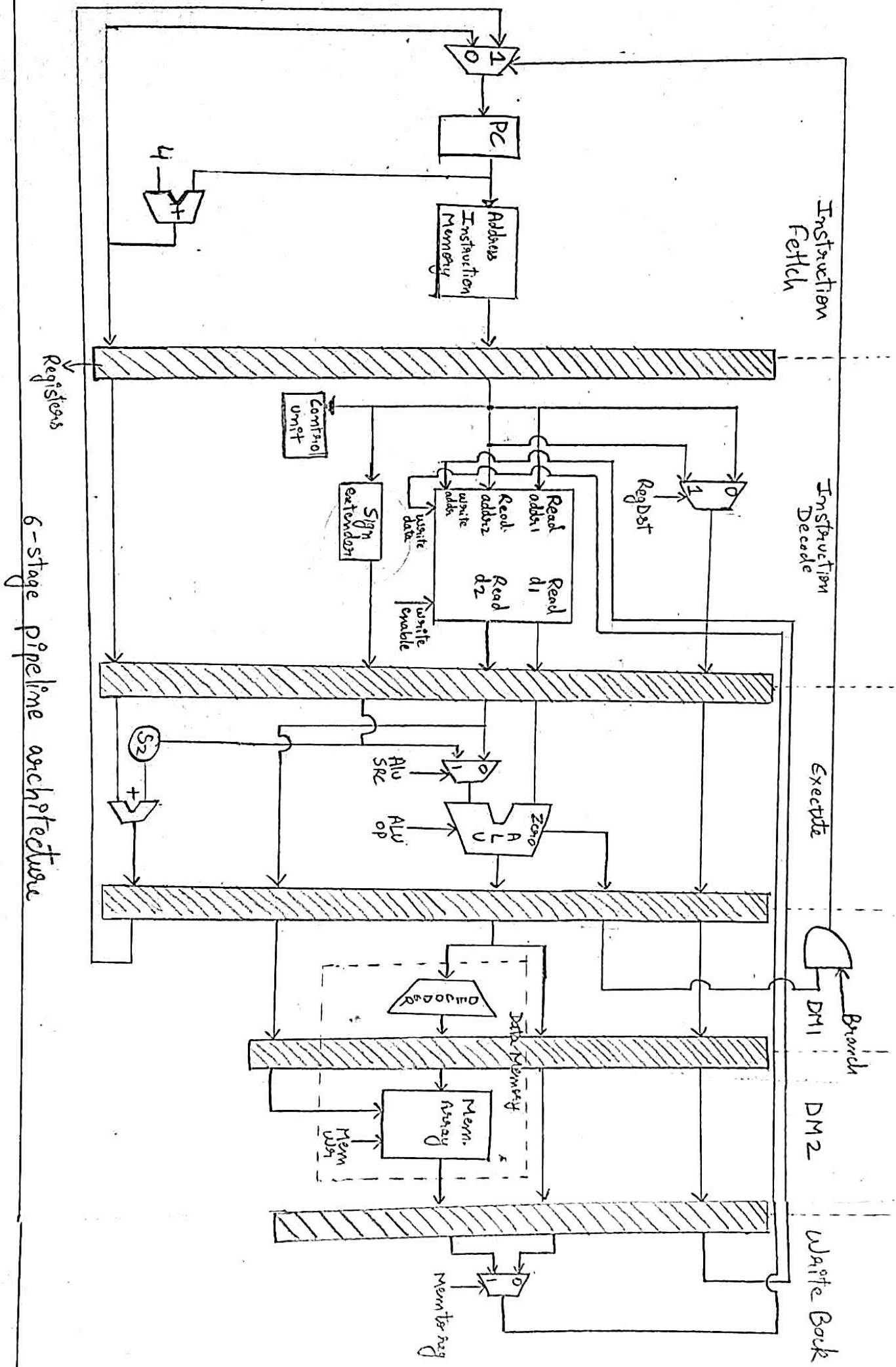
CS321 ESE TANISHQ MALU. 1901CS63



6-stage pipeline architecture

Pipelining is a process/technique that helps in implementing a form of instruction level parallelism. In this technique, multiple instructions are overlapped during execution.

The above schematic is an implementation of 6 stage pipelined architecture where data memory is divided into 2 stages DM_1 and DM_2. Pipelining increases the overall throughput

(b) Yes, the performance may increase on dividing the data memory into DM_1 and DM_2, given the circumstance that memory stage is major time defining step.

As we know that once the pipeline is full, one instruction is completed every cycle. Let us assume for our circuit $\underline{CPI = 1}$. (for large no of instruction)

∴ Total time for executing a set of instruction

$$= \text{Number of cycles} \times \text{time period}$$

$$= (n+k-1) \times T$$

$$\approx n \times T$$

If the data memory stage is the most time consuming step, then it will determine time period "T" of the circuit. Therefore, by dividing data memory into 2 parts, we can speed up our performance.

(P.T.O)

On dividing, DM_1 only decodes and provide address to memory array where the data is stored while DM_2 reads and outputs the data. Let time required by original data memory be "t_1", DM_1 be "t_2" and DM_2 be "$(t_1 - t_2) = t_3$".

Therefore the time required will change from "t_1" to "$\max(t_2, t_1 - t_2)$". This can considerably improve the throughput

For ex:- let us suppose that

$t_1 = 50$ ns

Time for (DM_1) $t_2 = 15$ ns

Time for (DM_2) $(t_1 - t_2) = t_3 = 35$ ns

Suppose Time for all other stage is 35ns, then the new time period will be $\max(35, 15) = 35$ ns.

$\therefore$ Initial execution time $= n \times 50$

after dividing into 2 stages $= n \times 35$

$\therefore$ $35n < 50n$

Thus performance increased considerably.

(P.T.O.)

Q2

Instruction set:-

1) lw ($3), 0($7)
2) add $7, ($3), $5
3) Sub $8, $5, ($3)

IF	ID	EX	DM_1	DM_2	WB		
	IF	ID	EX	DM_1	DM_2	WB	
		IF	ID	EX	DM_1	DM_2	WB

Hazard will occur since value of $3 will be accessed by add and sub before it is updated.

(a) No forwarding

	No. of cycles → C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}
1. lw $3, 0($7)	IF	ID	EX	DM_1	DM_2	WB					
2. add, $7, $3, $5		IF	ID	ID	ID	ID	EX	DM_1	DM_2	WB	
3. sub $8, $5, $3			IF	IF	IF	IF	ID	EX	DM_1	DM_2	WB

No. of cycles required when no forwarding = 11

Instruction "ID" of add operation had to wait until the value of $3 gets write back. Similary "IF" of sub will wait until ID is not free.

(b.) With forwarding

	No. of cycles C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	
lw $3, 0($7)	IF	ID	EX	DM_1	DM_2	WB					
add $7, $3, $5		IF	ID	ID	ID	EX	DM_1	DM_2	WB		
sub $8, $5, $3			IF	IF	IF	ID	EX	DM_1	DM_2	WB	

No. of clock cycle required = 10

"ID" stage of add instruction will wait until the value of 0($7) is fetched from DM2. and forwarded to it. Thus add can access value of $3 one stage before WB, thus it require 1 cycle less.