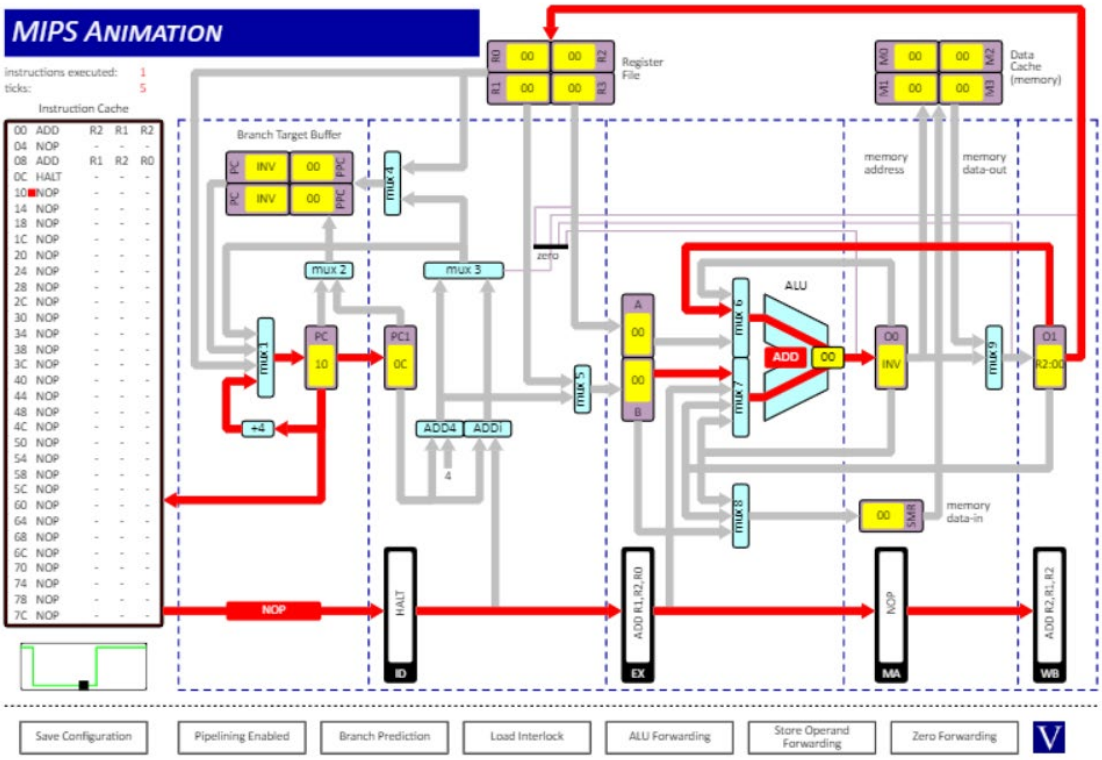
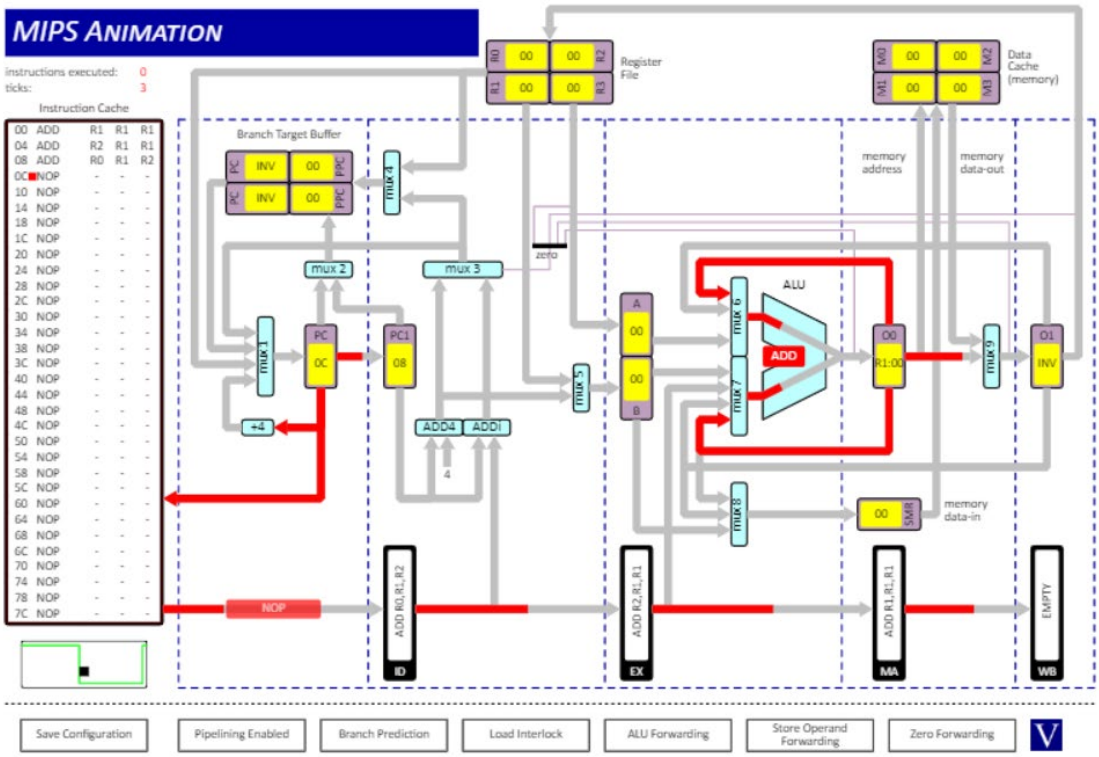
Q1.

1. O1 to MUX6



Instructions:
ADD R2, R1, R2
NOP
ADD R1, R2, R0

2. O0 to MUX7 and O1 to MUX6 (simultaneously)



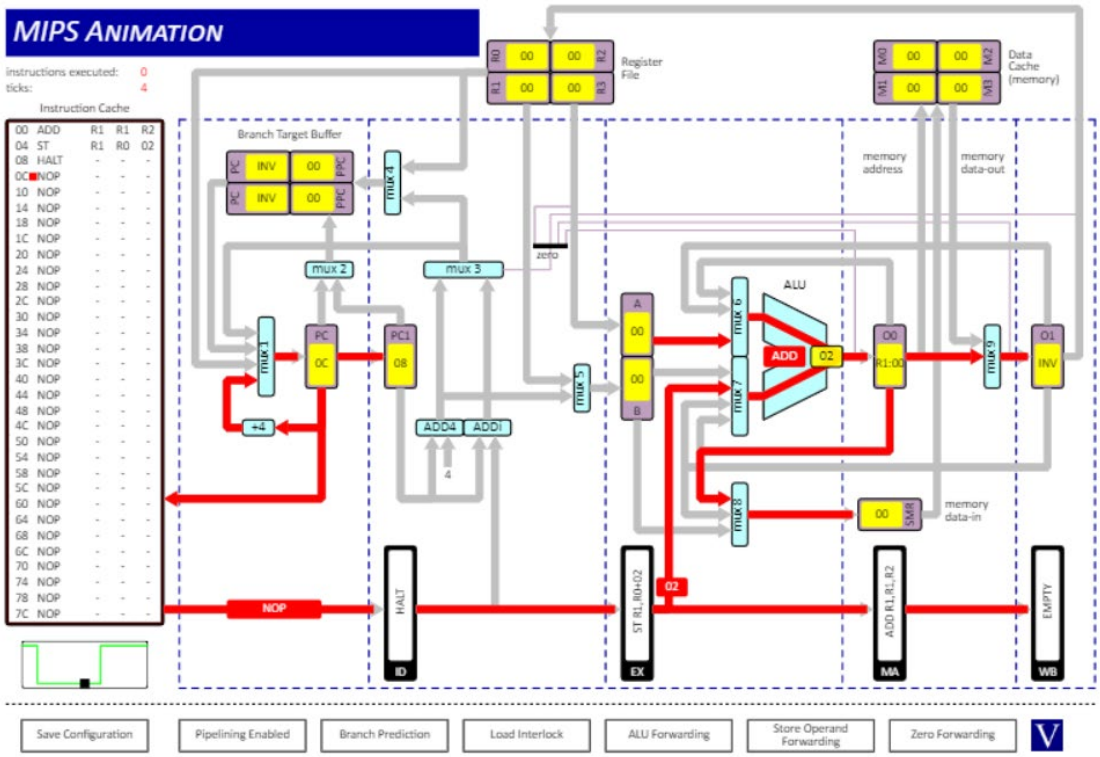
Instructions:

ADD R1, R1, R1

ADD R2, R1, R1

ADD R0, R1, R2

3. O0 to MUX8

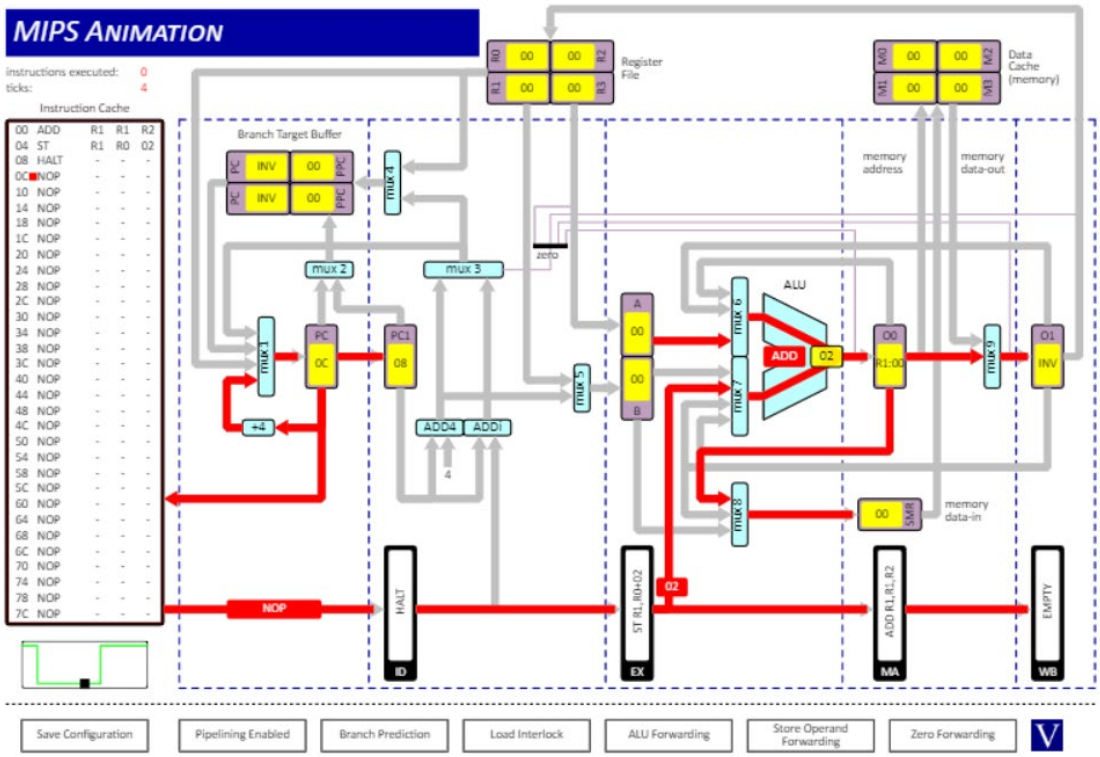


Instructions:

ADD R1, R1, R2

ST R1, R0, 02

4. EX to MUX7

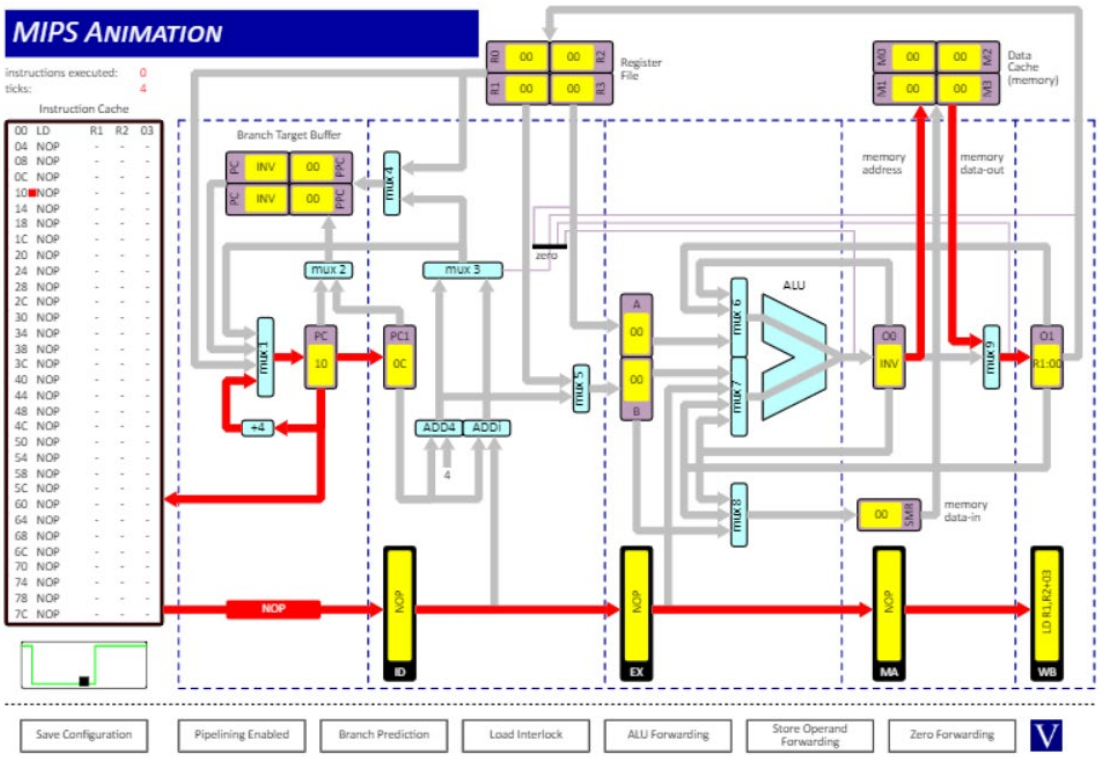


Instructions:

ADD R1, R1, R2

ST R1, R0, 02

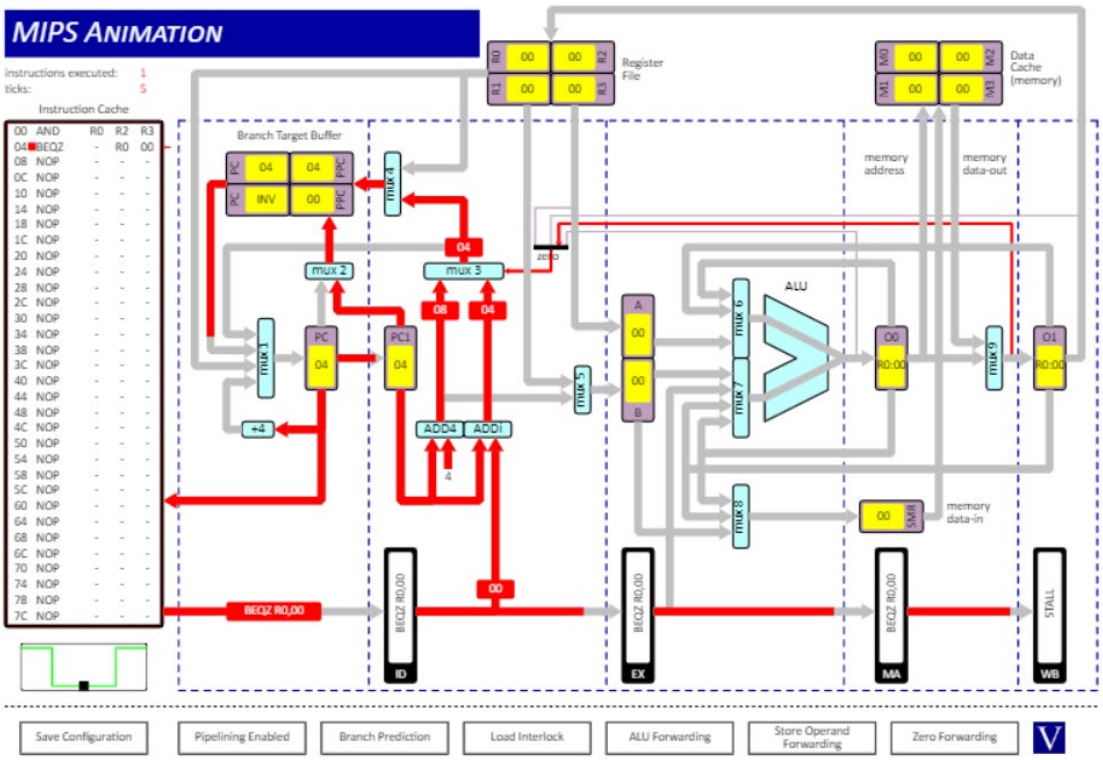
5. Data cache to MUX9 (memory data-out)



Instructions:

LD R1, R2, 03

6. O0 to Zero detector

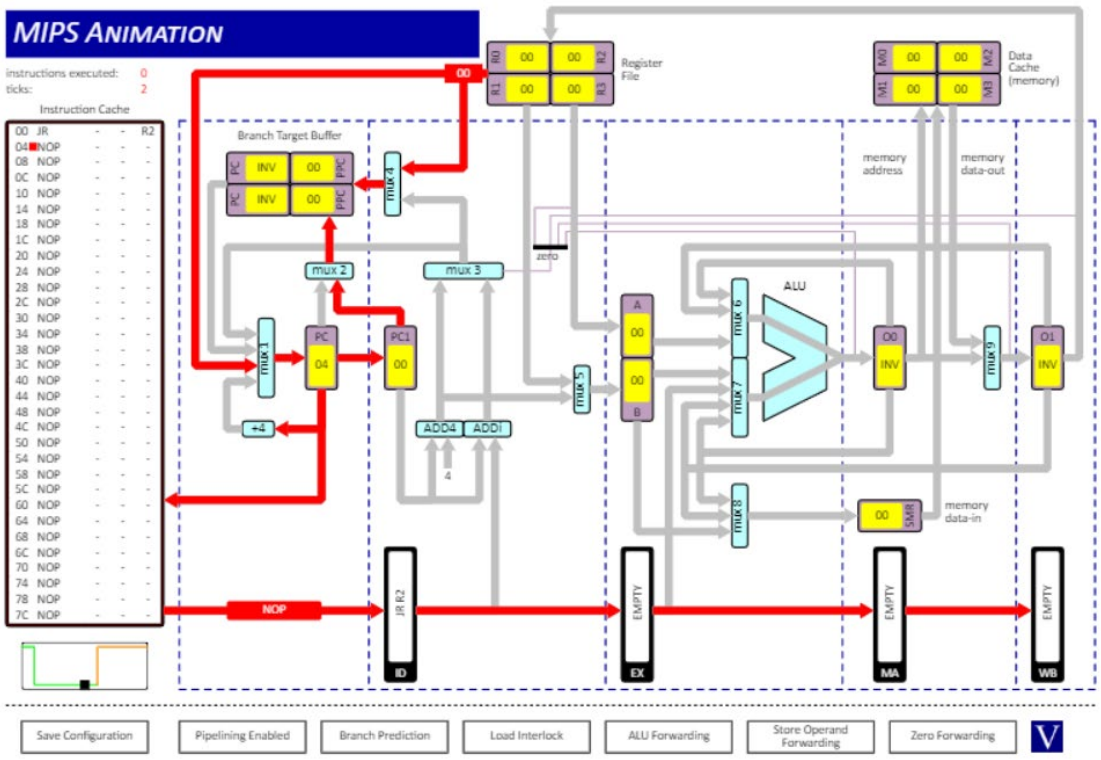


Instructions:

AND R0, R2, R3

BEQZ R0, 00

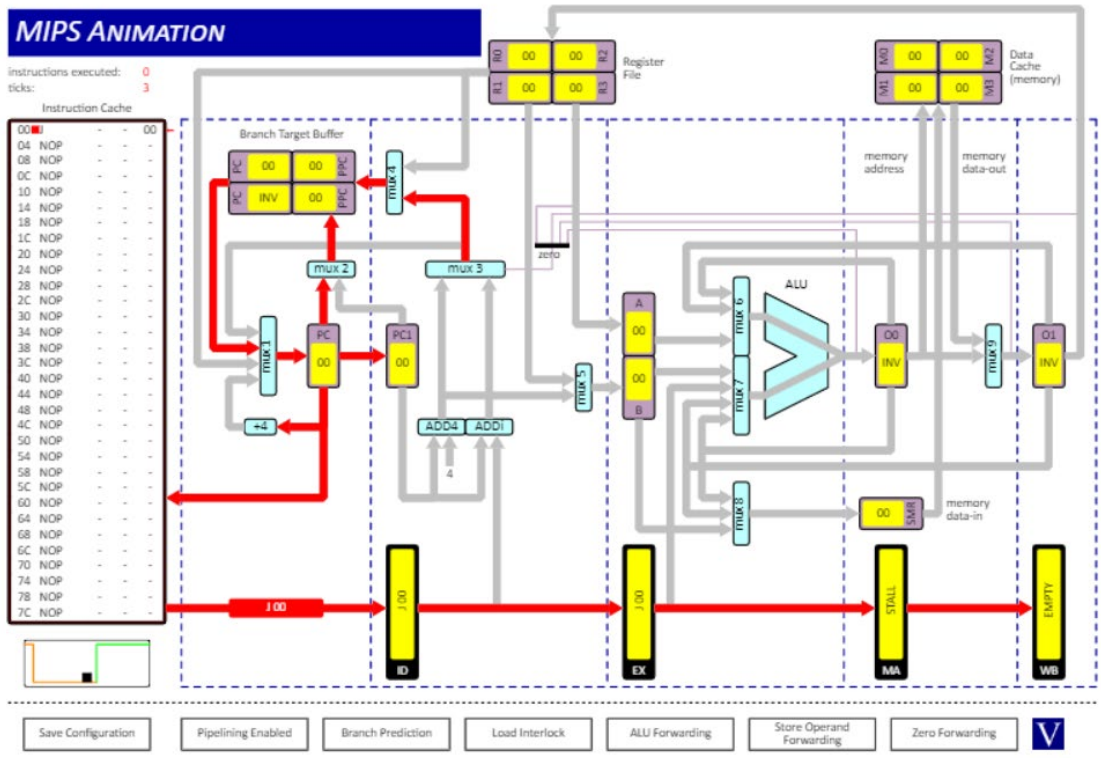
7. Register File to MUX1



Instructions:

JR R2

8. Branch Target Buffer to MUX1

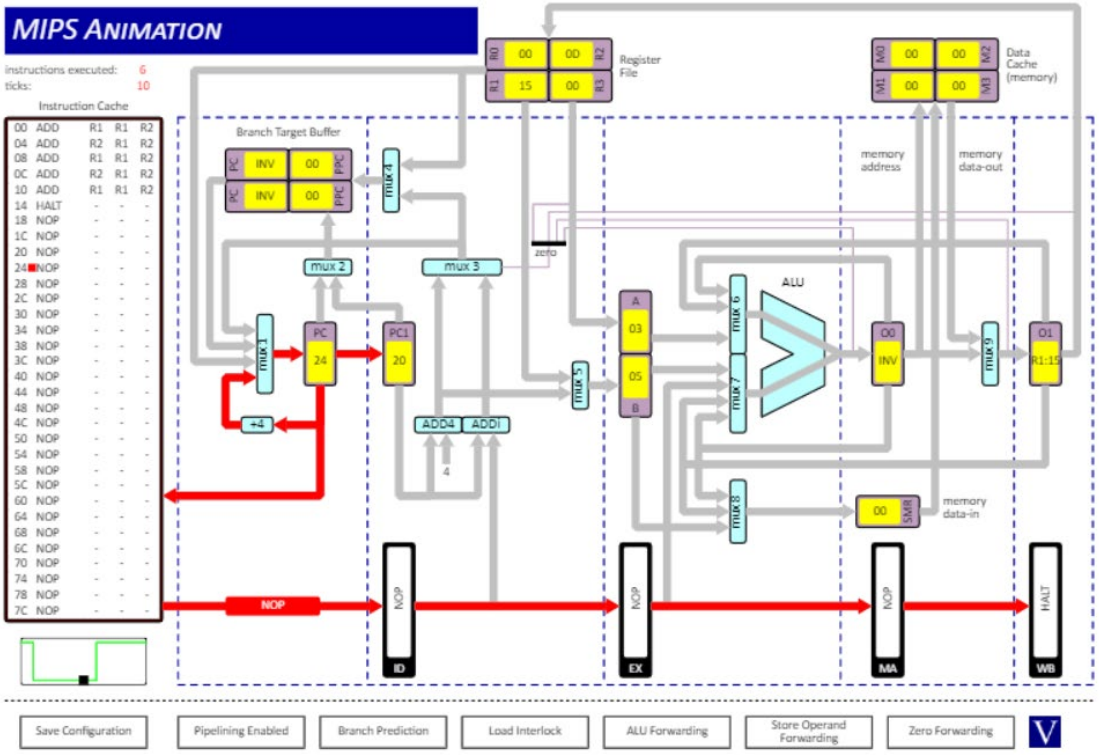


Instructions:

J 00

Q2

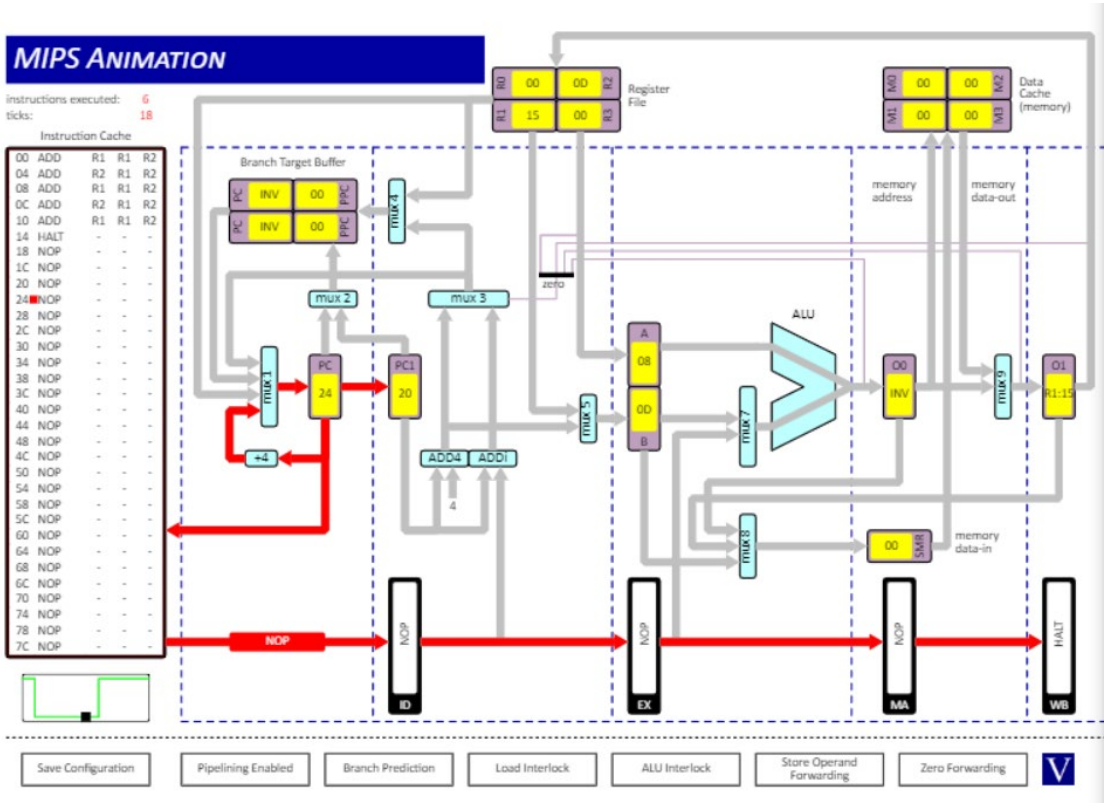
(i)



Ticks = 10

Result: R1 = 0x15

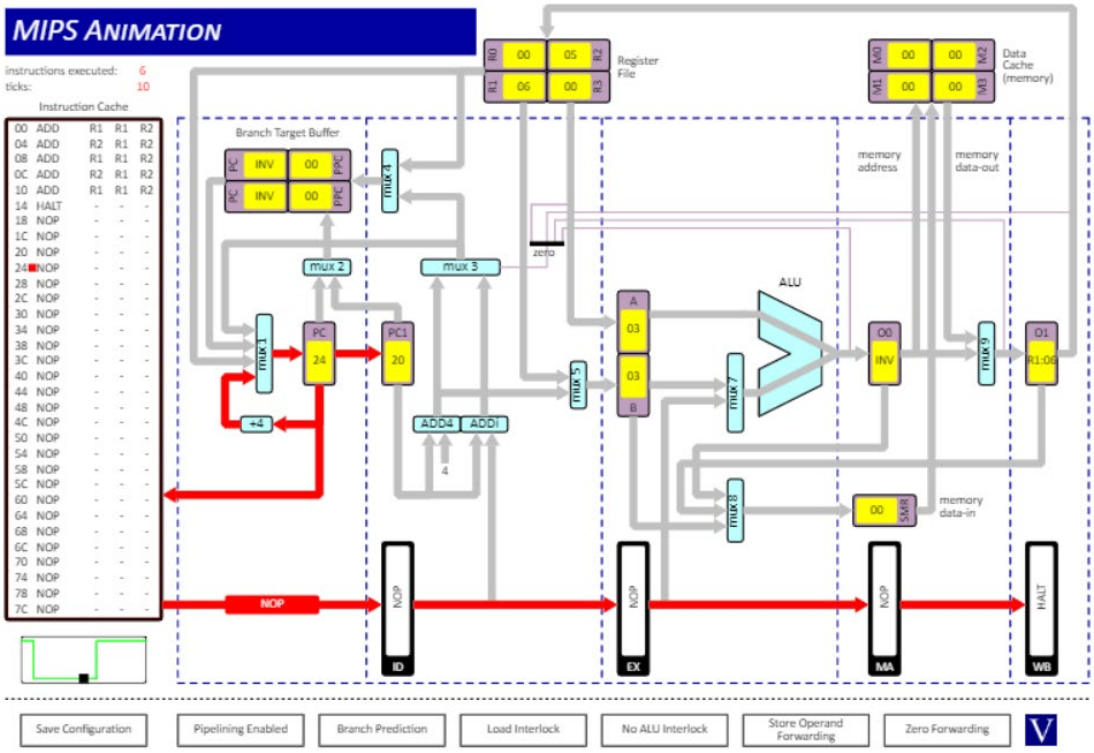
(ii)



Ticks = 18

Result: R1 = 0x15

(iii)



Ticks = 10

Result: R1 = 0x06

Explanation:

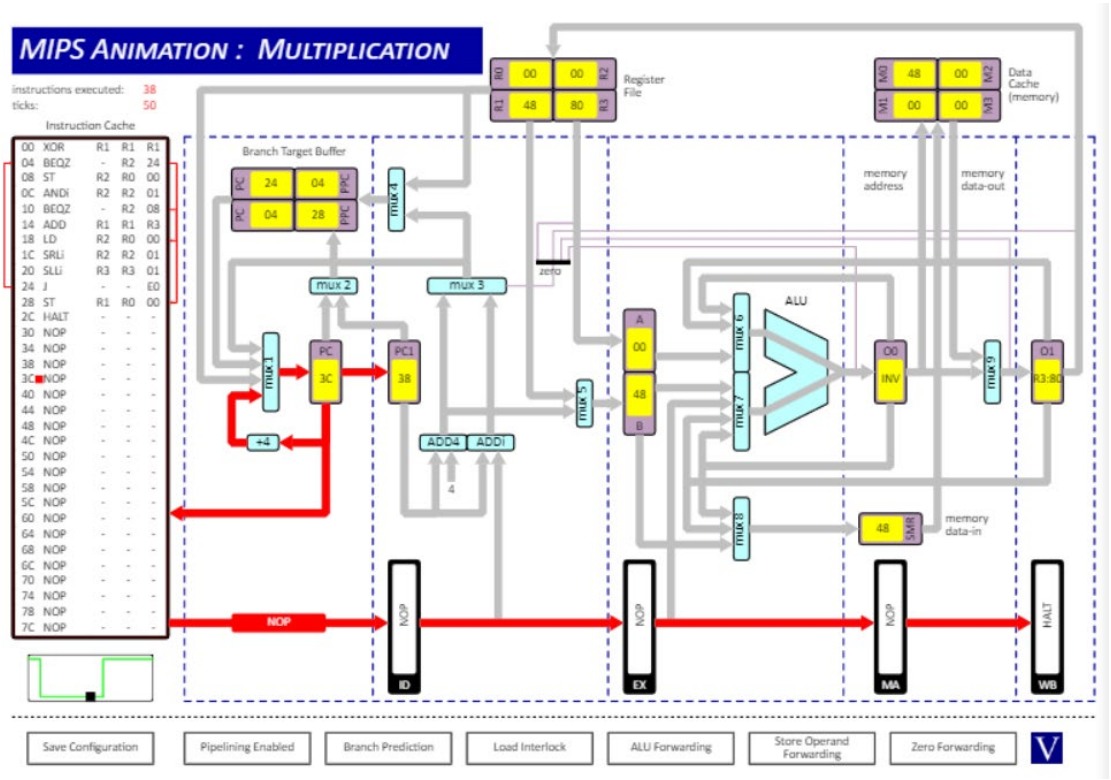
(i) When ALU forwarding is enabled, the result from the previous ALU operation can be stored in OUT0 and OUT1 and can be accessed immediately after it is calculated.

(ii) When ALU interlock is enabled, results cannot be stored into utilising OUT0 and OUT1, the processor needs to wait for the registers, due to pipeline stalls, to update from previous calculations until it can move on to the next instruction.

(iii) When there is no ALU interlock not CPU data dependency interlocks, the processor does not care about pipeline stalls, therefore does not wait for the implementations of the previous instruction to be fully complete when it moves on to the next instruction, thus the result is incorrect.

Q3

(i)



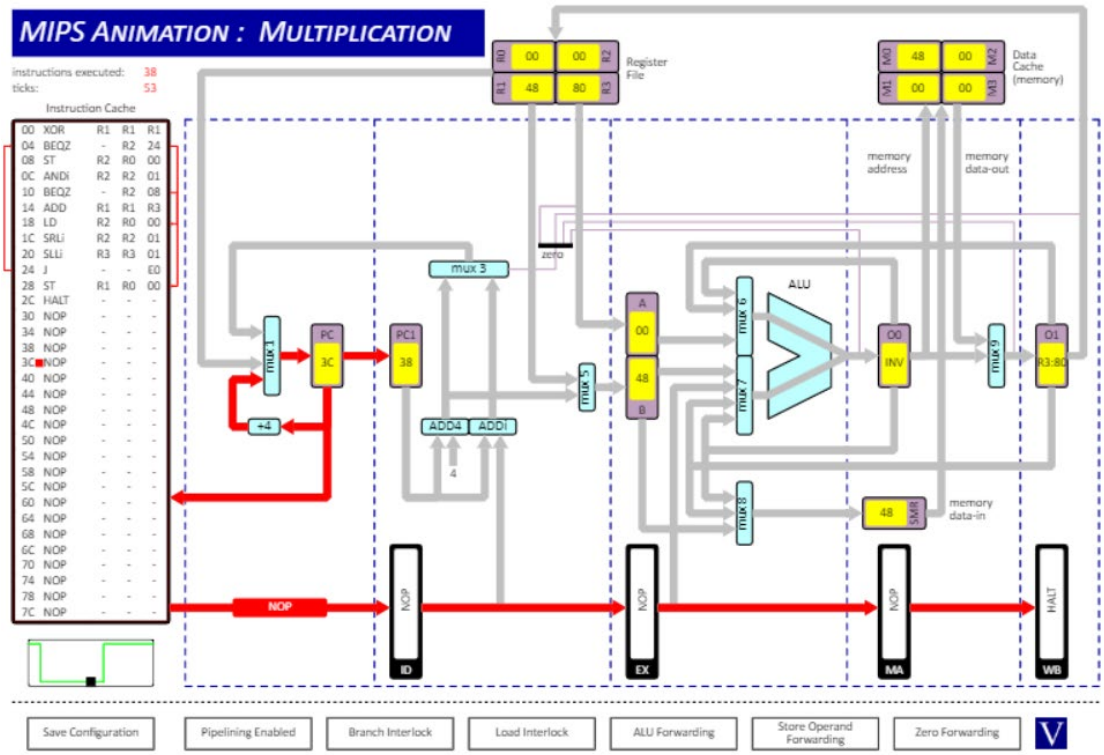
Instructions executed = 38

Ticks = 50

Reason: pipeline stalls.

Explanation: At the beginning before the pipeline has been loaded up to its capacity, there are 4 pipeline stalls. There is a stall is the between the instructions LD R2, R0, 00 and SRLi R2, R2, 01 to allow O1 to load the value from R2. The two instructions get executed 4 times, leading to 4 additional stalls. The branch instruction J E0 with branch prediction stalls 2 times as it predicts correctly. BEQZ also calls a stall in the pipeline to decide on the instruction it should execute next depending on its condition, in this situation this is 2. Thus 4+4+2+2=12=50-38 stalls.

(ii)



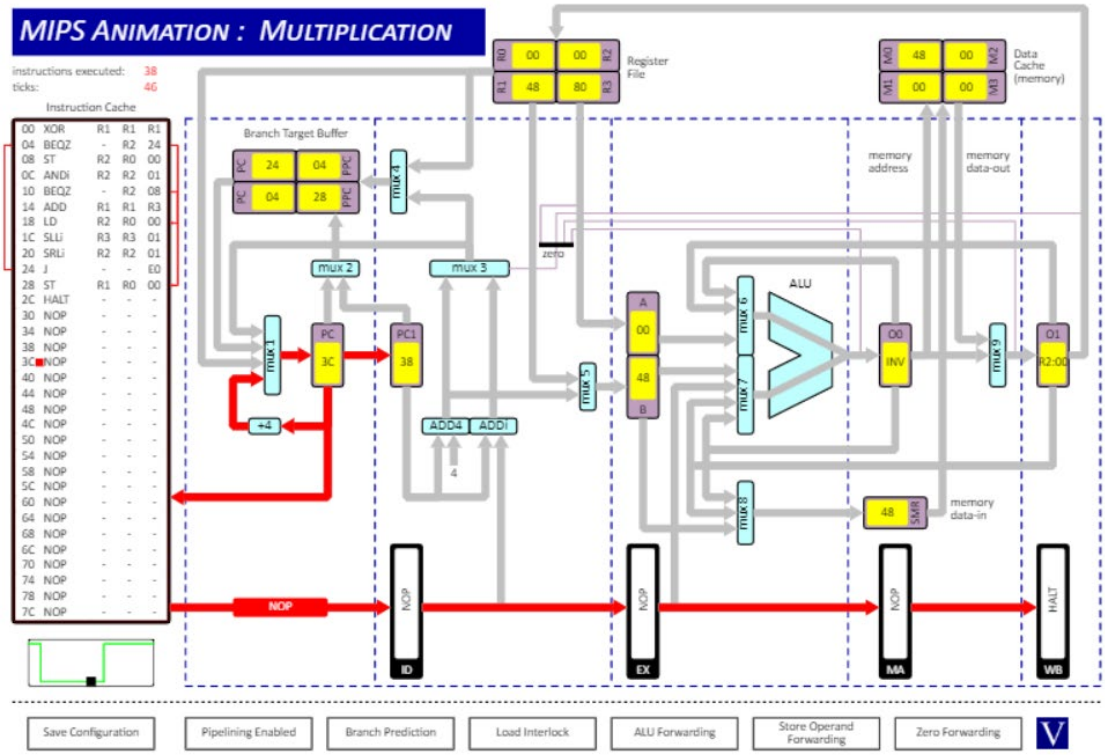
Instructions executed = 38

Ticks = 53

Reason: Branch interlock

Explanation: Of the difference in ticks and instructions 53-38=15, 4 come from the branch J E0 instruction due to the branch interlock, forcing it to stall 4 times before branching to the next instruction as it does not allow for predictions. Hence of the 11 that are left, 10 are exactly as described in (i), only that BEQZ stalls once less as their condition is not fulfilled and therefore does not require the branch to the next instruction.

(iii)



Instructions executed = 38

Ticks = 46

Explanation: The difference is 46-38=8. The swapping of SLLi R2, R2, 01 and SRLi R3, R3, 01 removes the data dependency causing stalls before. Before switching the operations, there is a stall is the between the instructions LD R2, R0, 00 and SRLi R2, R2, 01 to allow O1 to load the value from R2, which is executed 4 times. By swapping SRLi R2, R2, 01 and SLLi R3, R3, 01, dependency is removed as SLLi R3, R3, 01 does not involve any operands from LD, reducing the number of stalls in (i) by 4, hence having 8 stalls remaining in total.