Name : Ciarán Coady
Student Number : 17326951

Question 1

PT1

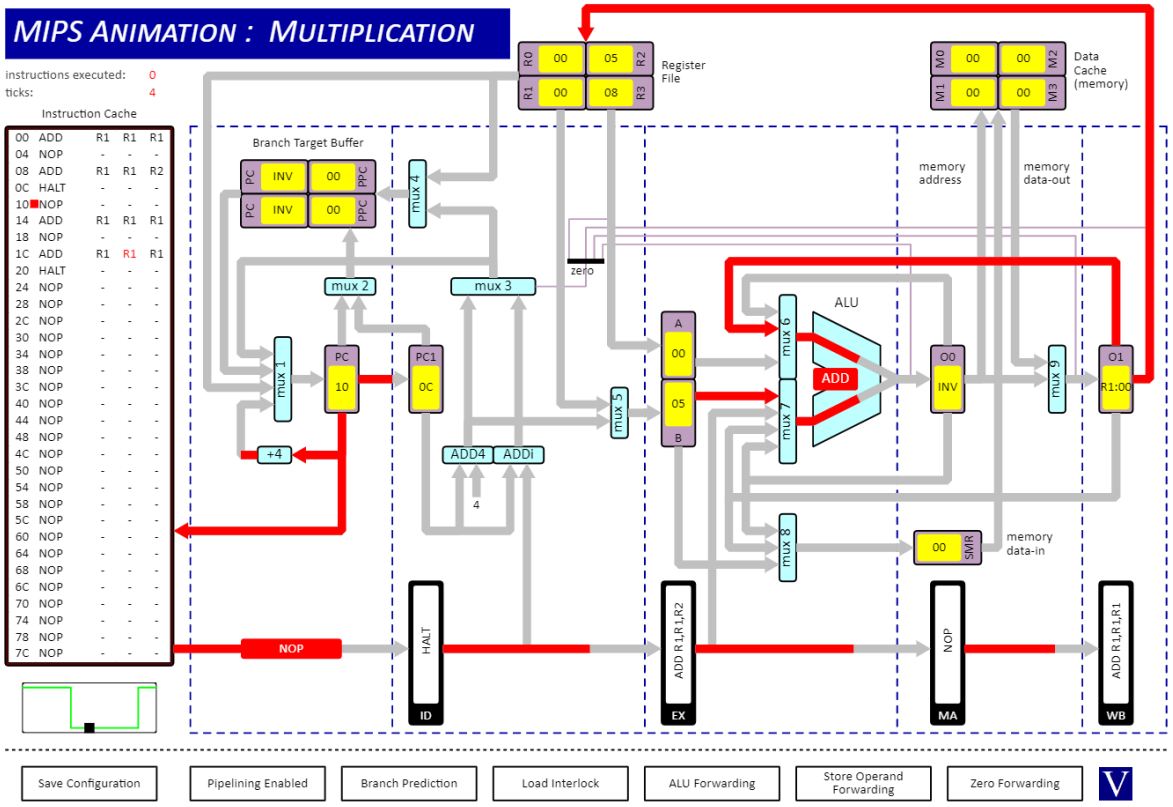
Instructions

ADD R1, R1, R1

NOP -, -, -

ADD R1, R1, R2

Screenshot



Name : Ciarán Coady
Student Number : 17326951

PT2

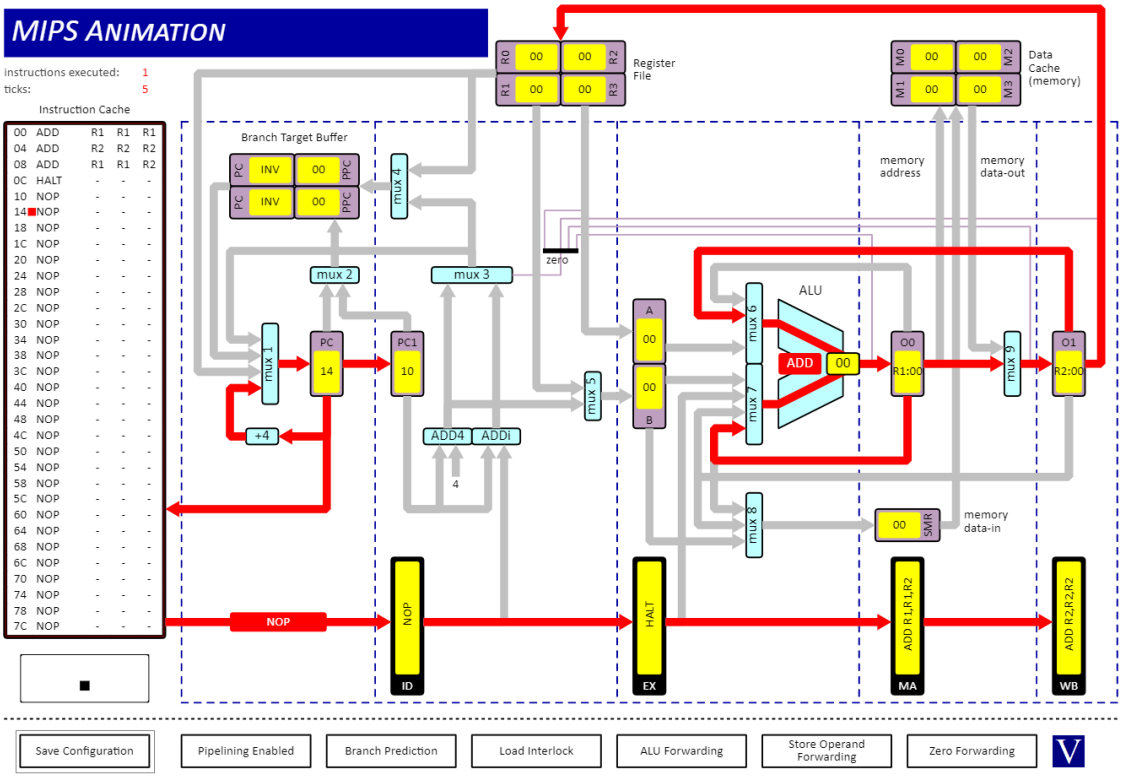
Instructions

ADD R1, R1, R1

ADD R2, R2, R2

ADD R1, R1, R2

Screenshot



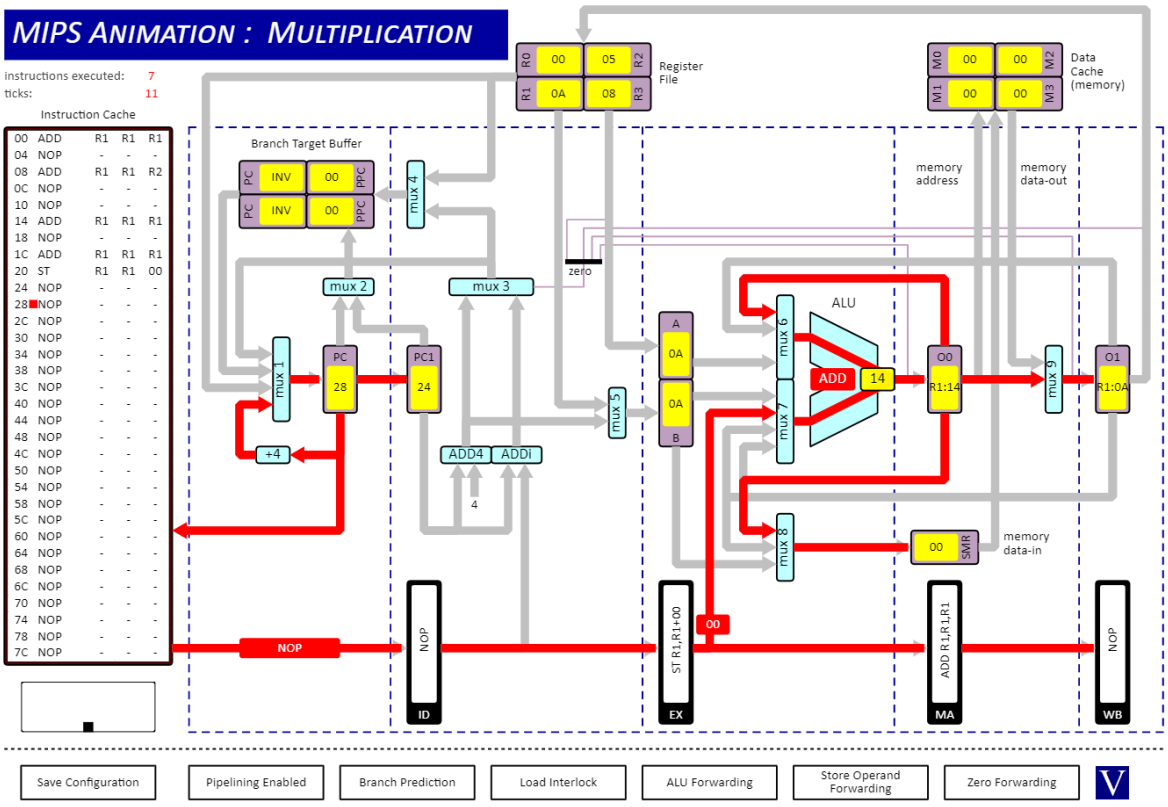
Name : Ciarán Coady
Student Number : 17326951

PT3

Instruction

ST R1, R1, 00

Screenshot



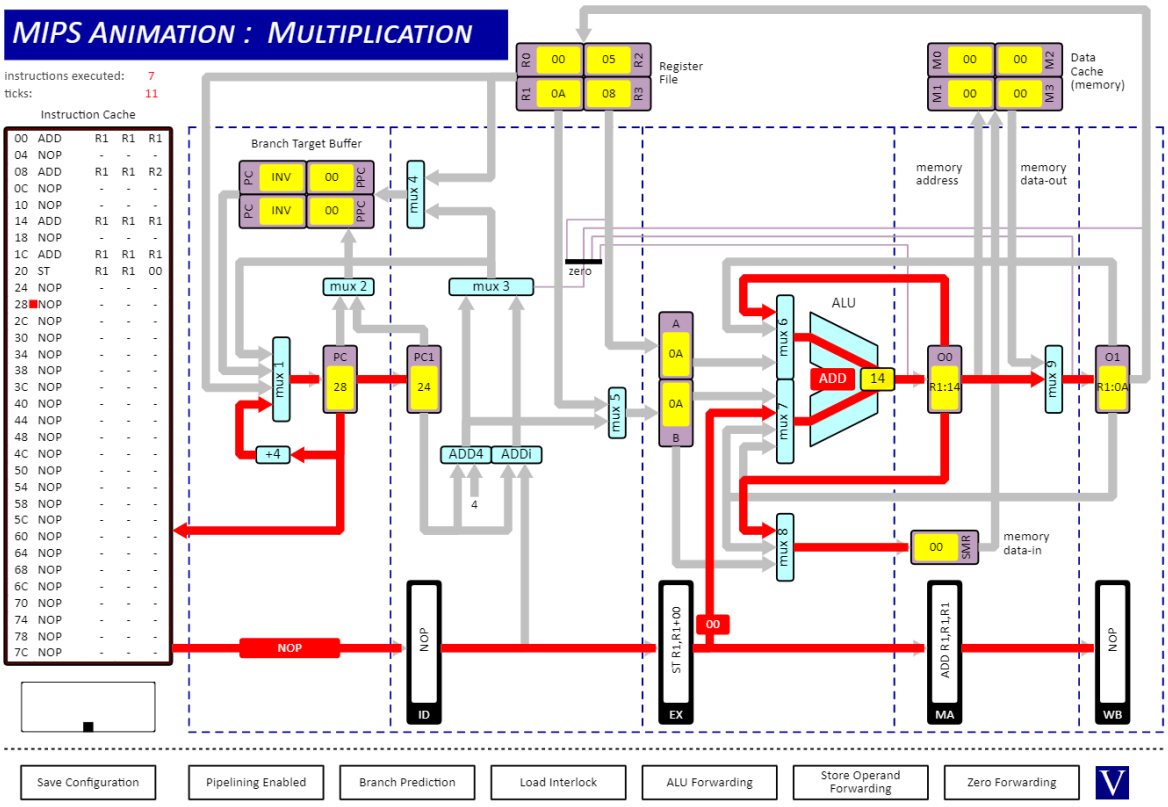
Name : Ciarán Coady
Student Number : 17326951

PT4

Instruction

ST R1, R1, 00

Screenshot



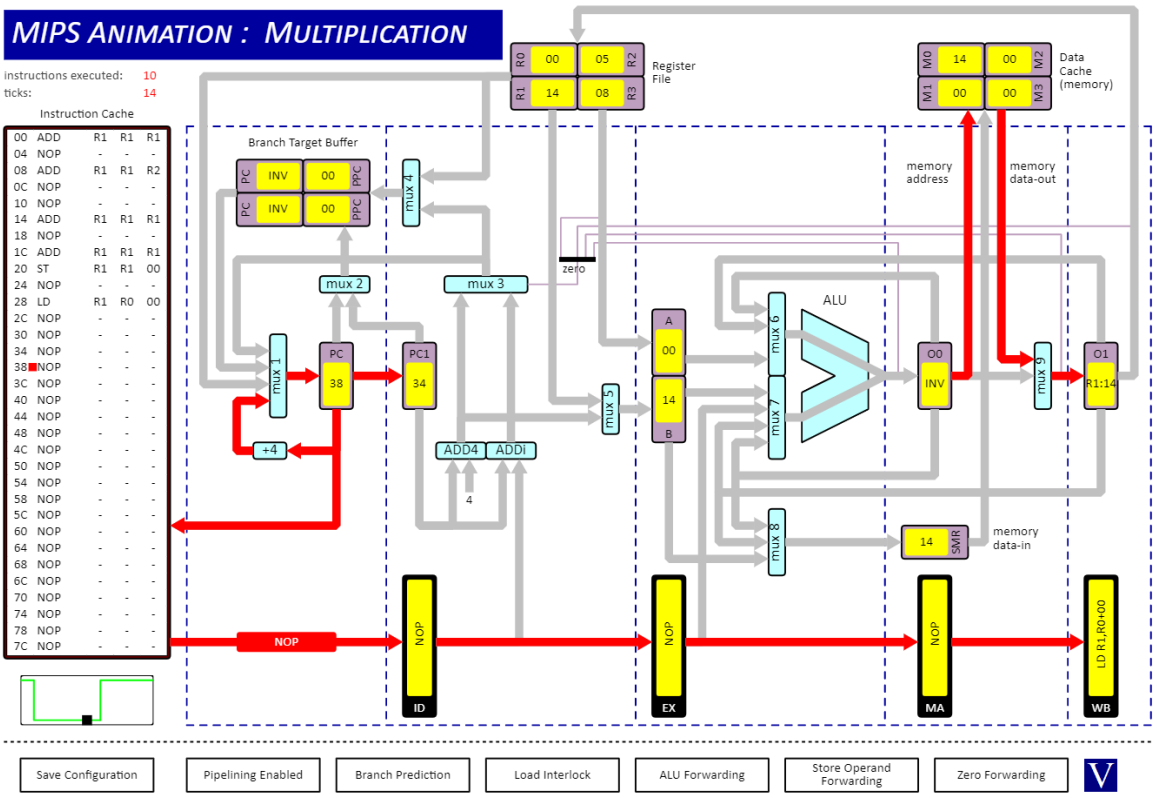
Name : Ciarán Coady
Student Number : 17326951

PT5

Instruction

LD R1, R0, 00

Screenshot



Name : Ciarán Coady
Student Number : 17326951

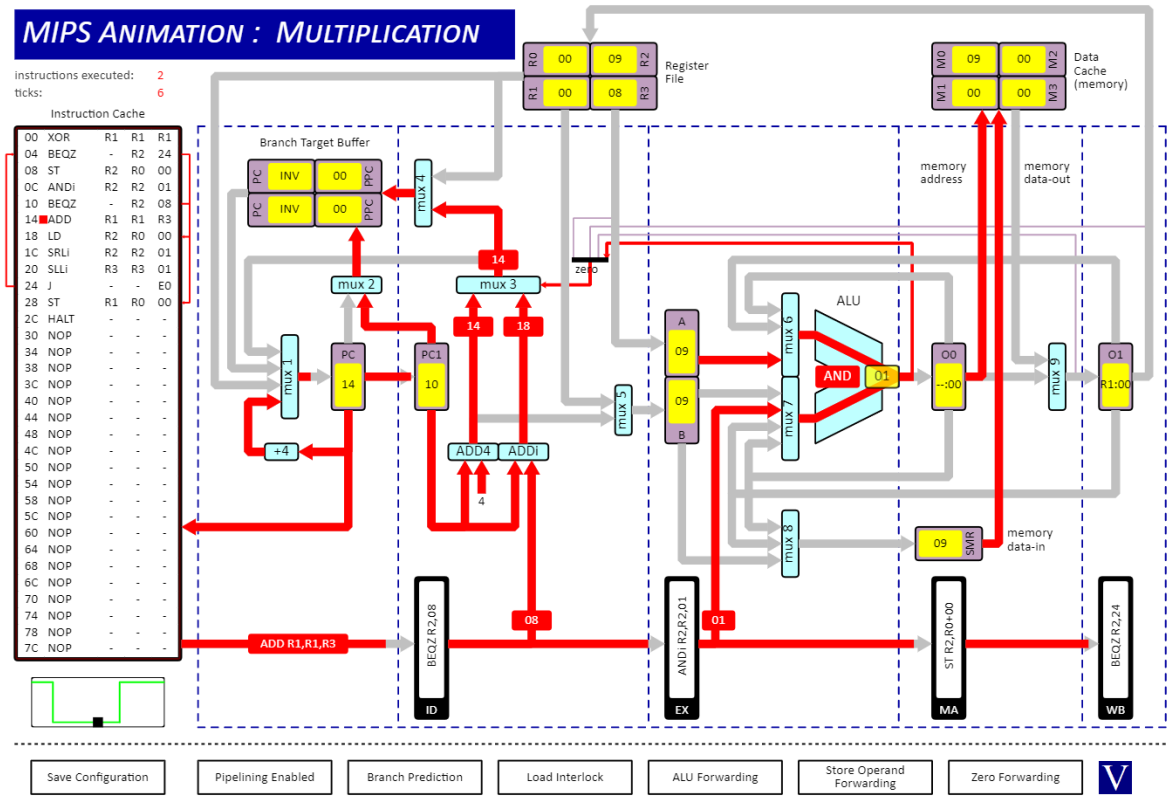
PT6

Instructions

ANDi R2, R2, 01

BEQZ -, R2, 08

Screenshot



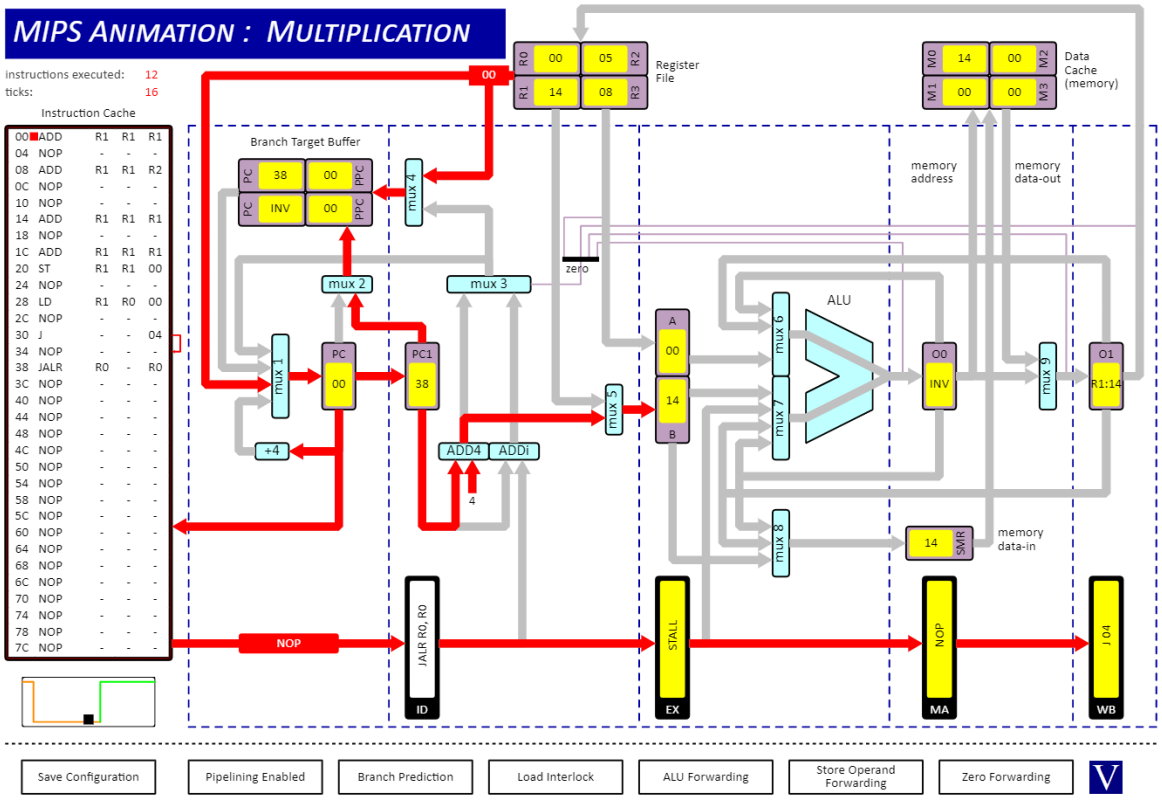
Name : Ciarán Coady
Student Number : 17326951

PT7

Instruction

JALR RO, -, R0

Screenshot



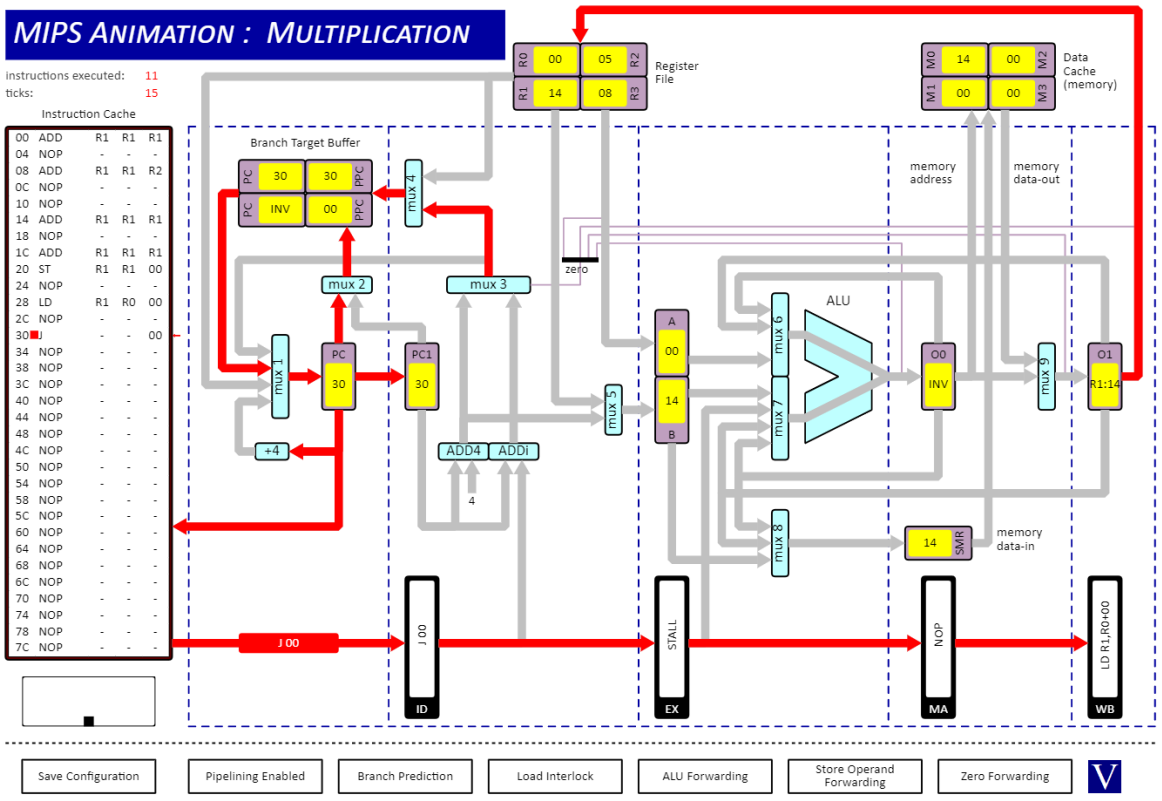
Name : Ciarán Coady
Student Number : 17326951

PT8

Instruction

J -, -, 00

Screenshot



Name : Ciarán Coady
Student Number : 17326951

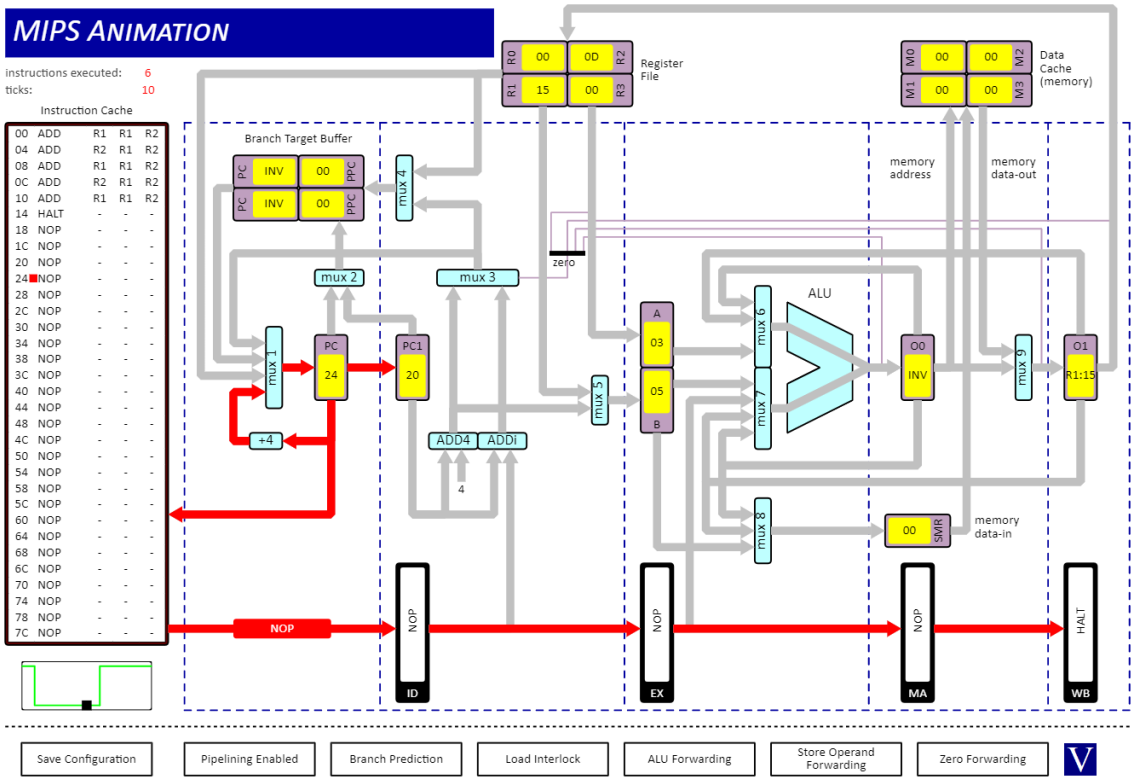
Question 2

Part I

Number of clock cycles needed to execute : 10

Resulting value of R1 : Hex 0x15 == 21

Screenshot



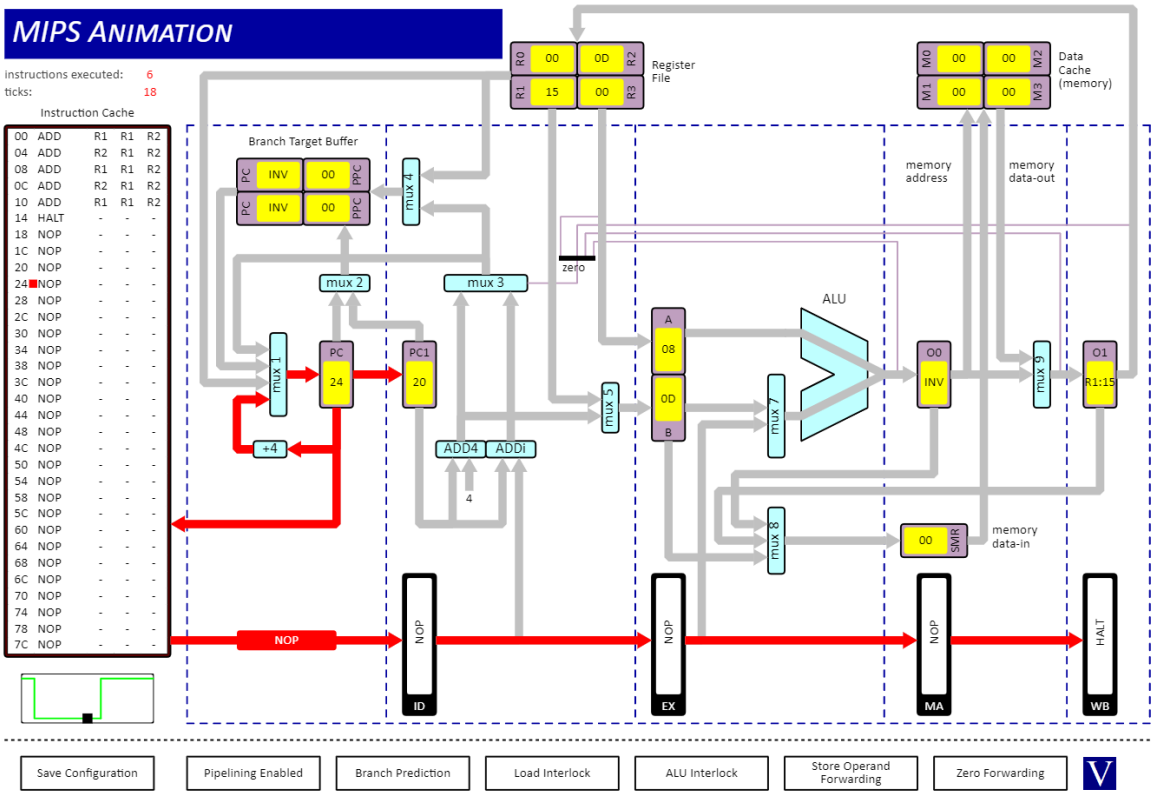
Name : Ciarán Coady
Student Number : 17326951

Part II

Number of clock cycles needed to execute : 18

Resulting value of R1 : Hex 0x15 == 21

Screenshot



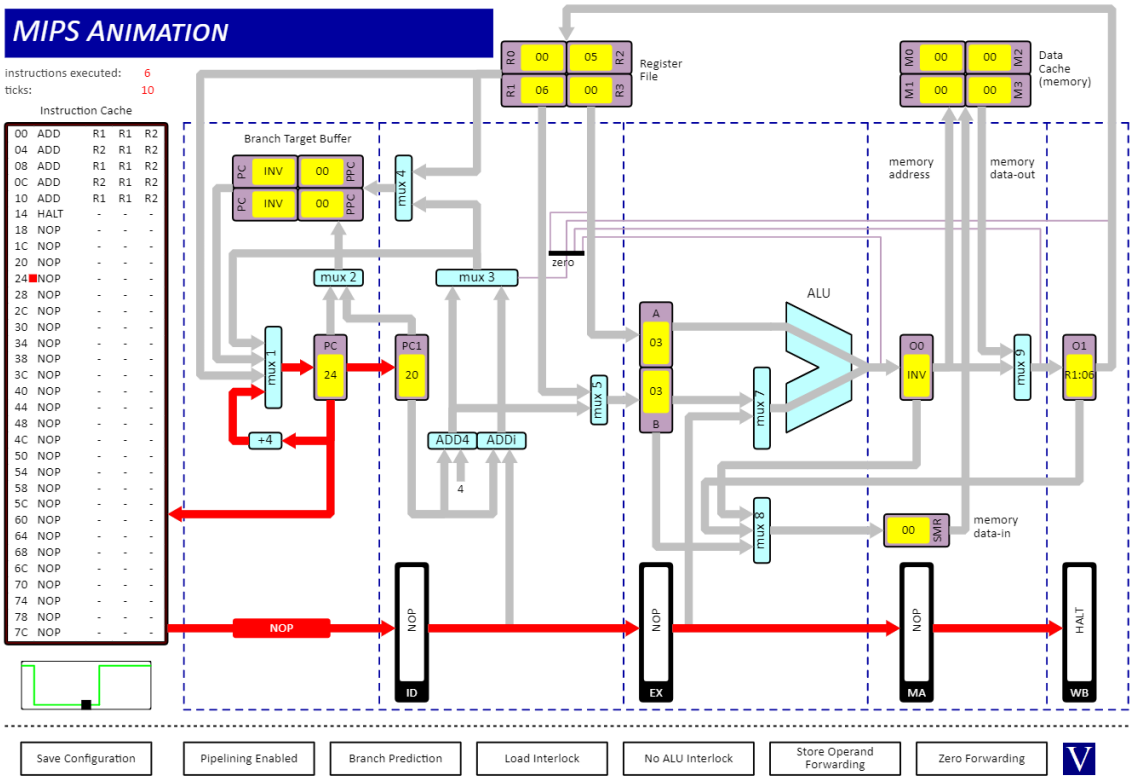
Name : Ciarán Coady
Student Number : 17326951

Part III

Number of clock cycles needed to execute : 10

Resulting value of R1 : Hex 0x06 == 6

Screenshot



Explain in detail why the results and number of clock cycles are different?

The number of clock cycles are different because in part 1 we have alu forwarding enabled. This means that we can immediately get access to the result of the alu operation after its is completed. This is facilitated by O0 and O1. This means that we can get the new values in the following instruction executions even before it has been written back, thus preventing data hazards and/or pipeline stalls.

In part 2 with alu interlock we have no such facility, and thus whenever an instruction is executed, we must wait for the alu operation to be written back (in that case where we have a data hazard) before we can continue with the next operation. This results in the same number of instructions taking more cycles to execute due to these pipeline stalls while waiting for the write back to be completed.

Regardless of execution time, we obtain the correct answer in both parts I and II. This is because we ensure that we use the most up to date version of each register and give adequate time for updates to propagate.

Name : Ciarán Coady
Student Number : 17326951

In part III, it takes the same amount of cycles as part I, but we do not care whether updates have propagated, thus we get an incorrect answer as we proceeded ahead with alu operations even though the results of the last alu operation have not been given time to update the registers.

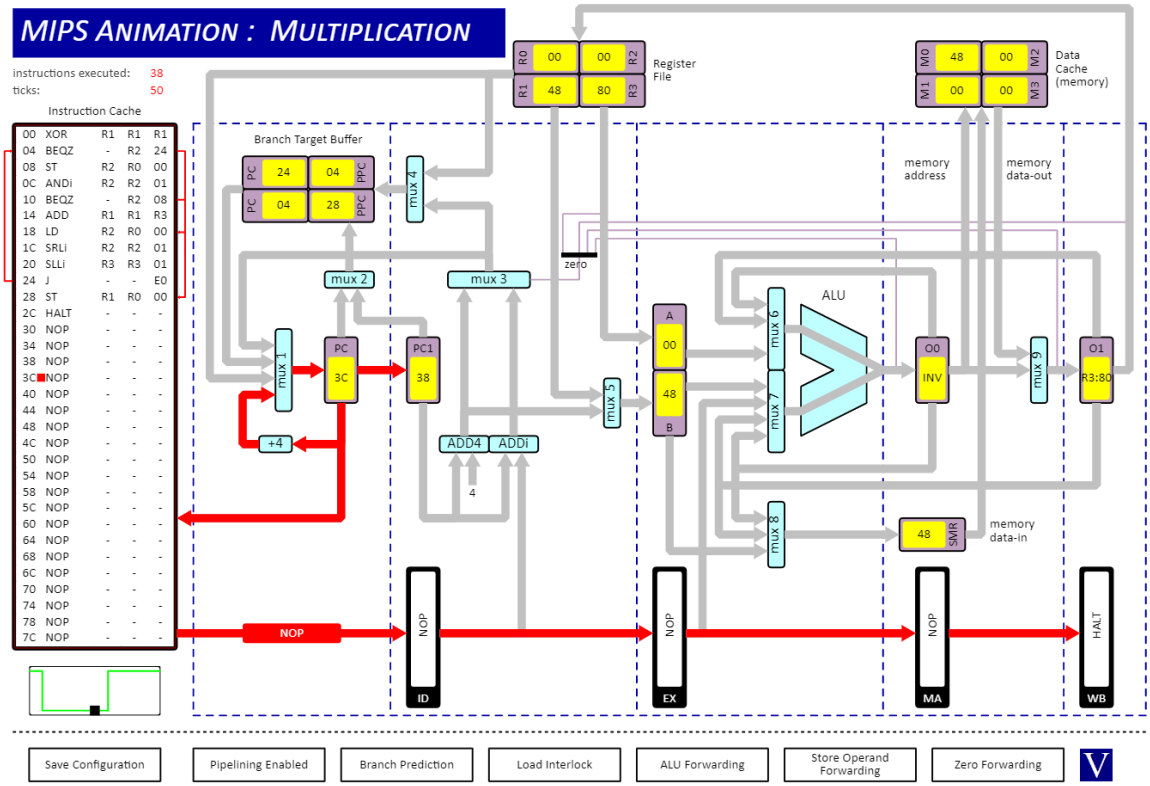
Question 3

Part I

Number of instructions : 38

Number of clock cycles : 50

Screenshot



Why are these numbers not equal?

The numbers are not equal because of pipeline stalls. Before the pipeline has been filled (at the beginning of execution) we have stalls, the number of stalls is equal to the number of stages in the pipeline, so this case 4.

There is data dependency between instruction LD R2, R0, 00 and SRLi R2, R2, 01 as the load instruction takes 2 cycles put the value from memory into O1. These instructions get executed 4 times each so that's 4 more stalls.

The branch instructions cause stalls when the branch prediction is incorrect. This happens a total of 4 times between all the branch instructions. This gives us a total of 4+4+4 = 12 stalls, which accounts for the discrepancy in clock cycles vs instructions.

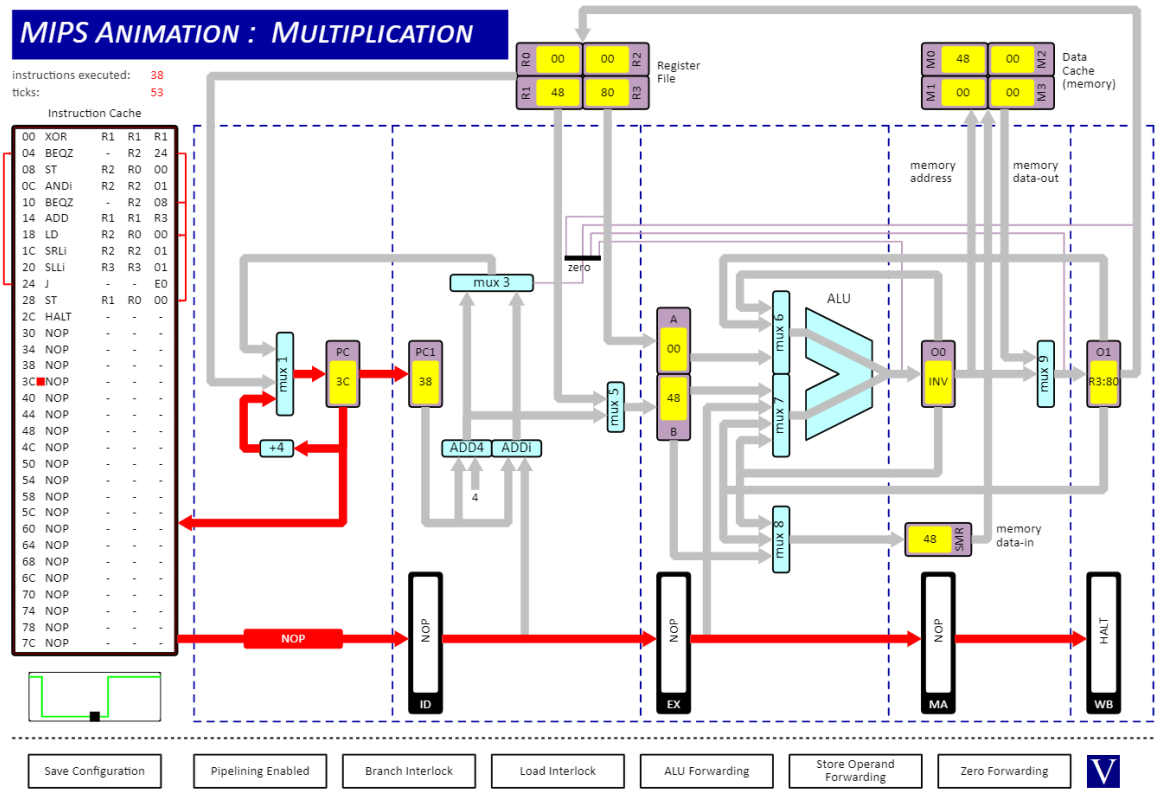
Name : Ciarán Coady
Student Number : 17326951

Part II

Number of instructions : 38

Number of clock cycles : 53

Screenshot



Why does the number of cycles needed to execute this program differ from part I?

This is because in this example we have turned off predictive branching. So now every time we hit a branch instruction it will always cause a pipeline stall unless the branch is not taken, in which case program execution will just continue. More stalls occur as we must compute the branch offset rather than predicting the jump as we could do before with the branch target buffer. In this case we have 3 extra stalls compared to part I giving us a total of 15 stalls.

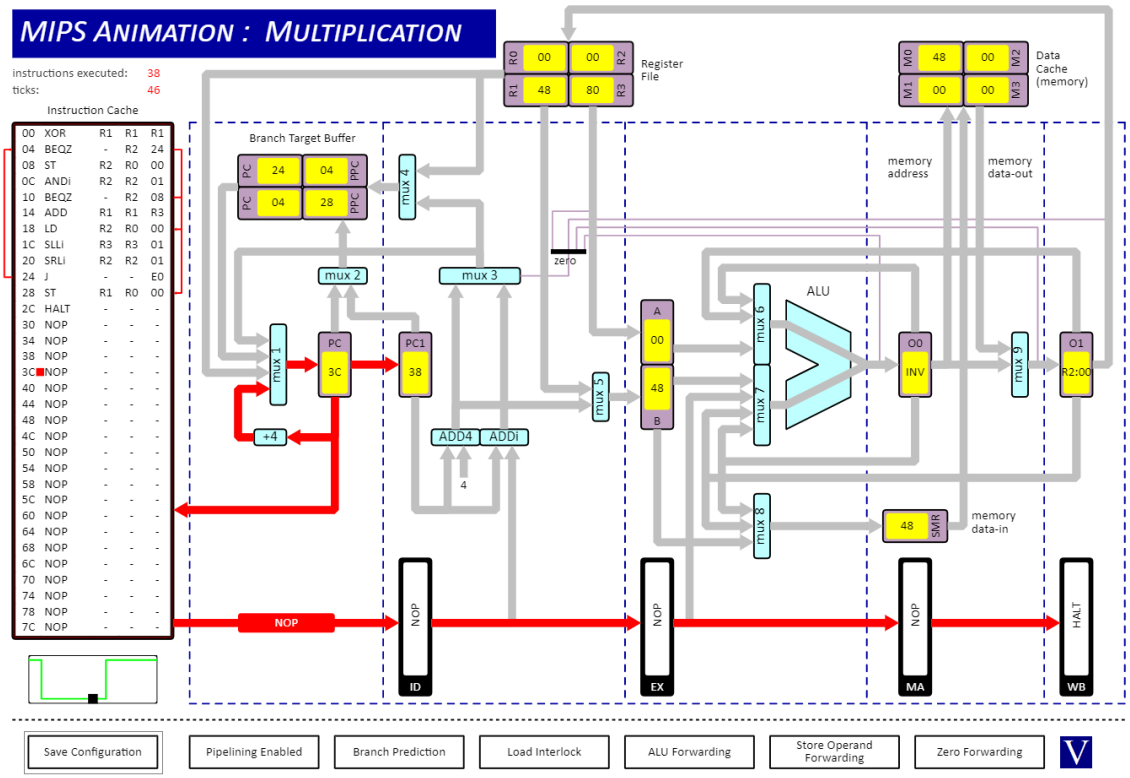
Name : Ciarán Coady
Student Number : 17326951

Part III

Number of instructions : 38

Number of clock cycles : 46

Screenshot



What is the effect on execution time if the two shift instructions are swapped and why?

This has the effect of reducing the execution time. This is because by swapping the two instructions we have removed a data dependency that was causing a pipeline stall. The LD R2, R0, 00 instruction takes 2 cycles to fetch the value from memory and put it in O1. Before we make the swap, the SRLi R2, R2, 01 instruction happens before the LD instruction has fetched the value. This causes a pipeline stall for 1 tick. By swapping these two instructions this data dependency is removed as the SLLI R3, R3, 01 does not involve r2.