- To detect the data - dependency condition some logic should be implemented in ID stage of the pipeline.

- Structure of a logic is -

Tomasulo Algorithm :-

- This algorithm is used to detect data - dependency in a pipeline. It work on ID stage of each Inst^n.

Eg:- I_1 : $r_0 \leftarrow r_1 + r_2$

I_2 : $r_3 \leftarrow r_0 * r_2$

(i) ID - Stage

(a) Inst^n decode

S.no	Func^n	Dest^n	Indep-endent Src1	Independ Src2	Depen-dent Src_1	Depend Sr_2	Status 0: EX not comple-ted	Status 1: EX comp-let-ed
I_1	ADD	r_0	r_1	r_2	—	—	0	
I_2	MUL	r_3	—	r_2	r_0	—	0	

(b) Access reg - file -

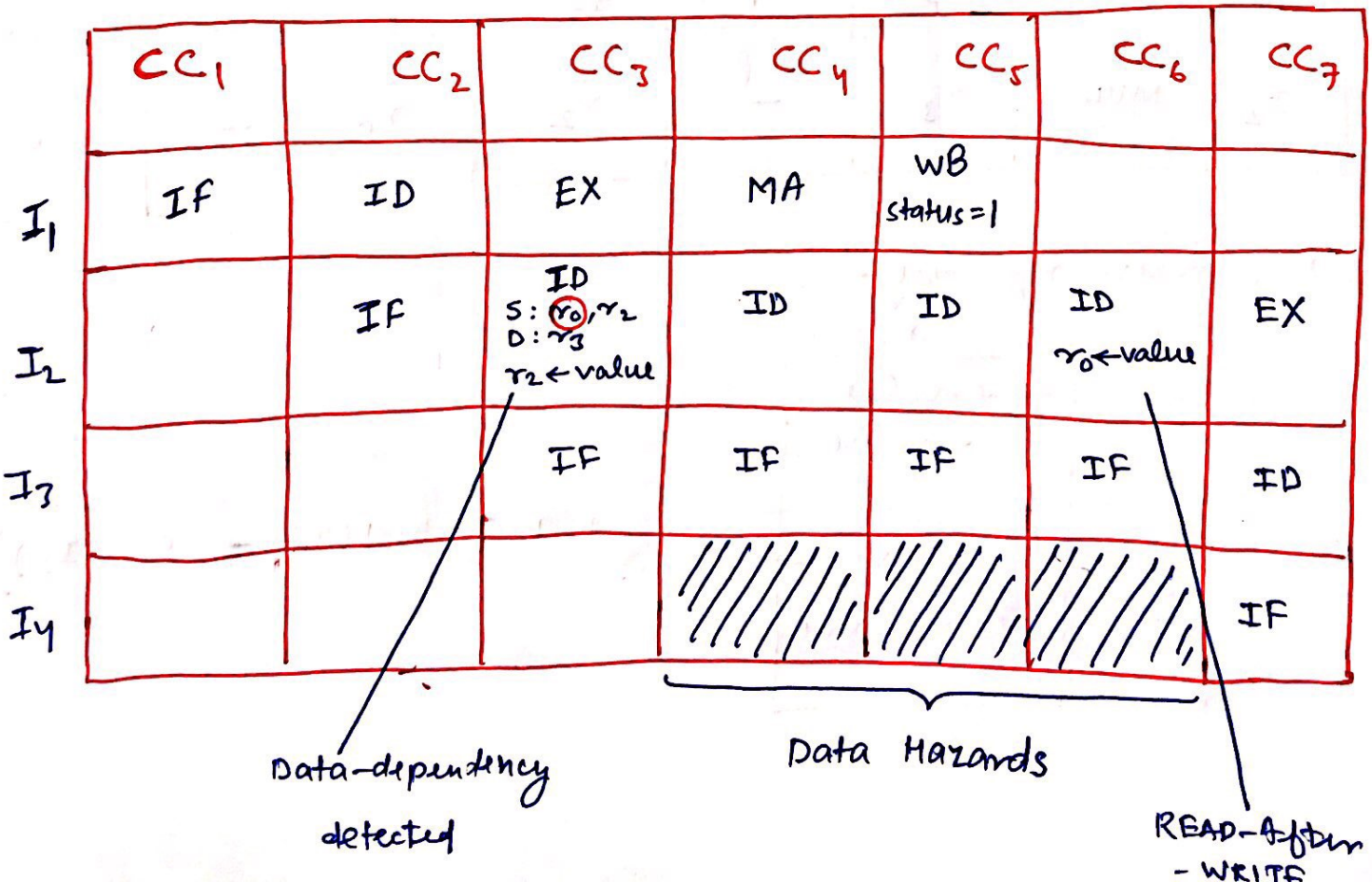
I_1	$r_1 \leftarrow$ value $r_2 \leftarrow$ value
I_2	$r_0 \leftarrow$ X $r_2 \leftarrow$ value

check (Dest(I_1) = Src(I_2))

(ii) EX - stage

I_1	EX - stage Allocated
I_2	EX - Stage is not Allocated (I_2 will be waiting on ID-stage until I_1 status becomes 1)

- By using this logic, we can stop the accessing of a unwanted data in the pipeline execution. Therefore, dependent Inst^n will wait on ID - stage until the data become available in register.



- To minimize the data-hazard in pipeline, H/w tech-niqne is used i.e operand forwarding also called as bypassing or short circuiting.

- Operand Forwarding techniques states that use the buffers b/w the stages to hold the intermediate result so that dependent inst^n will access the new value from the buffer before updating the register file. In this process last-stage op^n is modified to perform the i.e WRITE-READ op^n in the same cycle means updated data immediately available for READ access in the same cycle. Therefore buffer is not required at the end of last stage.

- Consider the following program code, execute in the pipeline using operand forwarding techniqne -

Code:-

I_1 : ADD r_0, r_1, r_2

I_2 : SUB r_3, r_0, r_2

I_3 : MUL r_4, r_3, r_0

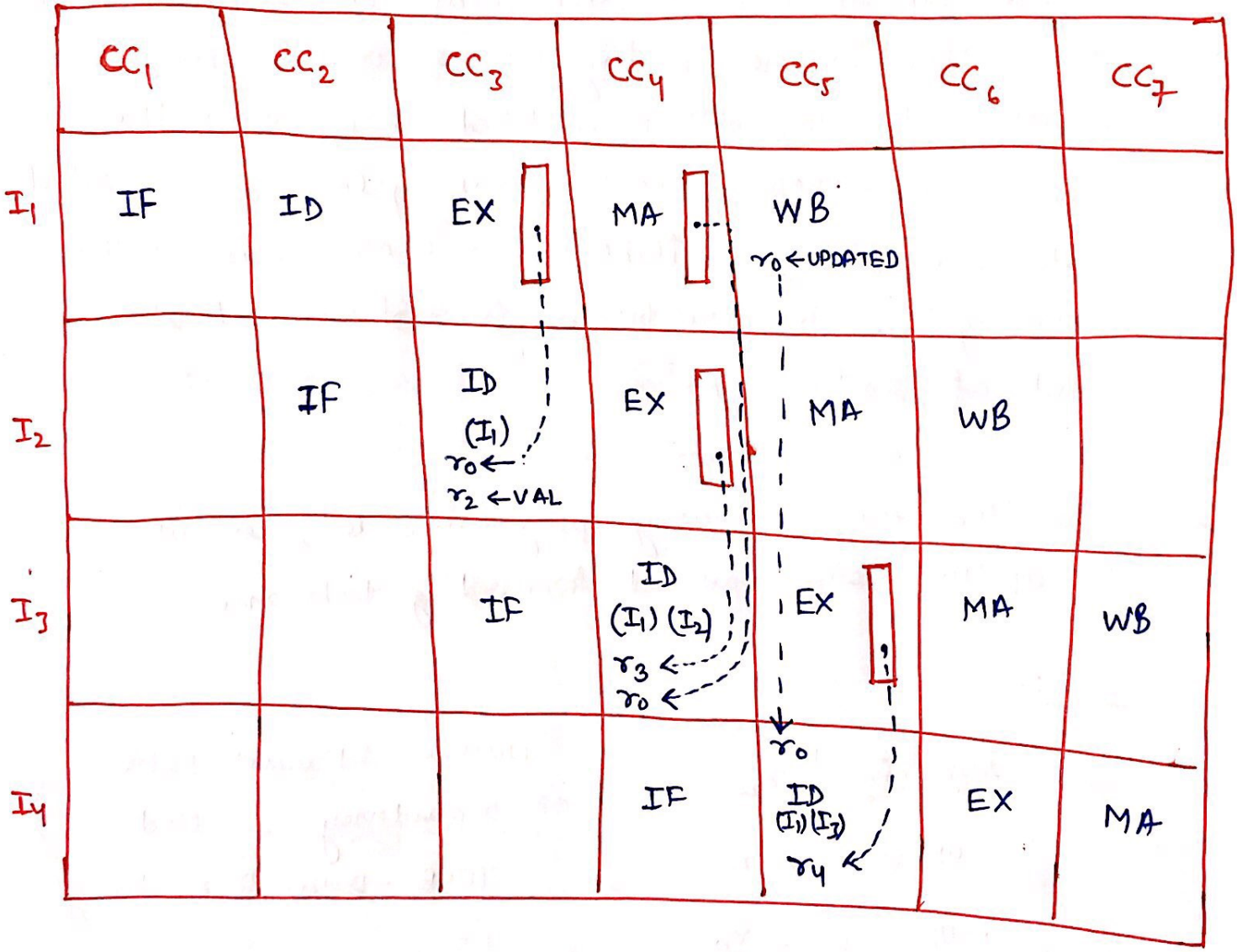
I_4 : DIV r_5, r_4, r_0

NOTE:- Adjacent Data-Dependency is called as TRUE-Data-Dependency i.e

$$\begin{bmatrix} I_2 - I_1 \\ I_3 - I_2 \\ I_4 - I_3 \end{bmatrix}$$

NOTE - Non-Adjacent Data dependency is data-dependency only. But it is eliminated through True-data-dependency

i.e $\begin{bmatrix} I_3 - I_1 & (r_0) \\ I_4 - I_1 & (r_0) \end{bmatrix}$



$$tp = \max\left(\begin{matrix}\text{stage} \\ \text{-Delay}\end{matrix} + \begin{matrix}\text{Buffer} \\ \text{Delay}\end{matrix}\right)$$

sns → 4ns → 1ns

Q: Consider following Program, execute on a RISC pipeline. How many cycles are required to complete the program. Assume that all the inst^n are spending 1 cycle on all the stages but LOAD inst^n takes 3-cycles on 4th stage of a Pipeline.

I_1 : LOAD r_0, 3(r_1)

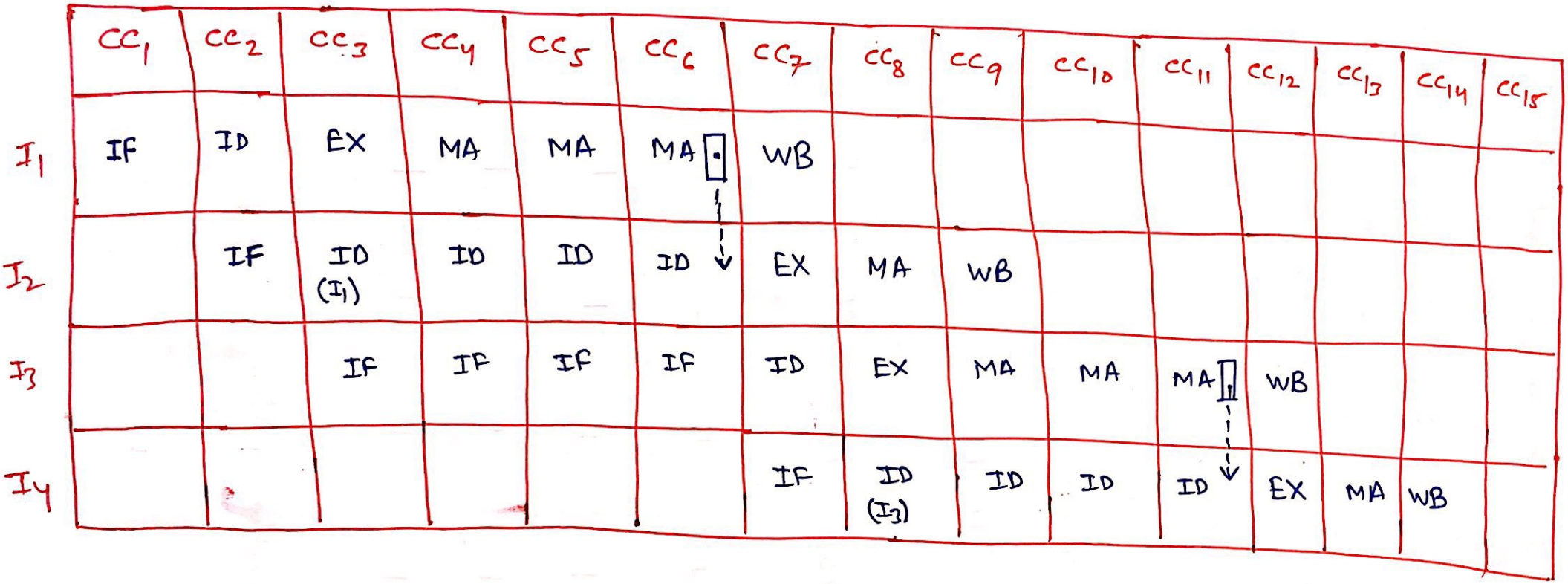
I_2 : ADD r_2, r_0, r_1

I_3 : LOAD r_3, 4(r_4)

I_4 : SUB r_5, r_3, r_4

	IF	ID	EX	MA	WB	
I_1	1	1	1	3	1	LOAD
I_2	1	1	1	1	1	ADD
I_3	1	1	1	3	1	LOAD
I_4	1	1	1	1	1	SUB

Now, generate reservation table—



* Since due LOAD opn data (dependent) will be present at 'MA' stage.

Q. Consider the following program executed on 4-stages (IF, ID, EX, WB) pipeline. Assume that all inst^n are spending 1-cycle on all the stages but 'MUL' Inst^n takes 4-cycle on EX stage and 'DIV' Inst^n takes 5-cycle at EX stage. Operand Forwarding mechanism is used as an optimization technique in the pipeline to handle the data depending.

(a) How many cycles are required to complete prog.

(b) How many cycles are saved using optimization over non-optimization.

I_1: MUL r_0, r_1, r_2

I_2: DIV r_3, r_1, r_2

I_3: ADD r_4, r_3, r_1

I_4: SUB r_3, r_4, r_2

	IF	ID	EX	WB	
I_1	1	1	4	1	MUL
I_2	1	1	5	1	DIV
I_3	1	1	1	1	ADD
I_4	1	1	1	1	SUB

With optimization (operand-Forwarding)

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6	CC_7	CC_8	CC_9	CC_{10}	CC_{11}	CC_{12}	CC_{13}	CC_{14}	CC_{15}
I_1	IF	ID	EX	EX	EX	EX	WB								
I_2		IF	ID	ID	ID	ID	EX	Ex	Ex	Ex	Ex	WB			
I_3			IF	IF	IF	IF	ID (I_2)	ID	ID	ID	ID ↓	EX	WB		
I_4							IF	IF	IF	IF	IF	ID ↓	Ex	WB	

(Total cycle = 14)

without optimization (no-operand Forwarding)

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6	CC_7	CC_8	CC_9	CC_{10}	CC_{11}	CC_{12}	CC_{13}	CC_{14}	CC_{15}	CC_{16}	CC_{17}	CC_{18}	CC_{19}
I_1	IF	ID	EX	EX	EX	EX	WB												
I_2		IF	ID	ID	ID	ID	EX	~~WB~~ EX	EX	EX	EX	WB							
I_3			IF	IF	IF	IF	ID (I_2)	ID	ID	ID	ID	ID	ID	EX	WB				
I_4							IF	IF	IF	IF	IF	IF	IF	ID (I_3)	ID	ID	EX	WB	

(Total cycles = 18)

Cycle - saved = 18 − 14

= 4 - cycles

(iii) Control Dependency :-

- It occurred in the pipeline when the transfer of control inst^n are executed in the pipeline.

Code:-

```
1000 : I1          Falling/Fall through path
1001 : I2
1002 : I3 ( JMP 2000)
1003 : I4
  .      .
  .      .
2000 : BI1         Taken path
2001 : BI2
  .      .
```

Expected o/p sequence –

$(I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow BI_1 \rightarrow BI_2 \cdots)$

'IF' Mprog is –

T_1 : PC → MAR

T_2 : M[MAR] → MBR
PC + I → PC

T_3 : MBR → IR

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6
PC:1000 I_1	IF PC:1001	ID	EX	MA	WB	
I_2		IF PC:1002	ID	EX	MA	WB
I_3			IF PC:1003	ID PC:2000	EX	MA
I_4				IF PC:1004	ID	EX
PC:2000 BI_1					IF PC:2001	ID
BI_2						IF PC: 2002

IF of I_4 make PC ← 1004 but during ID of I_3 PC ← 2000 (JMP Inst^n)

Actual o/p seq - $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow I_4 \rightarrow BI_1 \rightarrow BI_2$

I_4 → unwanted - Instⁿ

- In the above exec. sequence, unwanted Instⁿ is executed in the pipeline. So program output is missing (unwanted behaviour). This situation is known as control-dependency.
- To handle such problems, FLUSH opⁿ or FREEZE opⁿ is used. This opⁿ states that, insert the NOP instⁿ after the JMP instⁿ in the program to suspend the unwanted instⁿ fetch.

Code:-

```
1000: I1
1001: I2
1002: I3 (JMP 2000)
1003: I4 (NOP)
1004: I5
1005: I6
  ⋮    ⋮
2000 : BI1
2001 : BI2
  ⋮    ⋮
```

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6
PC:1000 I_1	IF PC:1001	ID	EX	MA	WB	
I_2		IF PC:1002	ID	EX	MA	WB
I_3			IF PC:1003	ID PC:2000	EX	MA
NOP				IF PC:1004	ID	EX
PC:2000 BI_1					IF PC:2001	ID
BI_2						IF PC:2002

Actual - o/p - seq : $I_1 \rightarrow I_2 \rightarrow I_3 \rightarrow$ NOP $\rightarrow BI_1 \rightarrow BI_2$

NOP → unwanted Instⁿ → stall

- Number of stalls created in the pipeline due to a branch inst^{n} is called as branch penality.

- It depends on the availability of target address in the pipeline i.e branch penality is equal to # at what stage target Address available − 1.

(branch penality → stage in which target Address available − 1)

NOTE:- In RISC pipeline branch penality is 1 as target address is available in 2nd (ID) stage.

NOTE:- For a hypothetical K-stage pipeline, branch penality is K-1.

- if stage number of TA^{given} :-

(Branch-penality = stage number − 1)

- if stage name of TA^{given} :-

(Branch-penality = corresponding stage number − 1)

- until the Inst^n is completed (all Inst^n are proceed through all stages) -

$$\text{Branch penality} = \text{Last stage in pipeline} - 1$$

NOTE:- Total no. of stalls created from the branches during the prog exec^n is -

$$\text{Branch freq} * \text{Branch penality}$$

Branch Inst^n/prog

stalls / branch inst^n

NOTE :- To minimize the control hazard (branch penality), h/w technique is used i.e branch prediction butter also called as branch target butter or loop butter.

- Branch-prediction butter is a high-speed butter present in IF(stage) used to hold the predicted target address. when target address present in first stage then there is not stall in pipeline.

- NOTE :- to minimize the control hazard, S/W mechanism is used i.e delayed branch.

Delayed Branch :- It is a compiler technique, so compiler will rearrange the code if possible, otherwise substitute the NOP inst^n after JMP inst^n to preserve the execution path.

User-code

1000 : I_1
1001 : I_2
1002 : I_3 (JMP 2000) — SWAP
1003 : I_4
⋮
2000 : BI_1
2001 : BI_2

COMPILER - M/c code

(a) Rearrangement

Addr : I_1
I_3 (JMP BI_1)
I_2
I_4
⋮
BI_1
BI_2
⋮

PC :	CC_1	CC_2	CC_3	CC_4	CC_5
I_1	IF PC : I_3	ID	EX	MA	WB
I_3		IF PC : I_2	ID PC : BI_1	EX	MA
I_2			IF PC : ~~I_4~~	ID	EX
BI_1				IF PC : BI_2	ID
BI_2					IF PC : BI_3

O/P - sequence = $I_1 - I_3 - I_2 - BI_1 - BI_2$

- O/p sequence is different but program result will be same, and there is no STALL.

NOTE:- Instⁿ ~~swapping~~ rearrangement may be possible in unconditional branch opⁿ but no rearrangment can be take place in case of conditional branch Instⁿ.

(b) NOP- substitution -

M/c generated-

Addr: I_1
I_2
I_3 (JMP → BI_1)
NOP
I_4
⋮
BI_1
BI_2

O/p sequence -

$I_1 - I_2 - I_3 -$ NOP $- BI_1 - BI_2$

Instruction Scheduling:-

- CPU always execute the program in a sequence from I_1 to I_n called as in order execution.

- In the in-order execution sequence, if any instⁿ is dependent then remaing instⁿ are also sharing the STALL cycle even they are independent.

I_1 : ADD r_0, r_1, r_2

I_2 : SUB r_3, r_0, r_4

I_3 : MUL r_4, r_5, r_6

I_4 : DIV r_3, r_7, r_8

In-order execution:-

$I_1 - I_2 - I_3 - I_4$

stalls

- In the above execution sequence, I_2 dependent on I_1 so I_2 waiting until the I_1 execution is completed. Waiting creates stall. These stalls are shared by I_3 and I_4 inst^n even they are independent.

- To handle the above problem inst^n scheduling concept is used. This concept states that execute the independent inst^n first called out of order (re-order) of inst^n execution.

$$(I_1 - I_3 - I_4 - I_2)$$

- Re-order execution creates two more dependencies in the pipeline.

(i) Anti- data dependency

(ii) Output data- dependency

Inst^n J **Inst^n I**

READ - before - WRITE

WRITE - before - READ

WRITE - before - WRITE

three types of hazard

- Anti data dependency will be occurred when Inst^n J WRITE the data before Inst I reads it. eg- I_3 exec before I_2, so I_3 updates the register r_4 before I_2 reads it ∴ I_2 incorrectly access new value from reg (r_4).

- O/p data-dependency will be occurred when the Inst^n J tries to WRITE the data before Inst^n I WRITE it. eg- I_4 exec before I_2, so I_4 updates the register (r_3) before I_2 WRITES it ∴ destination (r_3) updates with a old value (Data Loss)

NOTE:- To minimize the anti & O/p data-dependency & stalls H/w mechanism is used i.e register renaming.

- Register renaming states use re-order buffers to store the out-of-order inst^n o/p. Later update the reg-file with a re-order buffer contents after completion of dependent inst^n execution.

$I_1 \rightarrow r_0$

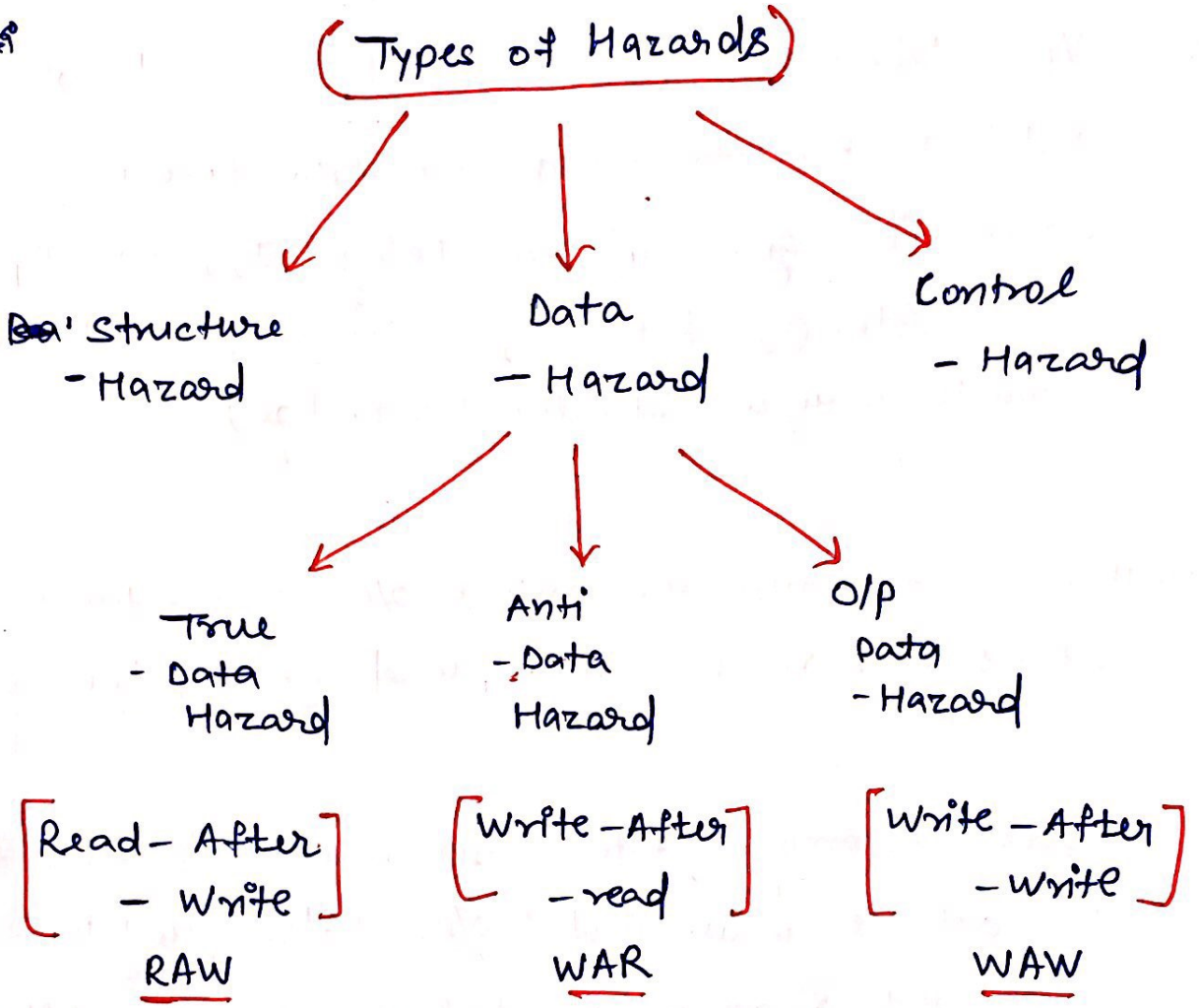
I_3, I_4 — Re-order Buffer

$I_2 \rightarrow r_4 3$ (STATUS = 1)

↳ re-order buffer → Reg-file {r_4, r_3}

Hazard :-

- Hazard is a delay.
- Delay is present in the pipeline due to a dependency problem.
-

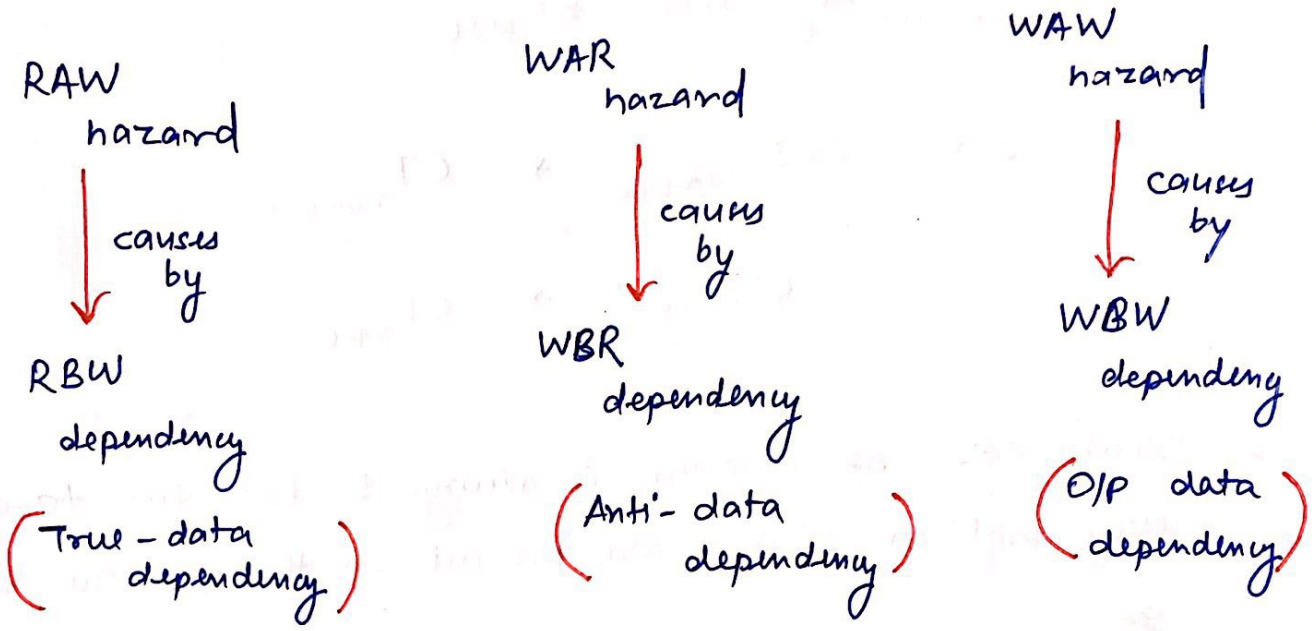


it is created when the inst^{n} J tries to READ the data before inst^{n} I WRITES it. (RBW).

it is created when inst^{n} J tries to WRITES the data before inst^{n} I READS it (WBR).

it is created when inst^{n} J tries to WRITE data before inst^{n} I WRITE it (WBW)

NOTE:- Read-After-Read (RAR) is not a hazard as Read-before-Read (RBR) is not any dependency problem.



Inst^{n} J	Inst^{n} I	Type of Dependency	Type of Hazards
I/p reg	O/p reg	True Data	RAW (in-order)
O/p reg	I/p reg	Anti-Data	WAR (out-order)
O/p reg	O/p reg	O/p Data	WAW (out-order)
I/p reg	I/p reg	—	—
ALU	ALU	Structural	Structural Hazard
JMP		control	control Hazard

Performance Analysis with stalls :-

$$S = \frac{\text{Avg. Inst}^{n} \text{ ET}_{\text{non-pipe}}}{\text{Avg. Inst}^{n} \text{ ET}_{\text{pipe}}}$$

$$S = \frac{CPI_{\text{non-pipe}} * CT_{\text{non-pipe}}}{CPI_{\text{pipe}} * CT_{\text{pipe}}}$$

- Ideal CPI of pipeline is always 1 but due to dependency problem stalls are present in the pipeline so

$$\left[S = \frac{CPI_{\text{non-pipe}} * CT_{\text{non-pipe}}}{\left(1 + \text{stalls}/\text{Inst}^{n}\right) CT_{\text{pipe}}} \right]$$

- when the pipelines stages are perfectly balanced, then 1-task ET is also equal to no. of stages in the pipeline (i.e $t_n = k \cdot t_p$). under this condition

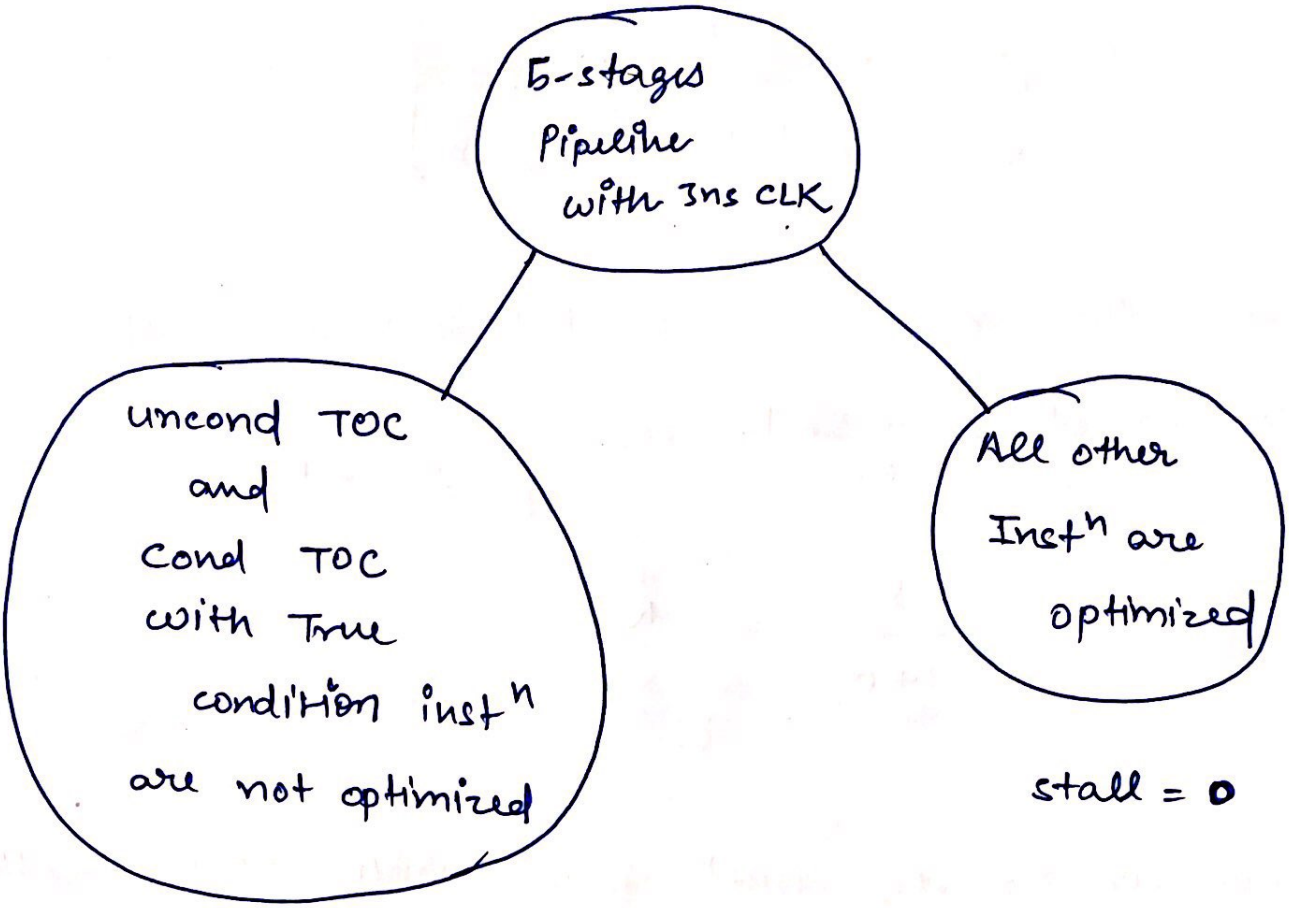
$$S = \frac{k \, t_p}{\left(1 + \text{stalls}/\text{Inst}\right) t_p}$$

$$S = \frac{K}{1 + \text{stalls}/\text{Inst}^n}$$

- when the system is operated with 100 % efficiency then no stalls present. So,

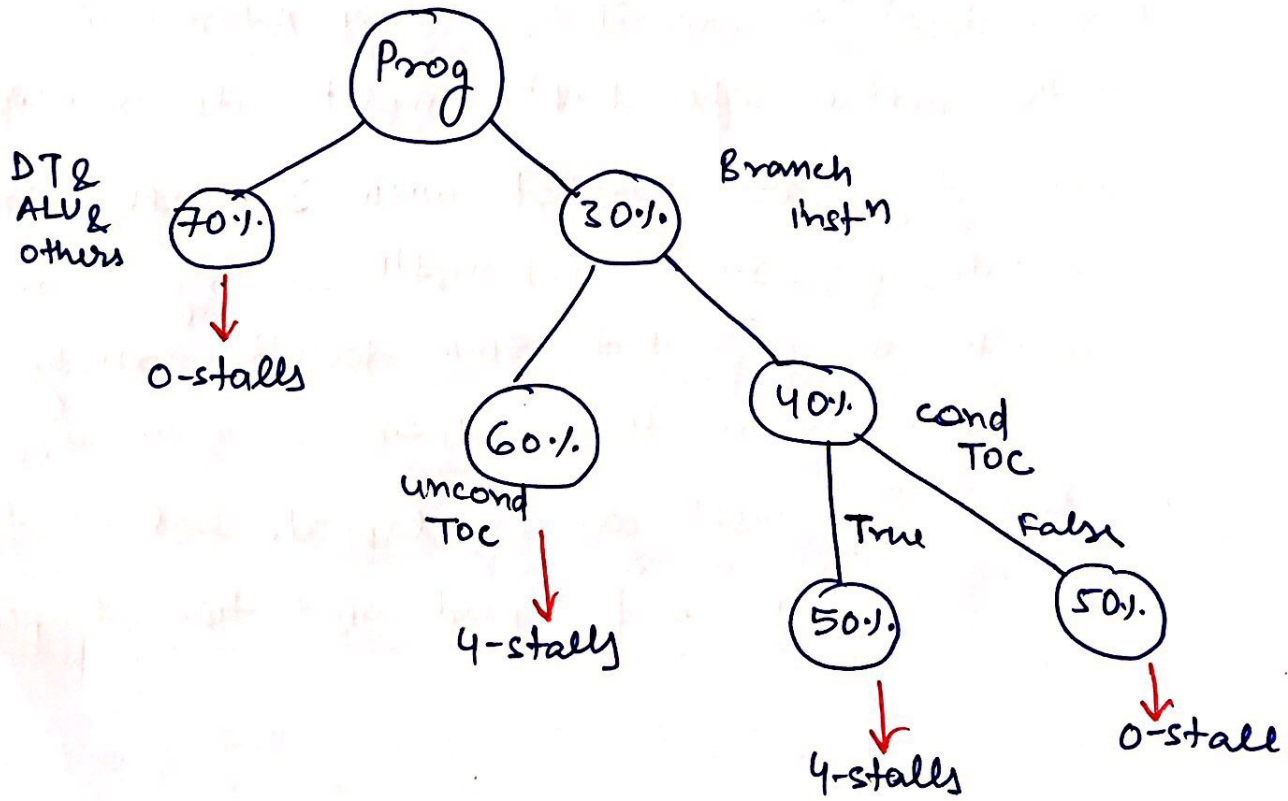
$$S = \frac{K}{1+0} = K$$

Q. Consider 5- stage inst^n pipeline which allows all the inst^n except branch inst^n. Processor stops the fetching of a following inst^n after the branch inst^n until the branch inst^n is completed. Target Address is available in the pipeline when Inst^n completes its execution. Pipe-line stages are operated with 3ns clk. Program contains & 30-% branch inst^n among them 40 % are conditional in -which 50-% doesn't satisfy the condition when the condition is false then the following inst^n are overlapped. Calculate the av. inst^n ET and speed-up factor of pipe-line.



stalls = K-1 = 5-1

= 4



$$\text{stalls / Inst}^n = (0.7 * 0) + (0.3 \times 0.6 \times 4) + (0.3 \times 0.4 \times 0.5 \times 4) + (0.3 \times 0.4 \times 0.5 \times 0)$$

$$= 0.96$$

(a) $$\text{Avg. inst}^n \text{ ET}_{pipe} = \left(1 + \text{stall/Inst}^n\right) \text{CT}_{pipe}$$

$$= (1 + 0.96)\ (3ns)$$

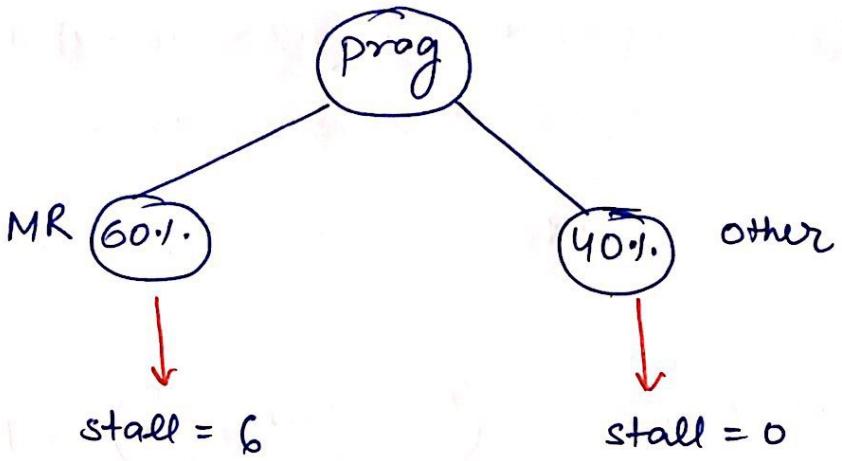
$$= 5.88ns$$

(b) $$\text{Performance gain } S = \frac{K}{1 + \text{stall/Inst}^n}$$

$$= \frac{5}{1 + 0.96}$$

$$(S = 2.55)$$

Q. Consider 8-stage pipeline operated with 4ns clk, which allows all the inst^n except the memory refrence inst^n. MR inst^n penality is 6-cycles. Prog contain 60% MR inst^n what is avg. inst^n execution time.



$CT_{pipe} = 4ns$

Avg. stalls / Inst^n = $(0.6 \times 6) + (0.4 \times 0)$

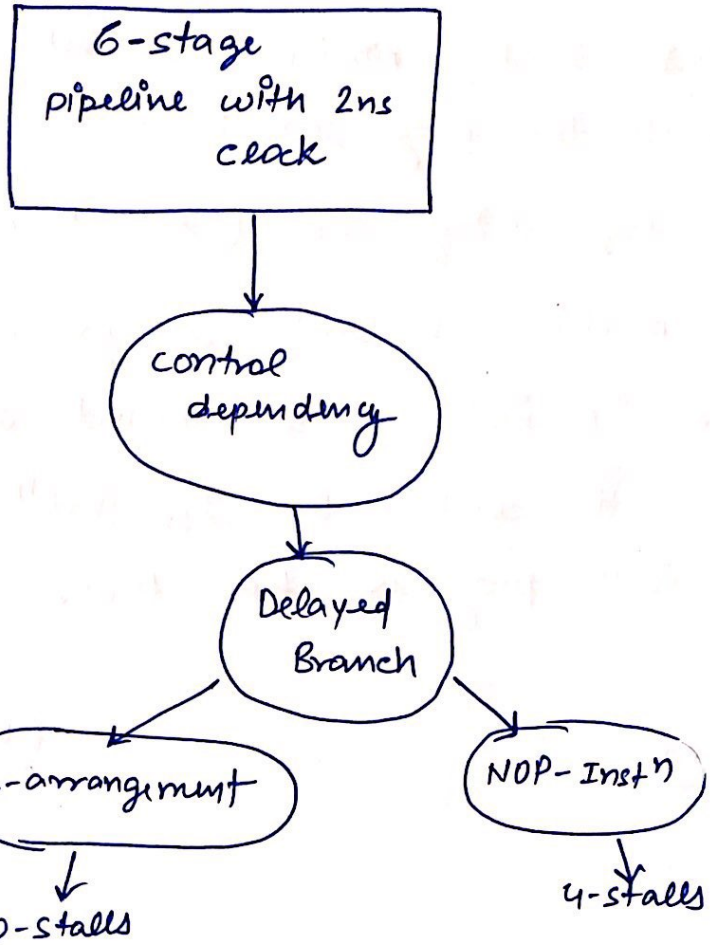
= 3.6

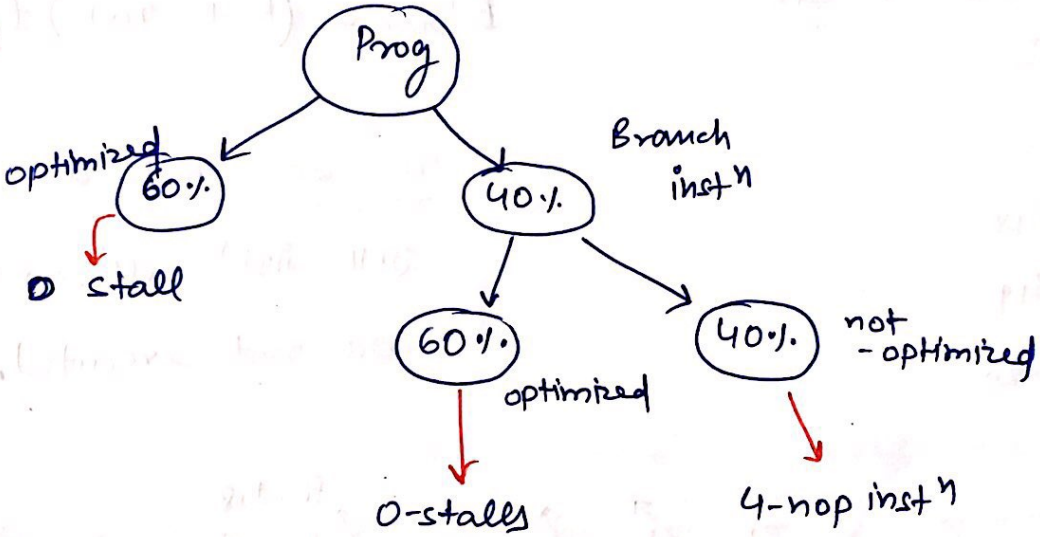
Avg. inst^n ET_{pipe} = $\left(1 + \text{stalls}/_{\text{Inst}^n}\right) CT_{pipe}$

= $(1+3.6)(4ns)$

= 18.4ns

Q. Consider 6-stage pipeline with 2ns clk uses, delayed branch concept to optimize the control dependency stalls. In the delayed branch re-arrangement process is used to avoid the stalls, if not possible to re-arrangement then 4 NOP inst^n are substituted after the branch inst^n in the program. Program contain 40% branch inst^n among them 60% are optimized, what is the av. inst^n ET.





$$\text{stalls/inst}^n = (0.4 \times 0.4 \times 4)$$

$$= \underline{0.64}$$

$$\text{avg. inst}^n \text{ ET}_{\text{pipe}} = (1 + \text{stalls/instn})\ tp$$

$$= (1 + 0.64)(2ns)$$

$$= \boxed{3.28ns}$$

Q. Consider 5-stage pipeline without branch-prediction, use to execute the prog which contain 20 inst^n ($I_1 - I_{20}$).

Pipeline stage delays are (2ns, 4ns, 8ns, 3ns, 1ns).

In the pipeline all inst^n are proceed through all stages. I_4 inst^n is a uncond JMP inst^n, it transfer the control to I_{18} inst^n during its execution. What is the prog execution time.

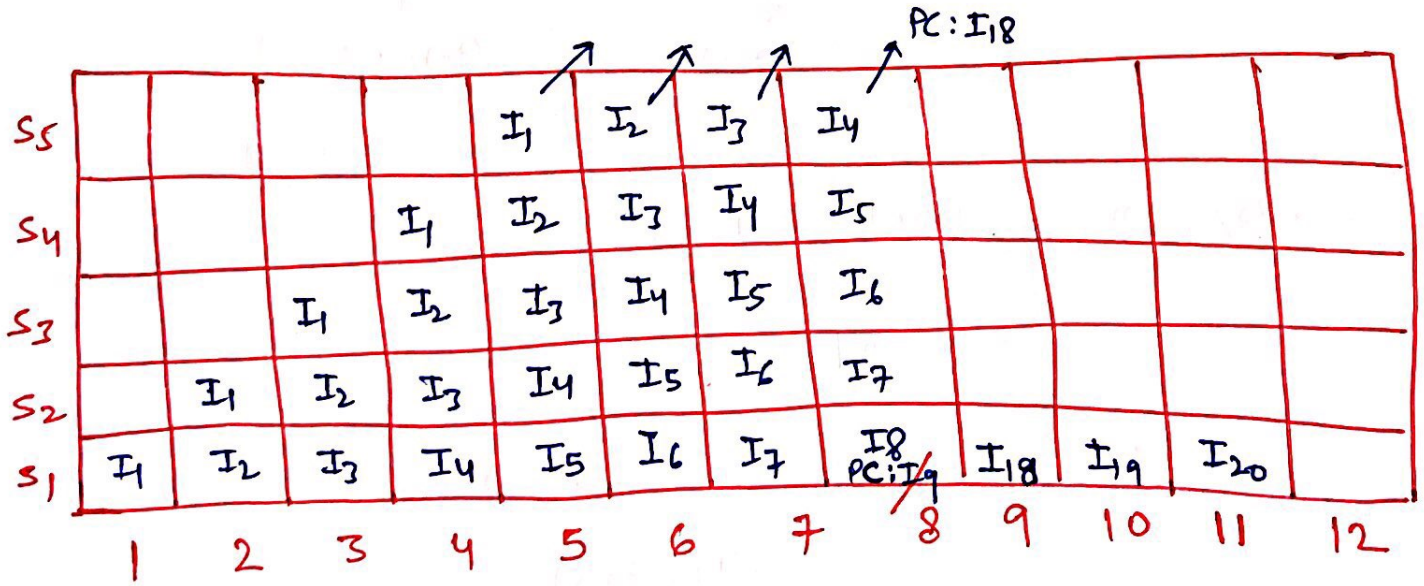
$(K = 5,\ t_p = 8ns)$

Program

I_1
I_2
I_3
I_4 : JMP I_{18}
I_5
⋮
I_{18}
I_{19}
I_{20}

$$ET_{pipe} = (K + n - 1)\, t_p$$

$n = 20$, but due to JMP inst^n all 20 inst^n are not executed.



so, effective no. of input (n) = 11

so, $ET_{pipe} = (k + n - 1)\, tp$

$= (5 + 11 - 1)(8ns)$

$= 15(8)$

$= (120ns)$

Q. Consider the following code, executed on the pipeline CPU.

I_1 : $r_0 \leftarrow r_1 + r_2$

I_2 : $r_1 \leftarrow r_0 - r_2$

I_3 : $r_0 \leftarrow r_1 * r_2$

I_4 : $r_2 \leftarrow r_0 * r_1$

I_5 : $r_1 \leftarrow r_2 - r_0$

I_6 : $M[X] \leftarrow r_1$

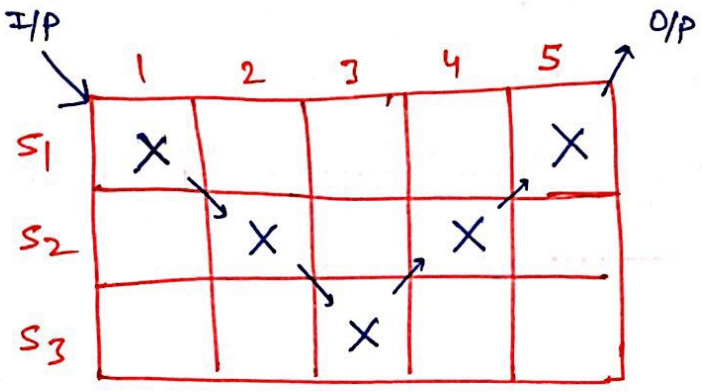
How many RAW, WAR & WAW Hazards possible?

RAW [In-order] (IN-OUT)	WAR [out-order] (OUT - IN)	WAW [out-order] (OUT - OUT)
$I_2 - I_1$ (r_0)	$I_2 - I_1$ (r_1)	$I_3 - I_1$ (r_0)
$I_3 - I_2$ (r_1)	$I_3 - I_2$ (r_0)	$I_5 - I_2$ (r_1)
$I_4 - I_3$ (r_0)	$I_4 - I_3$ (r_2)	
$I_5 - I_4$ (r_2)	$I_4 - I_2$ (r_2)	
$I_6 - I_5$ (r_1)	$I_4 - I_1$ (r_2)	
	$I_5 - I_4$ (r_1)	
	$I_5 - I_3$ (r_1)	
	$I_5 - I_1$ (r_1)	

Non-linear Pipeline :-

- This pipeline contains forward and backward connection, so reservation table is used to procus the inputs.

- Let us consider the following reservation table to process the inputs -



path : $S_1 - S_2 - S_3 - S_2 - S_1$

- Multiple checkmarks in the row, shows repeated use of same stage in different cycles.

- contiguous checkmarks in the row indicates extended use of same stage.

- Multiple checkmarks in the column indicates parallel use of diff stages in the same cycle.

Latency analysis :-

- Latency means clk cycle difference b/w to successive initiation in the pipeline.

- It depends on the reservation table.

- In the non-linear pipeline, some of the latencies will causes collisions called as forbidden latency (non-permisible latency). And some of the latency doesn't causes collision called as non-forbidden latency (permi-sible latency)

- To detect the forbidden latency we need to take the clock cycle difference b/w any two checkmarks in the row. i.e

row 1 (S_1) : $5-1 = 4$ ($C_4 = 1$)

row 2 (S_2) : $4-2 = 2$ ($C_2 = 1$)

row 3 (S_3) : —

- Collision vector is derived based on the forbidden latencies.

i.e $[C_n \ldots\ldots C_2 C_1]$ $\quad C \rightarrow \begin{cases} 0 \text{ ; permissiable} \\ 1 \text{ ; non-permissiable} \end{cases}$

length of collision vector = no. of cycles needed to execute one inst^n.

eg:- $\begin{matrix} C_5 & C_4 & C_3 & C_2 & C_1 \end{matrix}$

$[0 \quad 1 \quad 0 \quad 1 \quad 0]$

- Latency sequence is derived based on collision vector w.r.t permissiable latencies. i.e

(i) Latency seq w.r.t 'c1' :-

	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1	1	2			1	2	3	4			3	4	5
S_2		1	2	1	2			3	4	3	4		
S_3			1	2					3	4			

Latency cycle

(2-1) = 1

(7-2) = 5

(8-7) = 1

(13-8) = 5

(1,5) , (1,5) , (1,5) ....

(Avg -latency) = $\frac{1+5}{2} = \underline{3}$

(ii) Latency seq w.r.t 'c3' :-

	1	2	3	4	5	6	7	8	9	10	11	12	13
S_1	1			2	1		3	2		4	3		
S_2		1		1	2		2	3		3	4		4
S_3			1			2			3			4	

Latency cycle

3, 3, 3 .....

(4-1) = 3

(7-4) = 3

(10-7) = 3

(Avg-latency = 3)

(iii) Latency seq wrt 'C_5' :-

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S_1	1				1	2	3			2	3	4	5			4
S_2		1		1			2	3	2	3			4	5	4	5
S_3			1					2	3					4	5	

Latency - cycle

(6-1) = 5

(7-6) = 1

(12-7) = 5

(13-12) = 1

(5,1), (5,1), (5,1), ...

(Avg - latency = $\frac{5+1}{2}$ = 3)

Memory Organization :-

$$\text{Performance} \propto \frac{1}{\text{Access time}}$$

$$\text{Access-time} \propto \text{Execution time}$$

$$\text{Total Time} = \text{Hit/miss Latency} + \text{Access time} + \text{Transfer time}$$

Hit/miss Latency → (Negligible)

Transfer time → Depends on Bandwidth

Types of memory organization :-

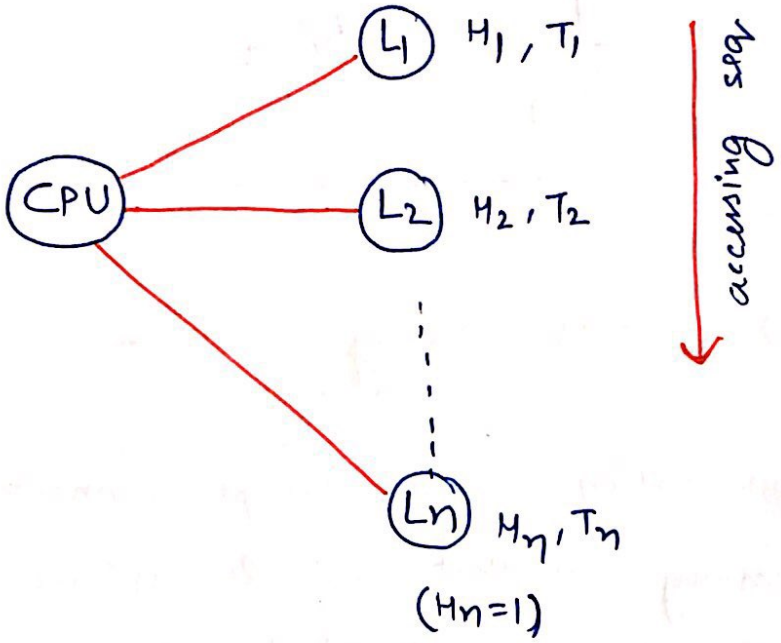
(i) Simultaneous Access mem org^n.

(ii) Hierarchal Access mem org^n.

(i) Simultaneous Access memory org^n :-

- In this org^n, CPU directly connected to all the levels of a memory, so that CPU will be accessing the data from any level of the memory in a sequence i.e
  - when there is miss in level 1 mem, then CPU directly access the data from level 2 mem without copying it into level 1 mem

- when there is a miss in level 2 [mem] then CPU access data directly from level 3 mem without copying it into level 2 and level 1 memory and so on.



H is hit-ratio
(1-H is miss-ratio)
T is access time

- $\text{Hit-Ratio (H)} = \dfrac{\text{Hits}}{\text{Total Access}}$

$\text{Hit-Ratio (H)} + \text{miss-ratio} = 100\%$

- $T_1 < T_2 < T_3 \text{ ----- } < T_n$

Time required to access 1 word Data from the mem is called as avg. memory access Time

$$T_{avg} = H_1T_1 + (1-H_1)H_2T_2 + (1-H_1)(1-H_2)H_3T_3 + \text{-----}$$
$$\text{----} (1-H_1)(1-H_2)\dots(1-H_{n-1})H_nT_n$$

1 word $\longrightarrow$ T_{avg}

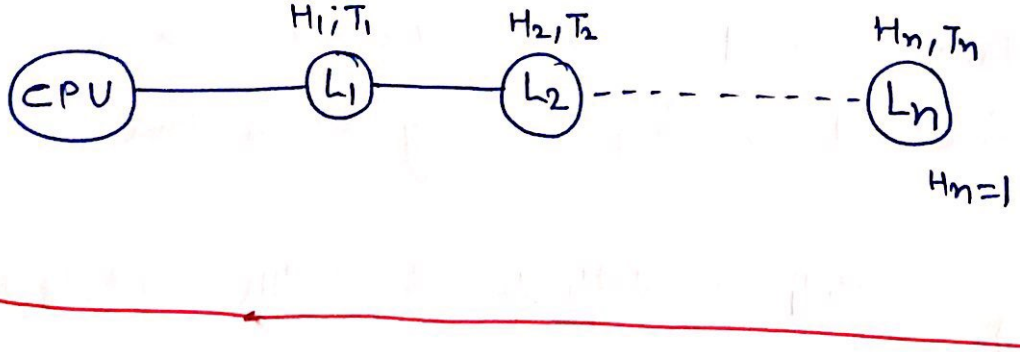
so in 1 sec = $\frac{1}{T_{avg}}$ words/sec

$$\eta_{mem} = \frac{1}{T_{avg}} \text{ words/sec}$$

(ii) Hierarchal Access memory org^n :-

- In this organization, CPU always connected to level 1 memory so that CPU always accessing the data only from the level 1 memory.

- When there is a 'miss' in level 1 memory then the respective data will be copied from higher levels to level 1 memory. Later CPU will access the data only from the level 1 memory.



$$T_{avg} = H_1 T_1 + (1-H_1) H_2 (T_2+T_1) + (1-H_1)(1-H_2) H_3 (T_3+T_2+T_1)$$
$$+ \cdots + (1-H_1)(1-H_2)\ldots(1+H_{n-1}) H_n (T_n + T_{n-1} + \cdots T_1)$$

Analysis:-

I_1 : Data (Assume that data present in L2 Mem)

I_2 :

I_3 : Data

I_4 :

I_5 : Data

I_6 :

I_7 : Data

} Re-use the I_1 Data

Assume $L_1 \rightarrow T_1$ (2ns)

$L_2 \rightarrow T_2$ (10ns)

Simultaneous

I_1 : M_{L_1}, T_2

miss in L1

I_3 : M_{L_1}, T_2

I_5 : M_{L_1}, T_2

I_7 : M_{L_1}, T_2

Time = $4T_2$

= (40ns)

Heierarchal

I_1 : $M_{L_1}, T_2 + T_1$

I_3 : T_1

I_5 : T_1

I_7 : T_1

} Temporal locality → access same data word

Time = $T_2 + 4T_1$

= (18ns)

I_1 : Data (W_1)

I_2 : " (W_2)

I_3 : " (W_3)

I_4 : " (W_4)

Assume all the words are present in L2, Mem

Simultaneous	Hierarchal
I_1 : M_{L_1}, T_2	I_1 : M_{L_1}, T_2+T_1
I_2 : M_{L_1}, T_2	I_2 : M_{L_1}, T_2+T_1
I_2 : M_{L_1}, T_2	I_3 : M_{L_1}, T_2+T_1
I_4 : M_{L_1}, T_2	I_4 : M_{L_1}, T_2+T_1
Time = $4T_2$ = (40ns)	Time = $4(T_2+T_1)$ = (48 ns)

So, ~~adjac~~ adjustment is done, data is transfer in the form of blocks.

Assume block size = 16 W so,

in one data - transfer 16 words will be move to L_1

I_1 : M_{L_1} , $T_2 + T_1$

I_2 : T_1

I_3 : T_1

I_4 : T_1

} Spatial Locality → adjacent data words Accessing

Time = $T_2 + 4T_1$

= (18ns)

NOTE :- In the hirarichal access memory org", access time is low because of the principal 'locality of refrence'.

- "Locality of refrence" means accessing the higher levels of a memory data from the lower-level memory (L_1). It is of two types :

  (i) Temporal locality

  (ii) Spatial locality

- To satisfy the above locality of refrence, we need to organise the data in the memory in form of blocks. Block contain multiple words and its size is decided by designer.

NOTE:- when the question contain hirarchy word or cache memory then use hirarchal org^n otherwise use simultaneous org^n.

Q. In a two-level memory org^n, Level 1 memory is 8 times faster than level 2 memory. And its access time is 30ns less than avg memory access time. Let L1 memory access time is 40ns. What is the Hit_ratio?

$(H_2 = 1)$

$$s = \frac{T_2}{T_1} = 8$$

$(T_2 = 8T_1)$

$(T_1 = 40ns)$

so, $T_2 = 320ns$

and

$T_{avg} = 40 + 32$
$= 70ns$

$$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2$$

$$T_{avg} = H_1 T_1 + 8(1-H_1) T_1$$

$$70ns = H_1(40) + 8(1-H_1)(40)$$

$$70 = 40H_1 + 320 - 320H_1$$

$$70 = -280H_1 + 320$$

$$280H_1 = 250$$

$$\left(H_1 = \frac{25}{28}\right)$$

Q. Three-level memory org^n has the following specification :

Level	Access time per word	Block-size in word	Hit-ratio
1	40 ns	—	0.6
2	100 ns	2	0.9
3	200 ns	4	1

If the retrieved data is not in L1 memory then transfer it from L2 to L1 and if not in L2 mem then transfer it from L3 → L2 → L1. What is T_{avg}?

$T_1 = 40 \times 1 = 40ns$

$T_2 = 100 \times 2 = 200ns$

$T_3 = 200 \times 4 = 800ns$

So,

$$T_{avg} = H_1T_1 + (1-H_1)H_2(T_2+T_1) + (1-H_1)(1-H_2)(T_2+T_1+T_3)$$

$$= 0.6(40) + (0.4)(0.9)(240) + (0.4)(0.1)(1040)$$

$$= 24 + 86.4 + 41.6$$

$$= \underline{(152\ ns)}$$