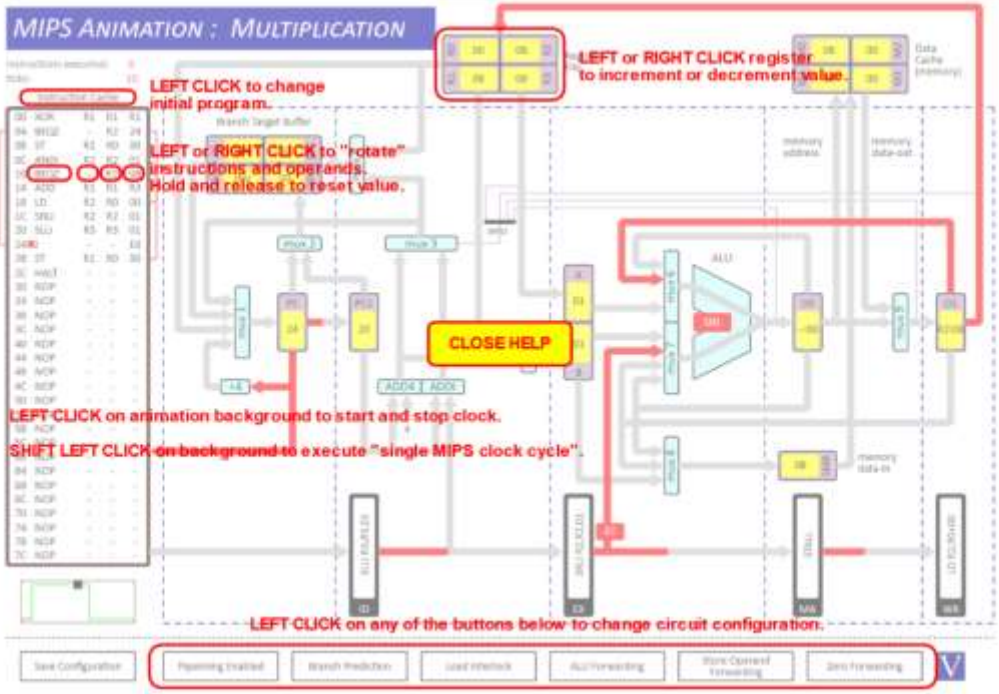
Q1.

(i)Interaction between SRli R2, R2, 01 and LD R2, R0 + 00 seems to cause pipeline forwarding between register 01(WB stage) and mux 6 (ALU stage) as the Srli has a dependency on the R2 register which is being changed by the LD instruction

---



LD R2, R0 + 00

SRli R2, R2, 01

stall of one pipline between them due to ld

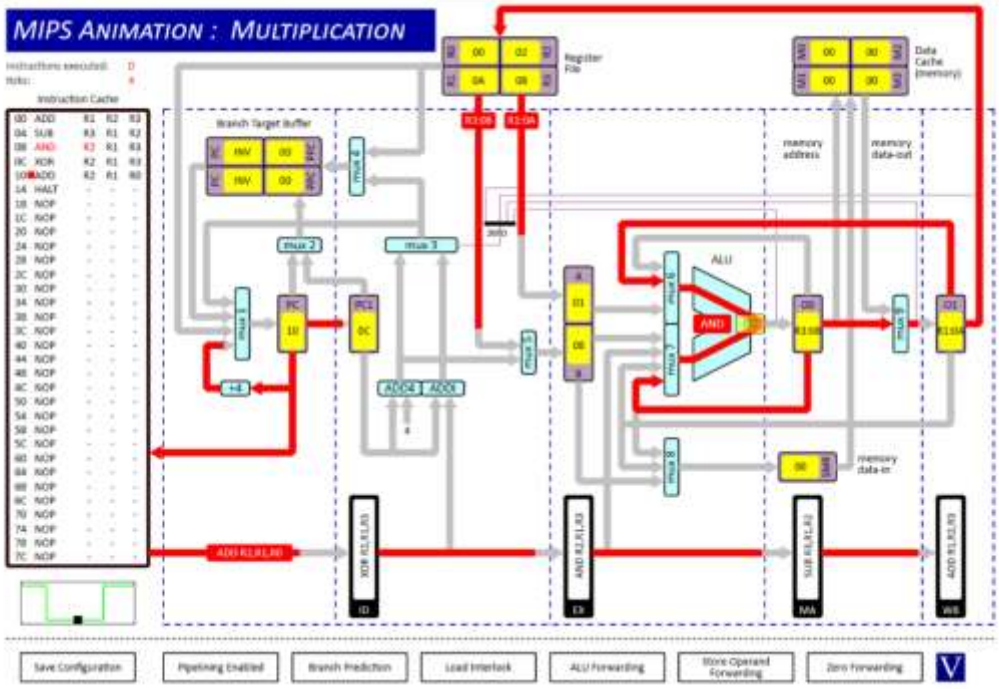
too long for detailed explanation each time so just giving instruction and dependencies from now on

(ii)01 to mux 6 00 to mux 7

ADD R1, R2. R3

SUB R3, R1, R2

AND R2, R1, R3

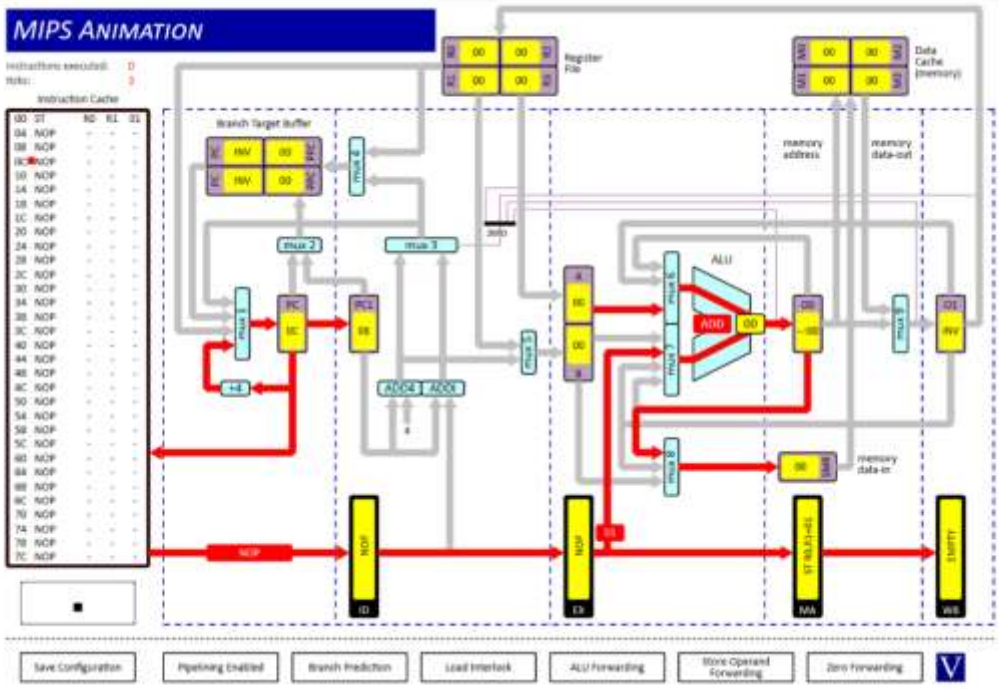


dependency of r1 and r2

(iii) 00 to mux8

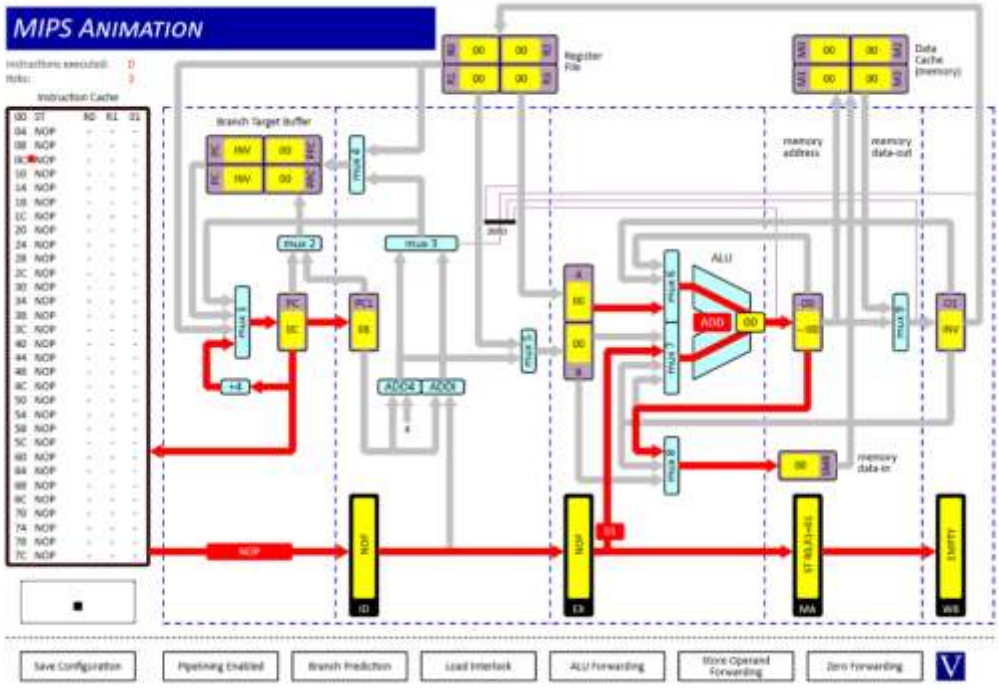
ST R0 R1 01

NOP



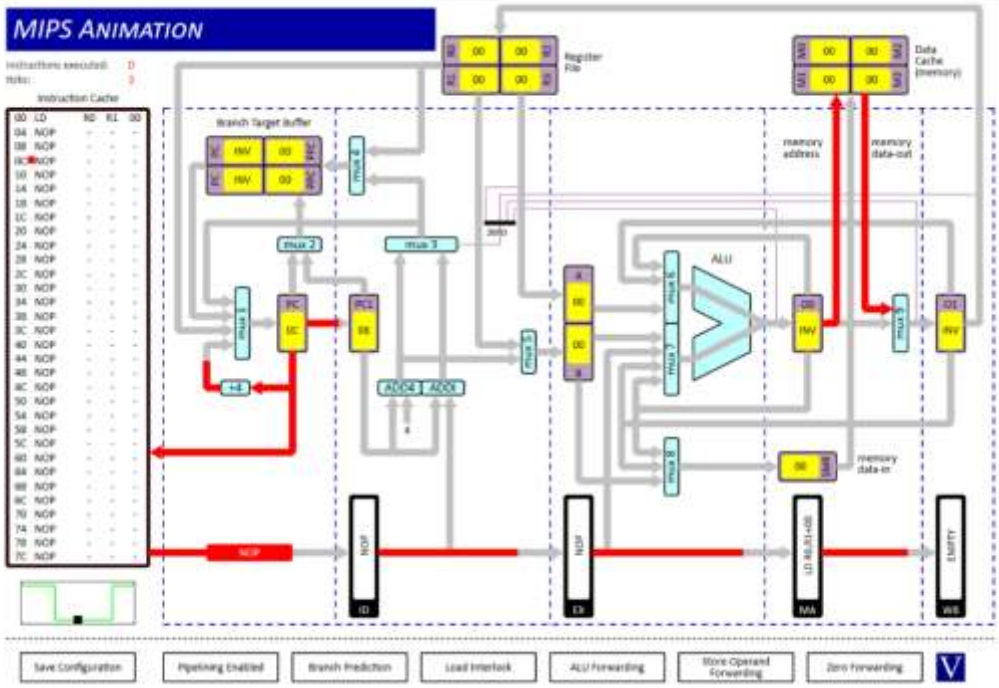
(iv)Ex to Mux 7

ST R0 R1 01



uses value 01 for alu addition

(v) LD R0 R1 00

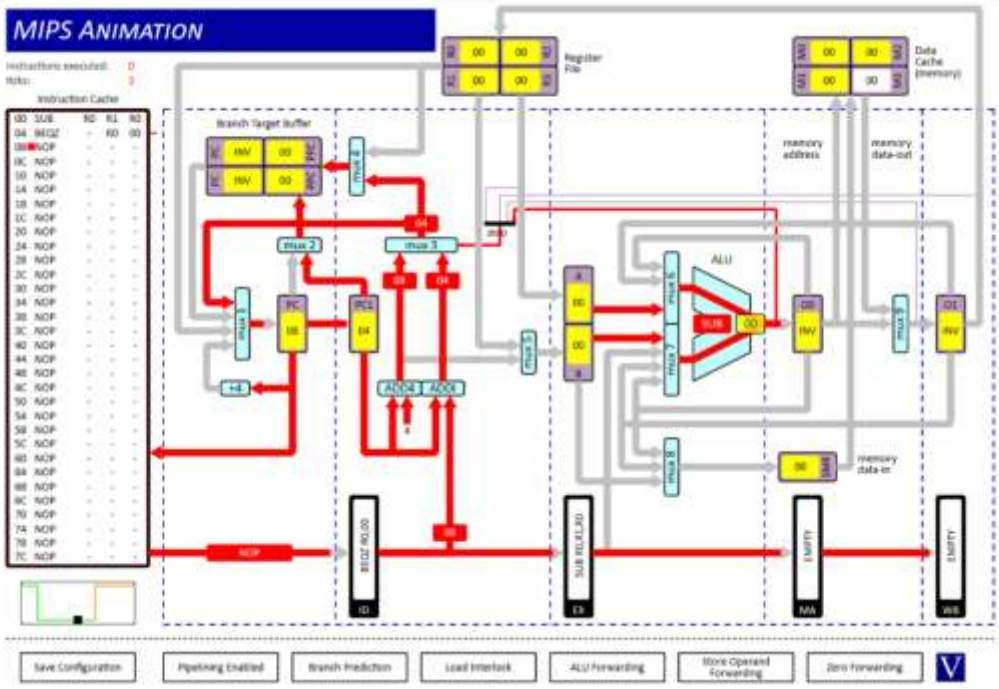


loads from memory

(vi) 00 to Zero detect

SUB R0 R1 R0

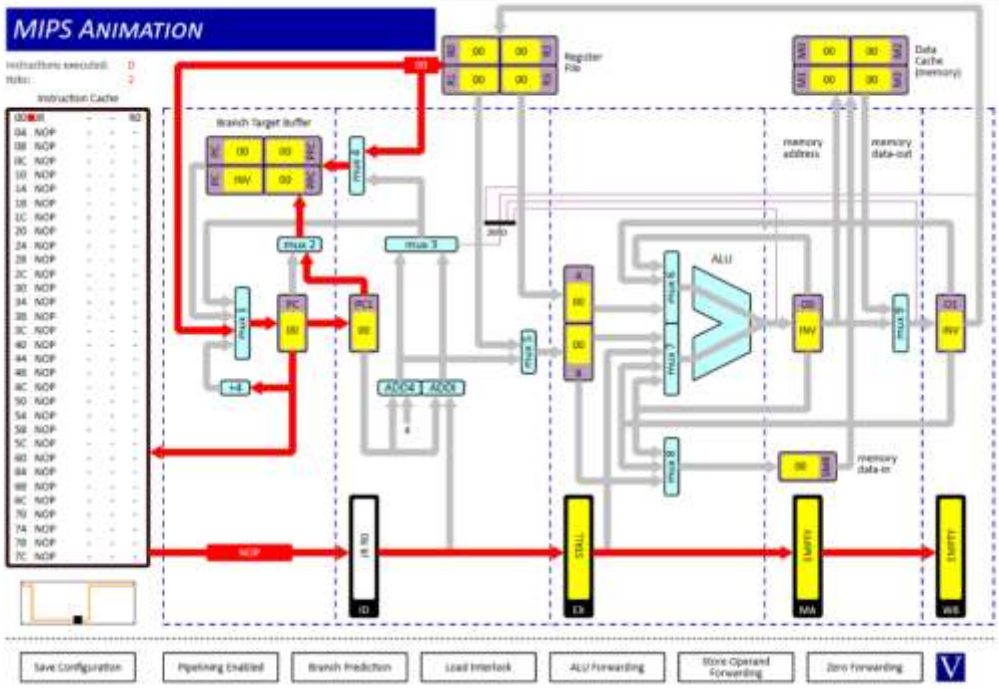
BEQZ - R0 00



BEQZ depends on zero flag being set

(Vii) Register file to mux 1

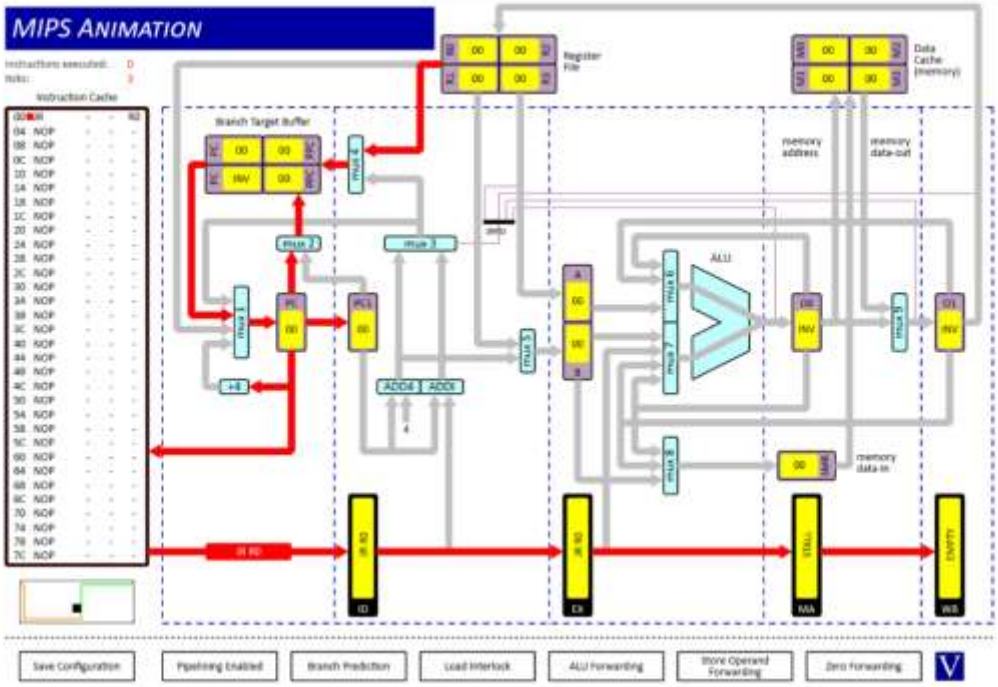
JR - - R0



Jump to address in R0, moves value in R0 to PC

(Viii) branch target buffer to Mux 1

JR - - R0



branch predicts to go to 0 since R0 is 0 and only instruction.

Q2.

(i) r1 = 0x15 cycles = 10

(ii) r1 = 0x15 cycles = 18

(iii) r1 = 0x06 cycles = 10

The first one has pipelining and alu forwarding meaning that R1 uses the results of the operations ahead of it, if the computation has a dependency on the computation of operations ahead of it.

This allows it to complete it in 10 cycles without any stalls between pipeline stages. The third result also takes 10 cycles to run since it simply runs all the instructions without taking into account any of the dependencies.

The second result adds stalls between operations to account for the dependencies since it has no alu forwarding mechanism to try figure out the computations before the wb phase of the dependent operations is complete.

Therefore the second result takes the longest at 18 cycles. The third result takes same time as first instruction at 10 cycles but has the wrong result due to not taking into account the dependencies. The first result has both the correct answer and takes only 10 cycles(no stalls) due to alu forwarding.

Q3.

39 instructions executed

(i)51 ticks

The two values are not the same due to pipeline stalls due to data dependencies on load and conditional jump.

When a branch is mispredicted theres a penalty of some stalls which is the equivalent of not executing an instruction in one cycle.

When theres a load data dependencies there are stalls introduced as the operation cannot continue until the value is acessed in the MA phase.

Additionally even with no stalls the value would not be the same because atleast the first instruction would take 4 cycles since there are 4 stages in the pipeline.

(ii)53 ticks

The program takes extra cycles as without branch prediction there are more stalls since there are more branch mispredictions without any branch predictions(ie. it has to throw away more instruction calculations as it always assumes the branch is not going to taken by default)

(iii) 47 ticks

The data dependencies between the ld instruction and the SRli is gone as the Srli has one instruction ahead of it. This means the stalls caused by the LD instruction are gone. The remaining stalls are caused by branch mispredictions.