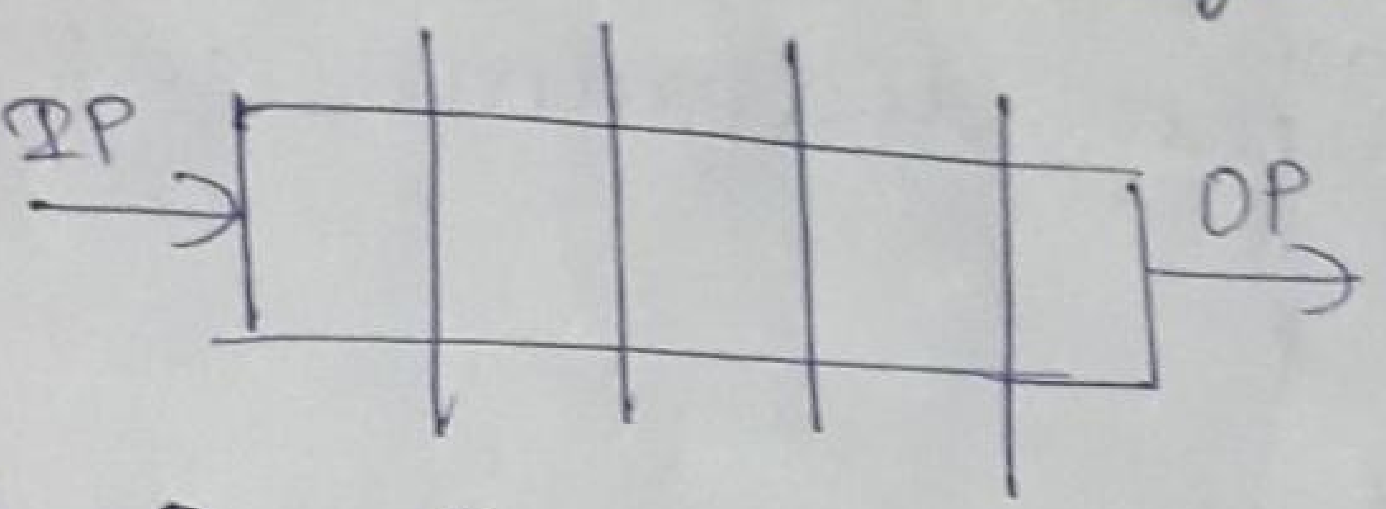
→ To enhance CPU performance

methods:

- * faster circuits
- * new hardware so as to increase performance

But these methods could be costly & difficult to implement

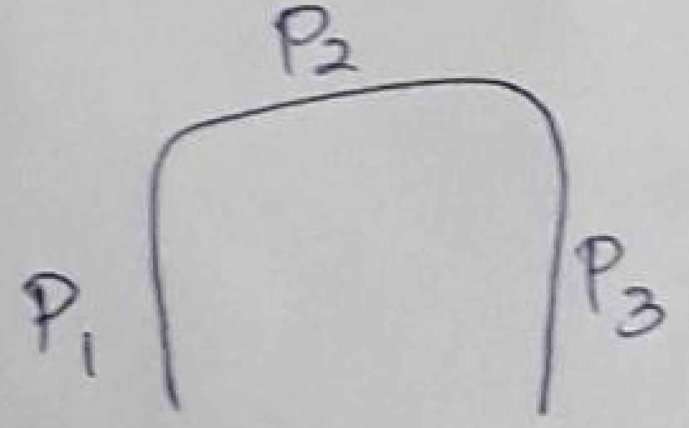
So we use pipelining.



(rough)

eg: P_1, P_2, P_3 ⇒ processes

C_1, C_2, C_3 → customers

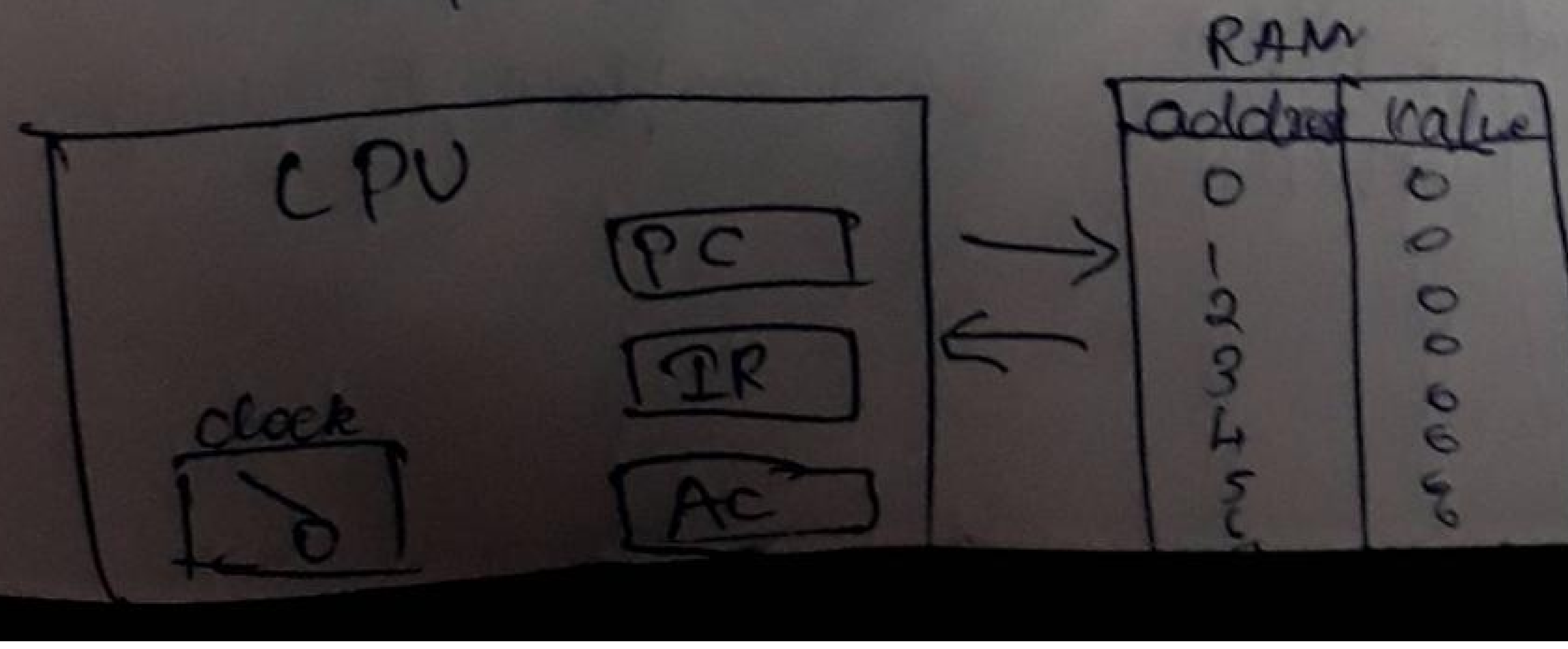


Time	1	2	3	4	5
C_1 →	P_1	P_2	P_3		
C_2 →	—	P_1	P_2	P_3	
C_3 →	—	—	P_1	P_2	P_3

no pipelining:

Time	1	2	3	4	5	6	7	8	9
C_1 ⇒	P_1	P_2	P_3						
C_2 ⇒	-	-	-	P_1	P_2	P_3			
C_3 →	-	-	-	-	-	-	P_1	P_2	P_3

Pipelining → Time ↓



Definition:

*Process of arrangement of existing/new hardware elements of CPU such that overall performance is increased.

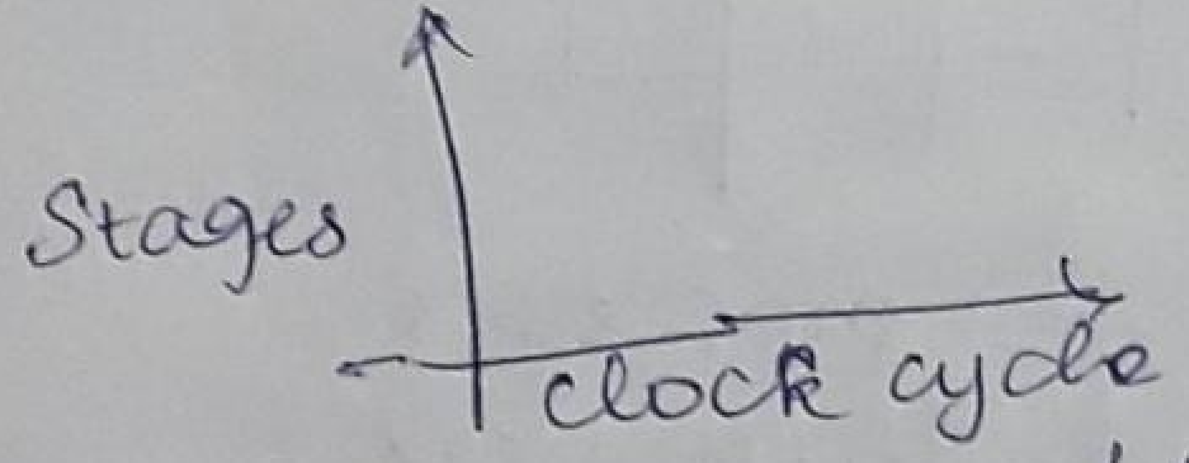
* Simultaneous execution of more than 1 instruction in the processor.

*overlapping instructions

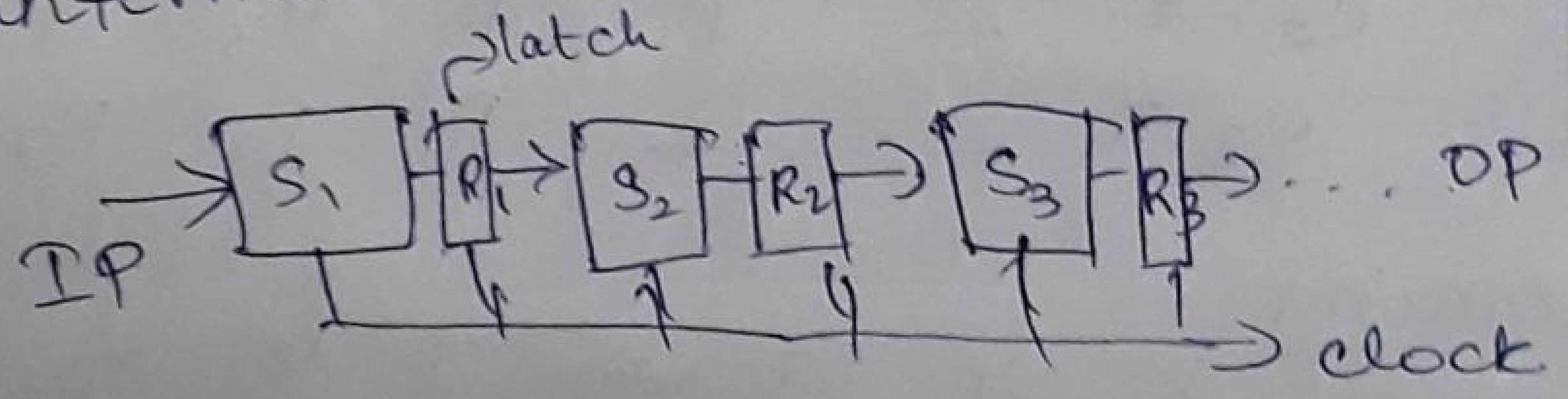
stages : parallel instructions take place

* To assess how many stages ⇒ we use RISC (reduced instruction set architecture)
↳ 5 stages

* Space time diagram : real time realization



* interface registers / latches store intermediate results



PIPELINING :

5 stages:

* IF
* ID
* EX
* MEM
* WB

Instruction fetched and stored in register, then we decode (calculated EA) using addressing mode we find operand, we perform this operation in ALU, execute it & write back it.

eg: If we have P = 8 I
P → program
8 I → 8 instructions (I_1 – I_8)

each instruction has to complete 5 stages

* non pipelined: 1 Stage = 1cc [1 cc = 1 stage = ⎍]

5 stages = 5 clock cycle [⎍⎍⎍⎍⎍]

so

8 × 5cc = 40 clock cycles

pipelined

* Draw space time / phase time diagram

	1	2	3	4	5	6	7	8	9	10	11	12
S1	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8				
S2		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8			
S3			I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8		
S4				I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	
S5					I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8

Time

OP

k = no of stages

n = no of instructions

Total CC = k + (n − 1)

= 5 + (8 − 1)

= 12 CC

CPI ≃ 1 (clock per instruction)

↳ If k = 5, n = 1000

eg: CC = 5 + (999) = 1004

$CPI = \frac{1004}{1000} \simeq 1$

} → This is practically not possible

Speedup: $\frac{\text{non pipelined}}{\text{pipe lined}} = \frac{40}{12} = 3.333$

efficiency / utilization : $\frac{\text{utilized boxes}}{\text{Total boxes}} = \frac{40}{60} = \frac{2}{3}$

Scanned by TapScanner

Hazards in pipelining

* In pipelining, $\boxed{CPI \cong 1}$, but its tough to achieve

* The hazards cause this difficulty which causes delay

- Data
  - RAW (true dependency)
  - WAR (anti dependancy)
  - WAW (output dependancy),
- structural
- control.

Read After Write:

eg: $\overset{3}{R_2} \leftarrow \overset{2}{R_2} \overset{+}{+} \overset{1}{R_3}$

$\overset{5}{R_5} \leftarrow \overset{3}{R_2} \overset{+}{+} \overset{2}{R_4}$

	1	2	3	4	5	6
I_1	IF	OF	Ex	MEM	WB	
I_2		IF	OF	Ex	MEM	WB

* I_1 enters stage 1 & IF happens.
@ 2nd ms, I_1 does OF & I_2 does IF, @ 3rd ms I_1 does execution, I_2 does OF (but the value of R_2 must be the one after written back which is @ 5th ms).

Structural Hazard :-

* when multiple instructions use same resources

	1	2	3	↓ 4	5	6	7	8
I_1	IF	ID	EX	MEM	WB			
I_2		IF	ID ~~EX~~	EX ~~MEM~~	MEM ~~WB~~	WB		
I_3			IF	ID	EX	MEM	WB	
I_4				IF	ID	EX	MEM	WB

ID → to find location of operand

@ CC4, I1 loads & stores value in memory.

I4 fetches instruction from memory

* both instructions using same resource (memory)

* To prevent this we stall (1+ pip stall per ins), we create bubbles

* If we stall, CPI ≠1.

* In this case, the hazard occurs in vann-Neuman. If harvard, data & instructions are saved seperately so no prob.

Solution :

* Resources duplication

* use pipelining for resources also

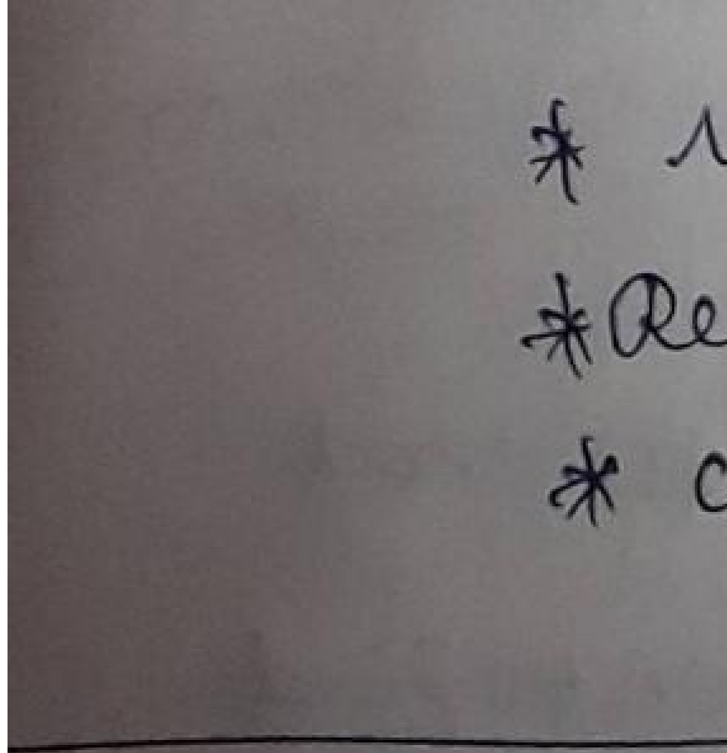
* Rename the resources

* change in ordering (I4 executed later)

Control Hazards:

* All instructions which change the PC leads to control hazard.

* branch conditions (eg: function call)

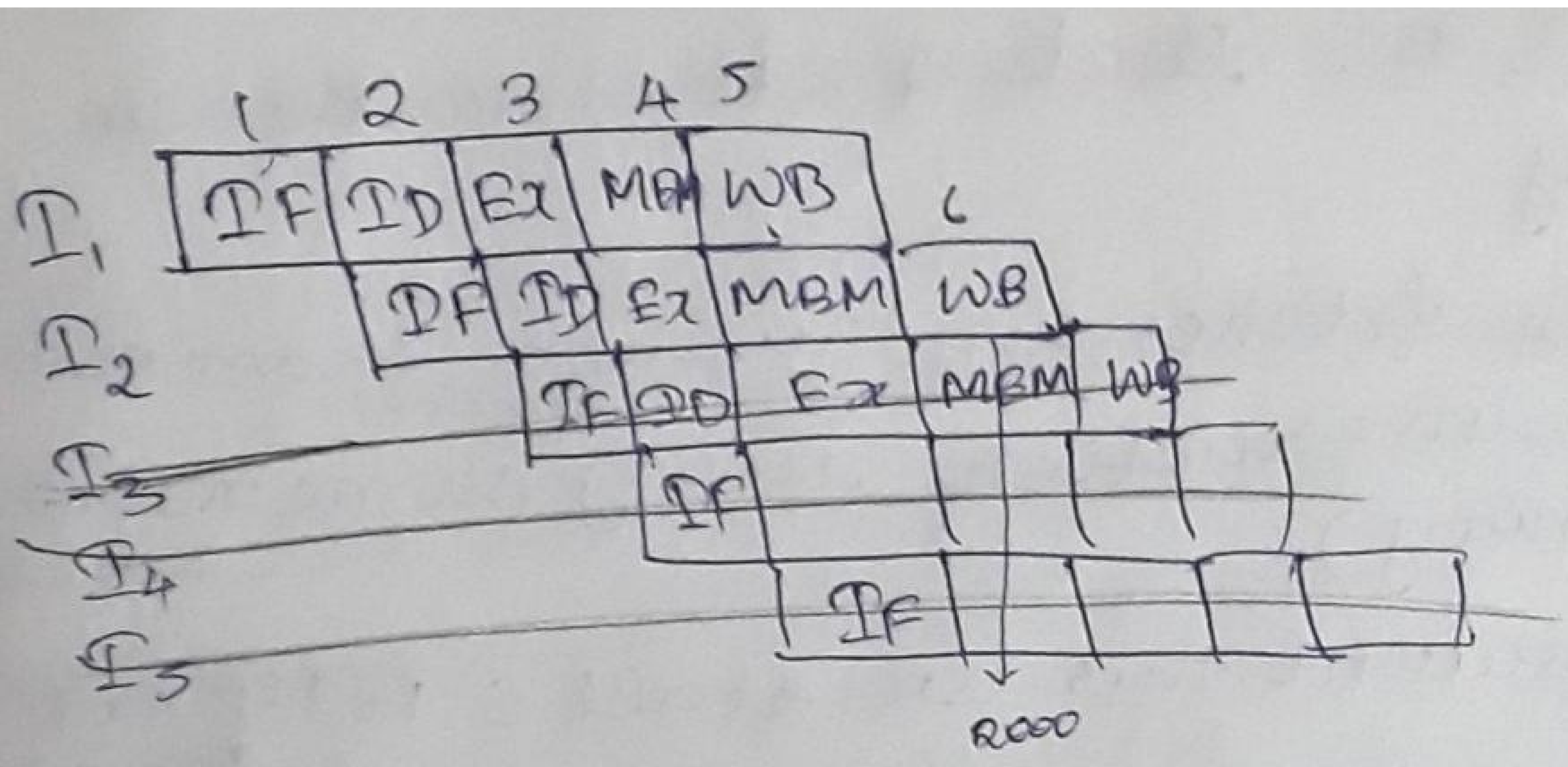


* When branch condition occurs, the sequential incrementation PC is broken.

eg: Bnez R1, 2000

↓
branch when R1 not equal to 0

$R_1 = 0$	no prob
$R_1 \neq 0$	branch (address 2000 ↓ data in 2000)



* consider, if I_2 ⇒ bnez R_1, 2000

* In I_2, (IF, ID, Ex, MEM happens) the value of add (2000) is written back in 6 th cc only.

* until then I_3, I_4, I_5 have already fetched instruction from PC (1008, 1012, 1016) This is now flushed (performance hampered)

Solution:

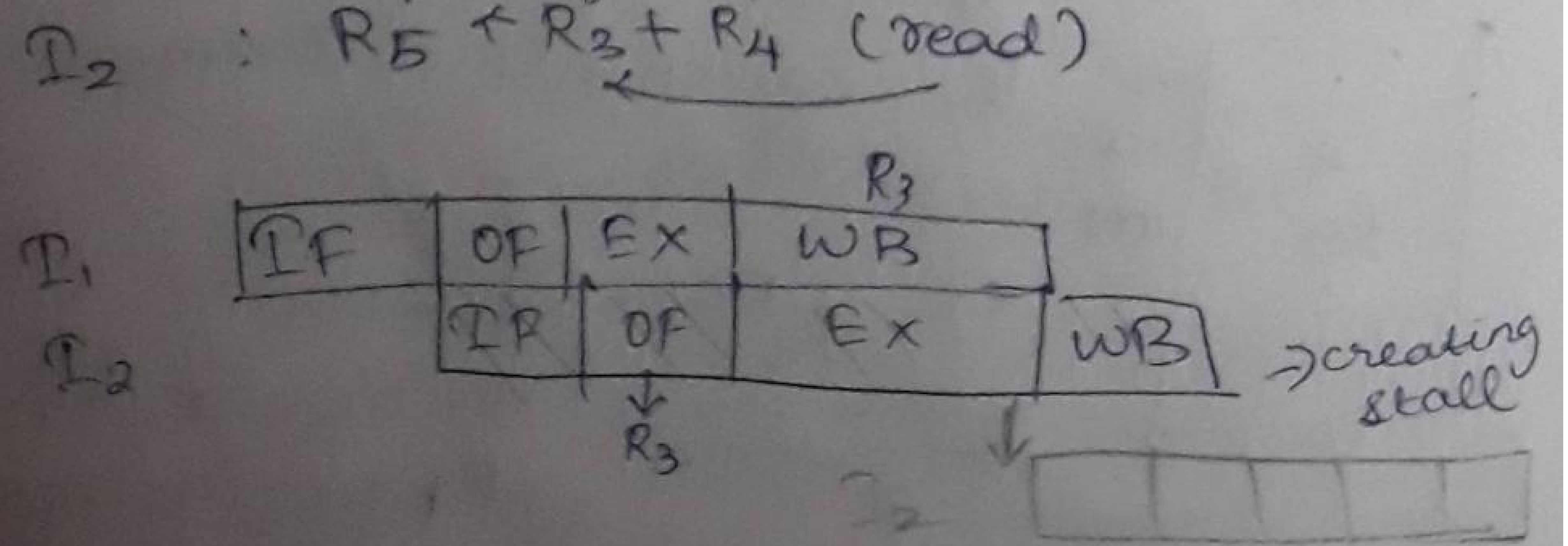
Stall:

* ALU performs Ex, so @ that time islf ALU would have identified that $R_1 \neq 0$. so we stall I_3 there islf.

Read After Write:

I_1 : $\overset{5}{R_3} \leftarrow \overset{2}{R_1} + \overset{3}{R_3}$ (write)

I_2 : $\overset{9}{R_5} \leftarrow \overset{5}{R_3} + \overset{4}{R_4}$ (read)



* data inconsistency

	~~IF~~ Source Domain	~~OF~~ dest Range
I_1	R_1 R_3	R_3
I_2	R_3 R_4	R_5

$$\boxed{I_1(\text{range}) \cap I_2(\text{Domain}) \neq \phi}$$

↳ * RAW data hazard

Write after Read hazard:

$I_1 : \overset{200}{R_1} \leftarrow \overset{\overset{(\text{read})}{10}}{R_2} * \overset{20}{R_3}$

$I_2 : \underset{\underset{(\text{write})}{70}}{R_2} \leftarrow \underset{30}{R_4} + \underset{40}{R_5}$

If, I_2 performs b4 I_1, WAR hazard occurs (generally doesnt occur in 4, 5 stage pipeline)

* occurs in auto increment addressing mode (any increment (write) happens first, then is read)

	Domain	Range
I_1	$R_2 R_3$	R_1
I_2	$R_4 R_5$	R_2

$$\boxed{\text{Domain}(I_1) \cap \{\text{Range}\}(I_2) \neq \phi}$$

↳ * ~~R~~ WAR hazard

Write after write:

$I_1 : \overset{200}{R_3} \leftarrow \overset{10}{R_1} * \overset{20}{R_2}$

$I_2 : \underset{100}{R_3} \leftarrow \underset{50}{R_4} + \underset{50}{R_5}$

* If delay (if * takes more time than +) (or) parallel instructions Concurrency occurs.

*correct value shld be 100, but due to ~~[illegible]~~ WAW, the val becomes 200 which is wrong

* doesnt generally occur in 4, 5 stage pipeline.

* occurs in parallel computing

	Domain	Range
I_1	$R_1 R_2$	R_3
I_2	$R_4 R_5$	R_3

$\boxed{\text{Range}(I_1) \cap \text{Range}(I_2) \neq \phi}$

WAW hazard

Addressing

* Addressing modes specify where an operand is located

* They can specity a constant, a register (or) memory location

* The actual location of operand is its EA

* certain addressing modes allow us to determine address of operand dynamically

Instruction types

* data movement
* arithmetic
* boolean
* bit manipulation
* I/O
* control transfer
* special purpose