THE AMERICAN UNIVERSITY IN CAIRO

School of Sciences and Engineering

Fall 2023

CSCE-3302: Computer Architecture Course

Project_2 Report

Submitted to:

Dr. Cherif Salama

Submitted by:

Name: Fekry Mohamed	ID: 900192372
Name: Mario Ghaly	ID: 900202178
Name: Freddy Amgad	ID: 900203088

12/12/2023

- A brief description of your implementation, including any bonus features included

“main.cpp” is a C++ program that implements an architectural simulator capable of assessing the performance of a simplified out-of-order 16-bit RISC processor that uses Tomasulo’s algorithm without
speculation. The program reads instructions from a file, simulates the execution of these instructions using the Tomasulo Algorithm, and prints the results.

Global Variables:

pc, branches, mispredictions: Keep track of program counter, branches count, and mispredictions count.

mem: A map representing memory.

stations: A map representing reservation stations for different instruction types.

reg: An array representing registers.

Inst Class: Represents an instruction with various attributes like instruction name, station, registers, offset, cycle span, issue, execution start, execution end, write-back, program counter, and branch flag.

Functions:

Print(vector<Inst> inst): Prints the details of instructions.

separate(string line): Splits a line into a vector of words.

getRegImm(string s): Extracts register and immediate values from a string.

FillFromFile(string filename): Reads instructions from a file and fills the instruction vector.

FillFromFile_data(string filename): Reads data from a file and fills the memory.

FillFromFile_hardware(string filename): Reads hardware configurations from a file and fills the reservation station map.

operation_excution(Inst& inst): Performs the execution of instructions.

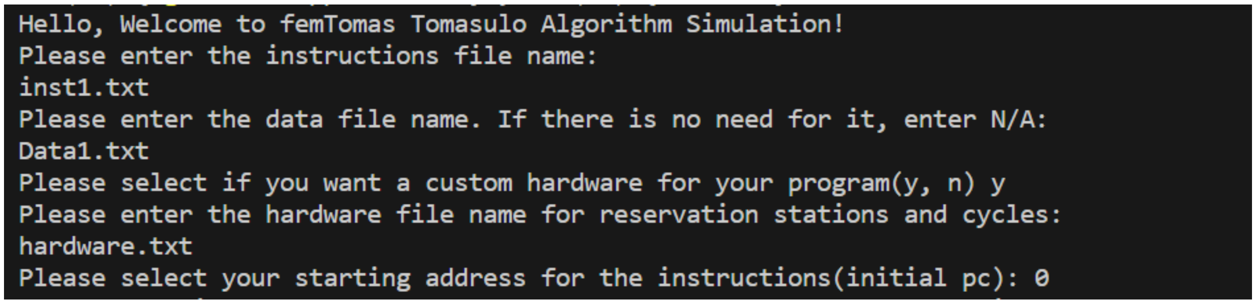
Main Function:

- Takes input file paths for instructions, data, and hardware configurations.
- Reads instructions, data, and hardware configurations from files.
- Simulates the Tomasulo Algorithm for out-of-order execution.
- Tracks issue, execution start, execution end, and write-back times for each instruction.
- Prints a table showing the execution details of each instruction, total execution time, branch misprediction percentage, and IPC (Instructions Per Cycle).

Bonus Features:

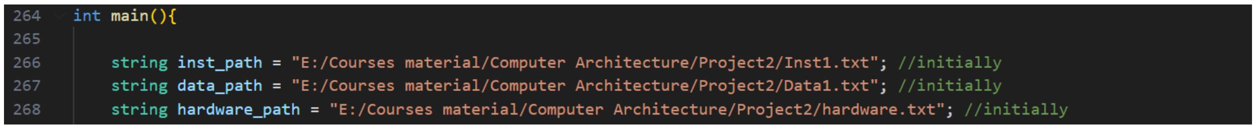
The program supports variable hardware organization. The user should specify the number of reservation stations for each class of instructions. Additionally, the user will specify the number of cycles needed by each functional unit type. (counted as two features)

A user guide including a full simulation example step-by-step with snapshots:



```
Hello, Welcome to femTomas Tomasulo Algorithm Simulation!
Please enter the instructions file name:
inst1.txt
Please enter the data file name. If there is no need for it, enter N/A:
Data1.txt
Please select if you want a custom hardware for your program(y, n) y
Please enter the hardware file name for reservation stations and cycles:
hardware.txt
Please select your starting address for the instructions(initial pc): 0
```

The user should run the “main.cpp” file, and then the program will request the instruction file name, which is inst1.txt in this example. Then, the user will enter the data file name in his program if needed. The user then specifies if he needs to enter a custom hardware file. In this example, I typed “y”, and entered the name of the file that includes the hardware specifications. Finally, you should select the initial PC. Note that all files must be included in the same path as the main file. Furthermore, the default locations are the paths of the files on our computers. If you need to use the default one, please include your paths here:



```
int main(){

    string inst_path = "E:/Courses material/Computer Architecture/Project2/Inst1.txt"; //initially
    string data_path = "E:/Courses material/Computer Architecture/Project2/Data1.txt"; //initially
    string hardware_path = "E:/Courses material/Computer Architecture/Project2/hardware.txt"; //initially
```

The results obtained from the simulation of each assembly program provided:

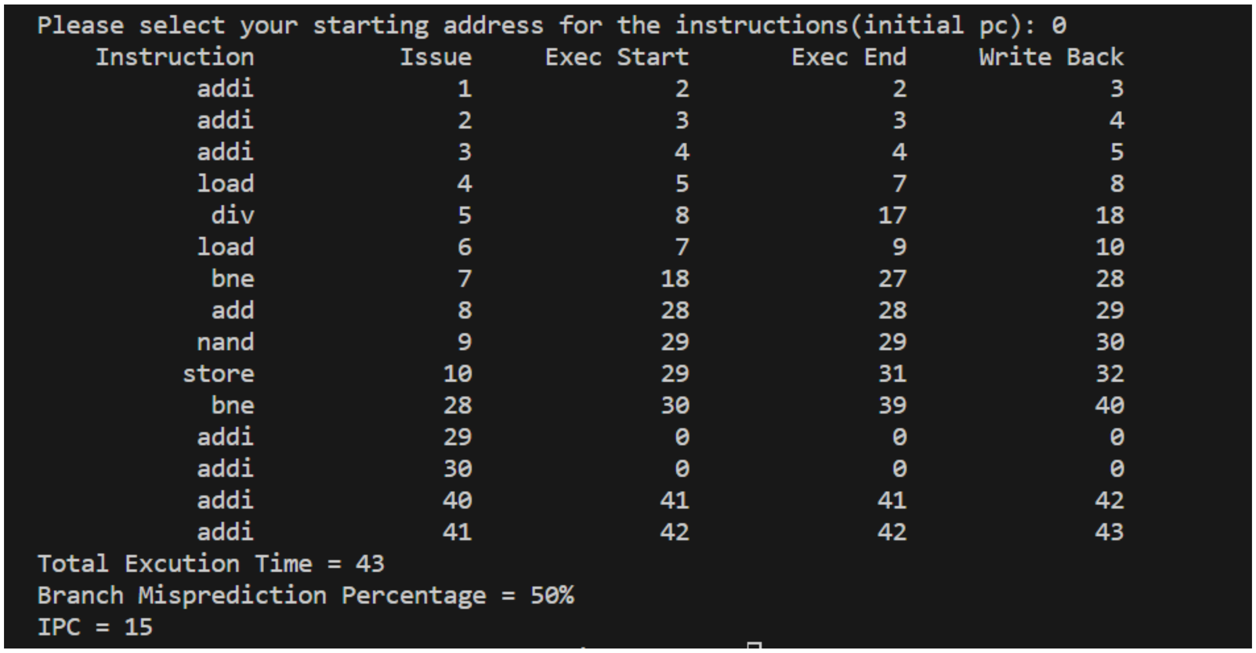
In all simulations, I used the customized hardware specifications, which are shown below:

	Number of reservation stations available	Number of Cycles needed
load	2	3
Bne	1	10
Call/ret	1	1
Add/addi	3	1
Nand	1	1
Div	1	10

Furthermore, Data1.txt file declares that data[50]=40 and Data[51] = 20

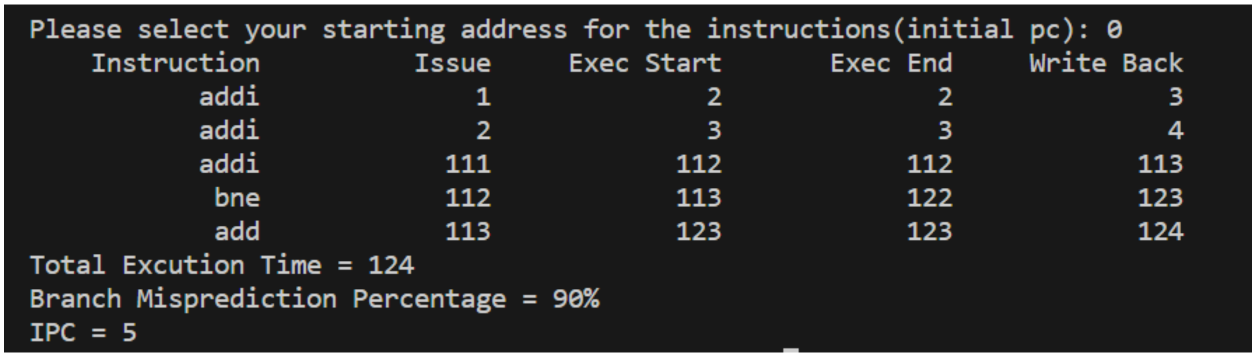
The result of each program is included below:

Inst1.txt file:



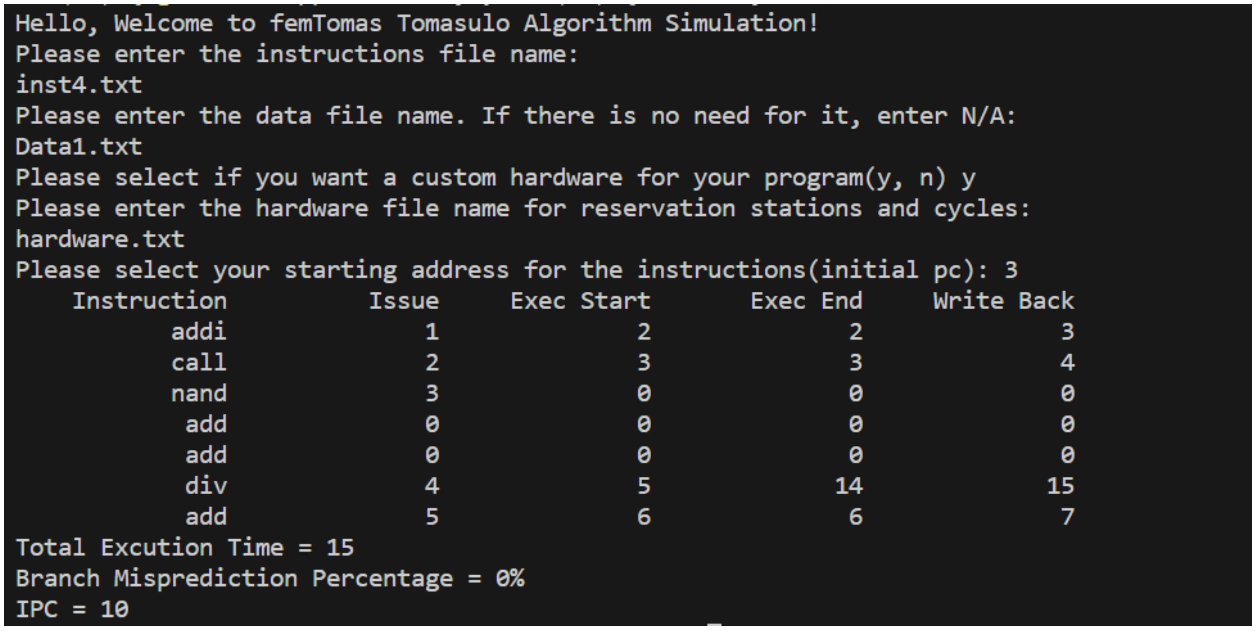
```
Please select your starting address for the instructions(initial pc): 0
    Instruction           Issue     Exec Start        Exec End      Write Back
           addi               1              2               2               3
           addi               2              3               3               4
           addi               3              4               4               5
           load               4              5               7               8
            div               5              8              17              18
           load               6              7               9              10
            bne               7             18              27              28
            add               8             28              28              29
           nand               9             29              29              30
          store              10             29              31              32
            bne              28             30              39              40
           addi              29              0               0               0
           addi              30              0               0               0
           addi              40             41              41              42
           addi              41             42              42              43
Total Excution Time = 43
Branch Misprediction Percentage = 50%
IPC = 15
```

Inst2.txt file:



```
Please select your starting address for the instructions(initial pc): 0
    Instruction           Issue     Exec Start        Exec End      Write Back
           addi               1              2               2               3
           addi               2              3               3               4
           addi             111            112             112             113
            bne             112            113             122             123
            add             113            123             123             124
Total Excution Time = 124
Branch Misprediction Percentage = 90%
IPC = 5
```

Inst3.txt file:



```
Hello, Welcome to femTomas Tomasulo Algorithm Simulation!
Please enter the instructions file name:
inst4.txt
Please enter the data file name. If there is no need for it, enter N/A:
Data1.txt
Please select if you want a custom hardware for your program(y, n) y
Please enter the hardware file name for reservation stations and cycles:
hardware.txt
Please select your starting address for the instructions(initial pc): 3
    Instruction          Issue      Exec Start        Exec End      Write Back
           addi              1               2               2               3
           call              2               3               3               4
           nand              3               0               0               0
            add              0               0               0               0
            add              0               0               0               0
            div              4               5              14              15
            add              5               6               6               7
Total Excution Time = 15
Branch Misprediction Percentage = 0%
IPC = 10
```

a brief discussion of the obtained results:

inst1.txt file:

all the instructions are normally issued except the “BNE R1 R4 2” because there is only one reservation station for the BNE instructions, and the previous BNE finished the write-back at cycle 28. Therefore, this instruction was issued in cycle 28.

“DIV R5 R3 R4” started the execution in cycle 8 because it has data dependence on the previous load instruction. It waited until the value was loaded to R3 to start the simulation. Similarly, “BNE R6 R5 3” starts execution in cycle 18 because the division instruction writes back on R5 in cycle 18.

All the executions end at the expected time based on the cycles it should take to be executed based on the hardware.txt file information explained in the previous section.

We can notice that the two instructions after “BNE R1 R4 2” are not executed because the branch is taken.

Branch misprediction is 50% because one is taken, and the other is not taken in this code.

Inst2.txt file:

This program contains a loop that will loop ten times until the R1 will equal R3. R1 is initialized to 20 and each iteration it decremented by 1, and R3 is initialized to 10.

The first two instruments are accessed once, so they are issued and executed normally. The loop will be taken 10 times, and each time, it will take 10 cycles to be executed and 1 cycle to be written back. Therefore, the last recorded issue for “ADDI R1 R1 -1” is in cycle 111, and the BNE instruction that will not be taken is issued in cycle 112.

Note that “BNE R1 R3 -2” is actually jump to the previous instruction but we made it -2 because our program increment the PC every clock cycle, so we need to make an extra “-1” for every branch instruction in addition to the required offset.

All the executions end at the expected time based on the cycles it should take to be executed based on the hardware.txt file information explained in the previous section.

Branch misprediction is 90% because nine instructions are taken, and one instruction is not taken in this code.

Inst3.txt file:

We tried in this code to set the initial PC to 3 instead of 0, so it begins the execution in cycle 3 with “addi r2 r1 5” instruction. The call is issued in cycle, but execution is finished in cycle 4. Therefore, the NAND operation is executed in cycle 3, but does not continue execution after the call instruction is finished. The call function jumps to PC = 8, which is the DIV instruction.

All the executions end at the expected time based on the cycles it should take to be executed based on the hardware.txt file information explained in the previous section.

Branch misprediction is 100% because call is unconditional branch, and there is no other branch.