: 0           X4: 110         X5: 1100                X6: 3210                X7: 19
----------------------------------------------------------------------------------------------------------------------------------

Number of Clock cycles = 21
Instruction per Cycle (IPC) = 0.714286
Branch Misprediction Percentage = nan
```

Test3:



```
[sherifpro@Sherifs-MacBook-Pro:tomasulo_simulator$ ./main
Usage: place instructions in a file called "instructions.txt"
Enter first instruction address: 19
Enter cycles for executing:
        LW :    2
        SW :    2
        BEQ :   2
        JALR/RET :      2
        ADD/NEG/ADDI :  3
        DIV :   8
Enter reservation station numbers for :
        LW :    1
        SW :    1
        BEQ :   1
        JALR/RET :      1
        ADD/NEG/ADDI :  2
        DIV :   1
Enter amount of data to add to memory: 0
Choose the mode of operation:
 Enter 1 for stepping in
 Enter 2 to the final stats
Choice: 2

*********************************************************************************************************************************
                                FINAL STATS OF PROGRAM
*********************************************************************************************************************************
Instructions            Issue   Execute         Write Back
ADDI x1, x0, 100        0       1 - 3           4
ADDI x2, x1, 100        1       4 - 6           7
ADD x3, x1, x2          5       7 - 9           10
ADD x4, x3, x3          8       10 - 12         13
DIV x6, x3, x1          9       10 - 17         18
NEG x5, x4              11      13 - 15         16
---------------------------------------------------------------------------------------------------------------------------------
REG Status
X0: -1 -1       X1: -1 -1       X2: -1 -1       X3: -1 -1       X4: -1 -1       X5: -1 -1       X6: -1 -1       X7: -1 -1
---------------------------------------------------------------------------------------------------------------------------------

REG file
X0: 0           X1: 100         X2: 200         X3: 300         X4: 600         X5: -600                X6: 3           X7: 0
---------------------------------------------------------------------------------------------------------------------------------

Number of Clock cycles = 16
Instruction per Cycle (IPC) = 0.375
Branch Misprediction Percentage = nan
```