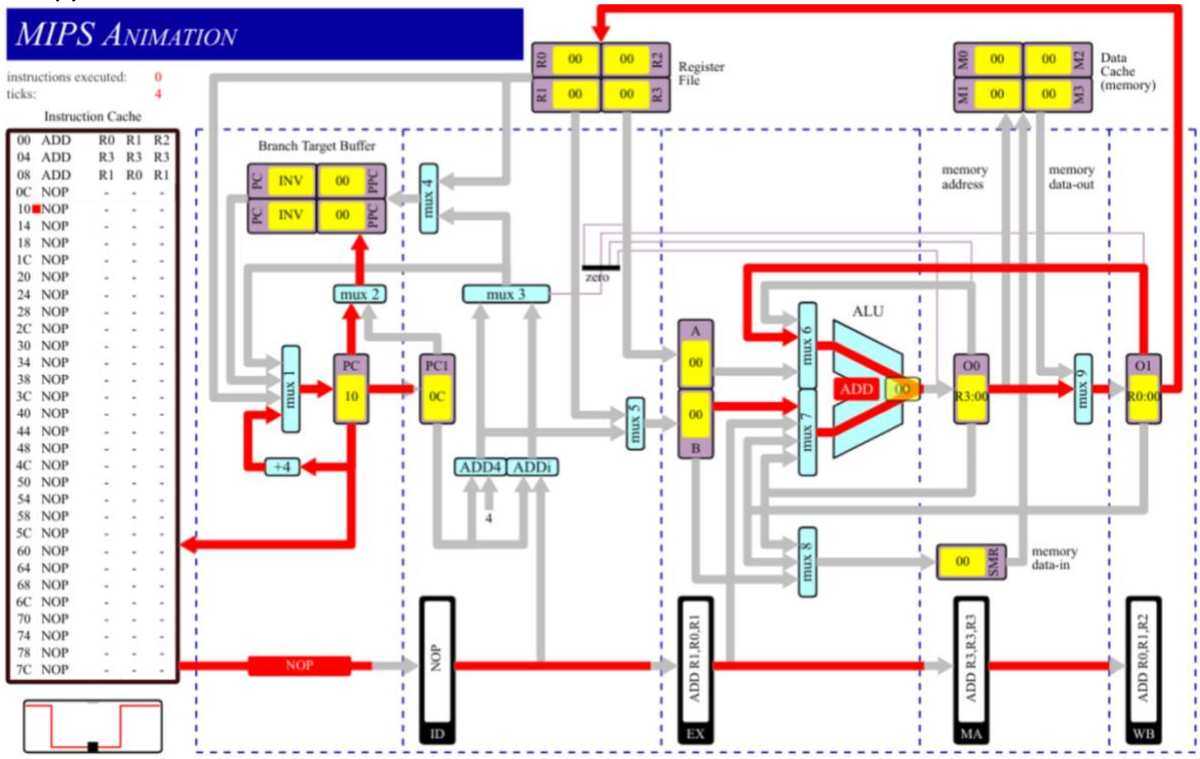
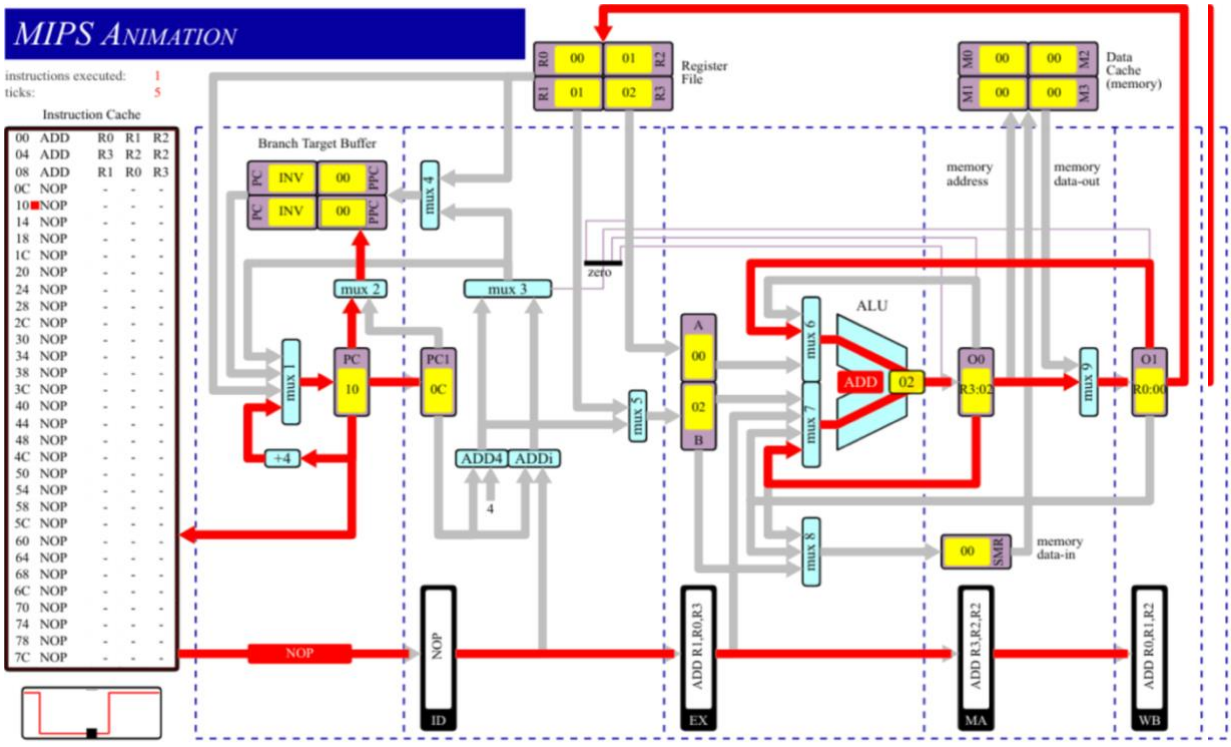
# Computer Architecture II Tutorial 4 – Davy Nolan

Q1. Show specified paths for:

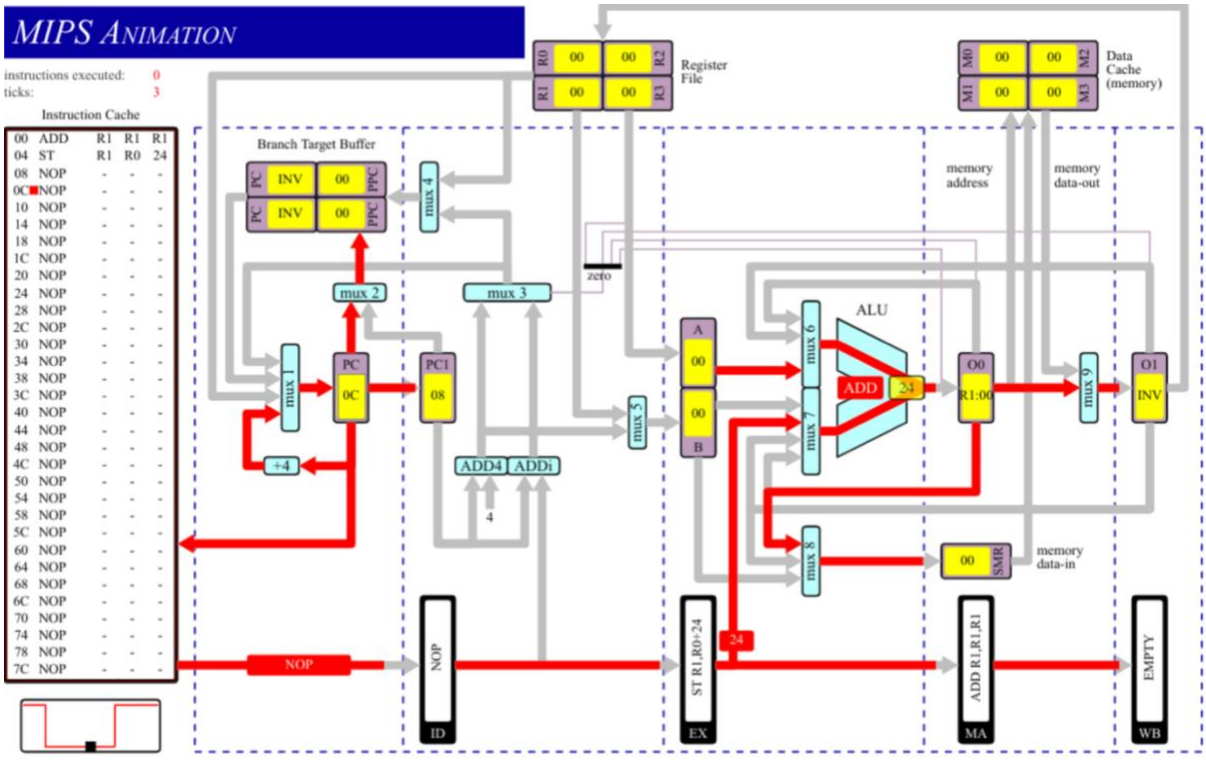
(i) O1 to MUX6



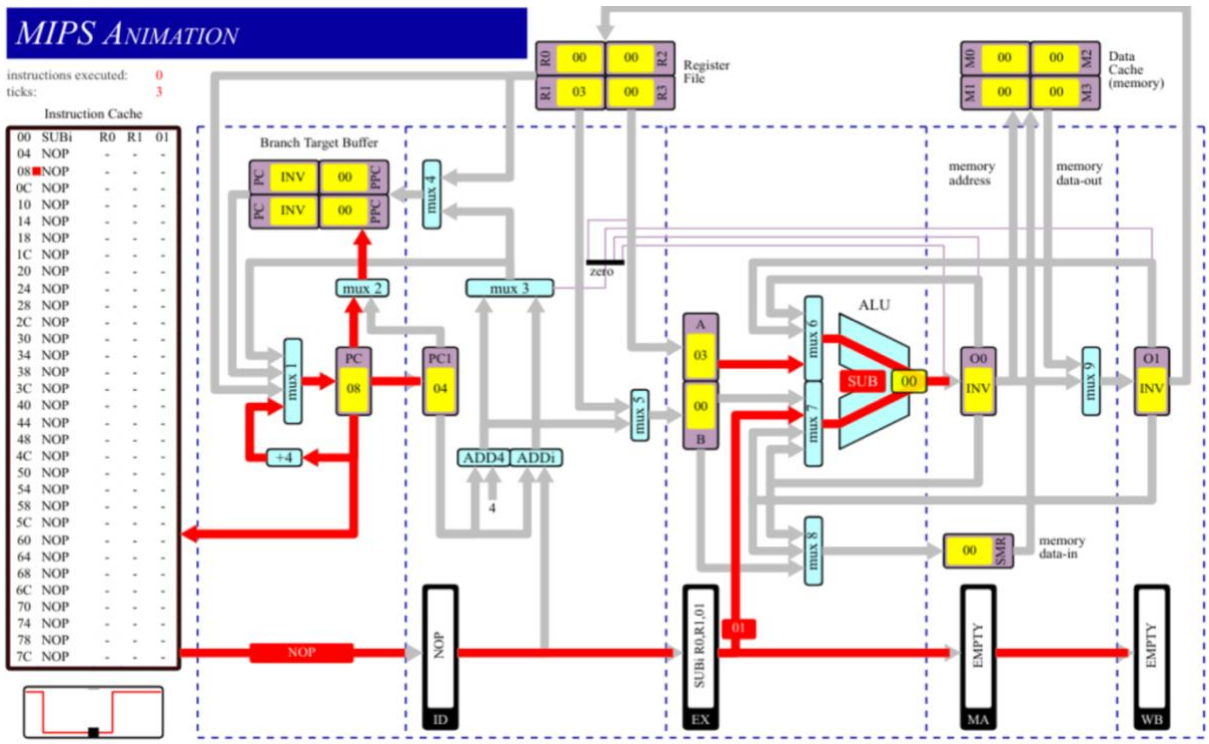

(ii) O0 to MUX7 and O1 to MUX6 (Simultaneously)



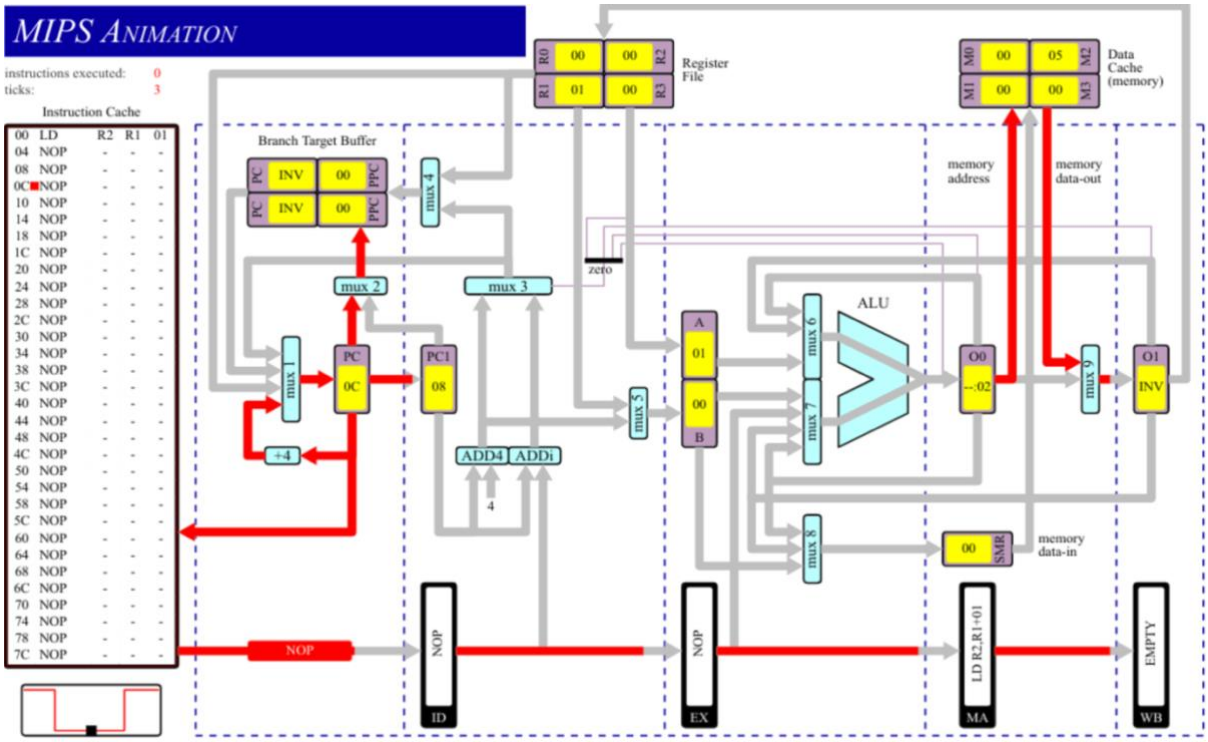
(iii) O0 to MUX8



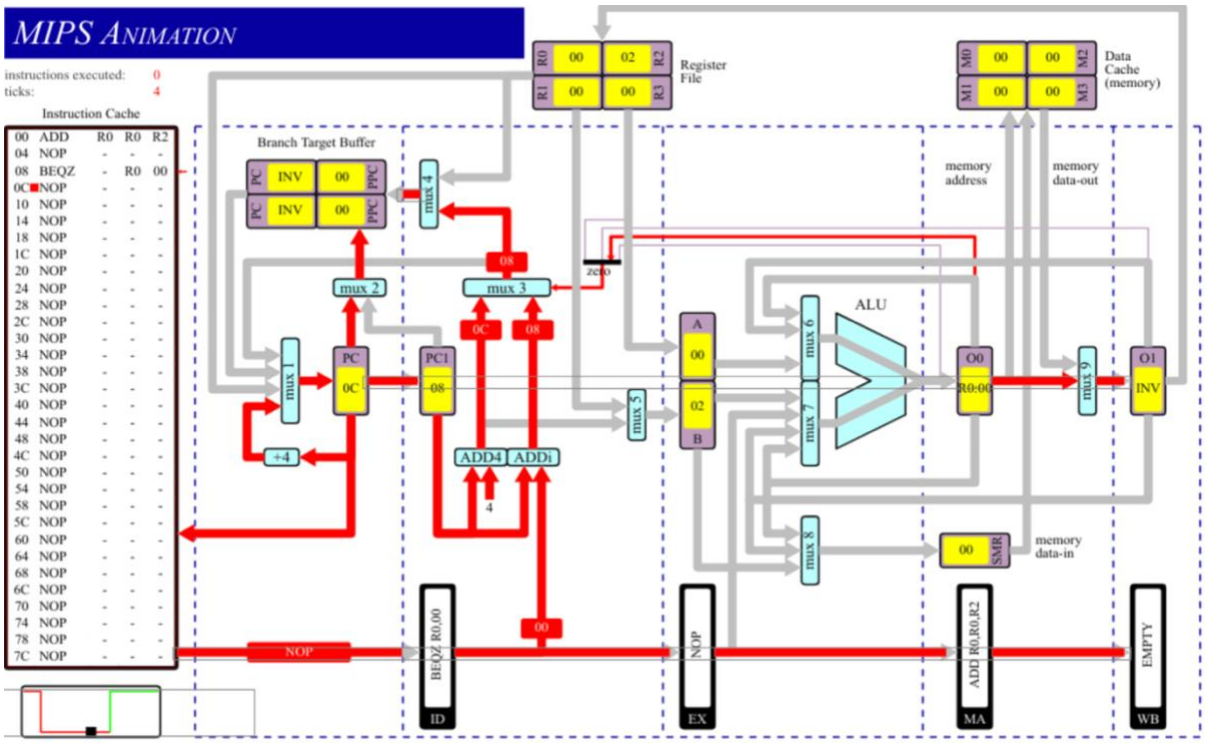

(iv) EX to MUX7



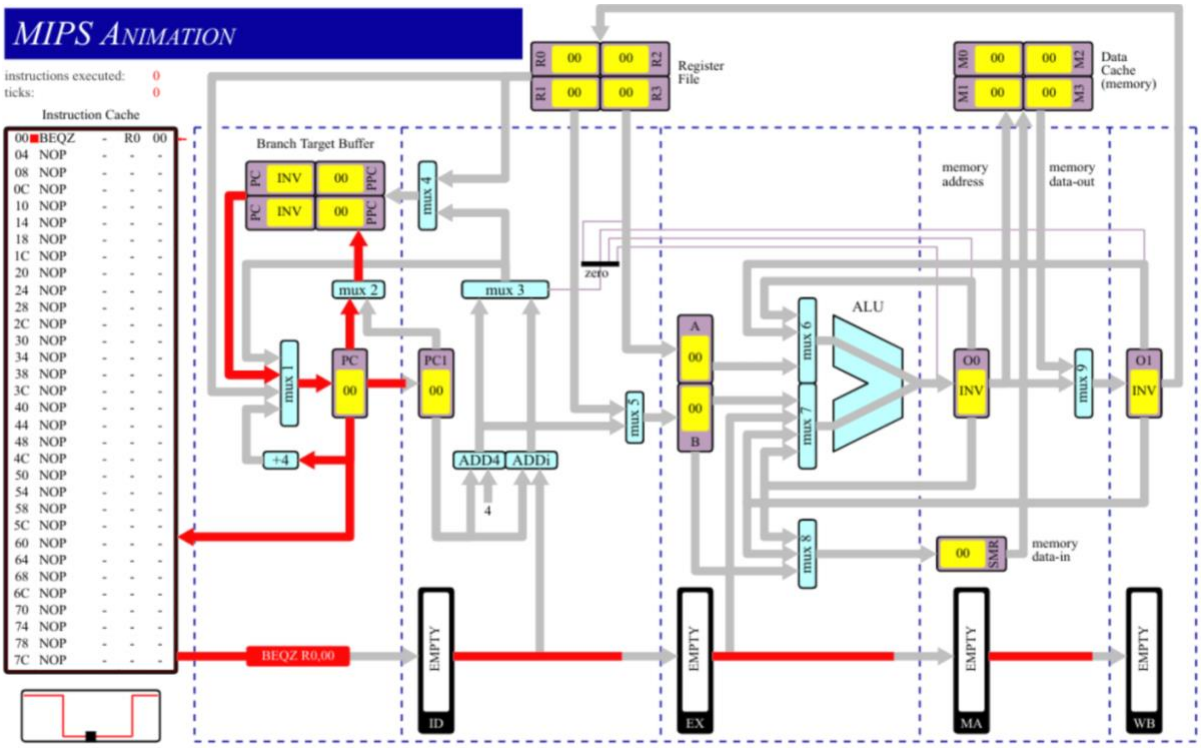
(v) Data cache to MUX9




(vi) O0 to Zero Detector



(vii) Branch target buffer to MUX1




Q2.

Add r1, r1, r2 ;r1 = r1 + r2
Add r2, r1, r2; r2= r1 + r2
Add r1, r1, r2; r1 = r1 + r2
Add r2, r1, r2; r2= r1 + r2
Add r1, r1, r2; r1 = r1 + r2
Halt

**ALU Forwarding enabled & CPU Data Dependencies enabled**

No. of clock cycles: 9 cycles
R1 = 21 (correct value)

**ALU Forwarding disabled & CPU Data Dependencies enabled**

No. of clock cycles: 17 cycles
R1 = 21 (correct value)

**ALU Forwarding disabled & CPU Data Dependencies disabled**

No. of clock cycles: 9 cycles
R1 = 6 (incorrect value)

The clock cycle nearly doubles whenever ALU Forwarding is disabled because every time there is a data hazard, the pipeline must wait for the data hazard to clear. This is called “stalling”. As instructions are blocked from entering the EX phase, they are stuck in the ID phase. As each phase is occupied by one instruction at a time, this tall propagates backwards so that the IF phase becomes congested.

When CPU Data Dependencies are disabled it gives an incorrect answer as the CPU cannot handle data hazards so it pushes through the pipeline with incorrect values for source operands being read from the register file.

Q3.

(i)

No. of instructions executed: 22
No. of clock cycles: 28

The pipeline is empty when the program begins. Only one phase is at work at a time until the pipeline is filled. The pipeline is only fully utilised by tick 5.
Since the **No. of clock cycles = No. of instructions + (pipeline size – 1)**
the answer is 26, therefore 2 clock cycles are not accounted for.

The 2 missing instructions are from the branch instructions J and BEQZ. When these are first executed, the pipeline stalls in the ID phase for a cycle. The branch target buffer maps the branch instruction address to the new PC value. This can prevent subsequent stalls whenever these branches are executed in the future as branch prediction occurs.

(ii) No. of clock cycles executed: 29

When Branch interlocking is enabled, it removes the branch target buffer availability. Therefore there are no cached branch instructions and thus, no branch prediction occurs. Branch prediction aids in reducing extra cycles by always taking a branch if there is a branch target buffer cache hit. There is an extra cycle in this scenario as the PC must be evaluated again for the 2$^{nd}$ J instruction, whereas before the destination address was cached and branch prediction cut out the extra cycle.

(iii) No. of instructions executed: 22
No. of cycles: 30 cycles

This increases the executed time by 2 cycles. This is because there is now a load hazard at every iteration. SRLi is dependent on the right value of R2 being loaded from memory by the previous LD instruction, hence there is a 1 cycle stall as the SRLi instruction stalls in the ID phase for the LD instruction to complete the MA phase. The correct value for R2 can be forwarded using ALU forwarding.