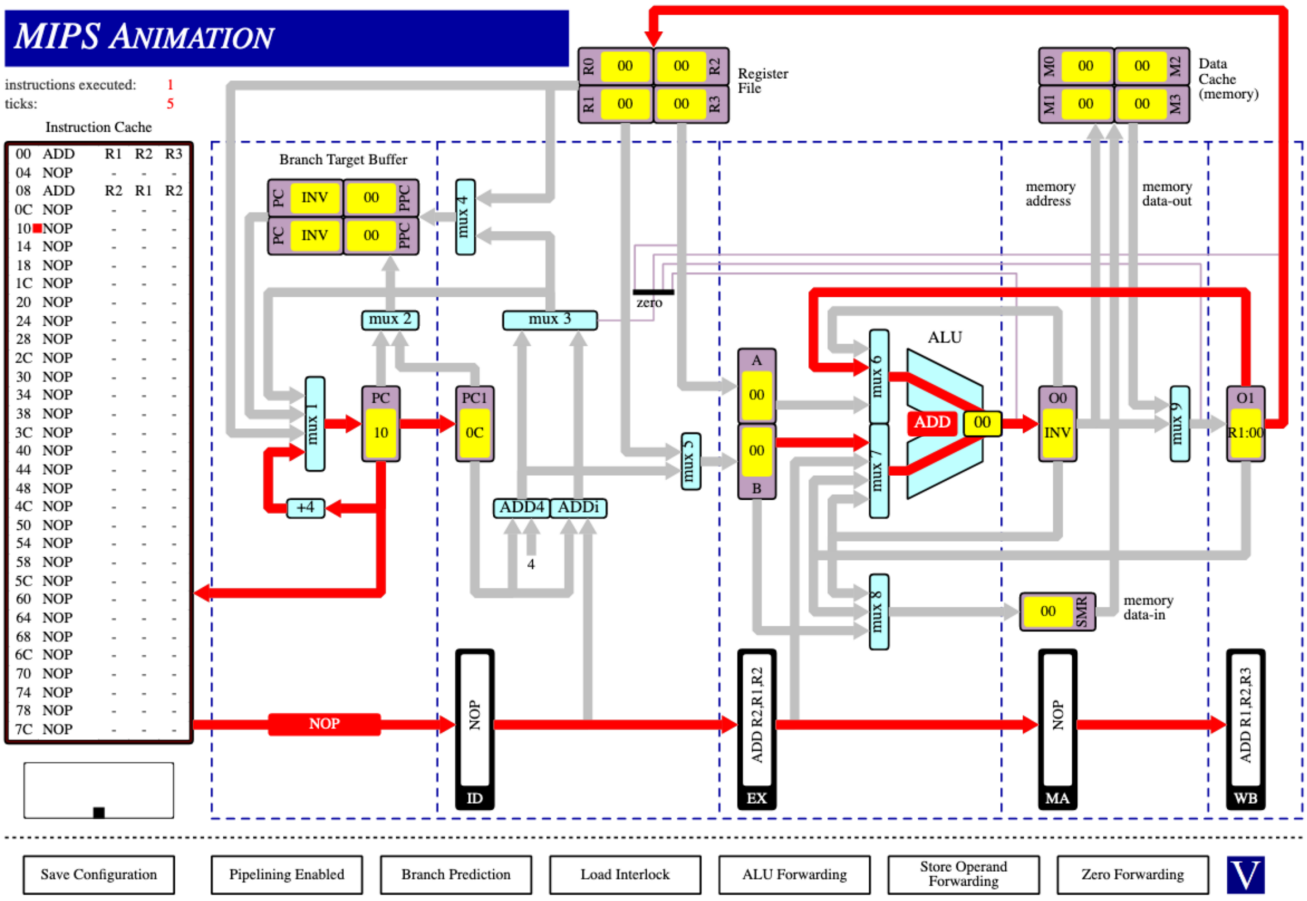
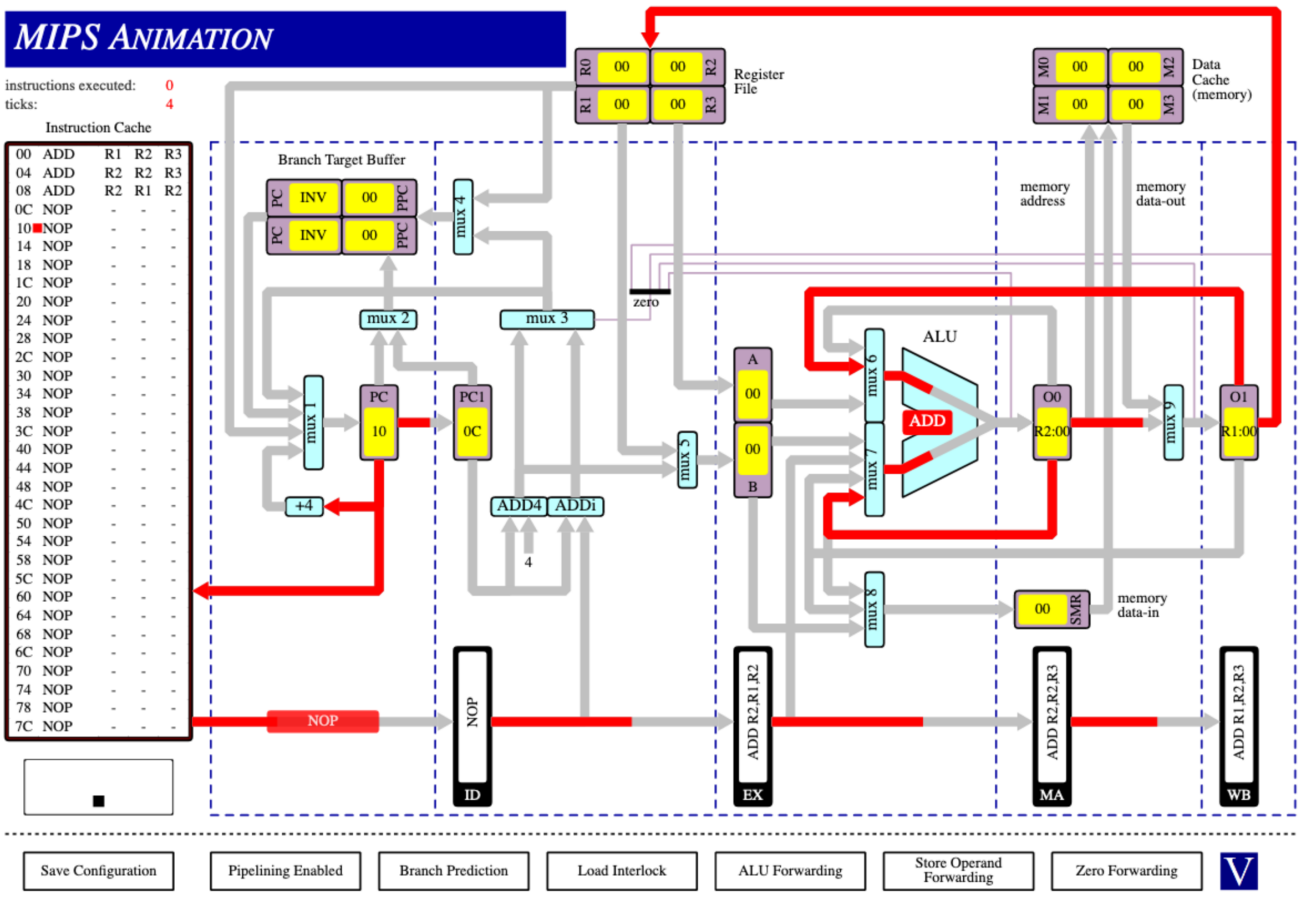
Q1

1.)



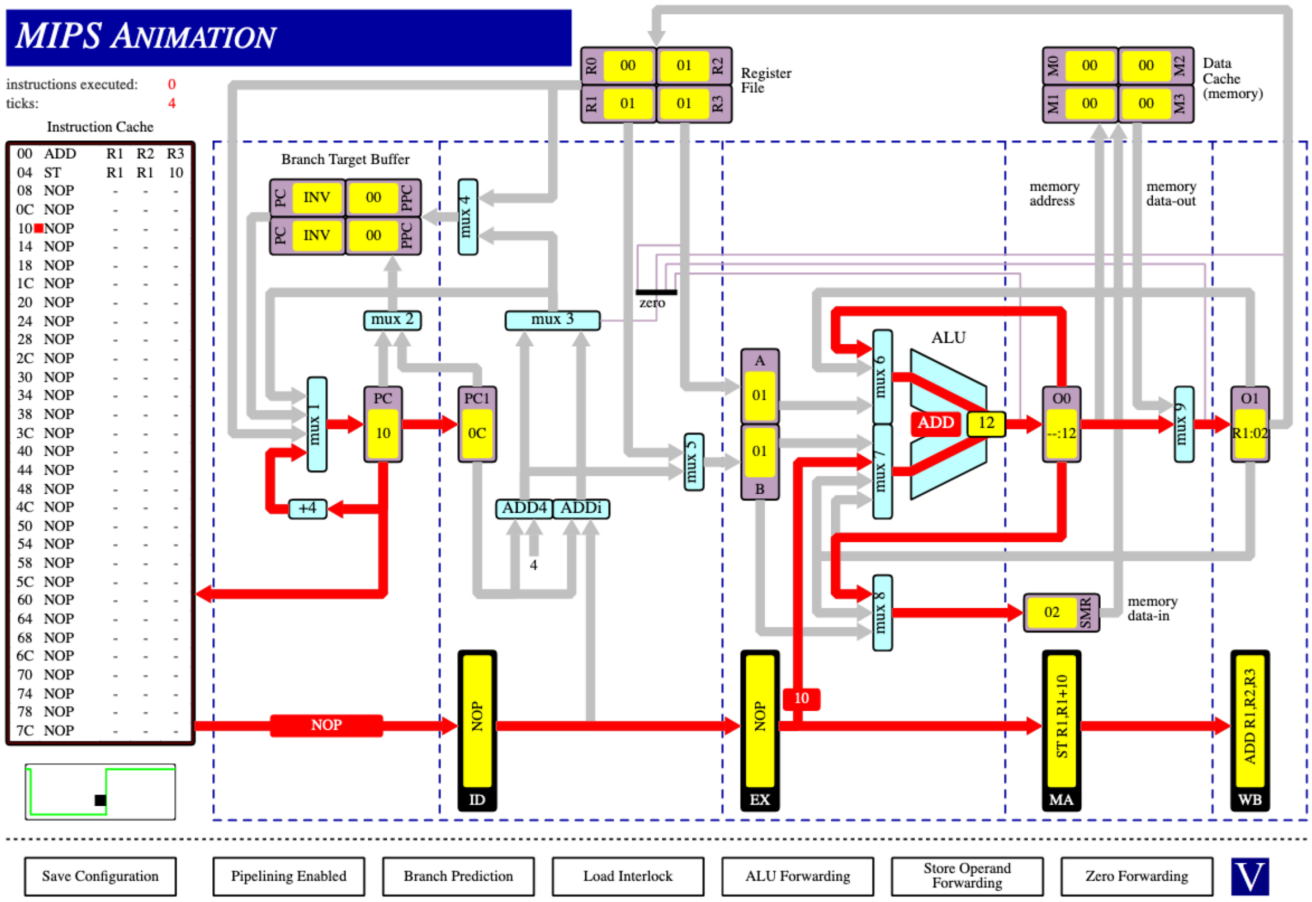

```
ADD R1 R2 R3
NOP -  -  -
ADD R2 R1 R2
```

2.)



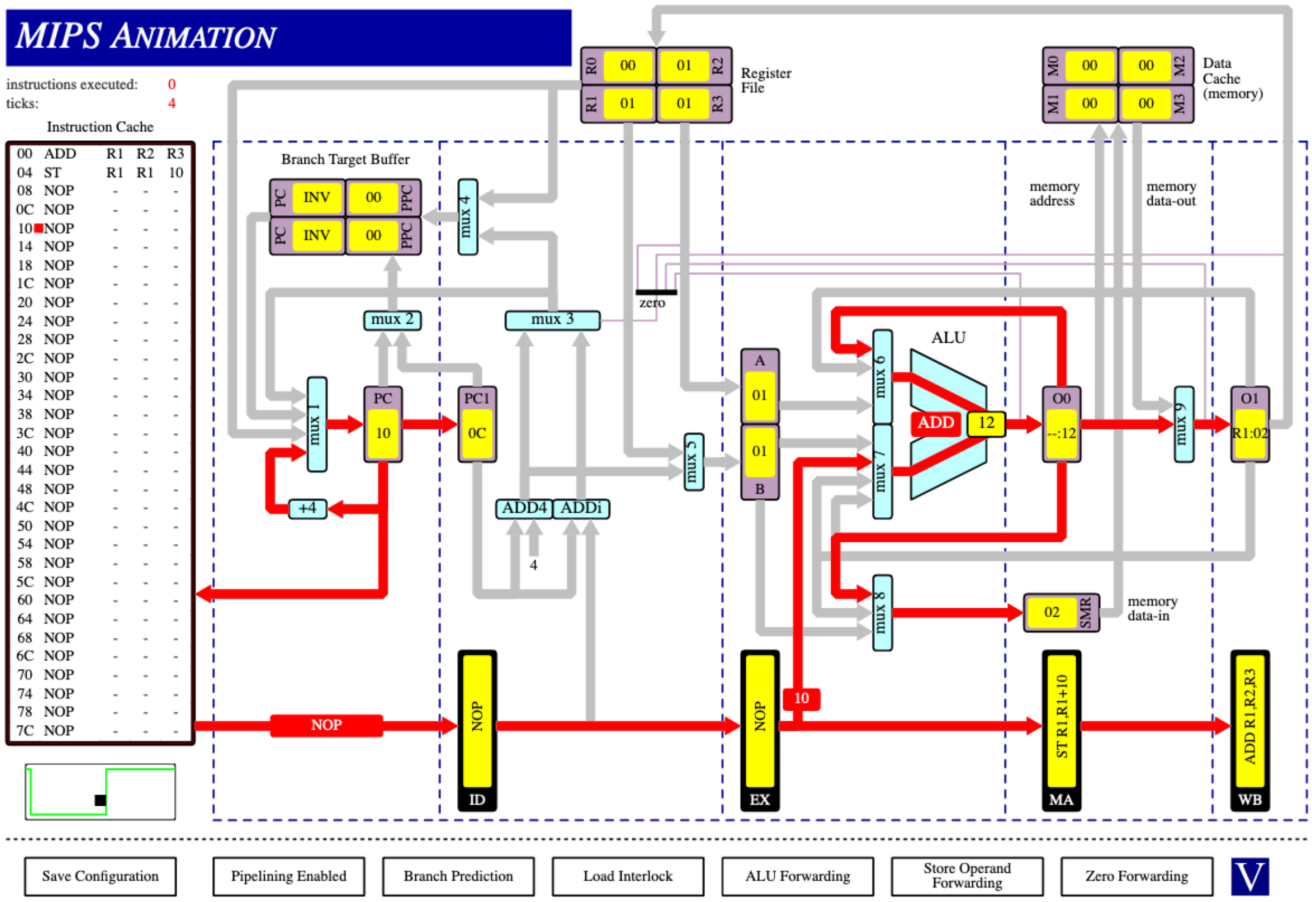

```
ADD R1 R2 R3
ADD R2 R2 R3
ADD R2 R1 R2
```

3.)



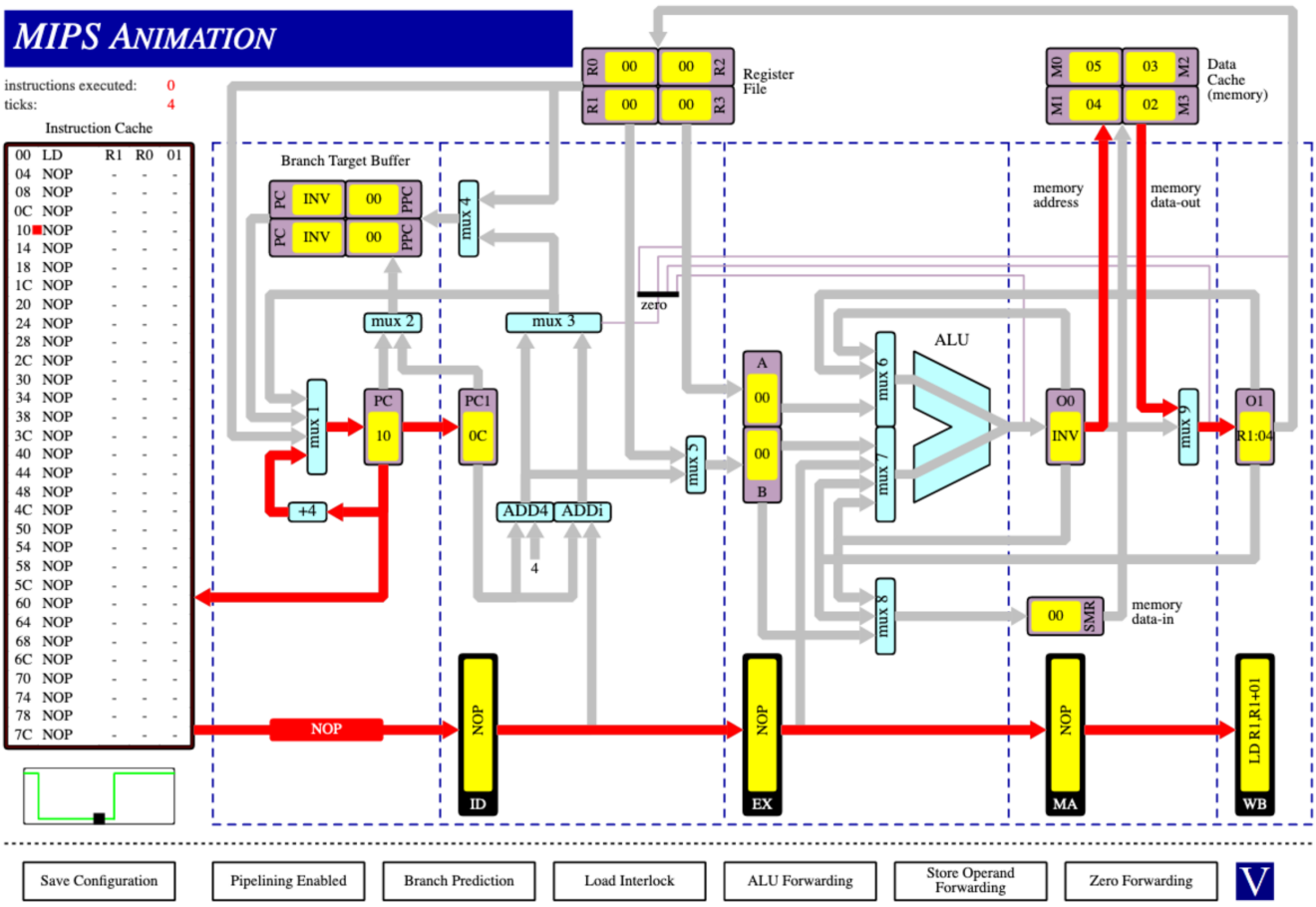

```
ADD R1 R2 R3
ST  R1 R1 10
```

4.)



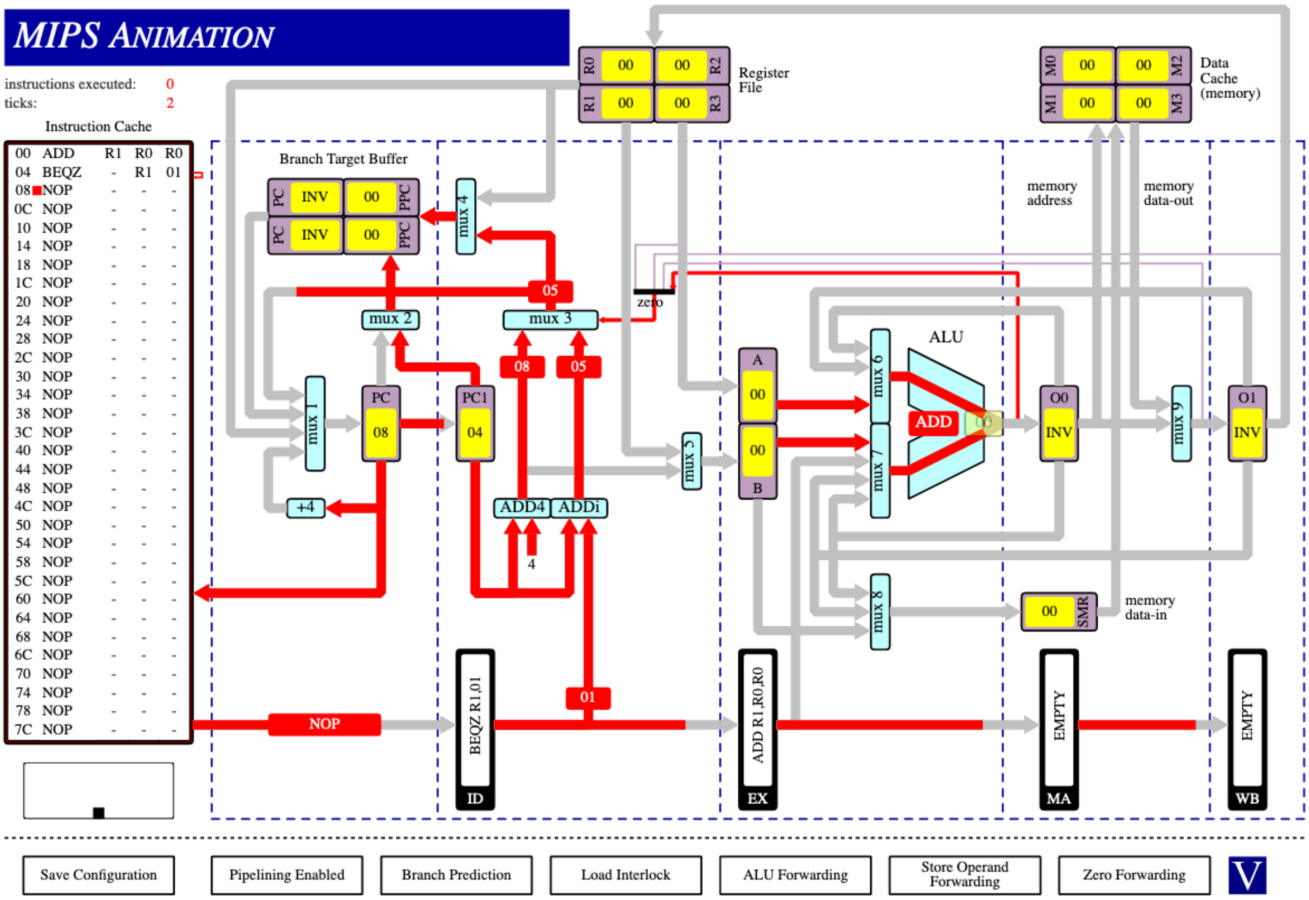

```
ADD R1 R2 R3
ST  R1 R1 10
```

5.)


MIPS ANIMATION
instructions executed: 0
ticks: 4
Instruction Cache
00 LD R1 R0 01
04 NOP
08 NOP
0C NOP
10 NOP
14 NOP
18 NOP
1C NOP
20 NOP
24 NOP
28 NOP
2C NOP
30 NOP
34 NOP
38 NOP
3C NOP
40 NOP
44 NOP
48 NOP
4C NOP
50 NOP
54 NOP
58 NOP
5C NOP
60 NOP
64 NOP
68 NOP
6C NOP
70 NOP
74 NOP
78 NOP
7C NOP
Register File
Data Cache (memory)
Branch Target Buffer
mux 1
mux 2
mux 3
mux 4
mux 5
mux 6
mux 7
mux 8
mux 9
ALU
zero
memory address
memory data-out
memory data-in
ADD4
ADDi
NOP
ID
EX
MA
WB
LD R1 R0 01
Save Configuration
Pipelining Enabled
Branch Prediction
Load Interlock
ALU Forwarding
Store Operand Forwarding
Zero Forwarding

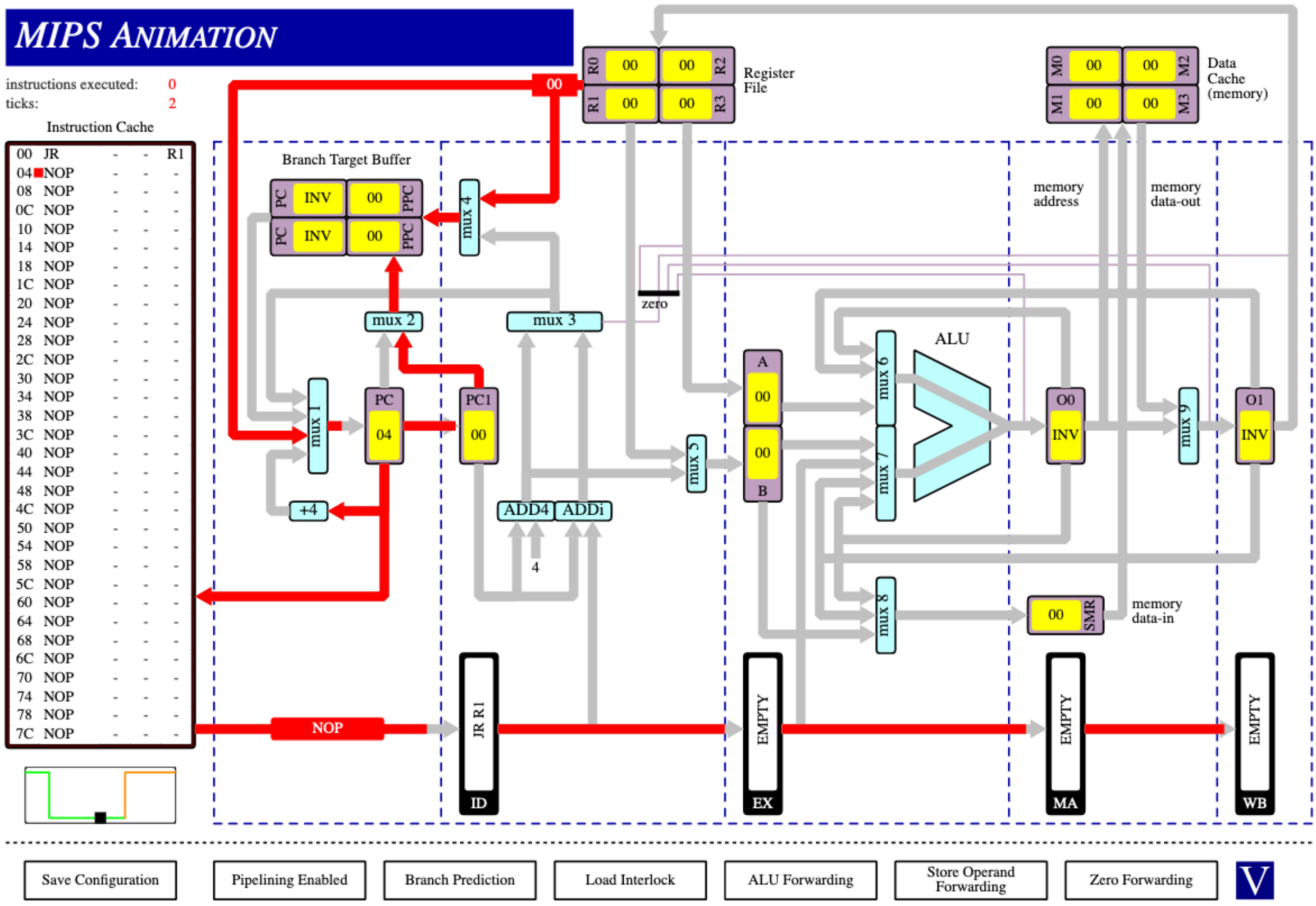
```
LD R1 R0 01
```

6.)



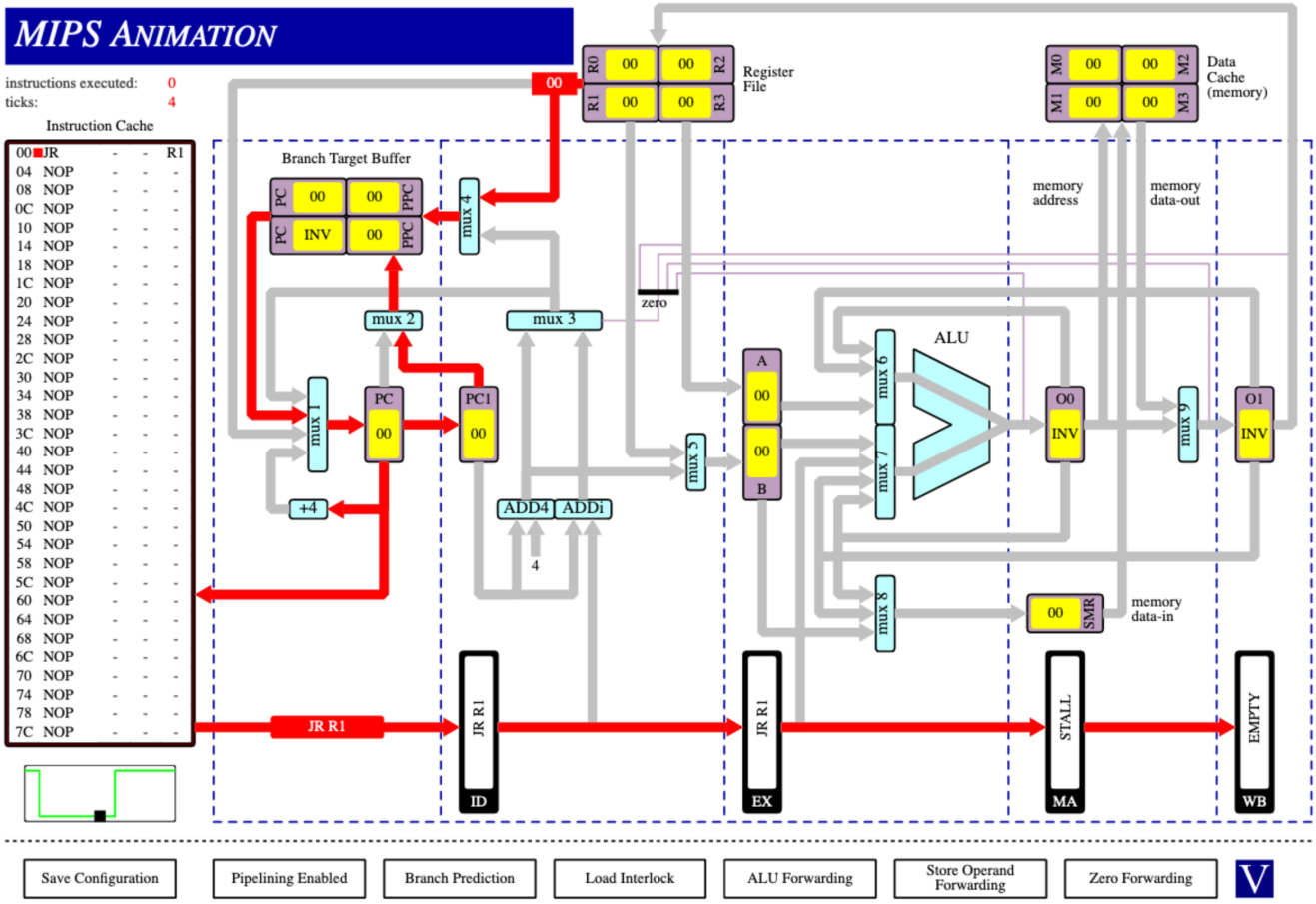

```
ADD  R1 R0 R0
BEQZ -  R1 01
```

7.)


MIPS ANIMATION
instructions executed: 0
ticks: 2
Instruction Cache
00 JR - - R1
04 NOP - - -
08 NOP - - -
0C NOP - - -
10 NOP - - -
14 NOP - - -
18 NOP - - -
1C NOP - - -
20 NOP - - -
24 NOP - - -
28 NOP - - -
2C NOP - - -
30 NOP - - -
34 NOP - - -
38 NOP - - -
3C NOP - - -
40 NOP - - -
44 NOP - - -
48 NOP - - -
4C NOP - - -
50 NOP - - -
54 NOP - - -
58 NOP - - -
5C NOP - - -
60 NOP - - -
64 NOP - - -
68 NOP - - -
6C NOP - - -
70 NOP - - -
74 NOP - - -
78 NOP - - -
7C NOP - - -
Branch Target Buffer
Register File
Data Cache (memory)
memory address
memory data-out
memory data-in
ALU
zero
mux 1
mux 2
mux 3
mux 4
mux 5
mux 6
mux 7
mux 8
mux 9
PC
PC1
ADD4
ADDi
+4
JR R1
NOP
EMPTY
ID
EX
MA
WB
Save Configuration
Pipelining Enabled
Branch Prediction
Load Interlock
ALU Forwarding
Store Operand Forwarding
Zero Forwarding

```
JR – – R1
```

8.)




```
JR - - R1
```

Q2

(i) cycles = 10 R1 = 0x15

The CPU can read values from the buffer O0 (and O1) and use that in the next instruction without waiting for the writeback to complete.

This also means the pipeline doesn't have to be stalled at any stage.

(ii) cycles = 18 R1 = 0x15

The CPU cannot read values from any buffer registers as they don't exist and is forced to wait for the writeback operations to complete. This causes an increase in cycles as the pipeline must be stalled to keep data integrity (thanks to the ALU Interlock).

(iii) cycles = 10 R1 = 0x06

The CPU cannot read values from buffer registers as they don't exist but also does not need to wait for writeback operations to complete as the ALU Interlock is disabled. This means there are no pipeline stalls but the value in R1 is not correct as arithmetic operations were performed without waiting for the results from previous operations to be written back.

Q3

(i) instructions = 38 cycles = 50

There are 12 more cycles than instructions (in the order they occur):

1. 3 cycles can be accounted for due to the 4 stage pipeline. 1 instruction will take 4 cycles to complete and therefore there will always be an additional 3 cycles.

2. 1 cycle is lost to a pipeline stall when the first load instruction is executed (`LD R2 R0 00` followed by `SRLi R2 R3 01` @ cycle = 10). Due to the data dependency (R2) of the second instruction (`SRLi`) on the `LD` instruction and the 2 cycle requirement of the `LD` instruction, the pipeline must be stalled for 1 cycle while waiting for the `LD` to complete.

3. 1 cycle is lost to an unconditional jump (`J – – E0` @ cycle = 13). The pipeline has to stall to load the correct instruction (`BEQZ – R2 24`) as opposed to the the previously loaded instruction (`ST R1 R0 00`).

4. 1 cycle is lost to a pipeline stall during a conditional branch (`BEQZ – R2 08` @ cycle = 18) as it had predicted to not take the branch and had to stall the pipeline while loading the correct next instruction (`LD R2 R0 00` as opposed to the originally loaded `ADD R1 R1 R3`).

5. 1 cycle is lost to the same reason as outlined in point 2 (@ cycle = 21).

6. 1 cycle is lost to the same reason as outlined in point 2 (@ cycle = 30).

7. 1 cycle is lost during a conditional branch (`BEQZ – R2 08` @ cycle = 37) as it had predicted to take the branch and had to stall while loading the correct next instruction (`ADD R1 R1 R3`) to replace the incorrectly loaded one (`LD R2 R0 00`).

8. 1 cycle is lost to the same reason as outlined in point 2 (@ cycle = 41).

9. 1 cycle is lost during a conditional branch (`BEQZ – R2 24` @ cycle = 45) as the CPU had predicted to not branch but the branch was taken. Because of this it had to stall the pipeline while the correct instruction (`ST R1 R0 00`) was loaded to replace the wrong instruction (`ST R2 R0 00`).

10. 1 cycle is lost to the `HALT` instruction as it takes time to execute but is not counted as an instruction.

(ii) instructions = 38 cycles = 53

There are 15 more cycles than instructions. This is an increase of 3 over the number in part (i). These occur as there is no branch prediction to improve the accuracy of loading. Due to this, the last `J`ump instruction is incorrect 4 times resulting in 4 stalls. The reason there is only an increase of 3 stalls is because the branch prediction must make a mistake to update the predicted path, the last `J`ump instruction is incorrect 1 time with branch prediction. This results in a difference of 3 branches.