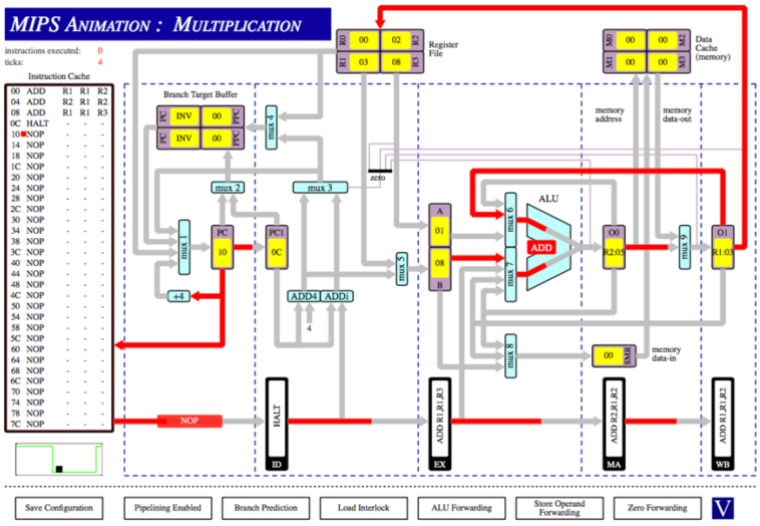
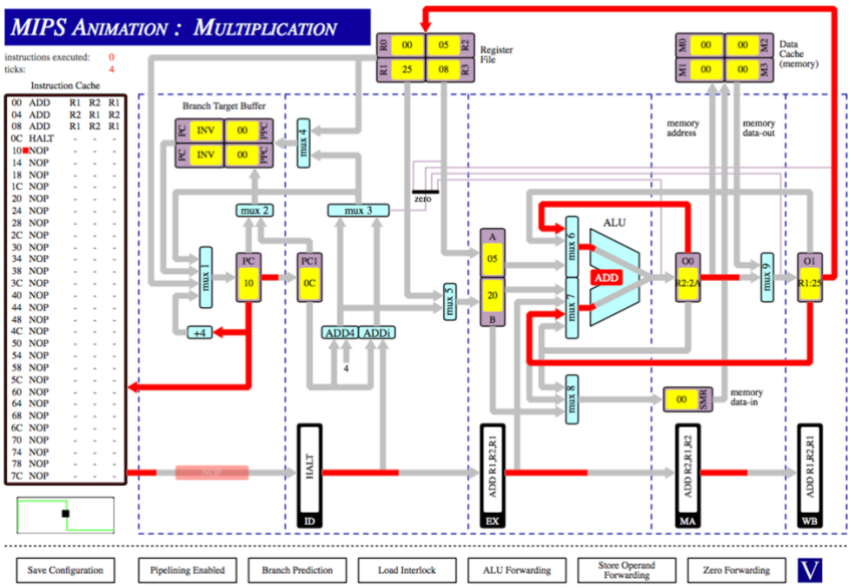
Q1 -
I)

ADD R1, R1, R2
ADD R2, R1, R2
ADD R1, R1, R3
HALT



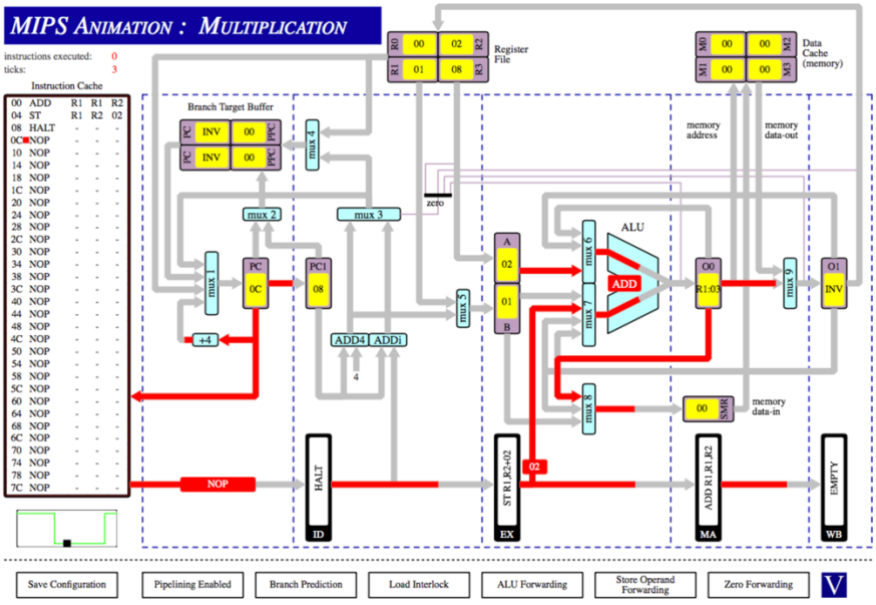
II)

ADD R1, R1, R2
ADD R2, R1, R2
ADD R1, R1, R3
HALT



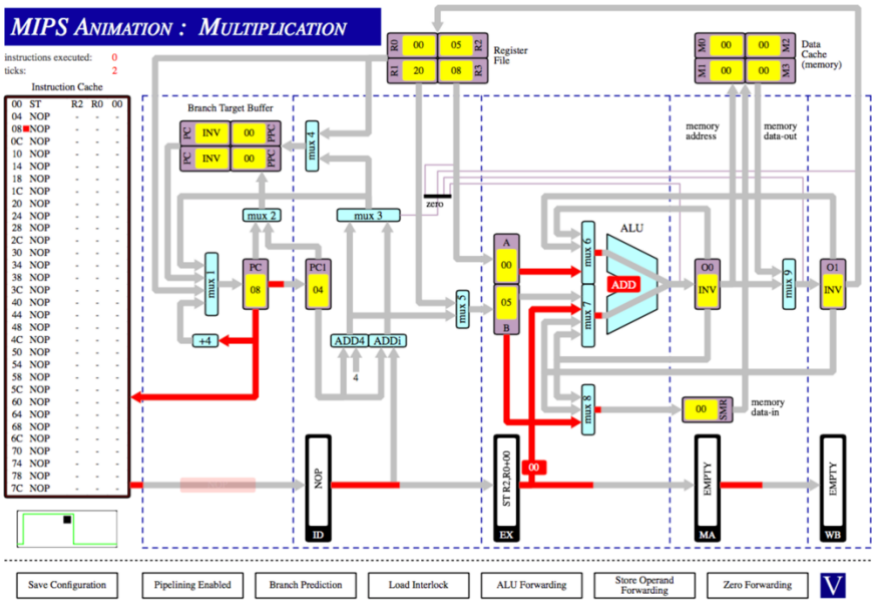
III)

ADD R1, R1, R2
ST R1, R2, 02
HALT



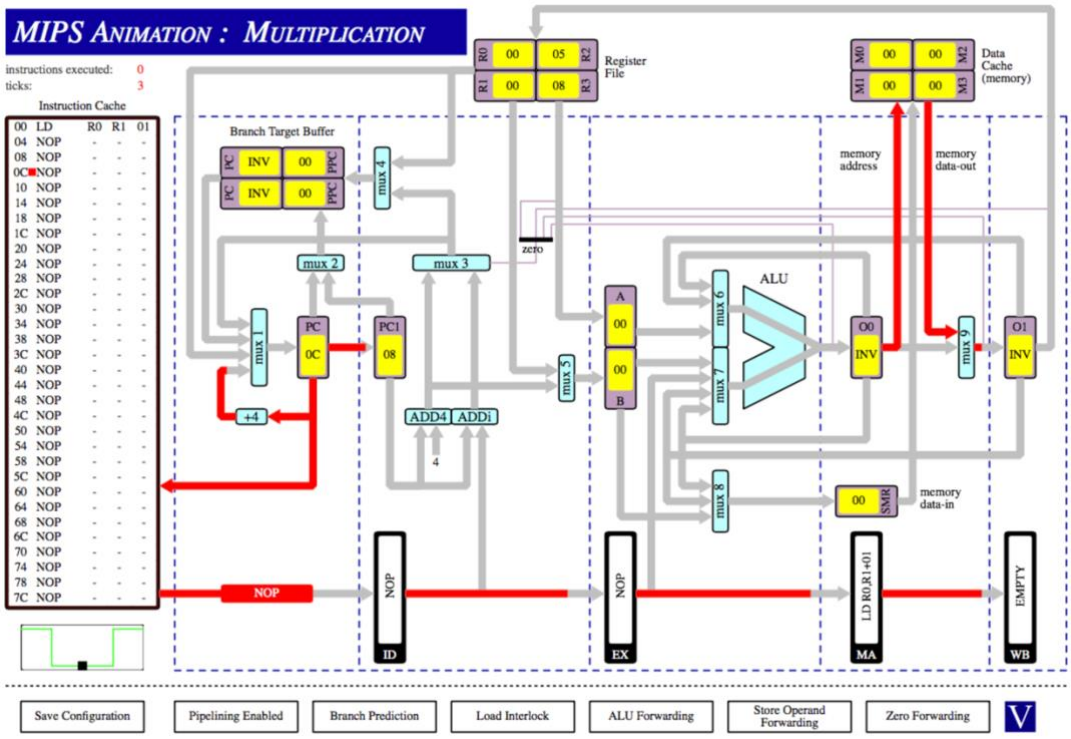
IV)

ST R2, R0, 00

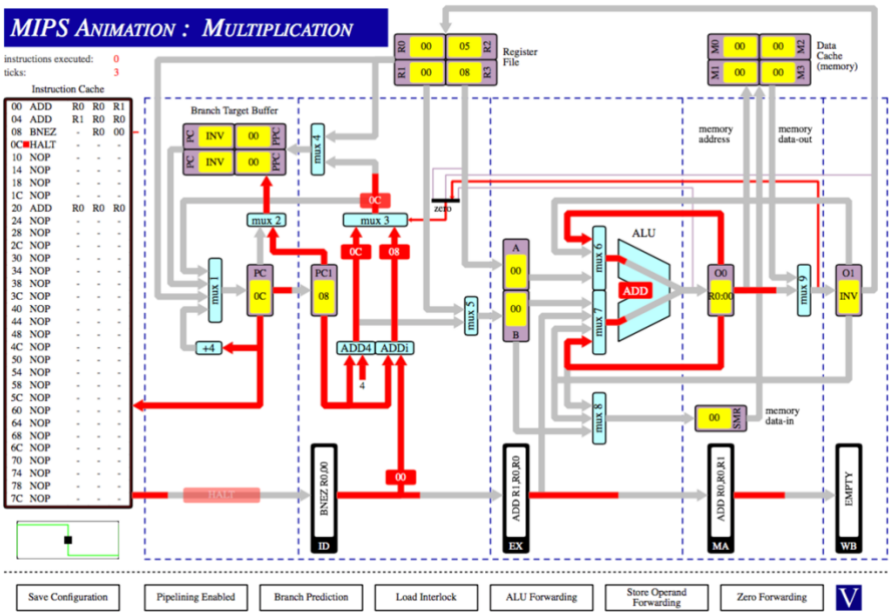


V)

LD R0, R1, 01

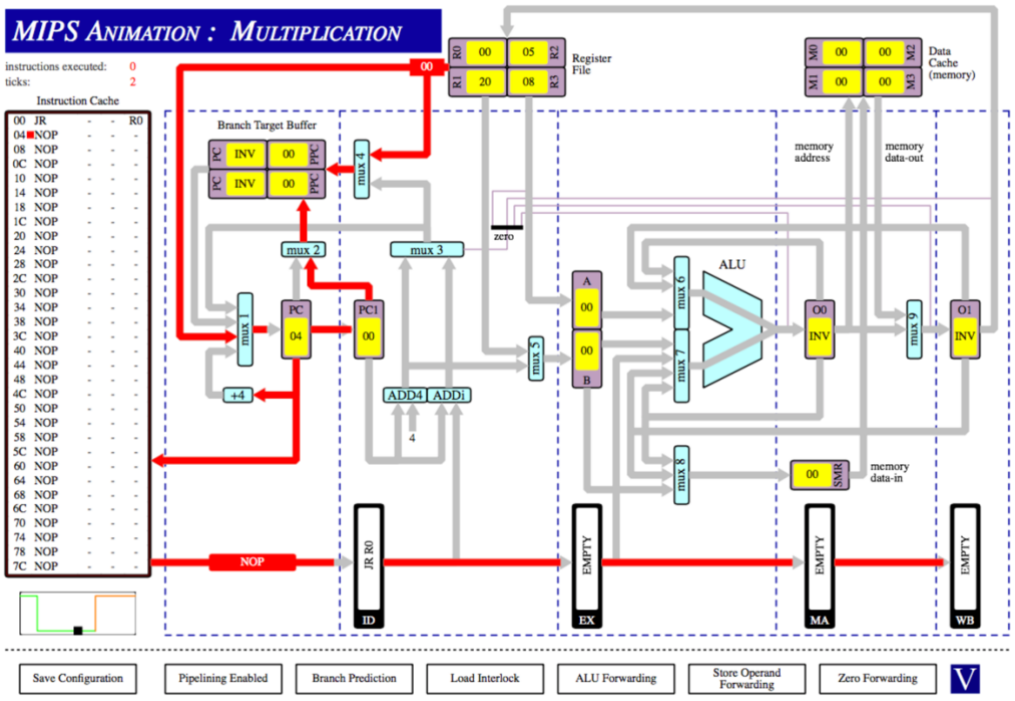


VI)
ADD R0, R0, R1
ADD R1, R0, R0
BNEZ R0, 00
HALT



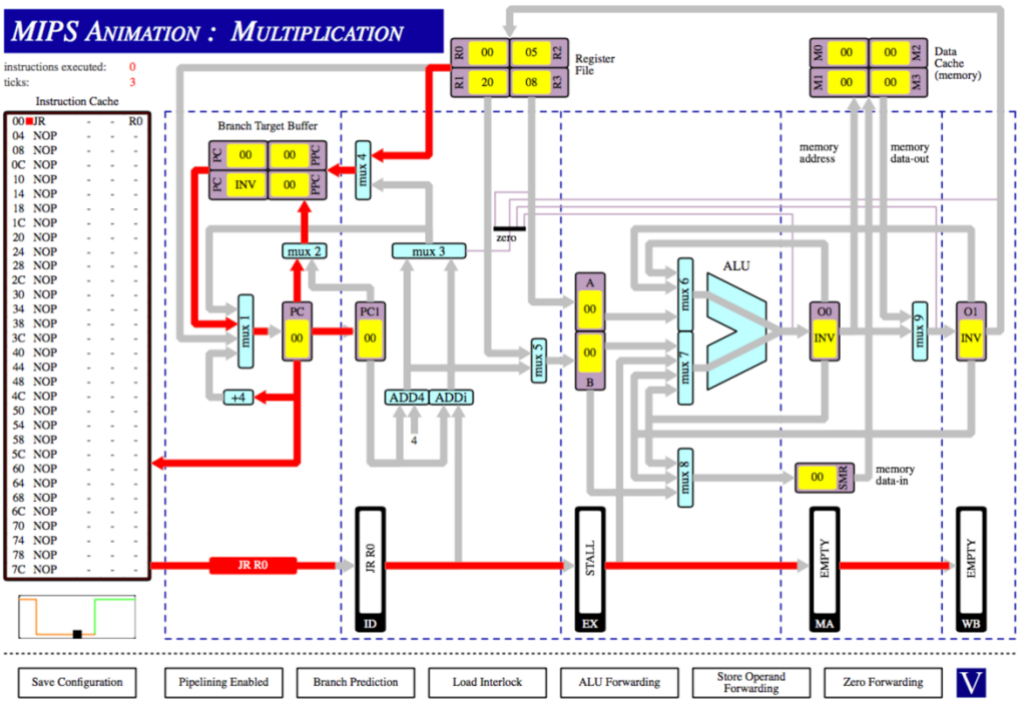
VII)

JR R0



VIII)

JR R0



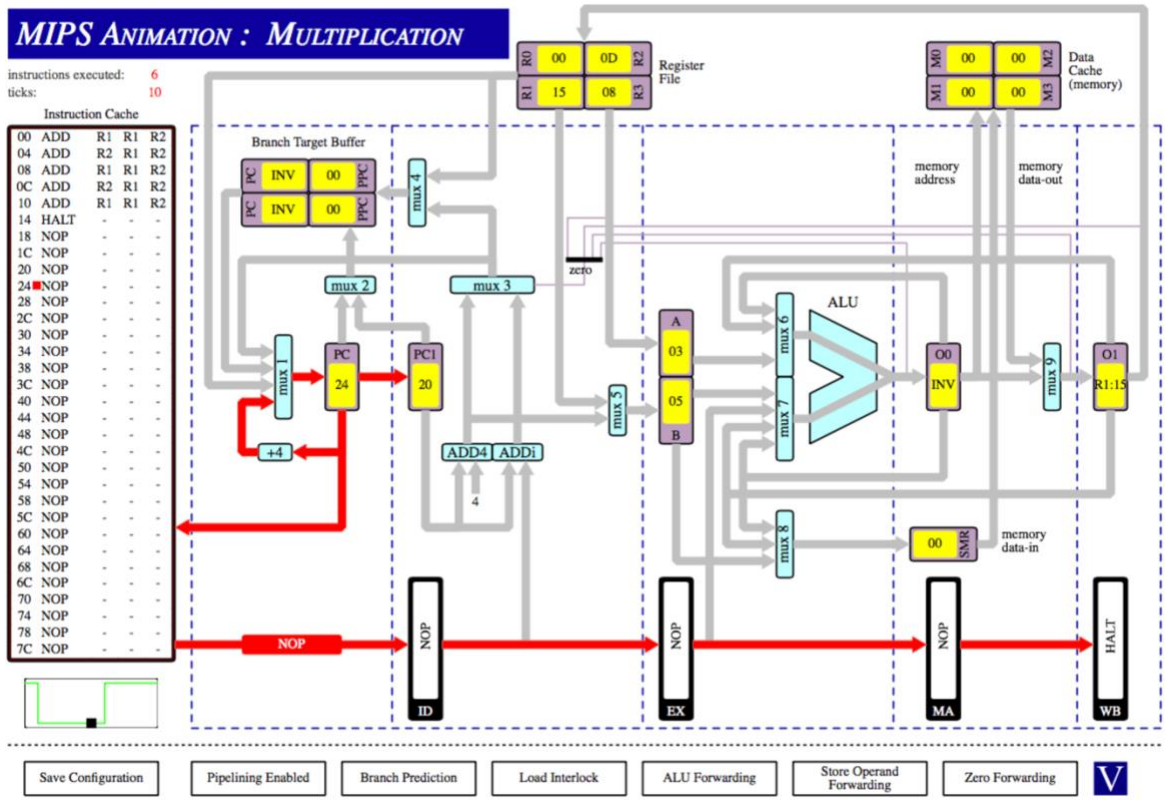
Q2 -

I)

result - 15
clock cycles - 10

The result is 15 as the answers from the previous instructions are used in the calculations.

With pipeline forwarding the ALU result from the previous instruction can be forwarded to the ALU inputs before being written back to the Register File in the WB stage (Of the previous instruction). This means that in the calculations the most recent values can be read before the previous instruction gets through the stages.

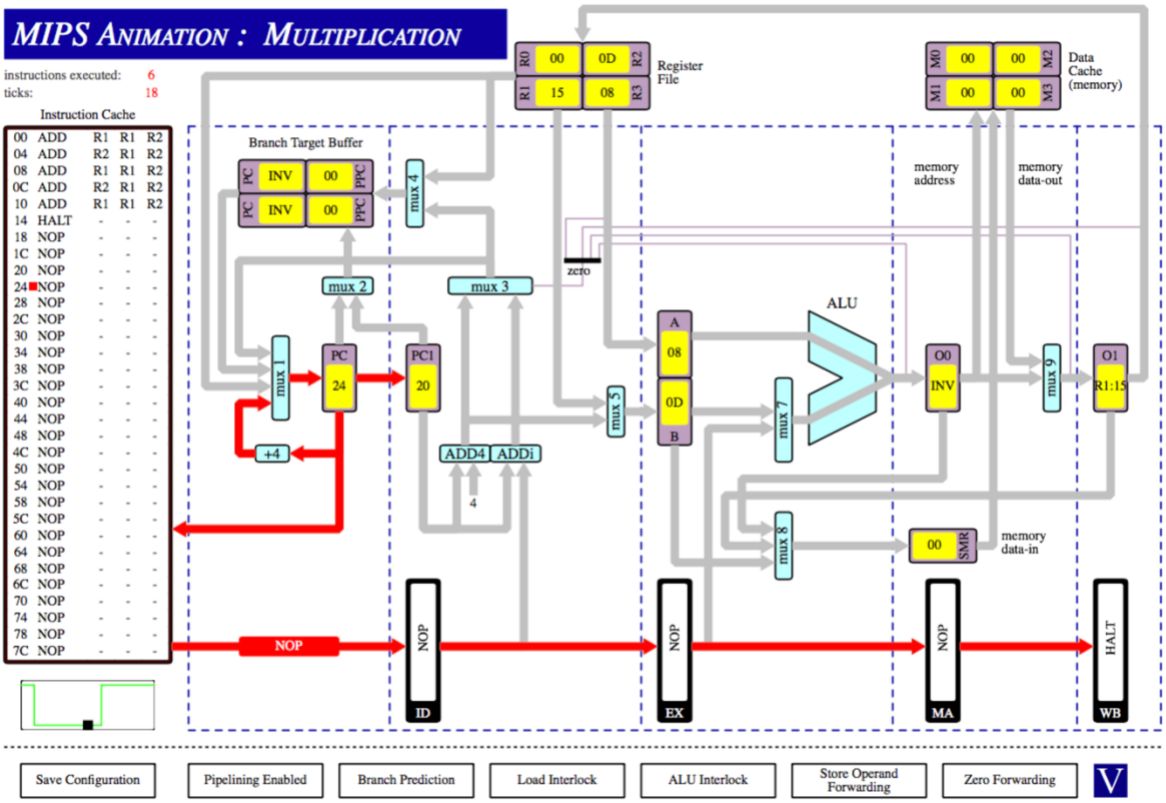


II)

result - 15
clock cycles - 18

The result is 15 as the answers from the previous instructions are used in the calculations.

Without pipeline forwarding we can't read the results from the previous instructions until they have been written back to the Register File. To use these recent values the pipeline is stalled until the previous instruction has made it'd way through the pipeline MA and WB stages. This stalling lasts 2 clock cycles and with the five instructions here we would have 8 extra clock cycles as opposed to part I.

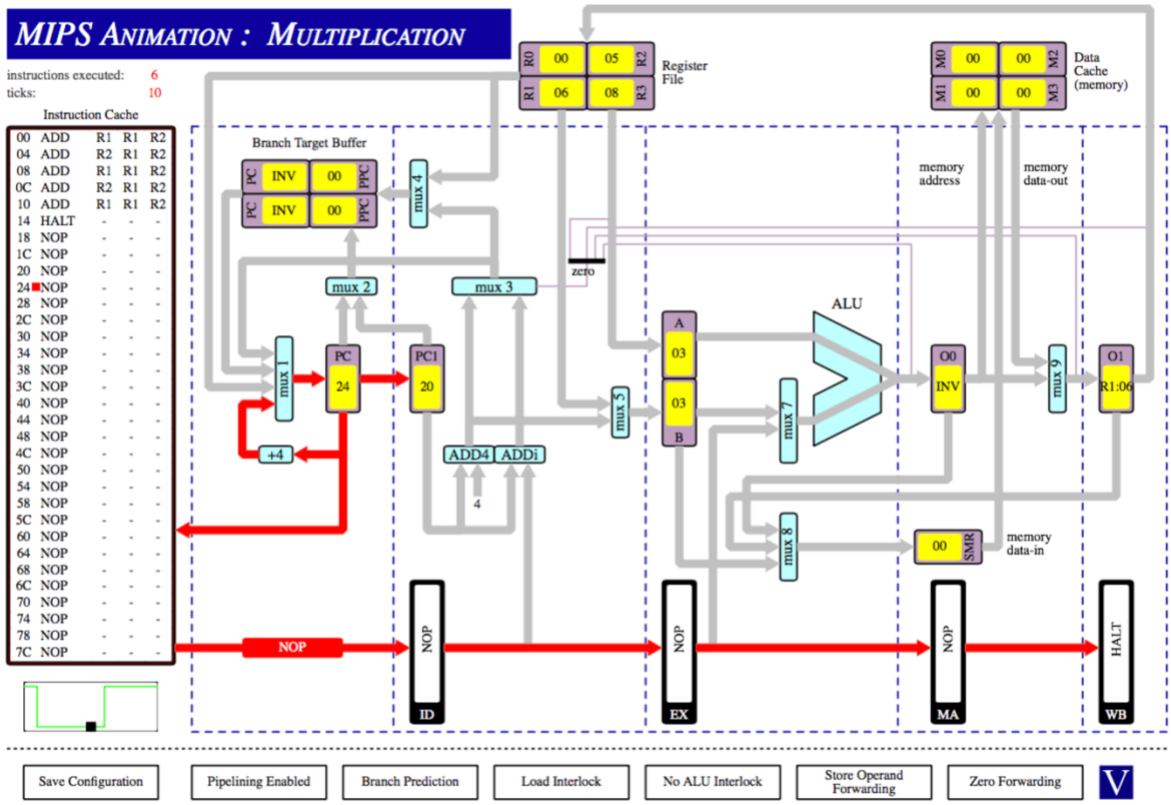


III)

result - 6
clock cycles - 10

Here we use neither forwarding or stalling so the instructions don't always use the answers from the previous instructions because they are not written back to the register file (in the WB stage of the instruction) before the next instruction attempts to read them in it's EX stage. Nothing has made it to the WB stage when the 3rd instruction executes When the 4th one executes the 1st answer has been written back to R1 - 3. At the 5th in the EX phase the answer to the 2nd has been written back as well so it's R2 - 3. Therefore the answer to the 5th instruction (R1+R2) is 6.

With no stalling the cycle count is 10.



Q3 -

I)

Instructions - 38
Clock Cycles – 50

It takes four clock cycles to execute an instruction fully. Each stage (ID, EX, MA, WB) executes in one clock cycle so to speed up we pipeline stages to work in parallel. So, when the ID phase is decoding an instruction *i*, the IF phase would be fetching the next instruction *i* + 1 in parallel. On the next clock cycle, the EX phase would execute instruction *i*, the ID phase would decode instruction *i* + 1 and the IF phase would fetch instruction *i* + 2. This means that without stalls there should be as many clock cycles as instructions.
The number of clock cycles is greater still than the number of instructions as there are 8 stall clock cycles and then four clock cycles to get the halt instruction to fully execute and the program to be terminated.

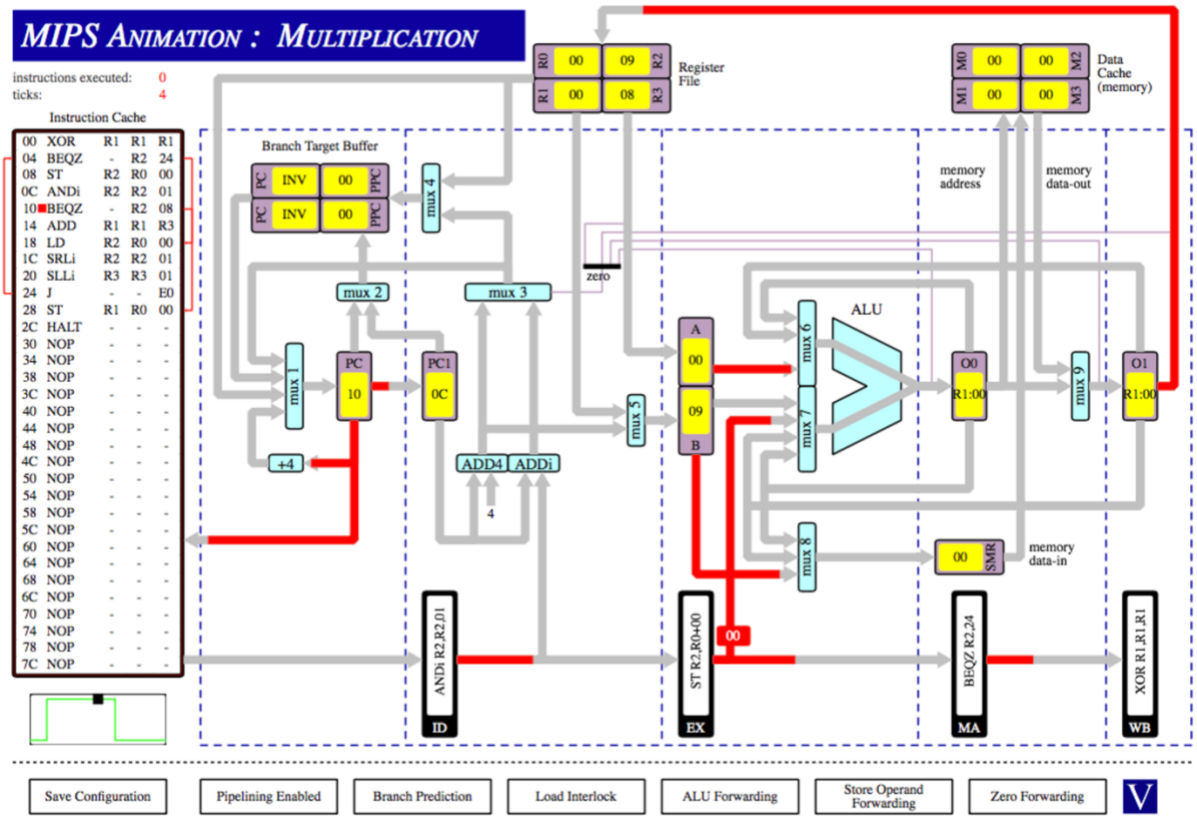
Stalls

After each LD instruction there is a stall cycle. This is to account for a data hazard that could occur if the data that is loaded is attempted to be read in the next instruction. We can use pipeline forwarding so that we don't have to wait for the instruction to be in the WB phase and have written it to the register file however the data is not loaded in until the MA phase

from memory so we need a stall cycle to stop an instruction like add that could come after the LD instruction from trying to read the data when before it was loaded from memory. There are four of these stalls total in the execution one for each time LD is executed.

Stall before j E0. In the round there is a stall cycle before the J E0 instruction. This is because the BTB expects the wrong PC but the branch target buffer eliminates all stalls when it fills up.

Stall before the conditional branch instructions.
The first of these stalls occurs when the condition of the branch is met and thus the the predicted PC does not match the actual PC and it takes a cycle to reconfigure. As above the BTB eliminates stalls when it fills up and expects the same as the last time and so there is no stall until the condition is not met and the predicted PC which expected the condition to be met is wrong. Essentially this stall occurs when the condition is met/not met and that is not what happened last time. There are 3 of these total, 1 with BEQZ R2, 24 and 2 with BEQZ R2, 08.



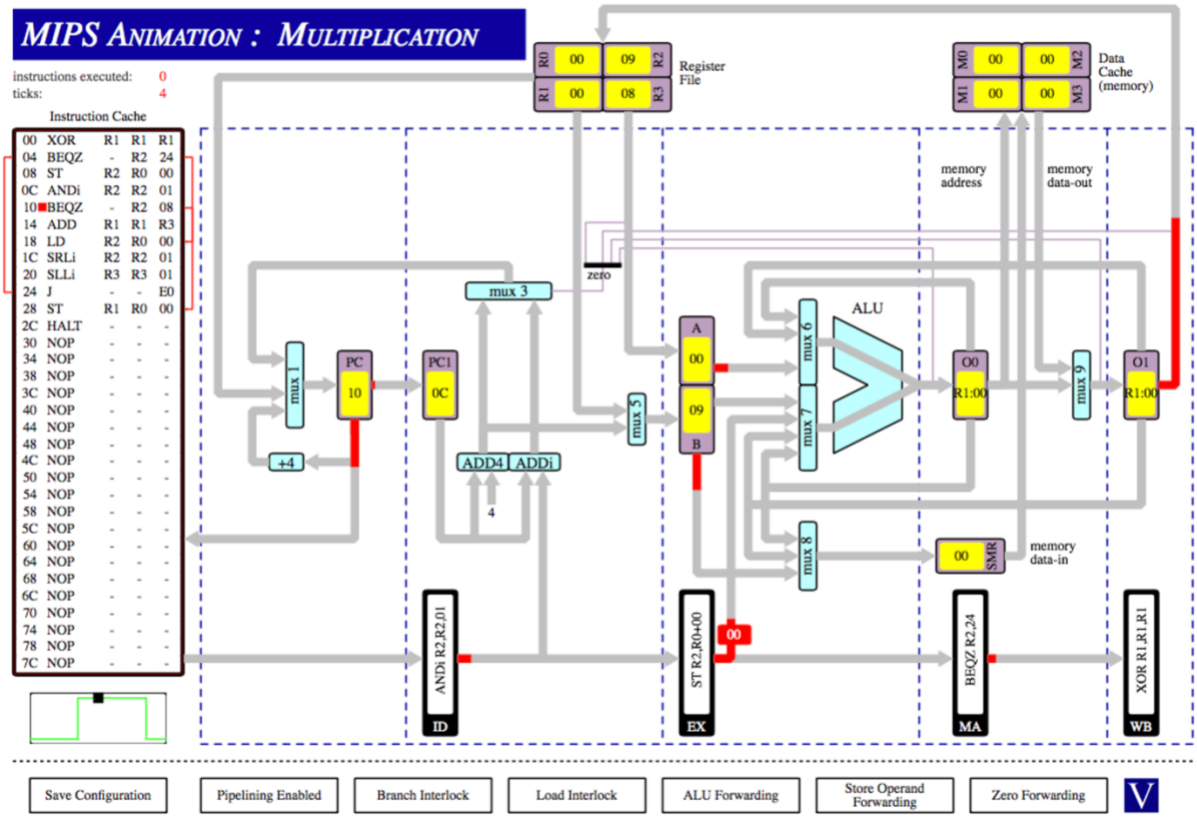
II)

Instructions - 38
Clock Cycles - 53

Implementing a branch target buffer resolves branches during the ID phase. The BTB searches for the current PC, and if found, uses the predicted PC to fetch the next instruction. If the branch is incorrectly predicted, a one cycle stall occurs as the correct instruction is

fetched. The BTB is updated. Without the BTB the PC is updated without prediction and this gives us extra clock cycles.

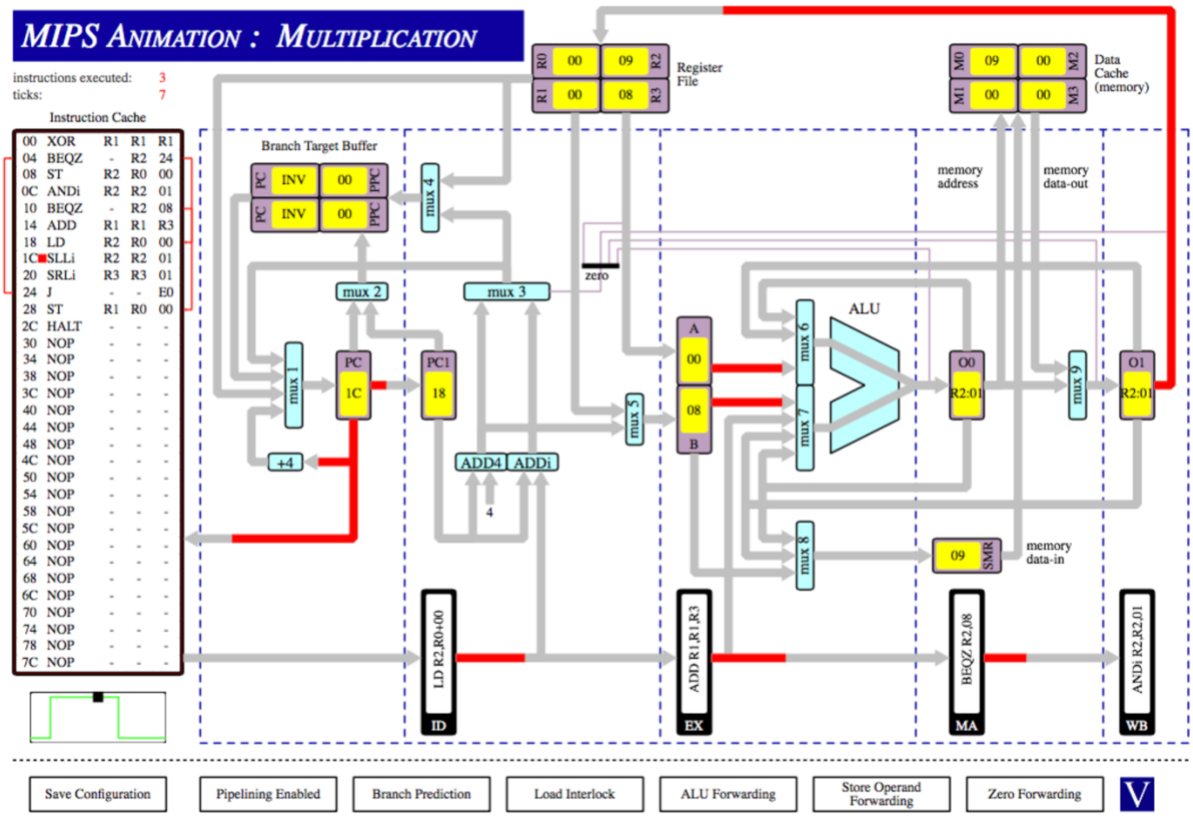
Without prediction there are 11 stalls as opposed to 8 which would account for the extra 3 cycles. These stalls occurred because without prediction there is no predicted target to be based on the last time the branch was executed. This means that stalls are introduced so that the PC can reconfigure whenever a jump occurs as opposed to when a condition is not the same as what happened last time.



III)

Instructions - 69
Clock Cycles - 84

When swapped there are 8 loops as opposed to 4 and 11 stalls as opposed to 8 when they weren't reversed.


MIPS ANIMATION : MULTIPLICATION
instructions executed: 3
ticks: 7
Instruction Cache
00 XOR R1 R1 R1
04 BEQZ - R2 24
08 ST R2 R0 00
0C ANDi R2 R2 01
10 BEQZ - R2 08
14 ADD R1 R1 R3
18 LD R2 R0 00
1C SLLi R2 R2 01
20 SRLi R3 R3 01
24 J - - E0
28 ST R1 R0 00
2C HALT - - -
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
zero
ALU
memory address
memory data-out
memory data-in
+4
ADD4
ADDi
4
LD R2,R0+00
ADD R1,R3
BEQZ R2,08
ANDi R2,01
ID
EX
MA
WB
Save Configuration
Pipelining Enabled
Branch Prediction
Load Interlock
ALU Forwarding
Store Operand Forwarding
Zero Forwarding