Tomasulo’s Algorithm Simulation

Tomasulo Algorithm

Bassant Samy , Mazen Amr Eid, Mohamed Hossam, Khaled Soliman

# Implementation

This simulation is written in Java. We took this decision as it is an Object-Oriented language and we thought that it would be perfect to model the algorithm. Indeed, using this fact, we can model the ROB, Reservation Station, Load Buffer, Memory, Register File and the Instruction Queue all as Objects. This simplifies greatly the design of the algorithm.

## Implementation Summary

The entry point of our program is obviously the Main class. You can find there a timer. This timer represents our Clock Cycles for the simulation. Every specified interval of time, this clock cycle is incremented by 1 and an interface function called “didUpdate()” is called. This last represents “always @ (posedge clk)” in Verilog and is implemented by the Controller class. This is where all the Tomasulo’s Algorithm lives.

The algorithm begins by first of all, at each cycle fetching 2 instructions at a time from the Instruction Queue by first checking if both the ROB & the Reservation Stations have empty slots, then if this is true, it issues the 2 fetched instructions to them. After this is done, we execute the 2 instructions at the next CC. And while the ROB is not empty, we try to commit if the instruction on its turn is ready to commit. We finally predict the next PC: If the branch offset is positive we predict that it is not taken and If it is negative we predict that it is negative (based on the usual behavior of a branch instruction). If a misprediction occurred, at the next CC, we flush and search for the instruction to be fetched (by either dequeuing if the instruction has not been executed or by enqueuing the instructions again until we find the matching PC). Finally, if the ROB and the Instruction Queue are empty, we stop the Timer and send the ROB and RS as a response to the website to be viewed.

## Output

We offered the user the option to choose either to see the output of the algorithm cycle by cycle or just see the algorithm running till it stops. This is done by entering 1 if you run it on console when asked to or by pressing on “” if you were on our website. The program then shows the user what happens in the processor at each Clock Cycle.

## Test

This program simulates:

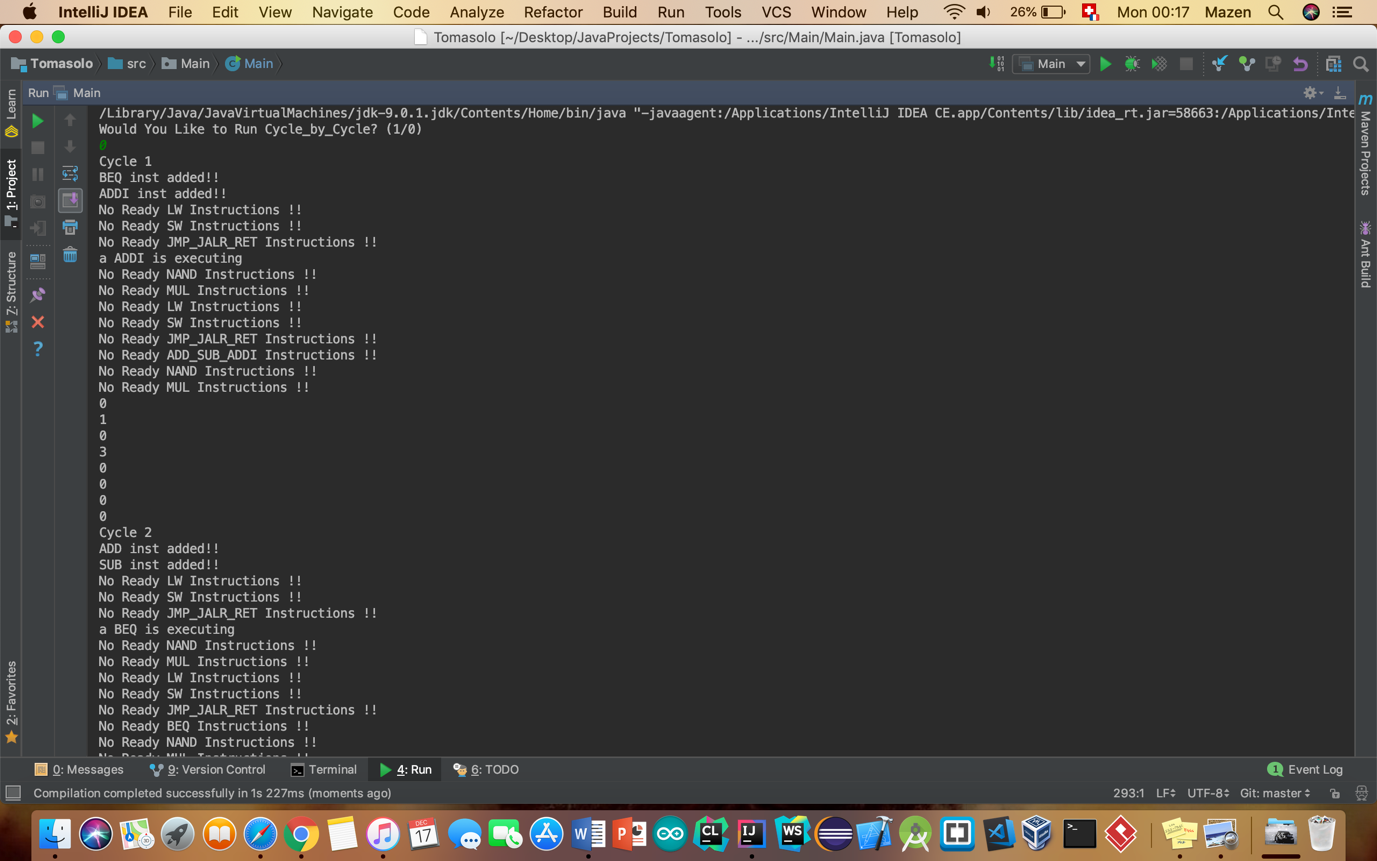
Beq x0, x0, 1

Addi x2, x1, 3

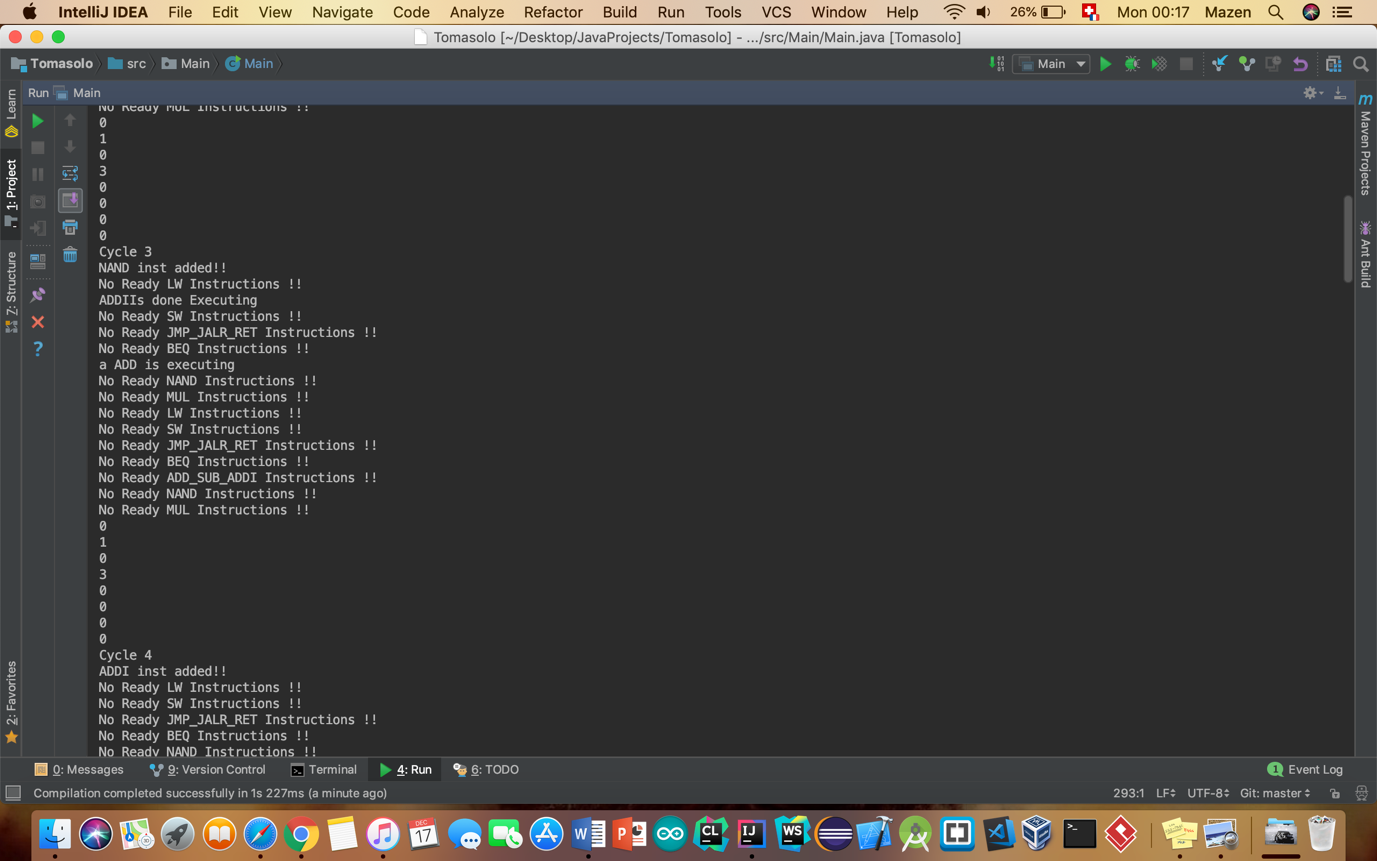
Add x5, x3, x3

Sub x6, x5, x6

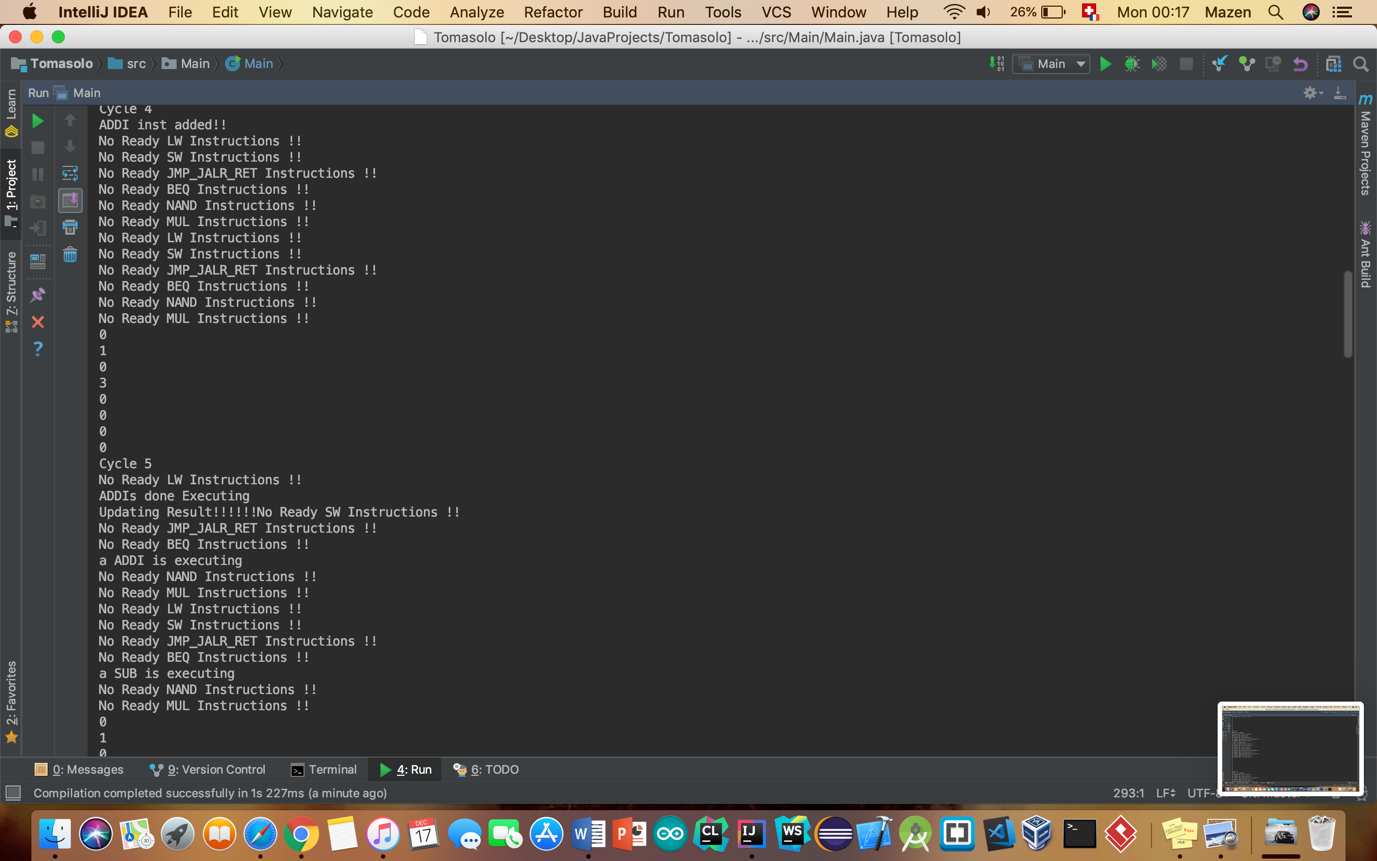
Nand x6, x5, x6



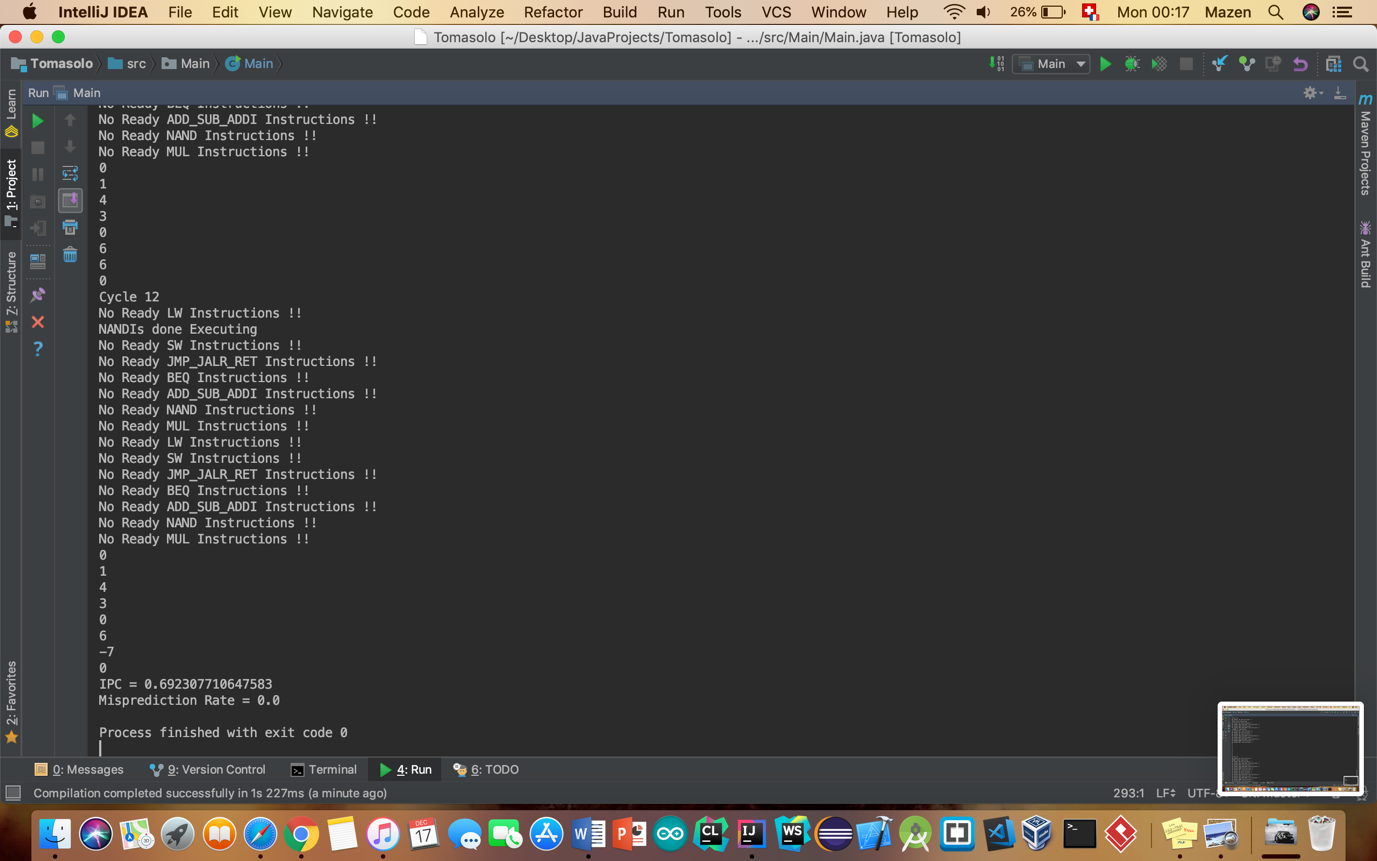
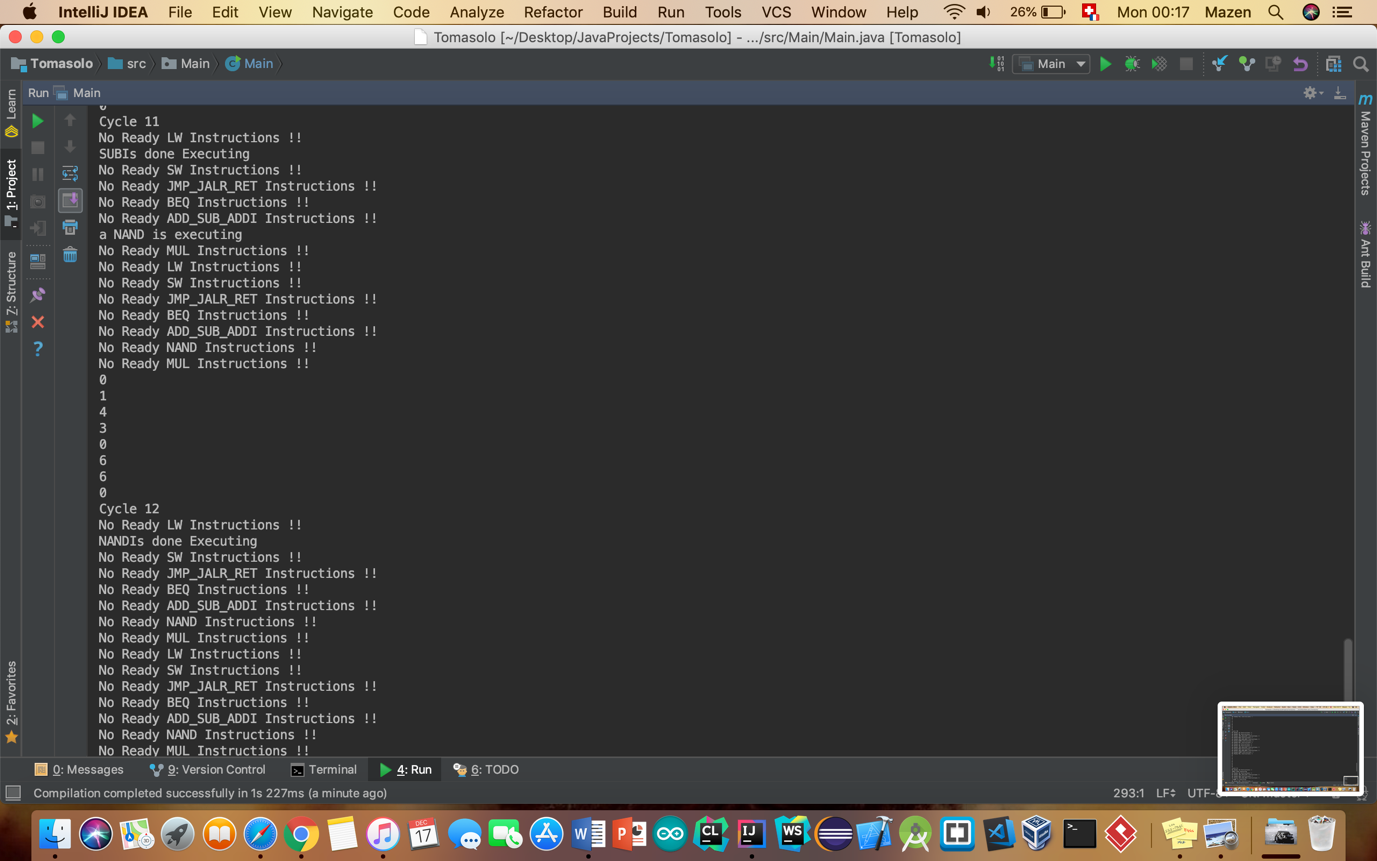
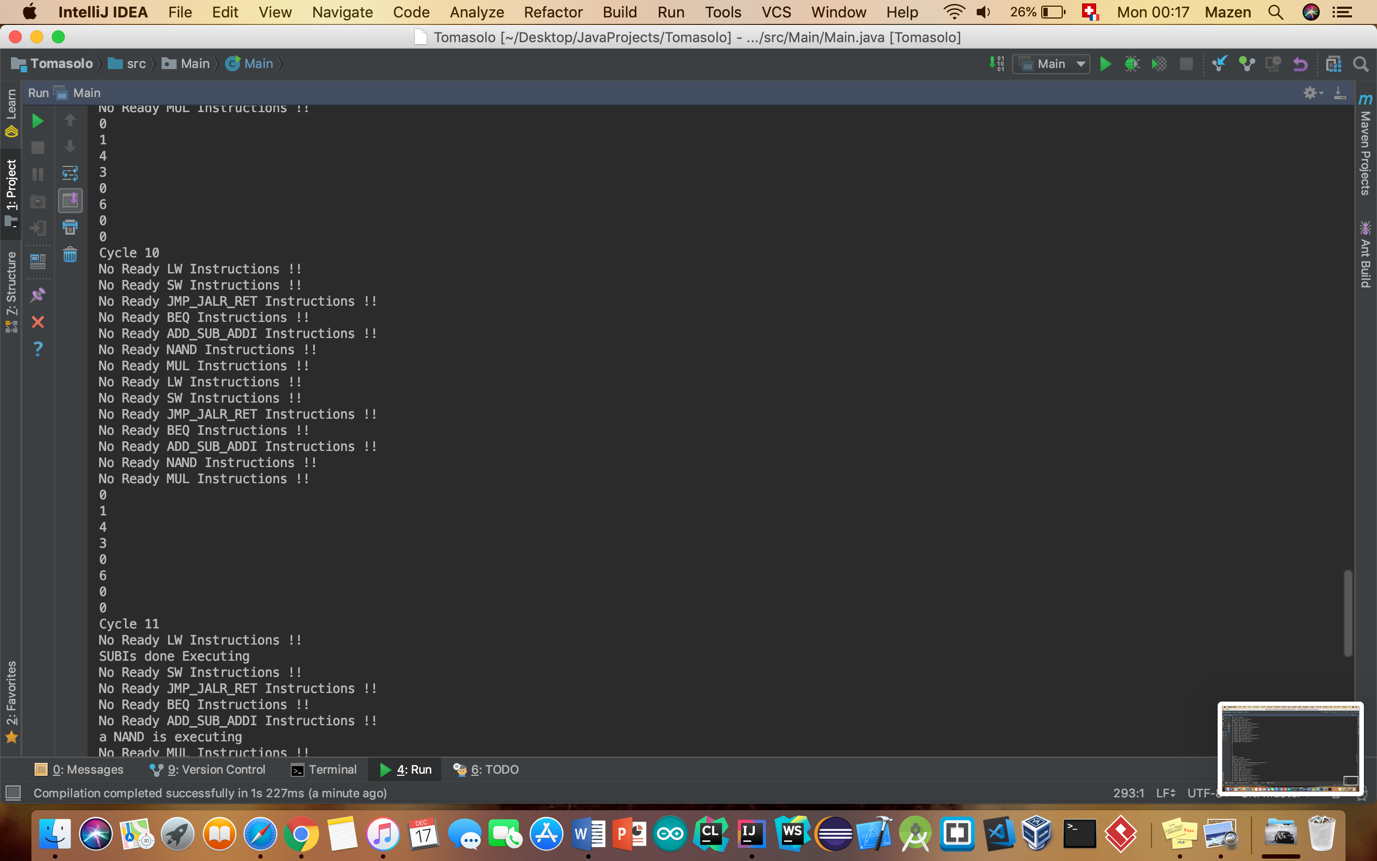
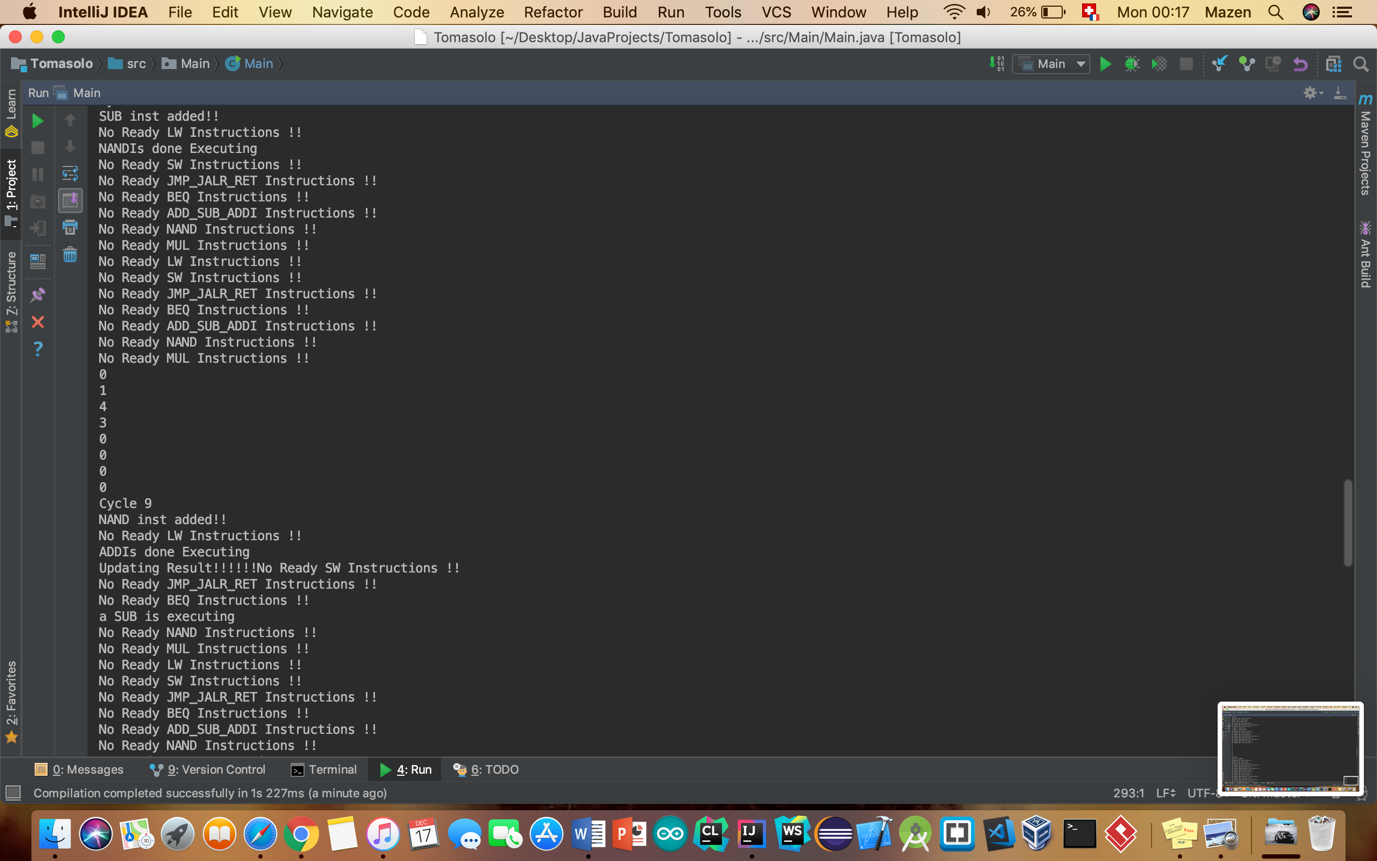
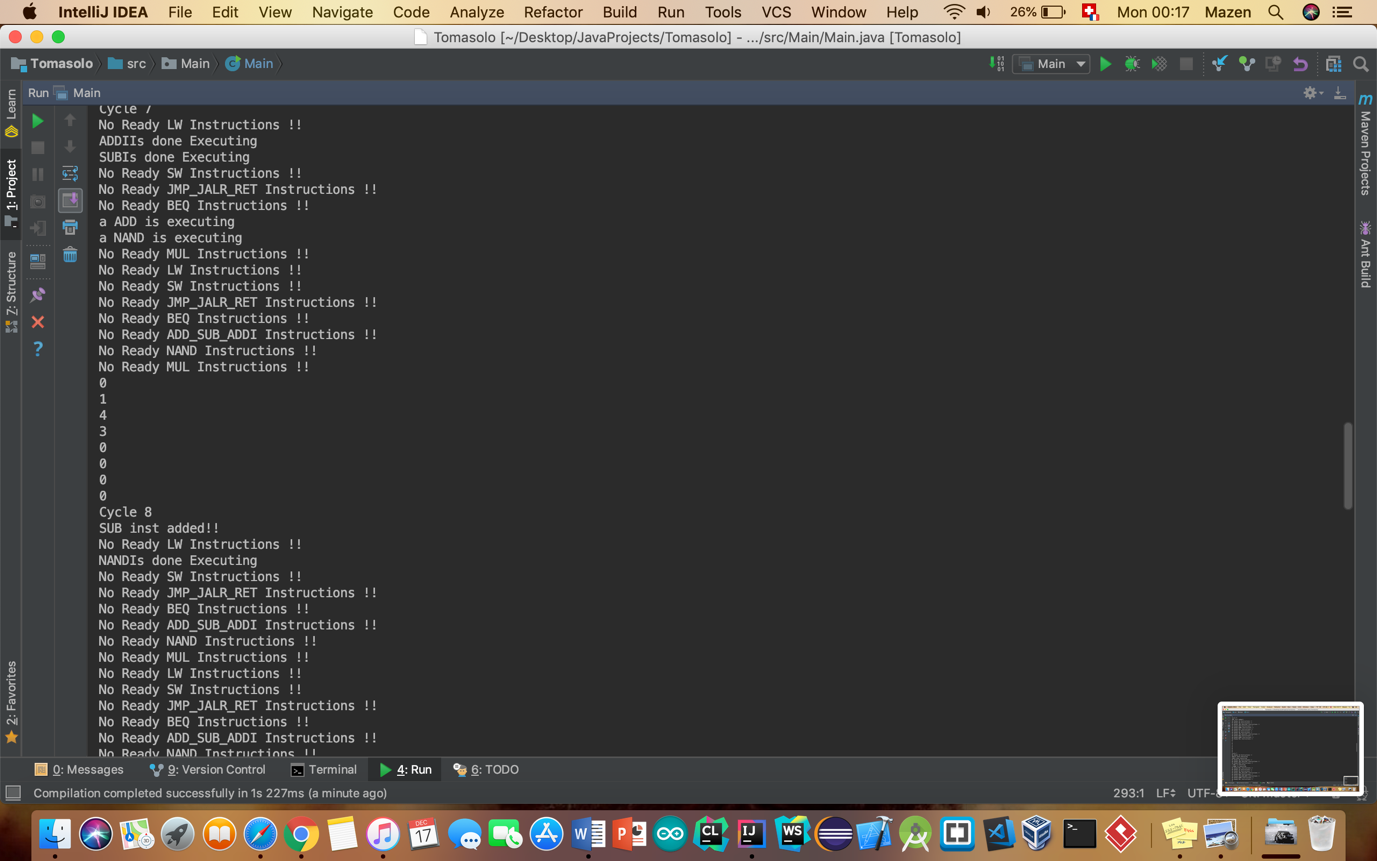
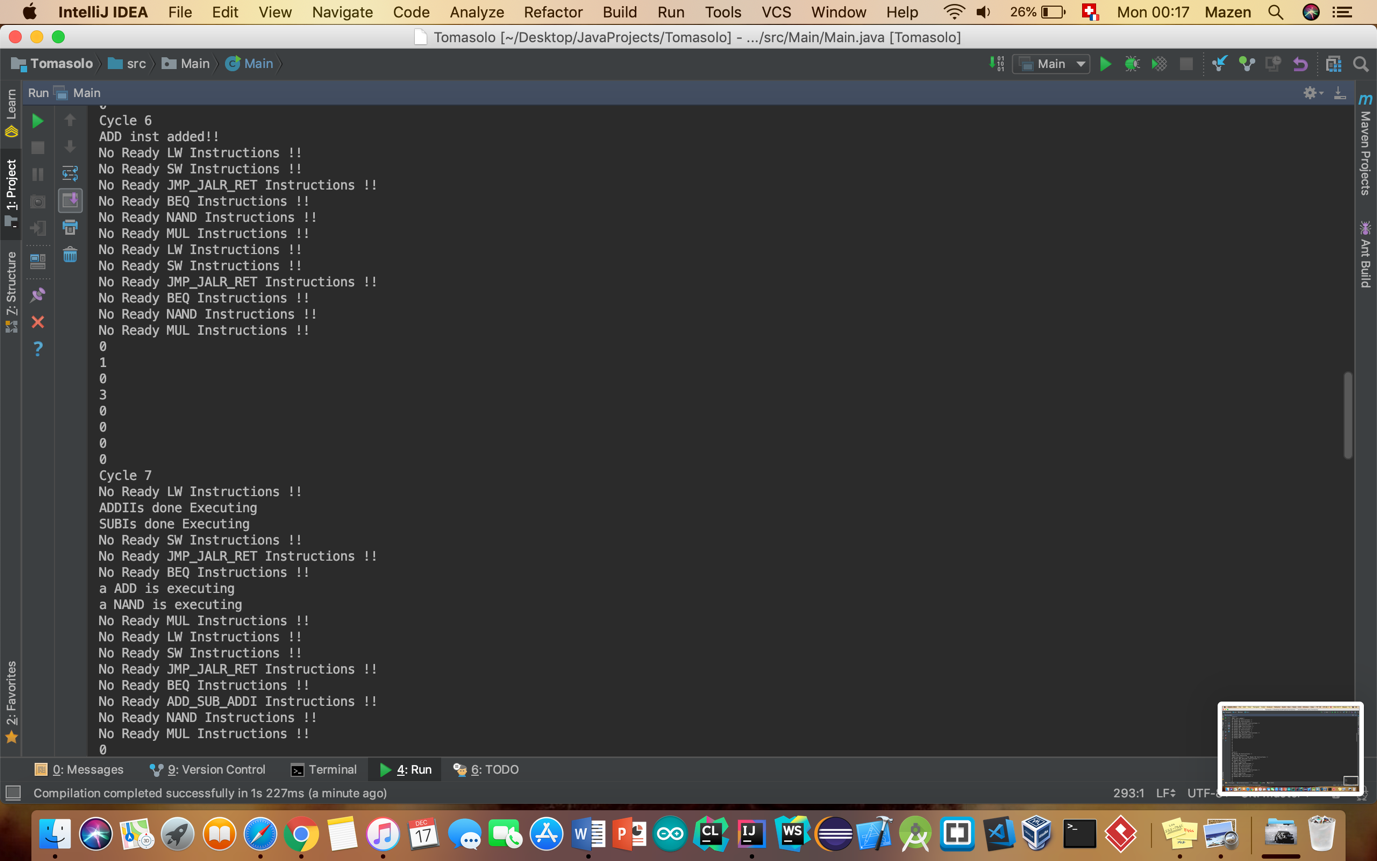
Beq and Addi are fetched and issued



Nand is fetched and issued and Beq and Addi are beginning execution



The Branch actually commited at the previous CC, therefore we jumped to the instruction with PC 1 which is addi in this place and issued it at CC 4 and beginned execution at CC 5.



IPC = 0.69

Misprediction Rate = 0 because we only had one branch instruction correctly predicted to be taken. In fact, this is a corner case as the branch was predicted to be not taken, but as the branch was wanting to branch to the immediately succeeding instruction and the pc prediction was the instruction succeeding the branch (because this is what predicting branch not taken is) so it hasn’t been captured as a misprediction.

## BONUS

We implemented a GUI: a website made in React.js where you can type your code manually or just upload the txt file containing your code. In both cases, your code will be previewed to you and highlighted before you click on start simulation. When you click on this last button, a request will be sent to our Back-End with all the assembly instructions then simulated in the backend then sent back to the front-end as a response. This response can then be displayed in the front-end and see even what happens at each clock cycle in the CPU.

The Back-End is made in Spring Framework as it is the most known server framework where you can code in JAVA.

## Repositories

Please visit our website to have a lot nicer GUI to deal with.

For the logfile, please visit the Algorithm’s github page.

Finally, have a look at our repositories on Github:

Tomasulo’s Algorithm repository: <https://github.com/djzenma/Tomasolo>

Front-End Repository: <https://github.com/djzenma/Tomasolo-s-Simulation>

Back-End Spring Repositoy: <https://github.com/KhaledSoliman/TomasuloSimSpring>