CA Assignment 2

(1.) Pipelining

Given:- time taken by unpipelined processor = 16ns for 1 instruct^n
latch latency = 0.2 ns.
sequential pipeline stages = 12.

1) cycle time for processors:-
unpipelined = time taken to complete 1 instruct^n + latch latency

$= 16 + 0.2 = \boxed{16.2 \text{ ns.}}$

pipelined = longest time taken to complete 1 instruct^n/stage + latch latency

$= 1.8 + 0.2 = \boxed{2 \text{ ns.}}$

[Since, its unequal stages the stage with longest time is considered to accomodate all other stages]

2) Clock speeds for both processors.

unpipelined = $\frac{1}{\text{cycle time}} = \frac{1}{16.2 \text{ns}} = \boxed{61.7 \text{ MHz}}$

pipelined = $\frac{1}{2 \text{ ns}} = \boxed{500 \text{ MHz}}$

3) IPCs in both processors.
unpipelined = 1 [~~takes~~ completes 1 instn in 16.2ns cycle time]
pipelined = 1 [completes effectively 1 instn in 2 ns cycle time]

4) time taken to complete 1 instruction.
unpipelined = time taken to complete 1 instruct^n + latch latency
$= 16 \text{ns} + 16.2 \text{ns}$ total
pipelined = ~~1.8 + 0.2 = 2 ns.~~ by time time for 1 inst^n $\times$ 12
$= 2 \times 12 = \boxed{24 \text{ns}}$

cycles taken to complete 1 instruction.
unpipelined = 1 cycle
pipelined = 12 cycles.

5) speedup of 12-stage pipeline = $\frac{\text{12-stage pipeline Perf.}}{\text{unpipeline perf}}$

$= \frac{\text{cycle time of unpipelined}}{\text{cycle time of pipelined}}$

$= \frac{16.2}{2} = \boxed{8.1}$

6) speedup of 1000-stage pipeline $= \frac{T + T_{ovh}}{T/N + T_{ovh}}$

$= \frac{16 + 0.2}{\frac{16}{1000} + 0.2} = \frac{16.2}{0.216} = \boxed{75}$

(2) Instructions in the 5-stage Pipeline.

1) A load instruction such as LD R3←[R2] computes the address of the main memory in the ALU stage of the basic 5-stage pipeline. The ALU stage will add some offset [0 in case of no offset given] to the register R2 and compute an address for the main memory location from where we fetch the data to be loaded into R3.

2) Instructions that don't write to registers.

⇒ ST R2 → 2[R1]

A store instruction basically stores writes to a memory location and not to a register. This instruction will simply read the input registers R1 & R2, compute the address to the memory location by adding offset to R1 i.e. R1+2, and then write the value in R2 to that memory location pointed by R1+2.

⇒ BEZ R1, [R2]

A branch instruction will only read the input registers and do a comparison to take appropriate action afterwards.

[by incrementing, PC]
Since it will move to some other branch, it won't perform any write instruction to memory or registers.

3) Instruction that writes to memory.

⇒ ST R3 → 2[R2]

This instruction will write the value in R3 to a memory location given by R2+2.

4) Instruction that doesn't use ALU stage of pipeline.

⇒ BEZ R2, [R3]

This instruction means move to the branch if equal to zero i.e. R2 = value in R3 memory location = 0. BEZ basically compares ~~the~~ R2 & R3 and then branches to the next instruction accordingly by incrementing the PC by an offset. So it doesn't use the ALU stage rather moves to the next instruct^n beforehand.

5) Instruction that does nothing in DM stage of pipeline.

⇒ ADD R3 ← R1, R2

This instruction will read input registers R1&R3 perform arithmetic operations (R1+R2) in ALU and then write the output to register R3. It doesn't write or read from main memory. Hence it does nothing in the DM stage.

6) input registers required

ADD - 2.
LD - 1
ST - 2

③ Data Dependencies

1) ADD R3 ← R1, R2
   ADD R5 ← R3, R4

without bypassing.

ADD R3 ← R1, R2 IF D/R AL DM RW

ADD R5 ← R3, R4 IF D/R D/R D/R AL DM RW

POP

POC

2 stalls.

with bypassing

ADD R3 ← R1, R2 IF D/R AL DM RW

ADD R5 ← R3, R4 IF D/R AL DM RW

POP

POC

0 stalls

2) LD R2, [R1]
   LD R3, [R2]

without bypassing

LD R2, [R1] IF D/R AL DM RW

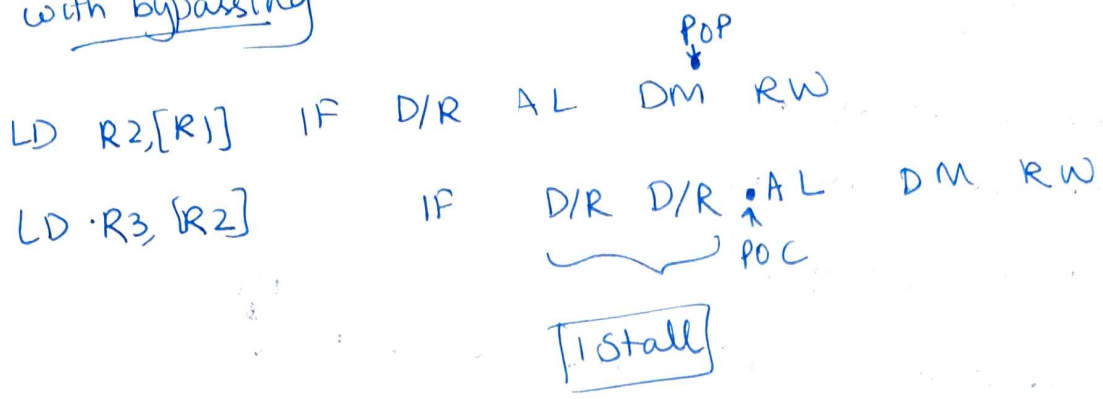
LD R3, [R2] IF D/R D/R D/R AL DM RW

POP

POC

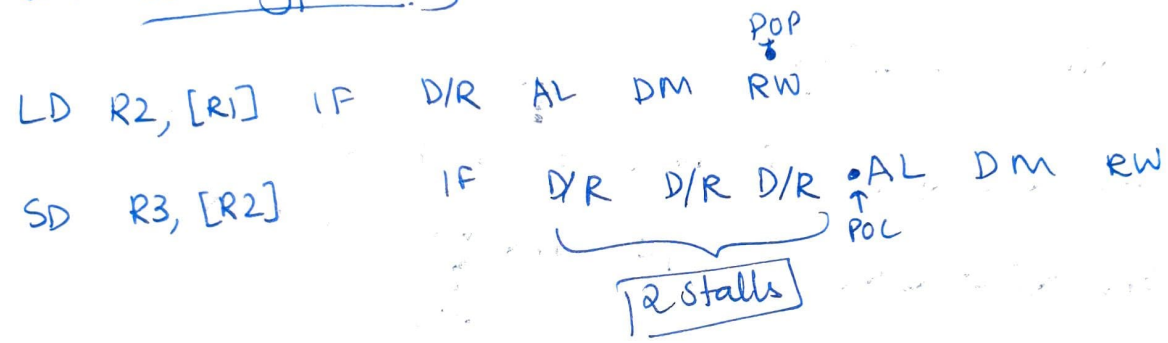
2 stalls

with bypassing

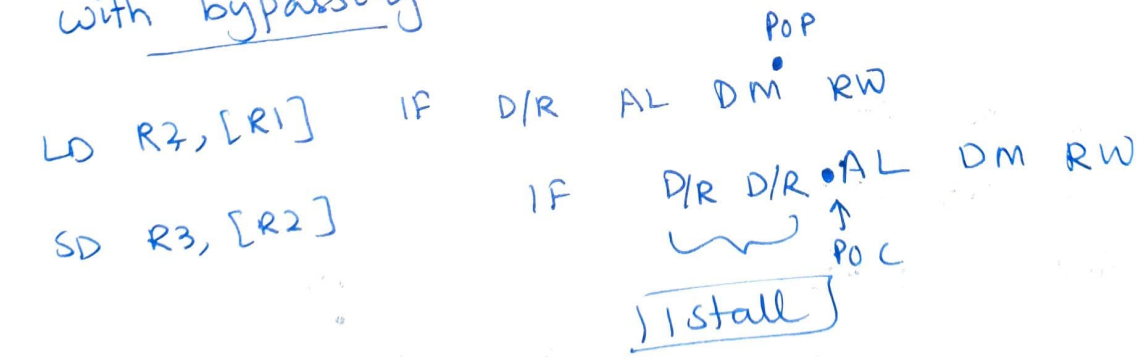


3) LD R2 ← [R1]
SD R3 → [R2]

without bypassing

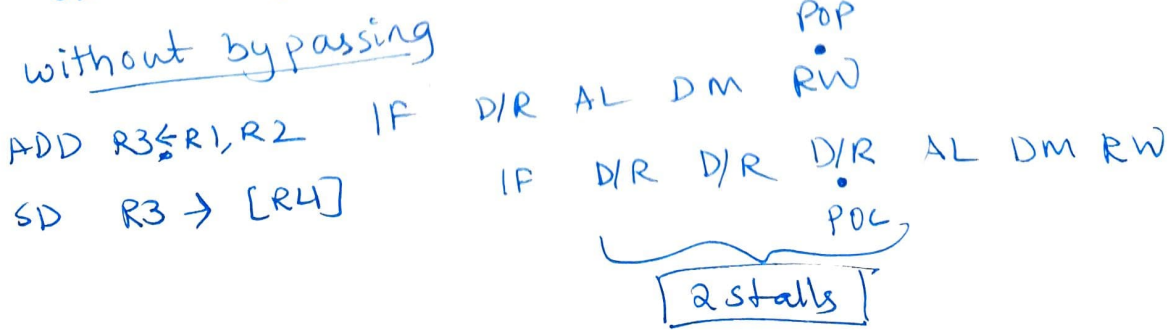


with bypassing



4) ADD R3 ← R1, R2
SD R3 → [R4]

without bypassing



with bypassing

ADD R3 ← R1, R2 IF D/R AL POP• DM RW

SD R3 → [R4] IF D/R AL • DM RW
POC.

1 0 stall