Report Project 2

Joseph Boulis 900182870
Mohamed Mansour 900172822
Mohamed Basuony 900182339

Brief description of the implementation:

Code is divided into the following stages:

- Parses input instructions into opcodes and register names/ immediates

- Starts issuing from the first instruction, placing it in a free reservation station and incrementing the instruction counter. If there is no free appropriate reservation station we stall the instruction counter and don't issue. If we issue an instruction, we add it to the output table containing the clock timers. The issued instruction is also added to a "write queue" which will be discussed later.

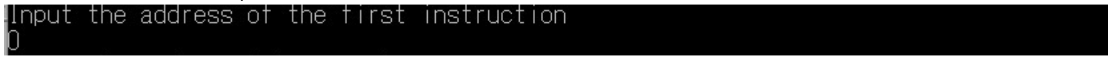
- The execution step loops over all reservation stations each clock cycle:
    1) If the reservation station didn't start executing but has all of it operands available. It starts executing
    2) If a reservation station is executing but didn't finish executing yet, its execution timer is decremented
    3) If a reservation station finished execution, the clock cycle is recorded and it is marked in the write queue as ready in the following clock cycle

- The writing step goes through all instructions in the write queue in order, and only writes from the first ready reservation station it encounters (ensuring there is only one write per clock cycle). When writing we either update the memory, the instruction counter, or the register file and any reservations stations awaiting a RAW dependency. Once we write, we record the current clock cycle in the appropriate output table entry.

- The program finishes execution once all entries in the output table have went through the writing stage, and the instruction counter passed the final instruction.

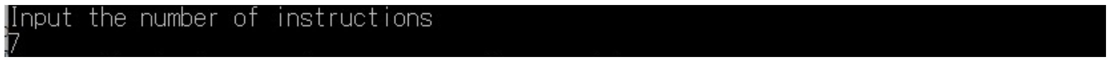
User's guide:

1- Upon running the code, the user is prompted to enter the address of the first instruction, here the address is 0:



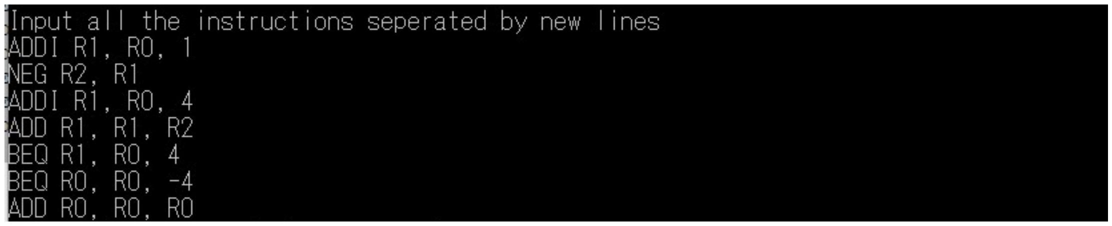
```
Input the address of the first instruction
0
```

2- The user enters the number of instructions, here, the number of instructions is 7:



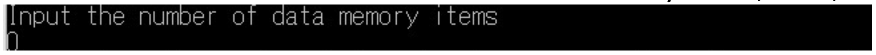
```
Input the number of instructions
7
```

3- The user enters the instructions that will be tested.



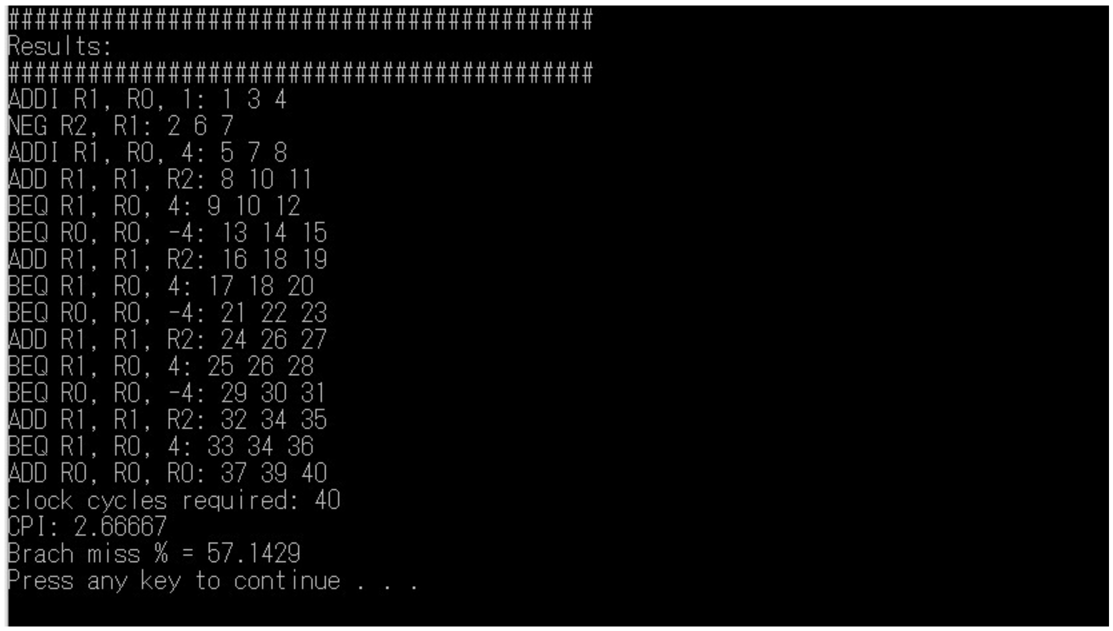
```
Input all the instructions seperated by new lines
ADDI R1, R0, 1
NEG R2, R1
ADDI R1, R0, 4
ADD R1, R1, R2
BEQ R1, R0, 4
BEQ R0, R0, -4
ADD R0, R0, R0
```

4- The user enters the number of data memory items, here, the number is 0.



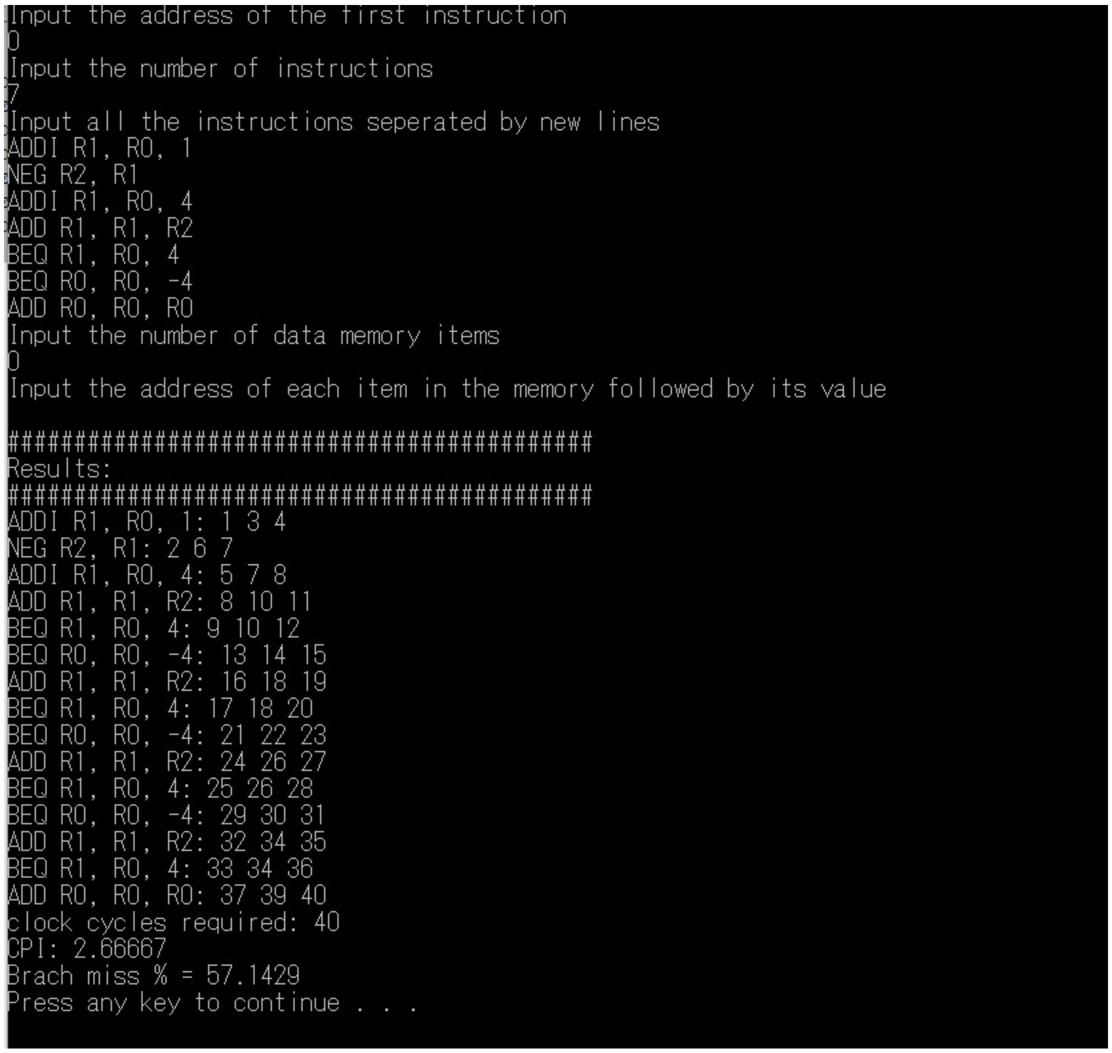
```
Input the number of data memory items
0
```

The result is then printed out for the user in the following form:

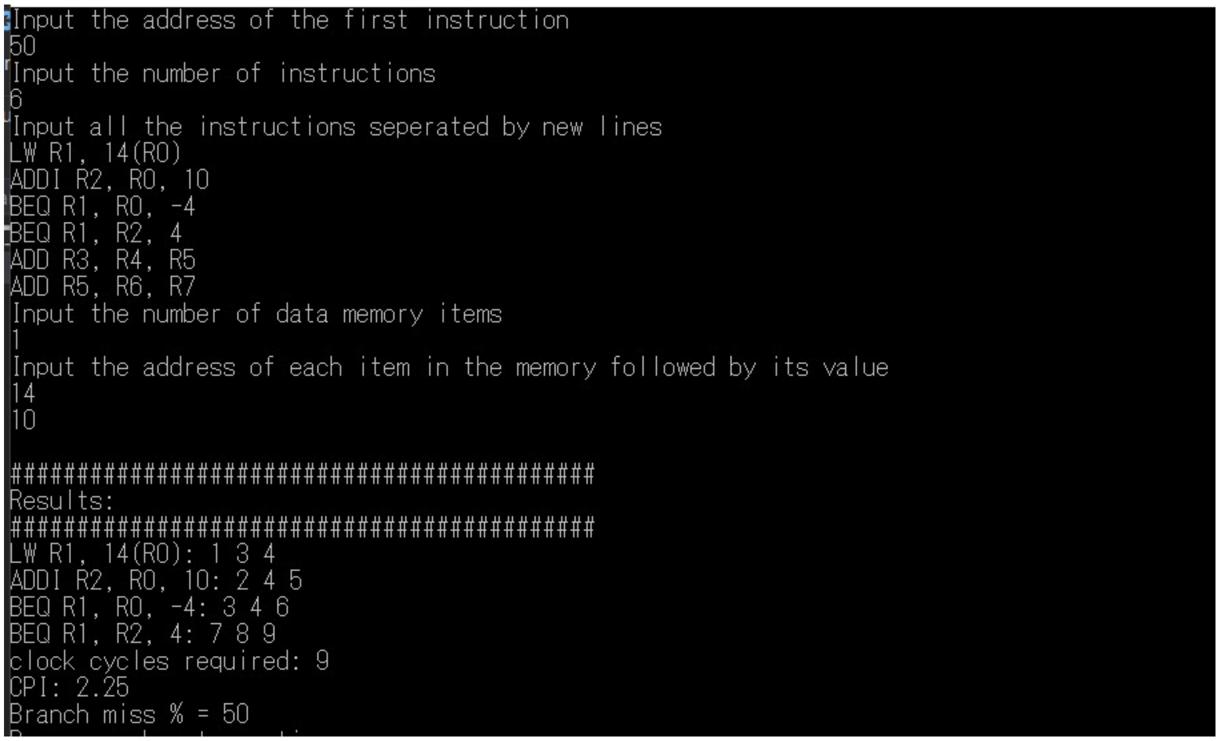


```
############################################
Results:
############################################
ADDI R1, R0, 1: 1 3 4
NEG R2, R1: 2 6 7
ADDI R1, R0, 4: 5 7 8
ADD R1, R1, R2: 8 10 11
BEQ R1, R0, 4: 9 10 12
BEQ R0, R0, -4: 13 14 15
ADD R1, R1, R2: 16 18 19
BEQ R1, R0, 4: 17 18 20
BEQ R0, R0, -4: 21 22 23
ADD R1, R1, R2: 24 26 27
BEQ R1, R0, 4: 25 26 28
BEQ R0, R0, -4: 29 30 31
ADD R1, R1, R2: 32 34 35
BEQ R1, R0, 4: 33 34 36
ADD R0, R0, R0: 37 39 40
clock cycles required: 40
CPI: 2.66667
Brach miss % = 57.1429
Press any key to continue . . .
```

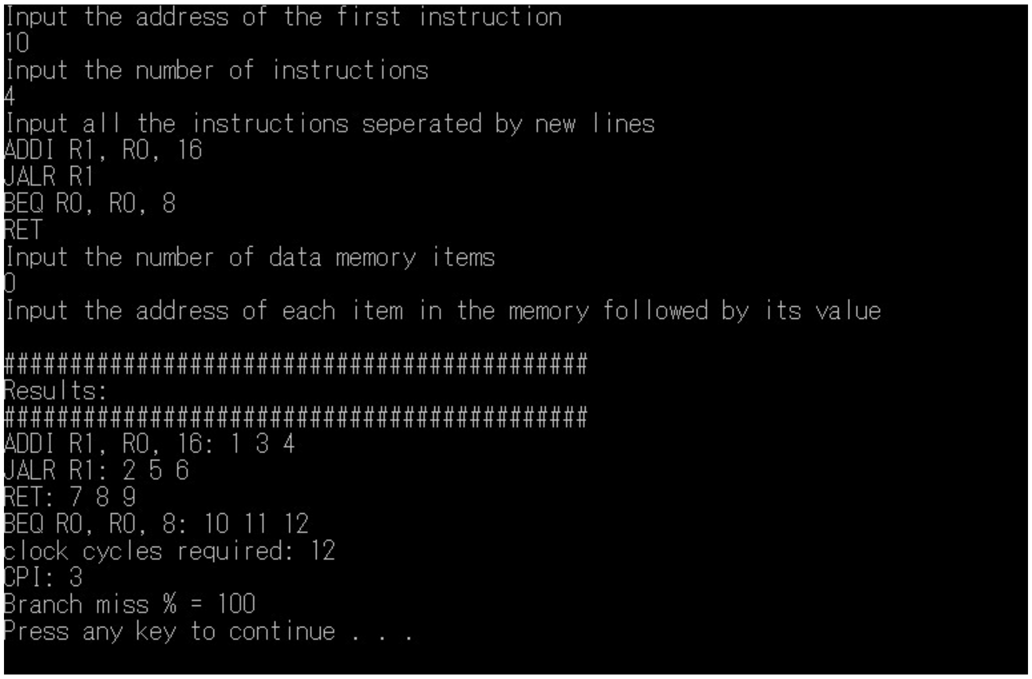
Results:



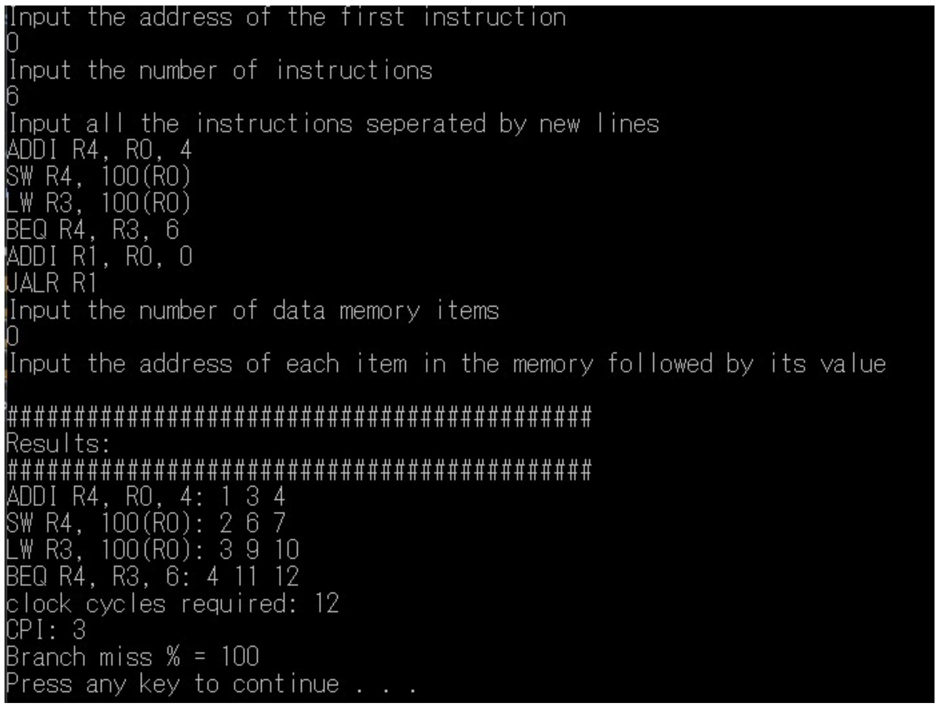
```
Input the address of the first instruction
0
Input the number of instructions
7
Input all the instructions seperated by new lines
ADDI R1, R0, 1
NEG R2, R1
ADDI R1, R0, 4
ADD R1, R1, R2
BEQ R1, R0, 4
BEQ R0, R0, -4
ADD R0, R0, R0
Input the number of data memory items
0
Input the address of each item in the memory followed by its value

############################################
Results:
############################################
ADDI R1, R0, 1: 1 3 4
NEG R2, R1: 2 6 7
ADDI R1, R0, 4: 5 7 8
ADD R1, R1, R2: 8 10 11
BEQ R1, R0, 4: 9 10 12
BEQ R0, R0, -4: 13 14 15
ADD R1, R1, R2: 16 18 19
BEQ R1, R0, 4: 17 18 20
BEQ R0, R0, -4: 21 22 23
ADD R1, R1, R2: 24 26 27
BEQ R1, R0, 4: 25 26 28
BEQ R0, R0, -4: 29 30 31
ADD R1, R1, R2: 32 34 35
BEQ R1, R0, 4: 33 34 36
ADD R0, R0, R0: 37 39 40
clock cycles required: 40
CPI: 2.66667
Brach miss % = 57.1429
Press any key to continue . . .
```



```
Input the address of the first instruction
50
Input the number of instructions
6
Input all the instructions seperated by new lines
LW R1, 14(R0)
ADDI R2, R0, 10
BEQ R1, R0, -4
BEQ R1, R2, 4
ADD R3, R4, R5
ADD R5, R6, R7
Input the number of data memory items
1
Input the address of each item in the memory followed by its value
14
10

###########################################
Results:
###########################################
LW R1, 14(R0): 1 3 4
ADDI R2, R0, 10: 2 4 5
BEQ R1, R0, -4: 3 4 6
BEQ R1, R2, 4: 7 8 9
clock cycles required: 9
CPI: 2.25
Branch miss % = 50
```

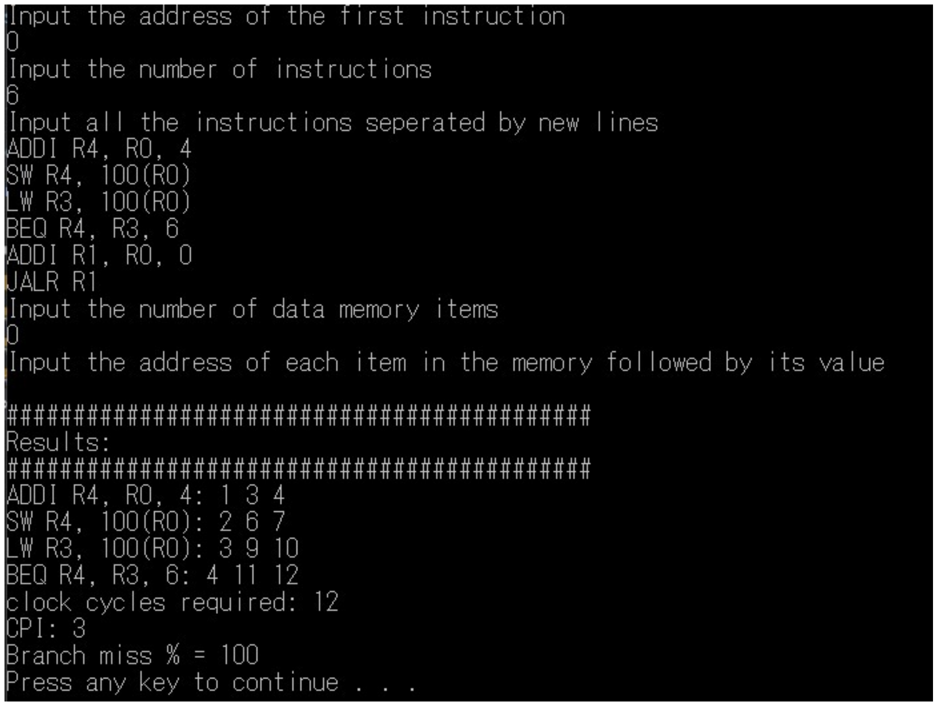


```
Input the address of the first instruction
10
Input the number of instructions
4
Input all the instructions seperated by new lines
ADDI R1, R0, 16
JALR R1
BEQ R0, R0, 8
RET
Input the number of data memory items
0
Input the address of each item in the memory followed by its value

###########################################
Results:
###########################################
ADDI R1, R0, 16: 1 3 4
JALR R1: 2 5 6
RET: 7 8 9
BEQ R0, R0, 8: 10 11 12
clock cycles required: 12
CPI: 3
Branch miss % = 100
Press any key to continue . . .
```



```
Input the address of the first instruction
0
Input the number of instructions
6
Input all the instructions seperated by new lines
ADDI R4, R0, 4
SW R4, 100(R0)
LW R3, 100(R0)
BEQ R4, R3, 6
ADDI R1, R0, 0
JALR R1
Input the number of data memory items
0
Input the address of each item in the memory followed by its value

############################################
Results:
############################################
ADDI R4, R0, 4: 1 3 4
SW R4, 100(R0): 2 6 7
LW R3, 100(R0): 3 9 10
BEQ R4, R3, 6: 4 11 12
clock cycles required: 12
CPI: 3
Branch miss % = 100
Press any key to continue . . .
```

Explanation:



```
Input the address of the first instruction
0
Input the number of instructions
6
Input all the instructions seperated by new lines
ADDI R4, R0, 4
SW R4, 100(R0)
LW R3, 100(R0)
BEQ R4, R3, 6
ADDI R1, R0, 0
JALR R1
Input the number of data memory items
0
Input the address of each item in the memory followed by its value

############################################
Results:
############################################
ADDI R4, R0, 4: 1 3 4
SW R4, 100(R0): 2 6 7
LW R3, 100(R0): 3 9 10
BEQ R4, R3, 6: 4 11 12
clock cycles required: 12
CPI: 3
Branch miss % = 100
Press any key to continue . . .
```

In this program, we have 6 instruction. We follow an in-order issuing so it can be seen that of there are two instructions I1 and I2, where I1 is before I2 in the instruction queue we cannot issue I2 before I1. Moreover, after we load the value R3 from 100(R0) and reach BEQ instruction that compares the value of R3 to the value of R4 (equal to 4) and found out that they are indeed equal, we jump to pc+6 which exceeds our final accessible pc address, so the program terminates. We finish exciting this program in 12 clock cycles. Since we follow an always not taken branching predictor, we fetched the instructions after BEQ while we did not need to do so. Thus, we have 1 branch miss out of total 1 branch, so we have 100% miss prediction rate.