Project 2 Report

Computer Architecture

Ahmed Jaheen - 900212943
Adham Samy - 900213258
Islam Hassan - 900213579

Submitted to Prof. Cherif Salama
This report is prepared for CSCE3301, section 2

Computer Science and Engineering Department
The American University in Cairo

11/12/2023

Implementation Brief Description

The implemented simulator is designed to assess the performance of a simplified out-of-order 16-bit RISC processor using Tomasulo's algorithm without speculation. The processor adheres to a RISC architecture. The simulator takes into account three backend stagesissue, execute, and writeand supports various instructions, including load/store, conditional branch, call/return, and arithmetic/logic operations. It handles reservation stations for functional units, registers, and memory operations. The simulator is developed in C++ and supports console application.

1. **Instruction Set:** The processor features a 16-bit word size, eight general-purpose registers (R0 to R7), and word-addressable memory (128 KB). The instruction set includes load/store, conditional branch, call/return, and arithmetic/logic instructions. Notable instructions include LOAD, STORE, BNE, CALL, RET, ADD, ADDI, NAND, and DIV.
2. **Simulation Process:** The simulator takes an assembly program as input, along with initial data for program execution. It processes instructions, populates reservation units, and executes instructions based on the Tomasulo algorithm. The simulation records the number of completed instructions, encountered branches, cycle counts, and branch mispredictions.

Here is a brief overview of the key components and functionalities in the code:

1. **Constructor (Tomasulo::Tomasulo)**
   - Initializes various data structures and parameters required for simulation.
   - Parses and categorizes the input assembly instructions into different instruction types (ADD, NAND, DIV, ADDI, LOAD, STORE, BNE, CALL, RET).
   - Initializes reservation units for functional units, registers, and memory operations.
   - Sets up initial values for registers, cycle delays, and cycles taken for each instruction type.
2. **runSimulation Method**
   - Executes the main simulation loop, iterating through each cycle of the processor.
   - Issues instructions to reservation units based on Tomasulo's algorithm.
   - Checks for readiness of source registers and executes instructions when ready.
   - Updates the instruction tracing vector with information about the execution of each instruction.
   - Handles control flow changes for branch (BNE), call (CALL), and return (RET)

instructions.
- Prints information about the simulation progress, including cycle number, PC, branches, mispredictions, and instruction tracing.

3. **execute Method**
   - Checks if any reservation unit is ready to execute.
   - Updates source register values based on readiness.
   - Executes instructions when conditions are met, updating the finish time and instruction tracing.
4. **write Method**
   - Writes the result of executed instructions to the register file or memory.
   - Handles cases where a store instruction may introduce a stall cycle.
   - Updates the instruction tracing with write result cycle information.
   - Clears reservation units after writing results.
5. **Instruction Parsing Methods (parse__R__Instructions, parse__I__Instructions, etc.)**
   - Parse different types of instructions (R-type, I-type, LOAD, STORE, etc.) to create a unified representation (Inst struct).
6. **Utility Methods (isFinished, flushReservationStation, etc.)**
   - Checks if the simulation has completed by verifying that all reservation units are free.
   - Flushes a reservation station when control flow changes.
7. **Function main()**
   - Reads filenames for instruction and data files, starting address, and optional delay.
   - Reads instructions and data from files.
   - Processes the instructions (removing empty lines, separating labels, and obtaining label addresses).
   - Creates an instance of the Tomasulo class and runs the simulation.

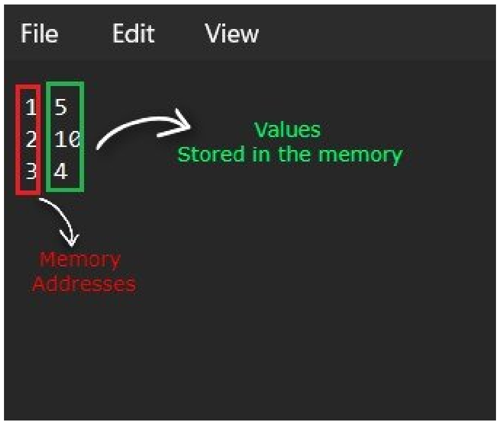
Bonus Features

(i) Implemented and integrated a parser to allow the user to provide the input program using proper assembly language including labels for jump targets and function calls. The assembly program can be entered directly in the application or in a text file read by the application.

User Guide

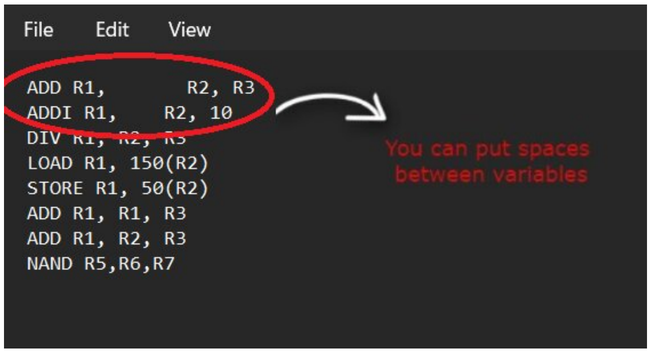
The project is written in C++, allowing you to compile it using any C++ compiler. Once the compilation process is complete, you have to add your implemented RISC instruction files and memory file to the same directiory as the .cpp file.
To initiate the program, follow these steps:

1. Firstly in the memory file, you write two numbers per line; the first number holds the memory address, and the second number holds the value stored in the memory address.



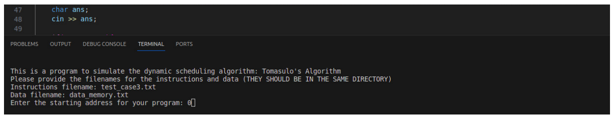
```
1 5
2 10
3 4
```

2. Secondly in the assembly test file, you can write any number of instructions. However, you have to be aware that you cannot write a space before any instruction or have an syntax error as it will not give an output. You can put any number of spaces between the registers. Also, you can use the registers from R0 to R7.



```
ADD R1,       R2, R3
ADDI R1,    R2, 10
DIV R1, R2, R3
LOAD R1, 150(R2)
STORE R1, 50(R2)
ADD R1, R1, R3
ADD R1, R2, R3
NAND R5,R6,R7
```

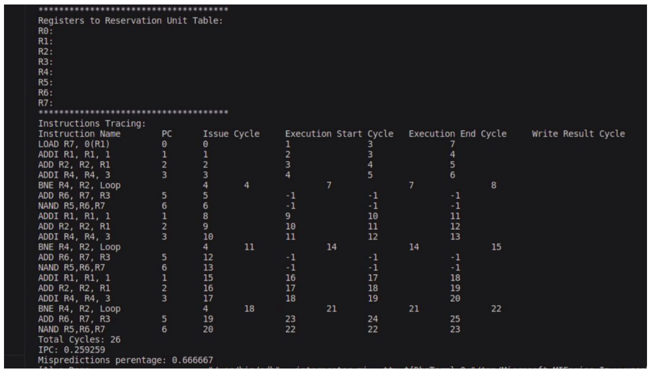
3. Thirdly, when you open the program, you will be prompted to enter the memory file's name, assembly instructions' file name, and starting address of the program as a decimal integer. Once you have entered the value, press the Enter key. As an example, I have entered the value 0 in this case.



```
    char ans;
    cin >> ans;
```

```
This is a program to simulate the dynamic scheduling algorithm: Tomasulo's Algorithm
Please provide the filenames for the instructions and data (THEY SHOULD BE IN THE SAME DIRECTORY)
Instructions filename: test_case3.txt
Data filename: data_memory.txt
Enter the starting address for your program: 0
```

4. Fourthly, For each cycle, you should expect to receive output according to the specified format, which will look like this.



```
*************************************
Registers to Reservation Unit Table:
R0:
R1:
R2:
R3:
R4:
R5:
R6:
R7:
*************************************
Instructions Tracing:
Instruction Name        PC      Issue Cycle     Execution Start Cycle   Execution End Cycle     Write Result Cycle
LOAD R7, 0(R1)          0       0               1               3               7
ADDI R1, R1, 1          1       1               2               3               4
ADD R2, R2, R1          2       2               3               4               5
ADDI R4, R4, 3          3       3               4               5               6
BNE R4, R2, Loop                4       4               7               7               8
ADD R6, R7, R3          5       5               -1              -1              -1
NAND R5,R6,R7           6       6               -1              -1              -1
ADDI R1, R1, 1          1       8               9               10              11
ADD R2, R2, R1          2       9               10              11              12
ADDI R4, R4, 3          3       10              11              12              13
BNE R4, R2, Loop                4       11              14              14              15
ADD R6, R7, R3          5       12              -1              -1              -1
NAND R5,R6,R7           6       13              -1              -1              -1
ADDI R1, R1, 1          1       15              16              17              18
ADD R2, R2, R1          2       16              17              18              19
ADDI R4, R4, 3          3       17              18              19              20
BNE R4, R2, Loop                4       18              21              21              22
ADD R6, R7, R3          5       19              23              24              25
NAND R5,R6,R7           6       20              22              22              23
Total Cycles: 26
IPC: 0.259259
Mispredictions perentage: 0.666667
```

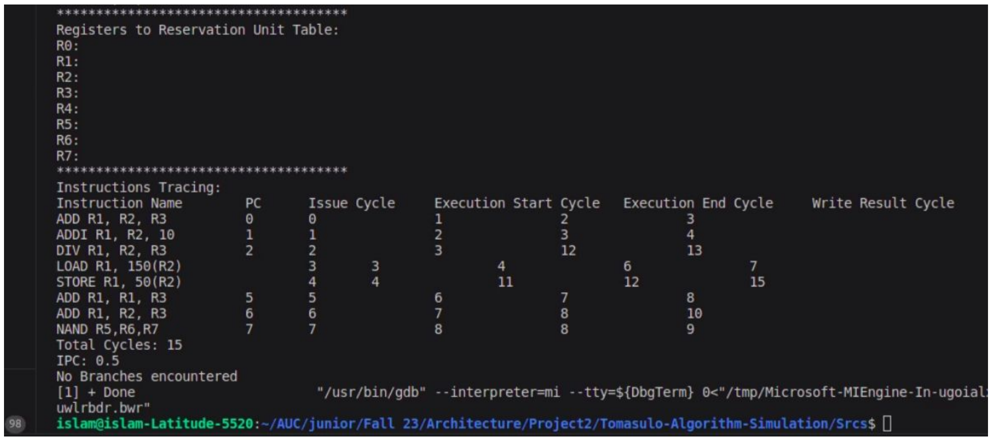
The results obtained from the simulation

1. **Assembly program I:**

   Assembly Code:

```
ADD R1, R2, R3
ADDI R1, R2, 10
DIV R1, R2, R3
LOAD R1, 150(R2)
STORE R1, 50(R2)
ADD R1, R1, R3
ADD R1, R2, R3
NAND R5,R6,R7
```

   Simulation Result:



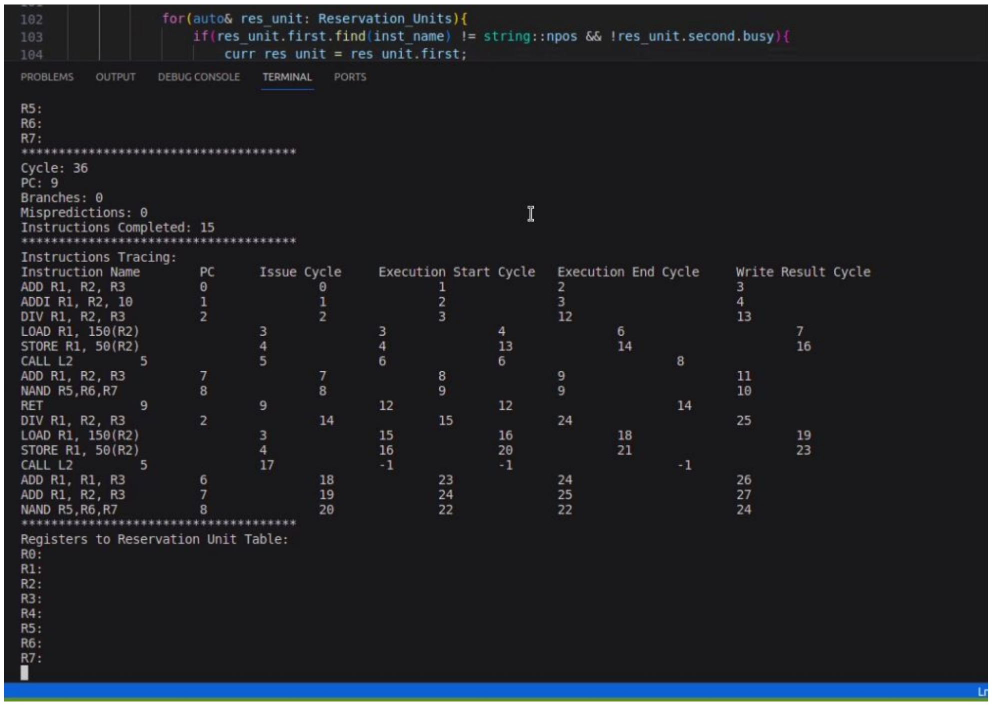
```
*************************************
Registers to Reservation Unit Table:
R0:
R1:
R2:
R3:
R4:
R5:
R6:
R7:
*************************************
Instructions Tracing:
Instruction Name        PC      Issue Cycle     Execution Start Cycle   Execution End Cycle     Write Result Cycle
ADD R1, R2, R3          0       0               1               2               3
ADDI R1, R2, 10         1       1               2               3               4
DIV R1, R2, R3          2       2               3               12              13
LOAD R1, 150(R2)                3       3               4               6               7
STORE R1, 50(R2)                4       4               11              12              15
ADD R1, R1, R3          5       5               6               7               8
ADD R1, R2, R3          6       6               7               8               10
NAND R5,R6,R7           7       7               8               8               9
Total Cycles: 15
IPC: 0.5
No Branches encountered
[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-ugoial
uwlrbdr.bwr"
islam@islam-Latitude-5520:~/AUC/junior/Fall 23/Architecture/Project2/Tomasulo-Algorithm-Simulation/Srcs$ 
```

2. **Assembly program II:**

Assembly Code:

```
ADD R1, R2, R3
ADDI R1, R2, 10
DIV R1, R2, R3
LOAD R1, 150(R2)
STORE R1, 50(R2)
CALL L2
ADD R1, R1, R3
L2: ADD R1, R2, R3
NAND R5,R6,R7
RET
```

Simulation Result:



```
        for(auto& res_unit: Reservation_Units){
            if(res_unit.first.find(inst_name) != string::npos && !res_unit.second.busy){
                curr res unit = res unit.first;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

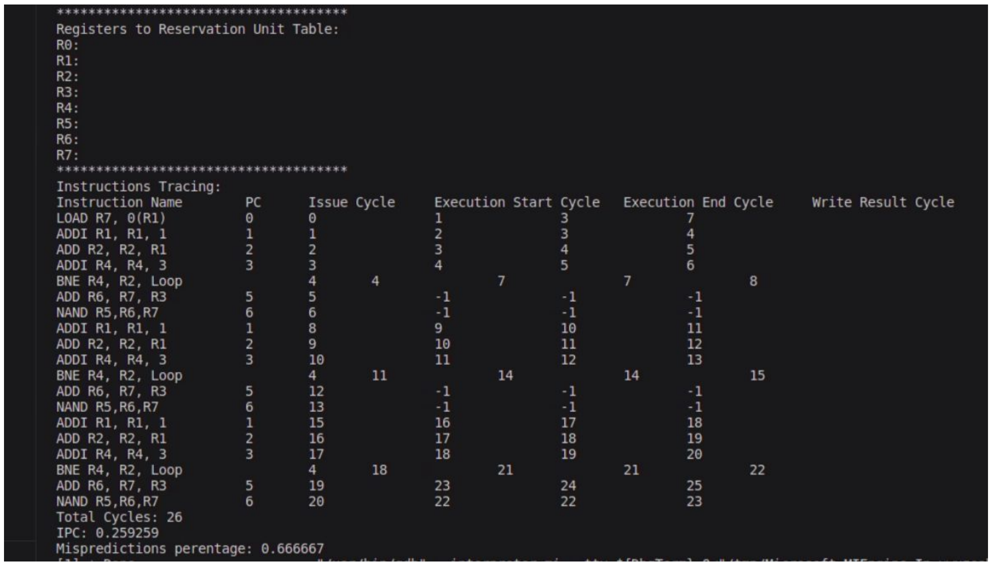
```
R5:
R6:
R7:
*************************************
Cycle: 36
PC: 9
Branches: 0
Mispredictions: 0
Instructions Completed: 15
*************************************
Instructions Tracing:
Instruction Name        PC      Issue Cycle     Execution Start Cycle   Execution End Cycle     Write Result Cycle
ADD R1, R2, R3          0               0               1               2                       3
ADDI R1, R2, 10         1               1               2               3                       4
DIV R1, R2, R3          2               2               3               12                      13
LOAD R1, 150(R2)                3               3               4               6                       7
STORE R1, 50(R2)                4               4               13              14                      16
CALL L2         5               5               6               6                       8
ADD R1, R2, R3          7               7               8               9                       11
NAND R5,R6,R7           8               8               9               9                       10
RET             9               9               12              12                      14
DIV R1, R2, R3          2               14              15              24                      25
LOAD R1, 150(R2)                3               15              16              18                      19
STORE R1, 50(R2)                4               16              20              21                      23
CALL L2         5               17              -1              -1                      -1
ADD R1, R1, R3          6               18              23              24                      26
ADD R1, R2, R3          7               19              24              25                      27
NAND R5,R6,R7           8               20              22              22                      24
*************************************
Registers to Reservation Unit Table:
R0:
R1:
R2:
R3:
R4:
R5:
R6:
R7:
```

3. **Assembly program III:**

Assembly Code:

```
LOAD R7, 0(R1)
Loop:
ADDI R1, R1, 1
ADD R2, R2, R1
ADDI R4, R4, 3
BNE R4, R2, Loop
ADD R6, R7, R3
NAND R5,R6,R7
```

Simulation Result:



```
*************************************
Registers to Reservation Unit Table:
R0:
R1:
R2:
R3:
R4:
R5:
R6:
R7:
*************************************
Instructions Tracing:
Instruction Name        PC      Issue Cycle      Execution Start Cycle   Execution End Cycle     Write Result Cycle
LOAD R7, 0(R1)          0       0                1               3               7
ADDI R1, R1, 1          1       1                2               3               4
ADD R2, R2, R1          2       2                3               4               5
ADDI R4, R4, 3          3       3                4               5               6
BNE R4, R2, Loop                4       4                7               7               8
ADD R6, R7, R3          5       5                -1              -1              -1
NAND R5,R6,R7           6       6                -1              -1              -1
ADDI R1, R1, 1          1       8                9               10              11
ADD R2, R2, R1          2       9                10              11              12
ADDI R4, R4, 3          3       10               11              12              13
BNE R4, R2, Loop                4       11               14              14              15
ADD R6, R7, R3          5       12               -1              -1              -1
NAND R5,R6,R7           6       13               -1              -1              -1
ADDI R1, R1, 1          1       15               16              17              18
ADD R2, R2, R1          2       16               17              18              19
ADDI R4, R4, 3          3       17               18              19              20
BNE R4, R2, Loop                4       18               21              21              22
ADD R6, R7, R3          5       19               23              24              25
NAND R5,R6,R7           6       20               22              22              23
Total Cycles: 26
IPC: 0.259259
Mispredictions perentage: 0.666667
```

Brief discussion of results

We solved each test case manually by pen and paper by showing the status of each instruction, the reservation stations (including load/store buffers), and the registers status at each cycle, and determining the number of cycles that each test case took to finish execution. Then, we compared our hand-written results with our Tomasulos Algorithm simulation, and the results was the same. Also, we traced every cycle on the program debugger to check that everything is working correctly.