

Tomasulo's Simulator  
Dr. Cherif Salama  
Ali Elkhoully, 900212679  
Moaz Hafez, 900214137

## Implementation:

We started by declaring our classes. We made a Reservation Station, with the basic symbols such as vj, vk and such. In the reservation station, we also have a variable how long each station requires. We built a functional Unit to execute the current operations. Instruction Queue to state which instructions we're queuing, also with the total instructions, in case we needed to jump. Finally, we created a register file to store the values, as well as the memory.

We chose to make the GUI as our bonus. We have a GUI built by python Tkinter. We built a text box for each input and output (Stated below in the "How to Run" section) We run the code, the GUI separates the input into lines to execute them easier. We built a "Simulation" Class, which includes all the other components to be used freely in the code.

We start by initializing the reservation stations, which are like those mentioned in the slides. Not to mention, we included many other components to store other parameters, such as memory, instructions and IPC.

In simulating, we start by checking the reservation stations, checking if there is any busy one that is done. We loop through all instructions and decrease the executing time left for the instructions. In case it is done, we write it back into the register file, and state the required registers as ready to use again.

In the second part, we go to fetch the next available instruction. We first check what type of instruction we have, then we include it with the required parameters and registers to use. The code also supports forwarding mechanisms.

Each press on the button iterates through the instructions, just like a for loop. This is done so we can check each iteration and instruction being done. It is also to check and output the state of the stations and instructions whether they are issued yet or not.

That is how each instruction works:

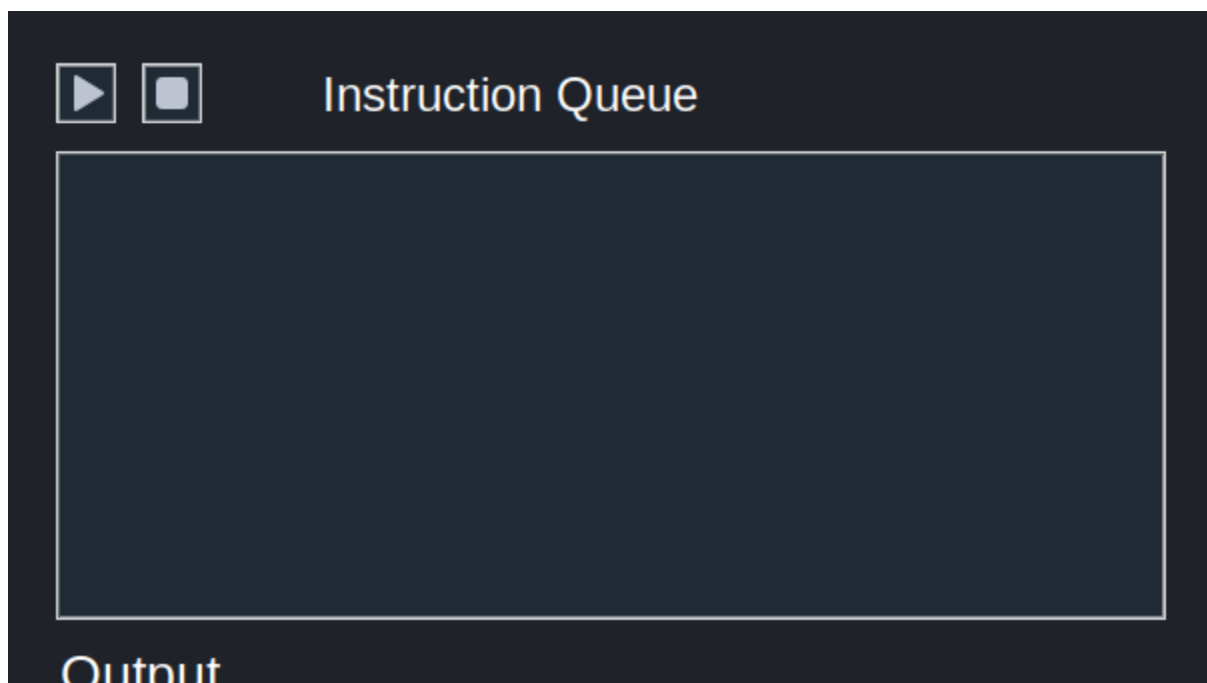
ADD/ADDI/MUL: they start first by reserving an ADD station, then reserve the registers. Then after the required number of clock cycles, we write back the result of the ADD/ADDI/MUL.

BEQ: start the same as the latter. Then, compare the 2 registers, then returns 1 if they are equal, 0 if else. If they are equal, we jump in the Instruction Queue to the required place. This is calculated by  $PC + 1 + \text{Offset}$ .

SD/LD: We start by first 2 clock cycles to calculate the address. Then when there are 4 clock cycles left, we then wait for the rest of the cycles, then write back either into the register or the memory.

RET: We jump directly into the instruction at index stored in R1.

## How to Run:



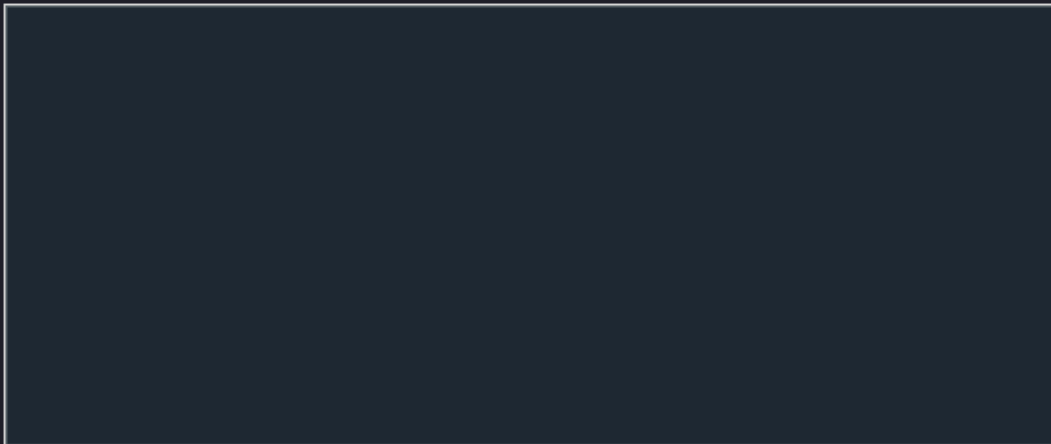
In this part, we write the instructions we want inside this box, then we click the play button each time we want to move 1 clock cycle. Click on the stop sign to start over,



Instruction Queue



Output



We can check the output of each state in the table right below.

▶

Memory  
Input in this format: "Address, Number"

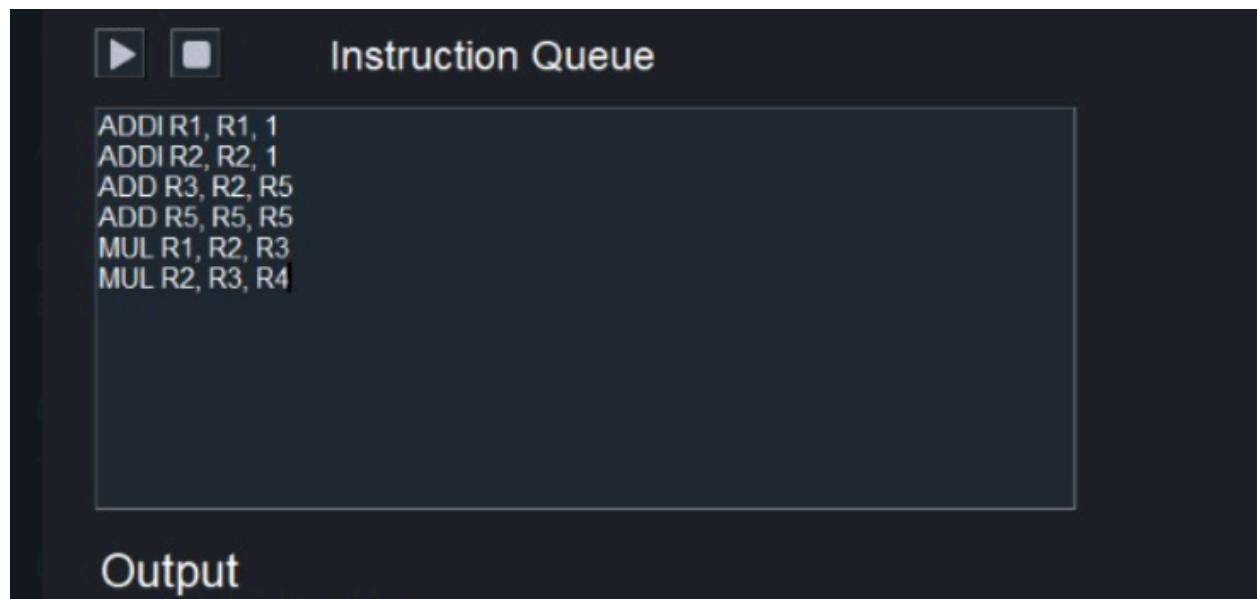
We write the memory as in the format written above, and then click on the button to write the memories.

Right below this Text Box, there is another Text Box that shows the main 3 outputs for the efficiency of the algorithm. As well as the state of each instruction.

### **Test Cases:**

Disclaimer: Here, we output only the final output, as showing each step will take a lot of memory and time.

Code 1:



The image shows a dark-themed window titled "Instruction Queue". At the top left of the window are two square buttons: the first contains a white right-pointing triangle (play icon), and the second contains a white square (stop icon). Below these buttons is a large rectangular area with a dark blue background, containing six lines of assembly code in white text. At the bottom of the window is a section labeled "Output" in white text, which is currently empty.

Instruction Queue

```
ADDI R1, R1, 1  
ADDI R2, R2, 1  
ADD R3, R2, R5  
ADD R5, R5, R5  
MUL R1, R2, R3  
MUL R2, R3, R4
```

Output

## Output

Writing back MUL in MUL

Result: 32

Cycle 23:

R0: 0

R1: 24

R2: 32

R3: 8

R4: 4

R5: 10

R6: 6

R7: 7



R0 status : None

R1 status : None

```
{ID: 0, 'Inst': 'ADDI, R1, R1, 1', 'Issue': 0, 'Exec': 2, 'Write': 3}  
{ID: 1, 'Inst': 'ADDI, R2, R2, 1', 'Issue': 1, 'Exec': 3, 'Write': 4}  
{ID: 2, 'Inst': 'ADD, R3, R2, R5', 'Issue': 2, 'Exec': 5, 'Write': 6}  
{ID: 3, 'Inst': 'ADD, R5, R5, R5', 'Issue': 3, 'Exec': 5, 'Write': 7}  
{ID: 4, 'Inst': 'MUL, R1, R2, R3', 'Issue': 4, 'Exec': 13, 'Write': 14}  
{ID: 5, 'Inst': 'MUL, R2, R3, R4', 'Issue': 14, 'Exec': 22, 'Write': 23}
```

Total Cycles: 23      IPC: 0.2608695652173913

Code 2:



### Instruction Queue

```
ADDI R3, R3, 1
ADDI R2, R0, R2
BEQ R0, R0, 2
ADD R0, R0, R0
ADD R1, R2, R3
ADD R4, R5, R6
SD R2, R4, 0
```

### Output

Cycle 18:

R0: 0  
R1: 1  
R2: 2  
R3: 4  
R4: 11  
R5: 5  
R6: 6  
R7: 7  
R0 status : None  
R1 status : None  
R2 status : None  
R3 status : None



```
{'ID': 0, 'Inst': 'ADDI, R3, R3, 1', 'Issue': 0, 'Exec': 2, 'Write': 3}  
{'ID': 1, 'Inst': 'ADDI, R2, R0, R2', 'Issue': 1, 'Exec': 3, 'Write': 5}  
{'ID': 2, 'Inst': 'BEQ, R0, R0, 2', 'Issue': 2, 'Exec': 3, 'Write': 4}  
{'ID': 3, 'Inst': 'ADD, R0, R0, R0', 'Issue': 3, 'Exec': 5, 'Write': 6}  
{'ID': 4, 'Inst': 'ADD, R1, R2, R3', 'Issue': None, 'Exec': None, 'Write': None}  
{'ID': 5, 'Inst': 'ADD, R4, R5, R6', 'Issue': 4, 'Exec': 6, 'Write': 7}  
{'ID': 6, 'Inst': 'SD, R2, R4, 0', 'Issue': 5, 'Exec': 13, 'Write': 14}
```

Total Cycles: 14      IPC: 0.42857142857142855

Code 3:

### Instruction Queue

```
ADD R0, R0, R0  
BEQ R3, R7, 4  
ADDI R3, R3, 1  
ADD R2, R2, R2  
RET  
ADD R0, R0, R0  
ADD R1, R2, R3  
SD R2, R4, 0 |
```

## Output

Cycle 34:

R0: 0

R1: 40

R2: 32

R3: 8

R4: 4

R5: 5

R6: 6

R7: 7

Memory Address 4: 32

R0 status : None

R1 status : None



R2 status : None

```
{'ID': 0, 'Inst': 'ADD, R0, R0, R0', 'Issue': 23, 'Exec': 2, 'Write': 4}  
{'ID': 1, 'Inst': 'BEQ, R3, R7, 4', 'Issue': 21, 'Exec': 22, 'Write': 23}  
{'ID': 2, 'Inst': 'ADDI, R3, R3, 1', 'Issue': 22, 'Exec': 24, 'Write': 25}  
{'ID': 3, 'Inst': 'ADD, R2, R2, R2', 'Issue': 18, 'Exec': 20, 'Write': 22}  
{'ID': 4, 'Inst': 'RET, None, None, None', 'Issue': 19, 'Exec': 20, 'Write': 21}  
{'ID': 5, 'Inst': 'ADD, R0, R0, R0', 'Issue': 23, 'Exec': 25, 'Write': 26}  
{'ID': 6, 'Inst': 'ADD, R1, R2, R3', 'Issue': 24, 'Exec': 27, 'Write': 28}  
{'ID': 7, 'Inst': 'SD, R2, R4, 0', 'Issue': 25, 'Exec': 31, 'Write': 32}
```

Total Cycles: 32      IPC: 0.25

Branches Misprediction: 0.2

Code 4:



### Instruction Queue

```
ADDI R1, R1, 1
ADDI R2, R2, 1
ADD R2, R2, R5
ADD R5, R2, R5
MUL R1, R2, R3
MUL R6, R1, R3
```

---



### Output

```
Cycle 23:
R0: 0
R1: 9
R2: 8
R3: 3
R4: 4
R5: 13
R6: 27
R7: 7
Memory Address 4: 0
R0 status : None
R1 status : None
R2 status : None
```

```
{ID: 0, 'Inst': 'ADDI, R1, R1, 1', 'Issue': 0, 'Exec': 2, 'Write': 3}  
{ID: 1, 'Inst': 'ADDI, R2, R2, 1', 'Issue': 1, 'Exec': 3, 'Write': 4}  
{ID: 2, 'Inst': 'ADD, R2, R2, R5', 'Issue': 2, 'Exec': 5, 'Write': 6}  
{ID: 3, 'Inst': 'ADD, R5, R2, R5', 'Issue': 3, 'Exec': 7, 'Write': 8}  
{ID: 4, 'Inst': 'MUL, R1, R2, R3', 'Issue': 4, 'Exec': 12, 'Write': 13}  
{ID: 5, 'Inst': 'MUL, R6, R1, R3', 'Issue': 13, 'Exec': 21, 'Write': 22}
```

Total Cycles: 22    IPC: 0.2727272727272727  
Branches Misprediction: 0

Code 5:



### Instruction Queue

```
ADD R0, R0, R0  
BEQ R3, R7, 4  
ADDI R3, R3, 1  
ADD R2, R2, R2  
RET  
ADD R0, R0, R0  
ADD R1, R2, R3  
SD R2, R4, 0
```

## Output

```
Cycle 33:  
R0: 0  
R1: 40  
R2: 32  
R3: 8  
R4: 4  
R5: 5  
R6: 6  
R7: 7  
Memory Address 4: 32  
R0 status : None  
R1 status : None  
R2 status : None
```

```
{ID: 0, 'Inst': 'ADD, R0, R0, R0', 'Issue': 23, 'Exec': 2, 'Write': 4}  
{ID: 1, 'Inst': 'BEQ, R3, R7, 4', 'Issue': 21, 'Exec': 22, 'Write': 23}  
{ID: 2, 'Inst': 'ADDI, R3, R3, 1', 'Issue': 22, 'Exec': 24, 'Write': 25}  
{ID: 3, 'Inst': 'ADD, R2, R2, R2', 'Issue': 18, 'Exec': 20, 'Write': 22}  
{ID: 4, 'Inst': 'RET, None, None, None', 'Issue': 19, 'Exec': 20, 'Write': 21}  
{ID: 5, 'Inst': 'ADD, R0, R0, R0', 'Issue': 23, 'Exec': 25, 'Write': 26}  
{ID: 6, 'Inst': 'ADD, R1, R2, R3', 'Issue': 24, 'Exec': 27, 'Write': 28}  
{ID: 7, 'Inst': 'SD, R2, R4, 0', 'Issue': 25, 'Exec': 31, 'Write': 32}
```

```
Total Cycles: 32    IPC: 0.25  
Branches Misprediction: 0.2
```

Code 6:



## Instruction Queue

```
BEQ R3, R7, 4  
ADDI R3, R3, 1  
ADD R2, R2, R2  
BEQ R0, R0, -4  
ADD R0, R0, R0  
ADD R1, R2, R3  
SD R2, R4, 0
```

## Output

```
Writing back SD in SD  
Result: 4  
Cycle 30:  
R0: 0  
R1: 40  
R2: 32  
R3: 8  
R4: 4  
R5: 5  
R6: 6  
R7: 7  
Memory Address 4: 32  
R0 status : None
```

```
{ID: 0, 'Inst': 'BEQ, R3, R7, 4', 'Issue': 20, 'Exec': 21, 'Write': 22}  
{ID: 1, 'Inst': 'ADDI, R3, R3, 1', 'Issue': 21, 'Exec': 23, 'Write': 24}  
{ID: 2, 'Inst': 'ADD, R2, R2, R2', 'Issue': 17, 'Exec': 19, 'Write': 21}  
{ID: 3, 'Inst': 'BEQ, R0, R0, -4', 'Issue': 18, 'Exec': 19, 'Write': 20}  
{ID: 4, 'Inst': 'ADD, R0, R0, R0', 'Issue': 19, 'Exec': 21, 'Write': 23}  
{ID: 5, 'Inst': 'ADD, R1, R2, R3', 'Issue': 22, 'Exec': 25, 'Write': 26}  
{ID: 6, 'Inst': 'SD, R2, R4, 0', 'Issue': 23, 'Exec': 29, 'Write': 30}
```

Total Cycles: 30      IPC: 0.23333333333333334

Branches Misprediction: 0.5555555555555556