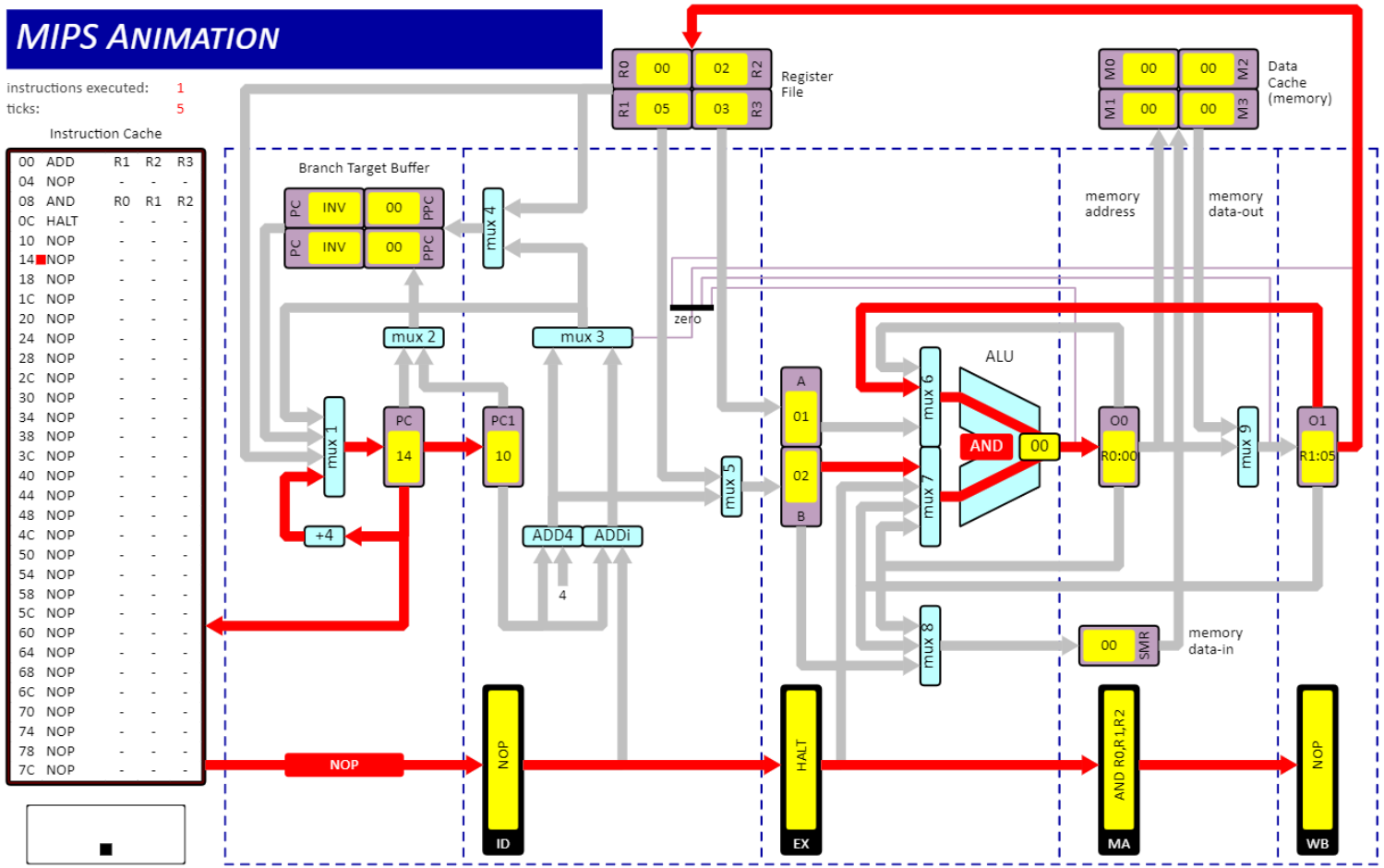
QUESTION 1.1 - O1 to MUX6

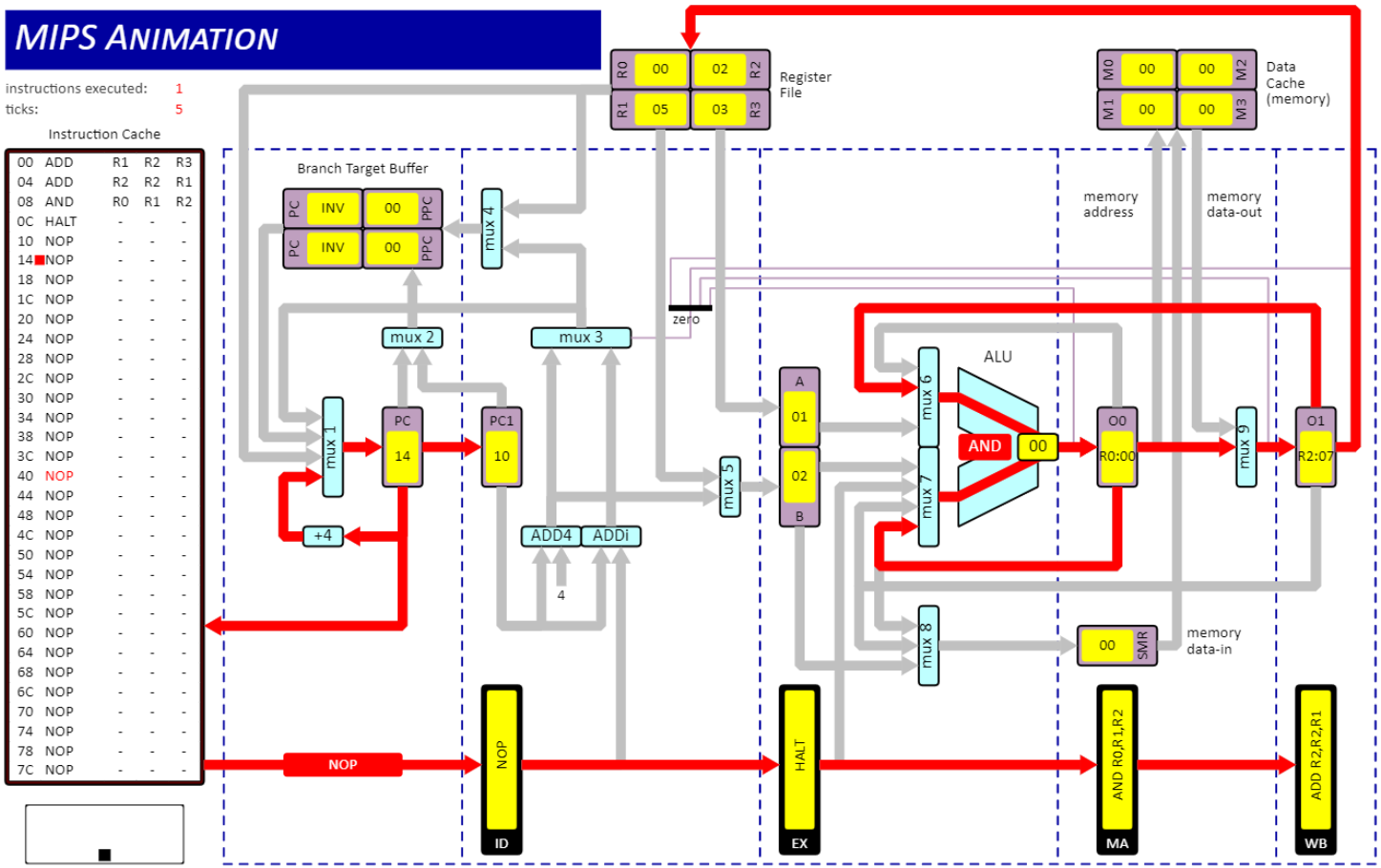



ADD R1, R2,R3

NOP

AND R0, R1, R2

QUESTION 1.2 - O0 to MUX7 and O1 to MUX6 (simultaneously)



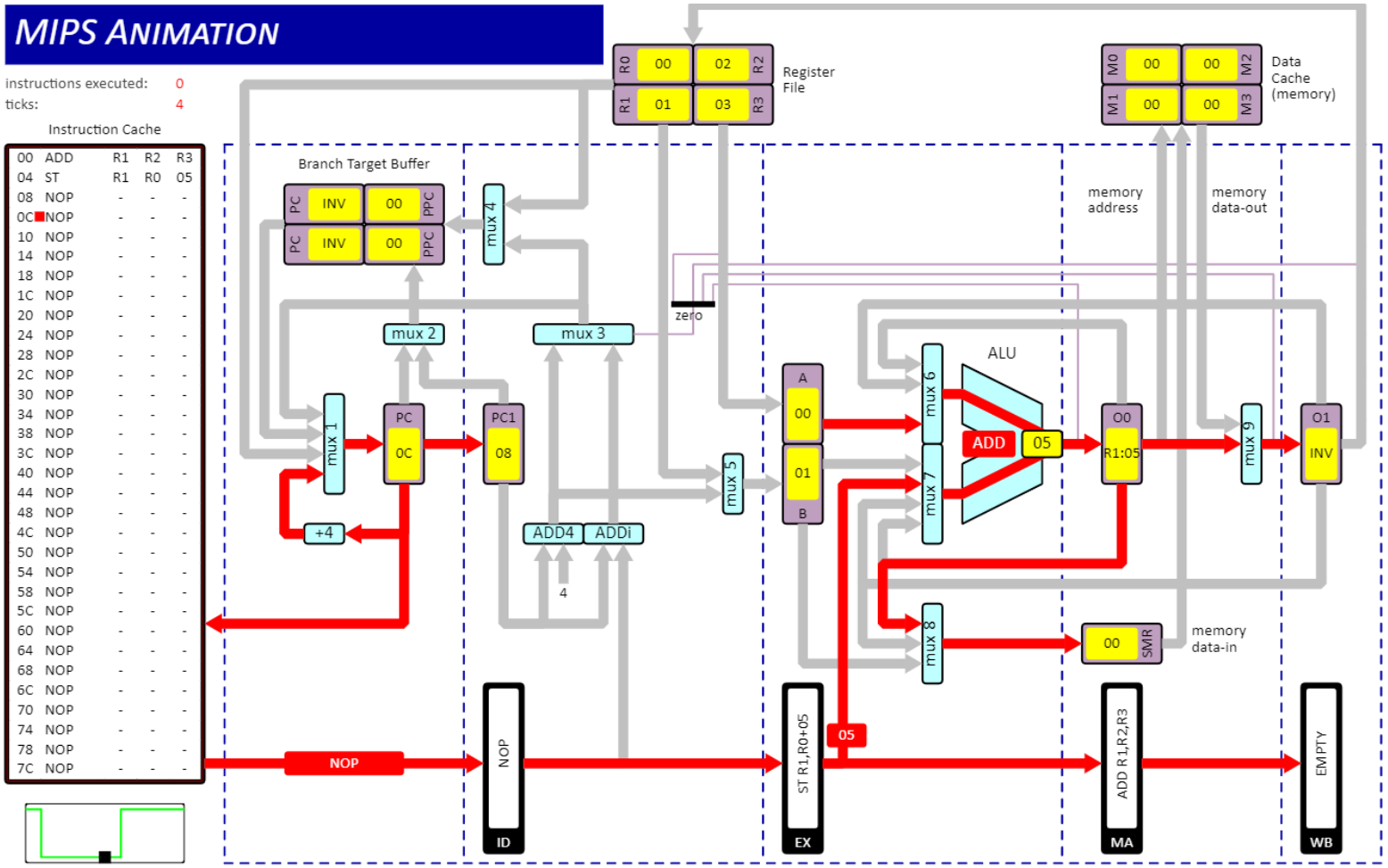

ADD R1, R2, R3

ADD R2, R2, R1

AND R0, R1, R2

QUESTION 1.3 - O0 to MUX8

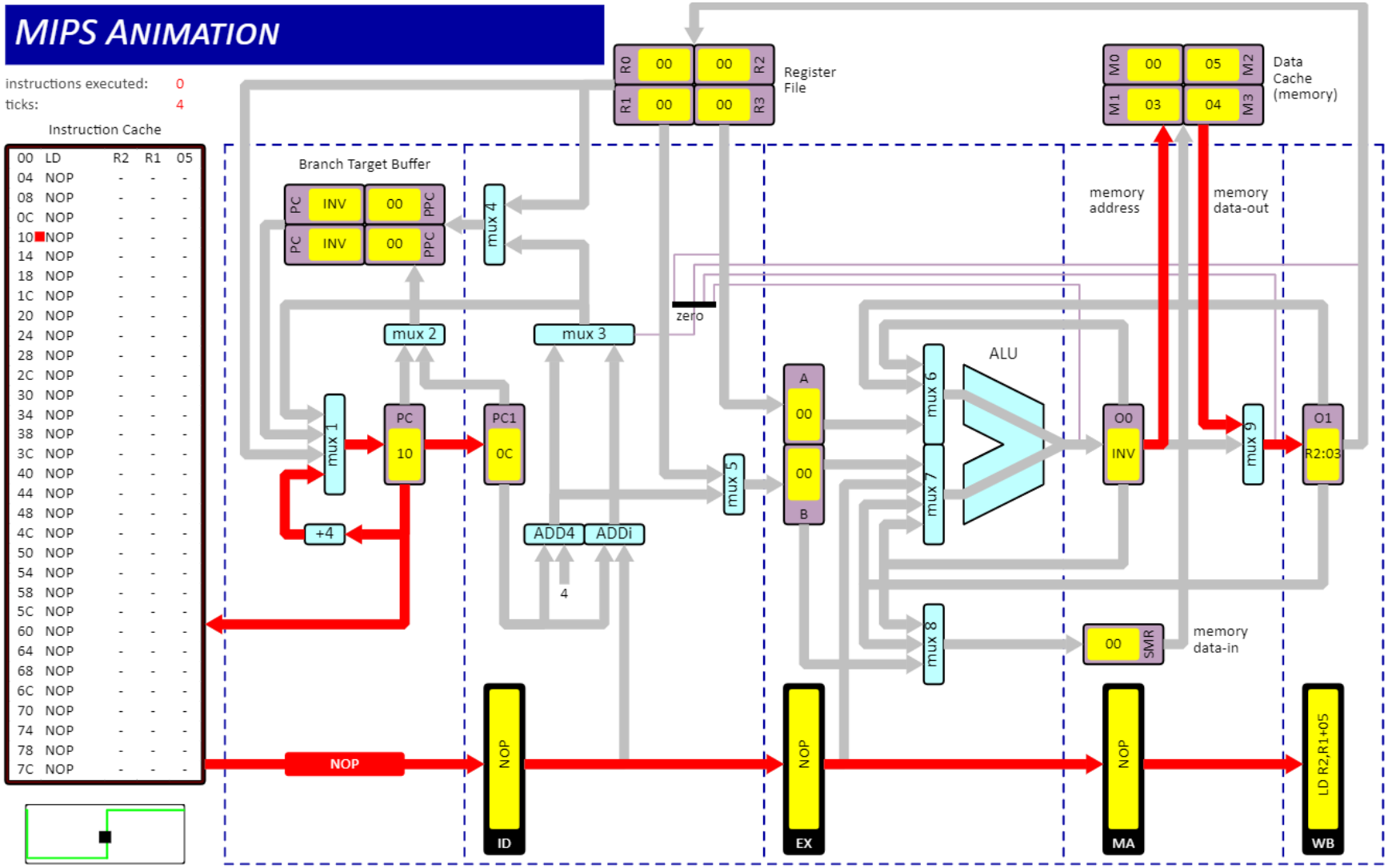
QUESTION 1.4 - EX to MUX7 (BOTH SHOW IN THIS SCREENSHOT)




ADD R1, R2, R3

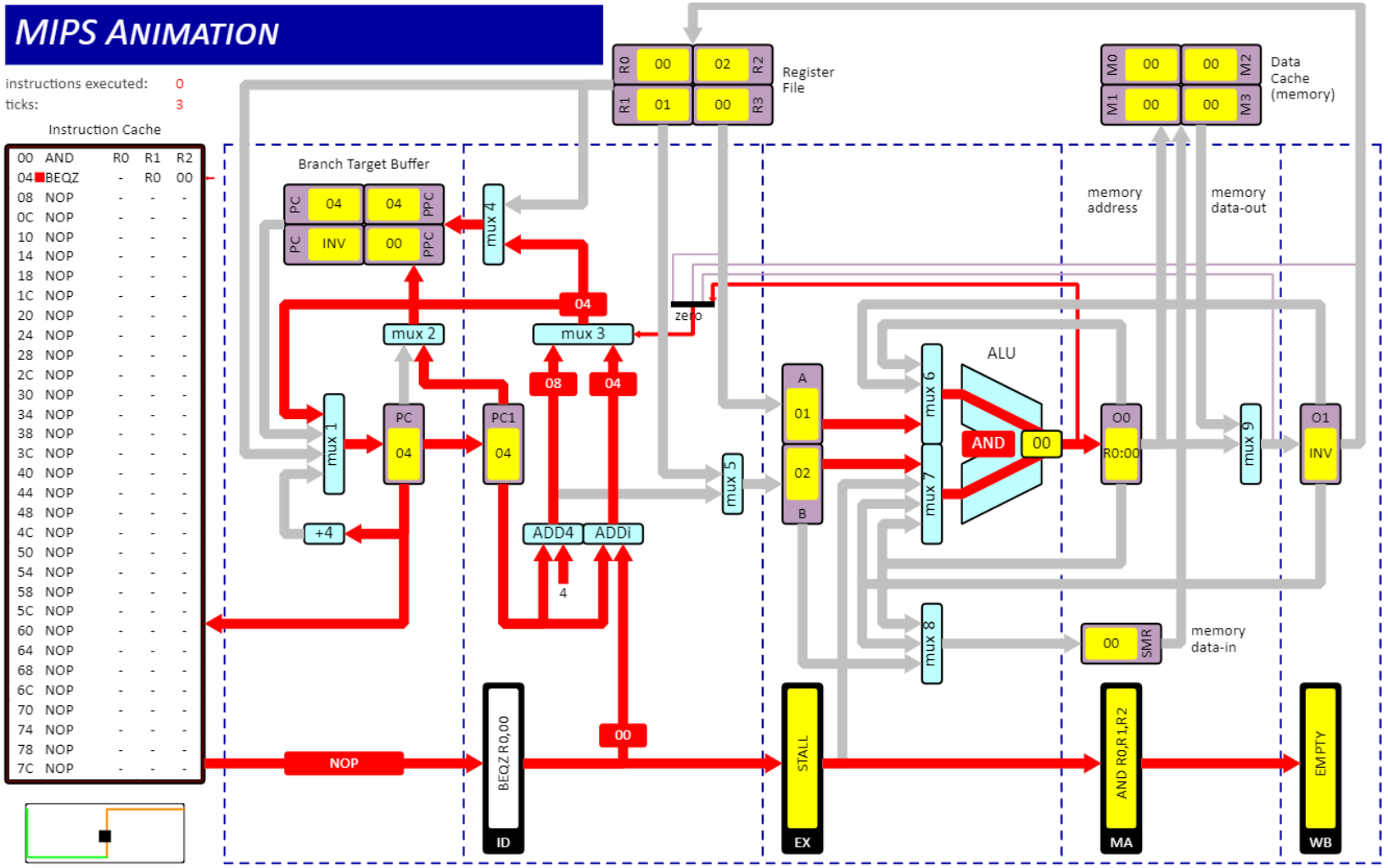
ST R1, R0, 05

QUESTION 1.5 - Data cache to MUX9 (memory data-out)




LD R2, R1, 05

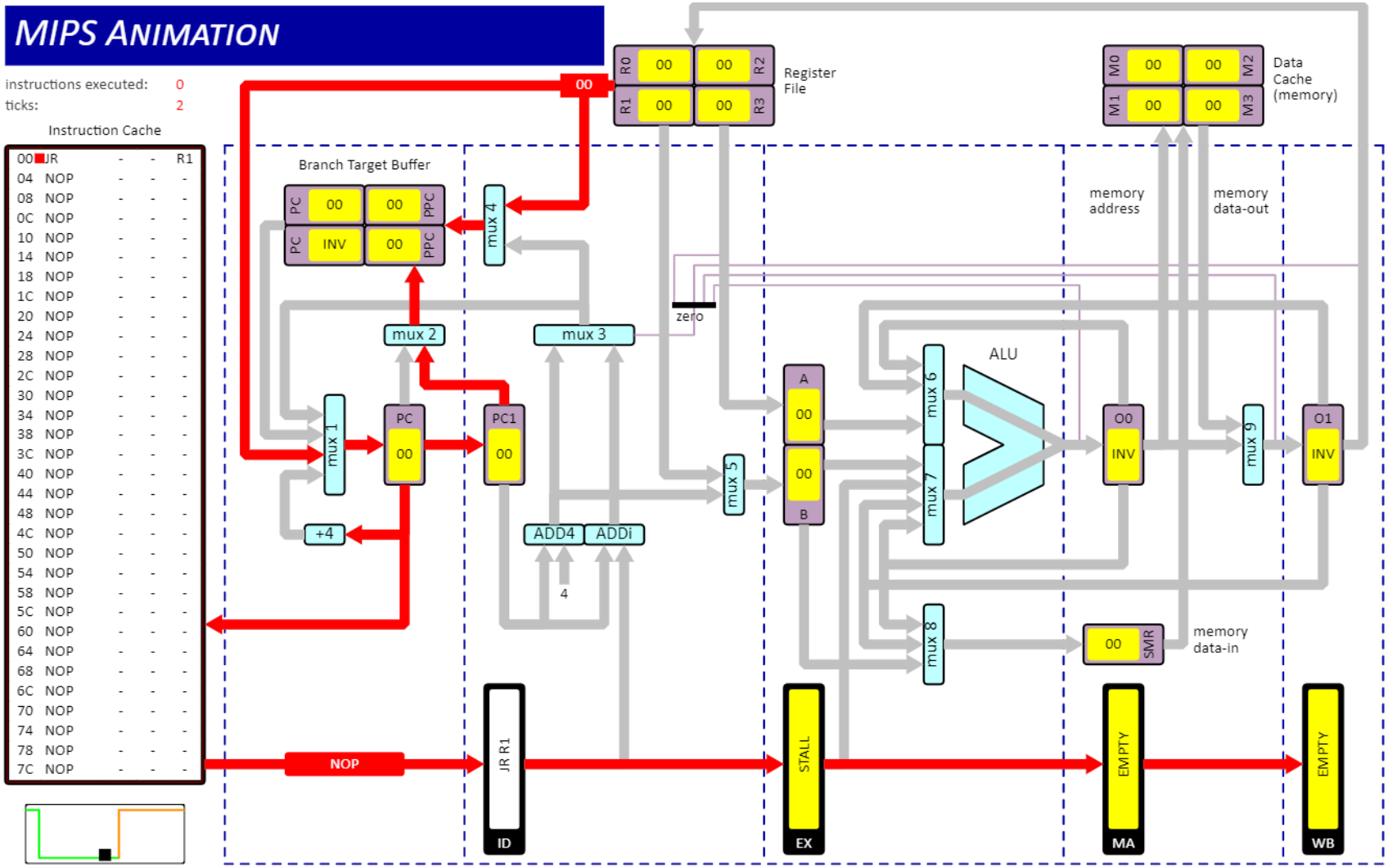
QUESTION 1.6 - O0 to Zero detector




AND R0, R1, R2

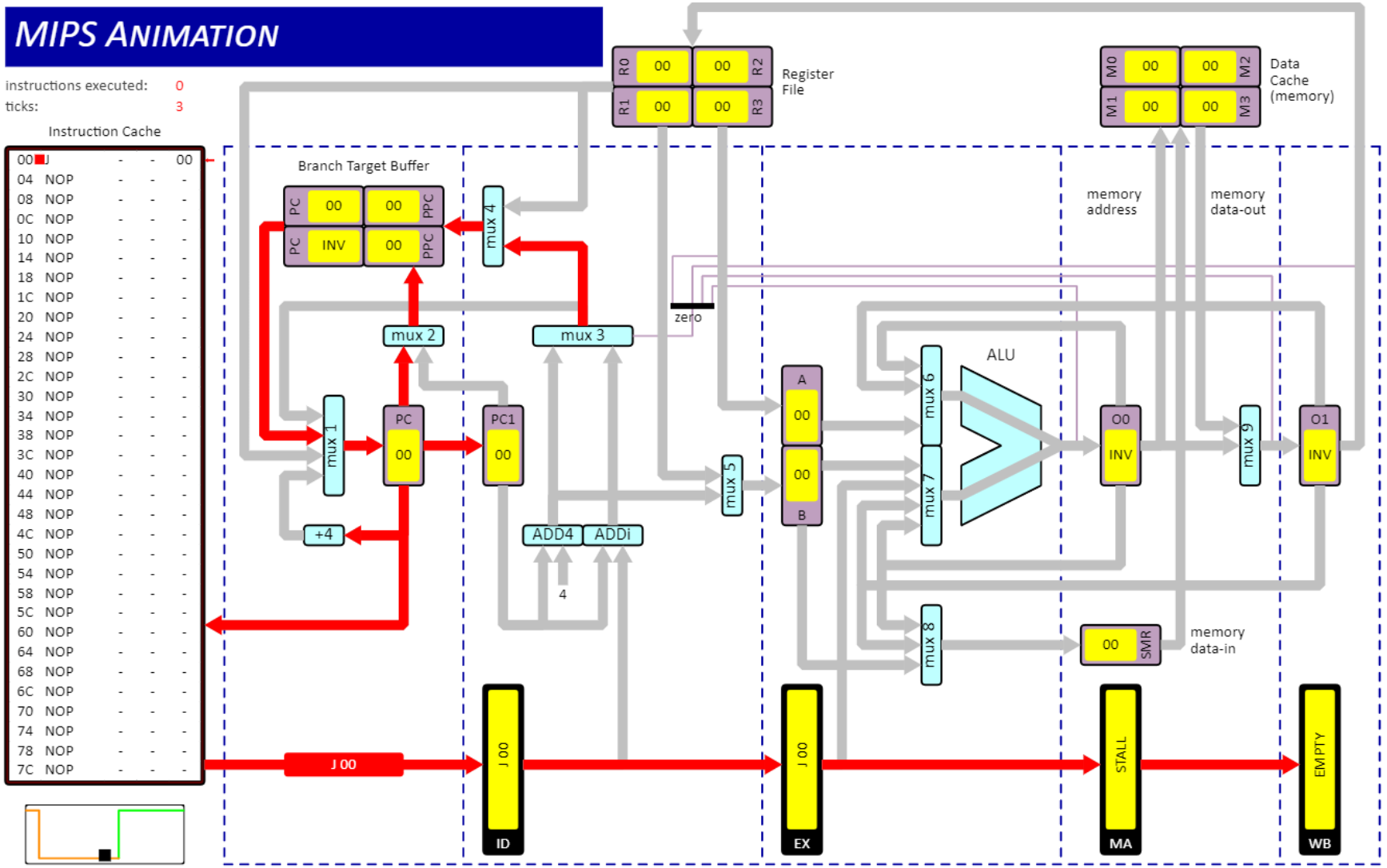
BEQZ R0, 00

QUESTION 1.7 - Register File to MUX1




JR R1

QUESTION 1.8 - Branch Target Buffer to MUX1




J 00

QUESTION 2

(i) Ticks = 10
Result = 0x15

(ii) Ticks = 18
Result = 0x15

(iii) Ticks = 10
Result = 0x06

When ALU forwarding is enabled, the results from previous instruction can be stored in O0 and O1 before the MA or WB phase.

When ALU forwarding is disabled but CPU data dependency interlocks enabled, the processor has to stall between instructions so that the previous instructions can update their registers before the following instruction is executed. There are 4 instructions after the first one, with a 2 cycle stall per instruction (4x2 = 8), this leads to an 8 tick delay.

When ALU forwarding is disabled and CPU data dependency interlocks are disabled too, the processor doesn't implement stalls and just executes the instructions "as is", leading to data hazards, which explains the different result.

QUESTION 3

(i) Ticks = 50
Instructions = 38

The numbers aren't equal because of stalls. The beginning of the pipeline includes 4 stalls as its being loaded up to its capacity. There is a stall between LD and SRLi to allow for the value in R2 to be stored in an immediate register O1. Since this is instruction is performed 4 times, there are 4 ticks of stalls. The unconditional jump J causes a stall before the jump is executed, so that the correct instruction can be fetched (BEQZ rather than ST). This happens 2 times as there is branch prediction which predicts correctly twice. There are also two stalls due to the 2 different BEQZ statements once the conditions for the branch are met, as they each need to fetch the correct next instruction.
So ultimately 50 – 38 = **12** (4+4+2+2) stalls.

(ii) Ticks = 53
Instructions = 38

11 of the stalls are the same as part (i), the other 4 come from the absence of branch prediction in the processor. The reason why it's only 11 is because only one of the BEQZ statements (BEQZ R2 24) is satisfied to branch, which means it requires a tick of stall to fetch the correct instruction. As there is no branch prediction, the unconditional jump J E0 will have a tick of stall 4 times so that the correct instruction can be fetched (BEQZ rather than ST). In total there is 53 - 38 = **15** (11 + 4) ticks of stall.

(iii) Ticks = 46
Instructions = 38

By swapping the shift instructions we can get rid of the stall between LD and SRLi as SLLi doesn't use any operands from LD. As such we get an overall 4 stalls less, totalling in 8 (46-38) which are equivalent to the first 8 from part (i).