CSCE3301 – Computer Architecture

Summer 2022

Project : femTomas Tomasulo Algorithm Simulation

<u>Project Report</u>

Project Overview:
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.

Implementation language:
C++

A Brief Description of Code Implementation

Main Data Structures Used:

Structs: we used structs to be able to control the variables we use in an easy and organized manner. For example, one of the structs we used was for the reservation station where we identified its contents. For that reason, it took name (for the name of the function), busy (to be able to later check whether or not the reservation station is ready to use), op(for the operation that's going to be used), and lastly the index and (vj,vk,qj,qk) to be able to track which registers are going to move on the executing phase and which will wait for a previous function to write back.

Vectors: Like structs, we constantly used vectors to be able to sequentially store information without worrying about its abundance. For example, one of the vectors we used was to store the instructions that were being read from the file.

Implementation:
Reading from the file: we used the concept of reading from the file to read both the instructions and the data memory.

Main Implementation:
After we finished with the preliminary functions with their corresponding variables, we started to implement tomasulo's algorithm. We created three main void functions, one representing each stage. The first is for issuing the function where we check whether the reservation station is free or not. For that reason we put "if statements" for every operation we have along with whether the reservation station is busy or not. The second is for executing where we used the logic of checking whether or not the operands themselves are free or not. We had to initialize variables to check when the starting of the execution will be and when they will end. All this was implemented by also using the concept of "if statements" , a tracking vector, and whether or not the function was issued. The third main function is for writing back where here we mainly check

whether or not the function has finished executing, and if that condition were satisfied, we implemented the functionality (NAND, negate, etc..), and then wrote back into the desired register.

Bonus Task Requirement

- We Implemented a parser to allow the program to use proper assembly language including labels for jump targets in case of BEQ instructions.

- We attempted to implement a GUI desktop application using QT.

Design Decisions and Assumptions

- Fetching and decoding take 0 cycles and the instruction queue is already filled with all the instructions to be simulated.
- No floating-point instructions, registers, or functional units
- No input/output instructions are supported
- No interrupts or exceptions are to be handled
- For each program being executed, assume that the program and its data are fully loaded in the main memory
- There is a one-to-one mapping between reservation stations and functional units. i.e., Each reservation
- station has a functional unit dedicated to it
- No cache or virtual memory

User guide:

1. User should follow this way of writing the test cases code:
    a. All instructions should be capitalized.
    b. Only one space between the instruction and the registers
    c. All registers should not be capitalized
    d. When writing a label, no space should be left between the colon and the instruction
    e. No spaces, just commas between the register named

An example that covers these specifications:

```
ADD r1,r2,r3
Label:MUL r2,r3,r1
```

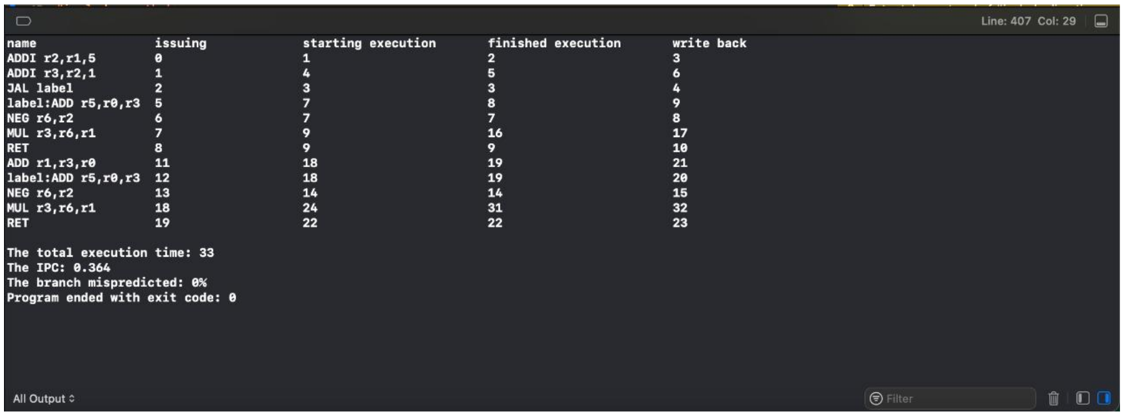
2. The user should write his code in a separate file and then use his/her own directory to read from the file they have chosen.

<u>**Test Cases and Screenshots:**</u>

Step 1: Assuming the user wrote his/her directory into the code, then pressed run.

Test case 1:

ADDI r2,r1,5
ADDI r3, r2,1
JAL label
label:ADD r5,r0, r3
NEG r6, r2
MUL r3,r6, r1
RET
ADD r1,r3, r0
label:ADD r5, r0, r3
NEG r6,r2
MUL r3, r6, r1
RET



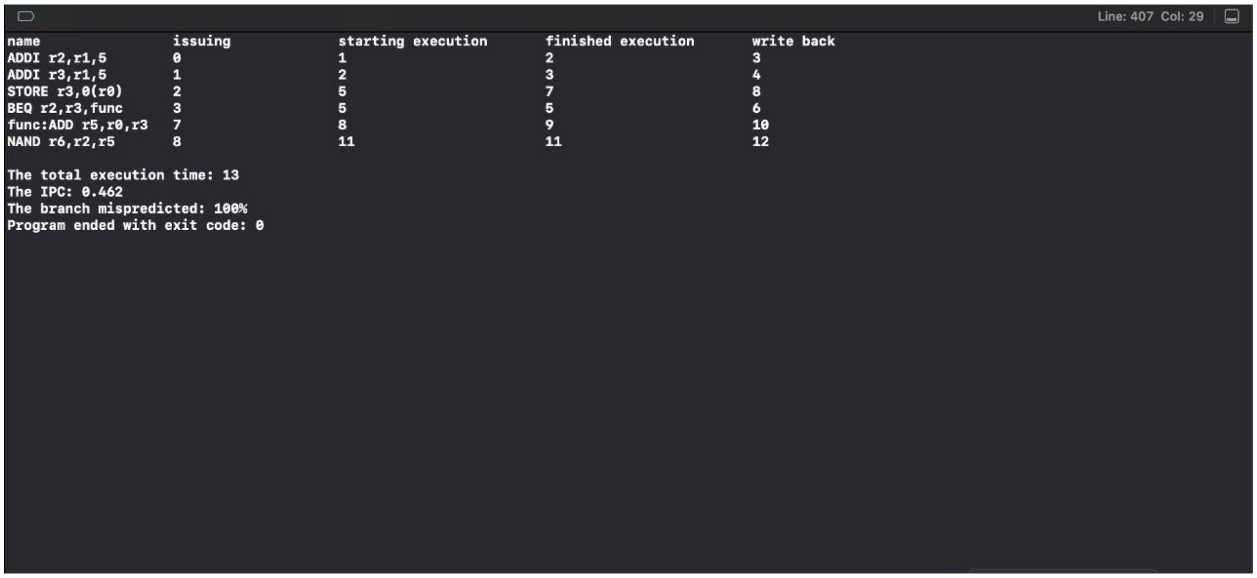
```
name               issuing             starting execution       finished execution       write back
ADDI r2,r1,5       0                   1                        2                        3
ADDI r3,r2,1       1                   4                        5                        6
JAL label          2                   3                        3                        4
label:ADD r5,r0,r3 5                   7                        8                        9
NEG r6,r2          6                   7                        7                        8
MUL r3,r6,r1       7                   9                        16                       17
RET                8                   9                        9                        10
ADD r1,r3,r0       11                  18                       19                       21
label:ADD r5,r0,r3 12                  18                       19                       20
NEG r6,r2          13                  14                       14                       15
MUL r3,r6,r1       18                  24                       31                       32
RET                19                  22                       22                       23

The total execution time: 33
The IPC: 0.364
The branch mispredicted: 0%
Program ended with exit code: 0
```

Test case 2:

A: to test misprediction when we put one branching function

ADDI r2, r1,5
ADDI r3, r1,5

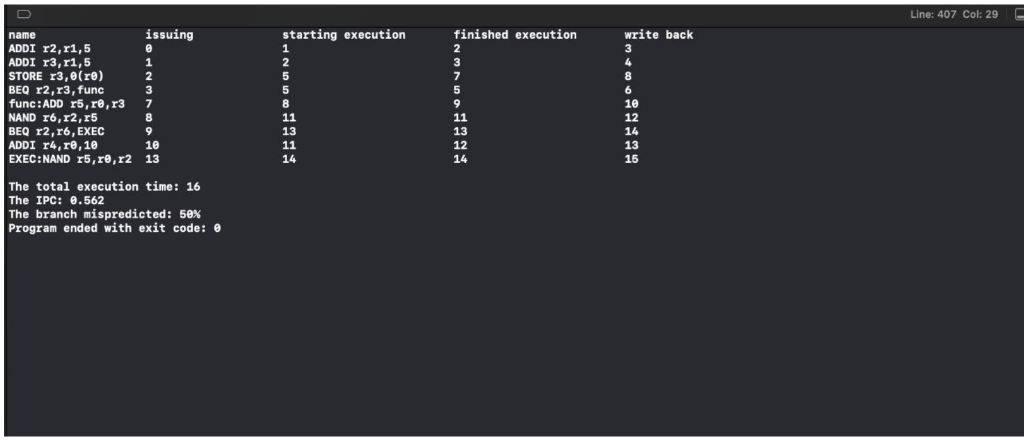
STORE r3,0(r0)
BEQ r2,r3,func
func:ADD r5,r0,r3
NAND r6,r2, r5



```
Line: 407  Col: 29
name                issuing             starting execution       finished execution       write back
ADDI r2,r1,5        0                   1                        2                        3
ADDI r3,r1,5        1                   2                        3                        4
STORE r3,0(r0)      2                   5                        7                        8
BEQ r2,r3,func      3                   5                        5                        6
func:ADD r5,r0,r3   7                   8                        9                        10
NAND r6,r2,r5       8                   11                       11                       12

The total execution time: 13
The IPC: 0.462
The branch mispredicted: 100%
Program ended with exit code: 0
```

B: to test misprediction when we put 2 branching functions

ADDI r2, r1,5
ADDI r3, r1,5
STORE r3,0(r0)
BEQ r2,r3,func
func:ADD r5,r0,r3
NAND r6,r2,r5
BEQ r2,r6,EXEC
ADDI r4,r0,10
EXEC: NAND r5, r0, r2

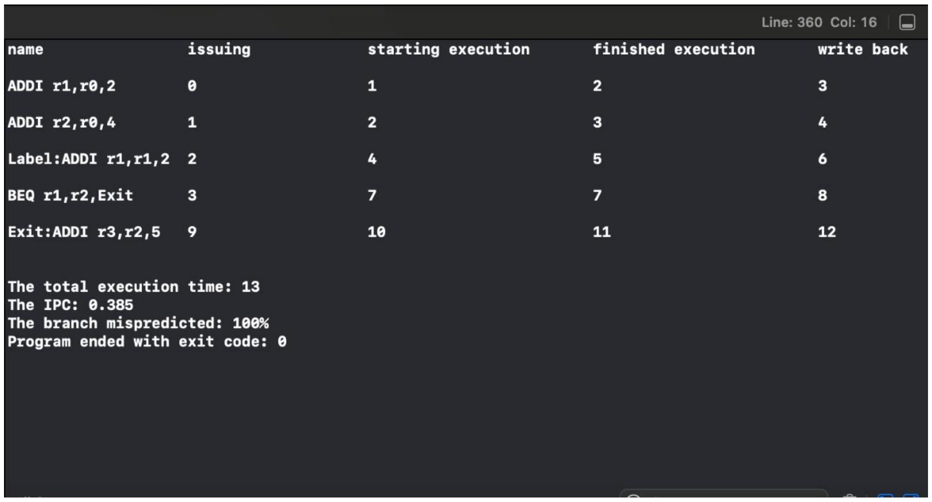


```
name                issuing             starting execution       finished execution       write back
ADDI r2,r1,5        0                   1                        2                        3
ADDI r3,r1,5        1                   2                        3                        4
STORE r3,0(r0)      2                   5                        7                        8
BEQ r2,r3,func      3                   5                        5                        6
func:ADD r5,r0,r3   7                   8                        9                        10
NAND r6,r2,r5       8                   11                       11                       12
BEQ r2,r6,EXEC      9                   13                       13                       14
ADDI r4,r0,10       10                  11                       12                       13
EXEC:NAND r5,r0,r2  13                  14                       14                       15

The total execution time: 16
The IPC: 0.562
The branch mispredicted: 50%
Program ended with exit code: 0
```

Test case 3:

Keeps adding to r1 till it equals r2 (therefore it should branch to exit the second time)

ADDI r1,r0,2
ADDI r2,r0,4
Label:ADDI r1,r1,2
BEQ r1,r2,Exit
BEQ r0,r0,Label
Exit:ADDI r3,r2,5



```
name                issuing             starting execution       finished execution       write back

ADDI r1,r0,2        0                   1                        2                        3

ADDI r2,r0,4        1                   2                        3                        4

Label:ADDI r1,r1,2  2                   4                        5                        6

BEQ r1,r2,Exit      3                   7                        7                        8

Exit:ADDI r3,r2,5   9                   10                       11                       12


The total execution time: 13
The IPC: 0.385
The branch mispredicted: 100%
Program ended with exit code: 0
```