"Control Dependency" :-

★ Control dependency Problem will be occured in the pipeline when the Transfer of control Instructions are executed.

★ Consider the following Program Segment :-

Code :

```
1000 : I1          | Falling / fall through
1001 : I2          | Path
1002 : I3 (Jmp 2000)
1003 : I4
1004 : I5
  .      .
  .      .
2000 : BI1         | Taken
2001 : BI2         | Path
```

→ Expected o/p sequence : $I_1 - I_2 - I_3 - BI_1 - BI_2$

	CC1	CC2	CC3	CC4	CC5	CC6
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 Uncond Jmp Pc: 1004 / 2000	Ex	MA
I_4				IF Pc: 1004	ID	Ex
BI_1					IF Pc: 2001	ID
BI_2						IF Pc: 2002

Actual O/P Sequence : $I_1 - I_2 - I_3 - \boxed{I_4} - BI_1 - BI_2$

I_4 ↓ unwanted

* In the above execution sequence unwanted instruction is overlapped with a jump instruction. when the unwanted instruction is executed in the pipeline then Program functionality is missing.

* To correct the above Problem, we need to insert the delay slot after the jump instruction to suspend the unwanted instruction fetch. that is:

Code:

```
1000 : I1
1001 : I2
1002 : I3 (JmP 2000)
1003 : NOP
       (Delay slot)
1004 : I4
  :    :
2000 : BI1
2001 : BI2
  :    :
```

	CC1	CC2	CC3	CC4	CC5	CC6
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 uncond T.C PC: 1004	EX	MA
NOP:				IF PC: ~~1004~~ 2000	ID	EX
BI_1:					IF PC: 2001	ID
BI_2:						IF PC: 2002

Actual O/p sequence :

$I_1 - I_2 - I_3 - \boxed{NOP} - BI_1 - BI_2$

Stall

* "NOP Instruction" is used for delay in the Program level or in the System level. It contain opcode so consume the cycle to execute the opcode but Nothing will be changed in the Hardware therefore this situation is considered as delay.

* To Handle the control dependency Problem, Nop Instn is inserted after the Jump Instruction, To suspend the unwanted instruction fetch. this Process is called as 'flush' or freeze.

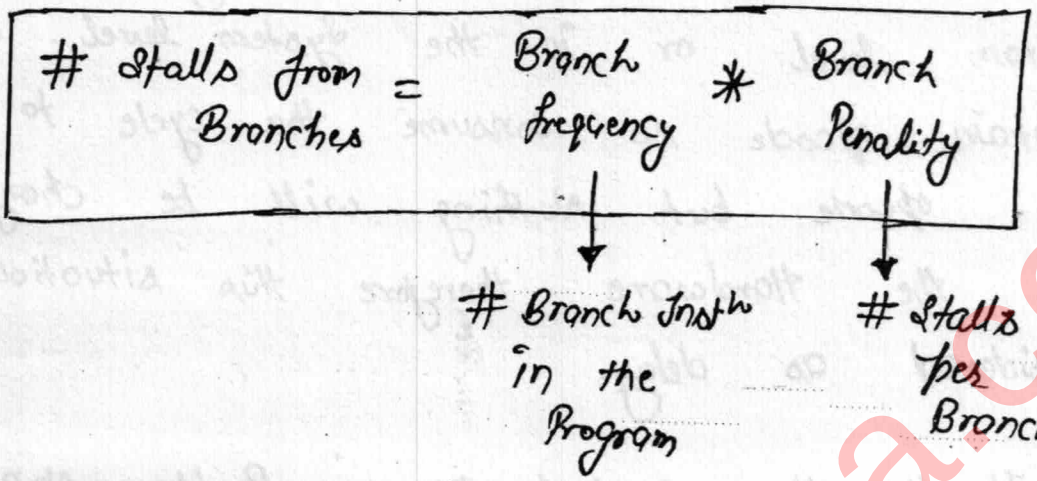
* flush operation creates stalls in the Pipeline. name as Branch Penalty.

* Branch Penality is depends on the availability of a Target Address in the Pipeline that is :

$$\text{Branch Penality} = \text{At what stage Target address is available in the Pipeline} - 1$$

** Note :- RISC Pipeline Branch Penality is always 1 because Target Address is in the 2nd stage.

* # of Stalls created in the Pipeline from the Branches during the Program execution is calculated as :



* To minimize the control dependency stalls Hardware Technique is used that is Branch Prediction buffer' (loop Buffer / Branch target Buffer).

* Branch Target Buffer / Branch Prediction Buffer :- It is a High speed buffer present in IF stage used to hold the Predicted Target Addresses when the Prediction is true then stall is not present otherwise stall is created.

** Note :- when the question contain Pipeline design with Branch Prediction then Assume that Target ~~Direct~~ Address is available in the 1st Stage so penality '0' otherwise (without Branch Prediction) Assume that Target Address not Present in the 1st Stage. So Penality depends on the Availability of the Target Address in the Pipeline.

Note :- To minimize the control dependency stall one of a software technique is used name as delayed Branch.

→ **"Delayed Branch" :-**

It is a compiler technique so compiler will be re-arranges the code if possible or substitute the NOP instruction after the Jump instruction if not possible to re-arrange, to preserve the Execution path.

User Code :

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

Compiler "Re-Arrangment" Code :

machine generated addr.

I_1
I_3 (Jmp BI_1)
I_2
I_4
⋮
BI_1
BI_2

Expected o/p :-

$I_1 - I_2 - I_3 - BI_1 - BI_2$

Actual o/p :-

$I_1 - I_2 - I_3 - I_4 - BI_1 - BI_2$

Pc: I_1	CC1	CC2	CC3	CC4	CC5
I_1:	IF Pc: I_3	ID	EX	MA	WB
I_3:		IF Pc: I_2	ID uncond Toc Pc: ~~I_4~~ BI_2	EX	MA
I_2:			IF Pc: ~~I_4~~	ID	EX
BI_1:				IF Pc: BI_2	ID
BI_2:					IF Pc: BI_3

Actual o/p sequence :- $I_1 - I_3 - I_2 - BI_1 - BI_2$

Compiler "Nop Substitution"

Code:

I_1
I_2
I_3 (Jmp BI_1)
Nop
I_4
⋮
BI_1
BI_2
⋮

⇒

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
PC: I_1							
I_1:	IF PC: I_2	ID	EX	MA	WB		
I_2:		IF PC: I_3	ID	EX	MA	WB	
I_3:			IF PC: Nop	ID uncond TC PC: I_4	EX	MA	
Nop:				IF PC: I_4	ID	EX	
BI_1:					IF PC: BI_2	ID	
BI_2:						IF PC: BI_3	

Actual o/p sequence :- $I_1 - I_2 - I_3 - Nop - BI_1 - BI_2$

Stall

"Instruction Scheduling" :-

* CPU always executes the Program in a sequence. (Top-down Approach). called as In-order Execution.

* In this Execution sequence, If any Instruction is dependent then the Remaining Instructions are also sharing the stall cycles even they are independent.

Eg:

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 sequence is :

$I_1 - I_2 - I_3 - I_4$

Stalls

* In the above code I_2 is data dependent on I_1 so I_2 will be waiting until I_1 is completed. This waiting creates Stalls. These Stalls are also shared by the I_3, and I_4 Even they are independent.

* To Handle the Above Situation, In-order to utilize the Stalls, we can insert the independent Instructions first causes out of order (Re-order) Execution i.e.

$I_1 - I_3 - I_4 - I_2$

* Re-order Execution creates two more dependencies in the Pipeline:

1.) Anti-Data dependency.

2.) O/p Data dependency.

"Instⁿ-J" "Instⁿ-I"

Read-before-write [True data loss]

write-before-Read [ANTI data loss]

write-before-write [o/p data loss]

↓

After

↓

Delay

↓

Stall

↓

Hazard

* "Anti-Dependency" will be occured when the Instⁿ-'J' tries to write the data before Instruction-'I' reads it

eg:- I_3 executes before I_2

so I_3 updates the Register r_4 before I_2 reads it therefore I_2 'incorrectly read' the New value from the r_4. (Data Loss)

★ "Output Data dependency" will be occured when the Instruction 'J' tries to write the data before Instruction -'I' writes it.

eg:- I_4 executing before I_2.

So, I_4 updates the Register r_3 before I_2 writes it. therefore Destination is incorrectly updated with a old value (Data Loss).

★ To Handle the above dependency Problems Hardware technique is used that is Register Renaming.

★ Register Renaming :-

→ This technique States that use the Re-order buffers to store the output of a out of order Instructions. later the Re-order Buffers update the Register file with a Re-order Buffer contents after the completion of a dependent Instruction.

Execution :-

$I_1 \rightarrow r_0$

I_3, I_4] → Re-order Buffer

$I_2 \rightarrow r_3$ [Status = 1]

⇓

Re-order Buffer → Reg. file { r_4, r_3 }

"Hazards" :-

* Hazard is a delay. Delay is present in the Pipeline due to a dependency Problem so Hazard is classified into three types:

1.) Structural Hazard
2.) Data Hazard
3.) Control Hazard

* Data Hazard is further classified into three types based on the order of Read and write operation :

1) RAW (Read after write) Hazard
2.) WAR (write after read) Hazard
3) WAW (write after write) Hazard

* RAW Hazard is created when the instruction 'J' tries to read the data before Instruction 'I' writes it { True Data dependency }

* WAR Hazard is created when the Instruction 'J' tries to write the Data before Instruction 'I' reads it. { Anti-Data dependency }.

* WAW Hazard is created when the Instruction 'J' tries to write the Data before Instruction 'I' writes it. { O/P data dependency }.

★ Note :- RAR (Read after Read) is not a Hazard because RBR is not a problem.

★★

Instⁿ 'j'	Instⁿ 'I'	
IN-Reg	out-Reg	[True Data dependency]
out-Reg	IN-Reg	[Anti Data dependency]
out-Reg	out-Reg	[O/P Data dependency]

"Performance Analysis" :- "with Stalls"

$$S = \frac{\text{Avg Instⁿ E.T. Nonpipe}}{\text{Avg Instⁿ E.T. Pipe}}$$

$$S = \frac{CPI_{Nonpipe} * \text{Cycletime}_{NonPipe}}{CPI_{pipe} * \text{Cycle time}_{Pipe}}$$

★ Ideal CPI of Pipeline is always 1 but due to dependency Problem stalls are created in the Pipeline therefore ;

$$S = \frac{CPI_{NonPipe} * \text{Cycle time}_{NonPipe}}{\left(1 + \#\text{stalls}/\text{Instⁿ}\right) * \text{Cycle time}_{Pipe}}$$

when the Pipeline Stages are perfectly balanced then 1 task execution time in Non-Pipe is also equal to # of stages in Pipeline i.e

$$\boxed{t_n = k \cdot t_p}$$

therefore,

$$S = \frac{k \cdot t_p}{\left(1 + \#\text{ stalls}/\text{Inst}^n\right) * t_p}$$

$$\boxed{S = \frac{k}{\left(1 + \#\text{ stalls}/\text{Inst}^n\right)}}$$

* when the system is 100% efficient, then No stalls present therefore.

$$\boxed{S = k}$$

Que:- Conpider 6-Stage Pipeline which allows all the Instructions except Branch Instruction the following Instruction after the Branch is stopped until the Target Address is available. In the Pipeline Target Address is available at the End of Execution. Program contain 40% Branch Instruction among them 70% are Conditional in which 40% of Instructions does not satisfy the condition. when the condition is false then the following ~~condition~~ Instructions are overlapped. Pipeline is operated with a 2ns

clock.

Date: 20/08/2017

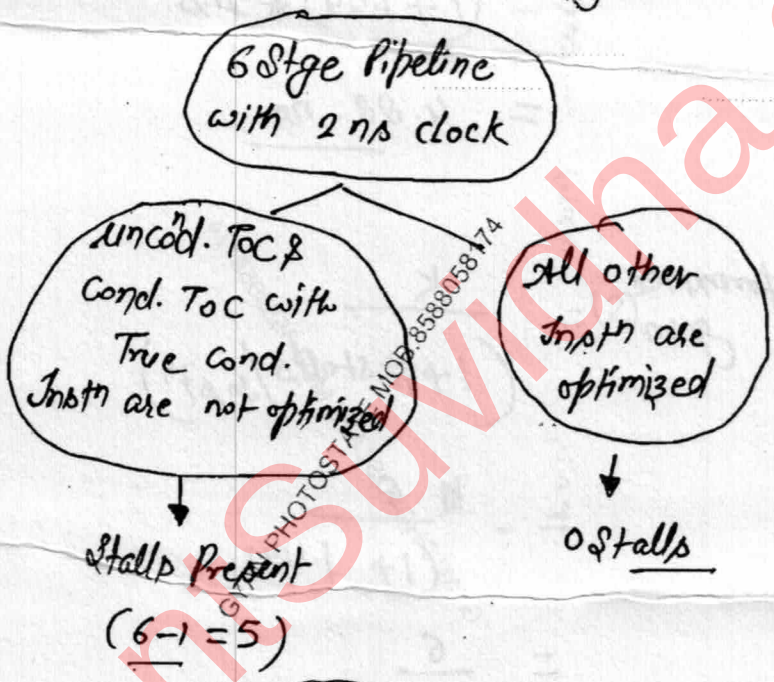
a) what is the Average Instn execution time ?

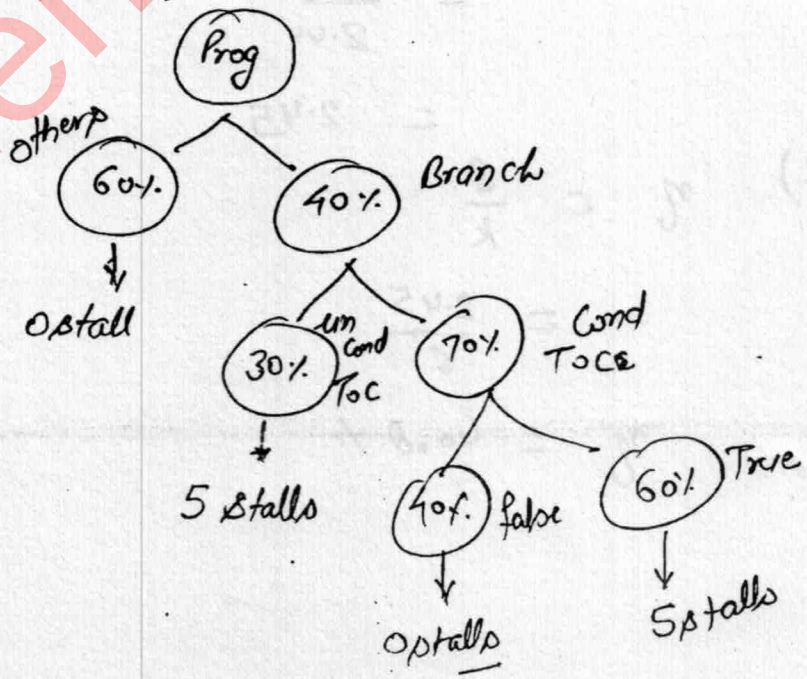
b) what is the performance gain ?

c) what is the efficiency ?

Soln :-

If Stage of the Target Address not given Assume it as last i.e for 6 stage Pipeline, Target addr. will be present at 6th stage.





$$\#\ \text{Stalls/Instn} = \Big[((0.6)*0) + (0.4*0.3*5) + (0.4*0.7*0.4*0) + (0.4*0.7*0.6*5)\Big]$$

$$= 1.44$$

a) $\text{Avg Instn}_{\text{Pipe}} = \left(1 + \#\ \text{Stalls/Instn}\right) \text{cycletime}$

$$= (1 + 1.44) * 2\text{ns}$$

$$= 4.88\ \text{ns}$$

b.) $\text{Performance gain}(S) = \dfrac{k}{(1 + \#\ \text{Stalls/Instn})}$

$$= \frac{6}{(1 + 1.44)}$$

$$= \frac{6}{2.44}$$

$$= 2.45$$

c) $\eta = \dfrac{S}{k}$

$$= \frac{2.45}{6}$$

$$\eta = 40.8\ \%$$

Note :-

In the Question Target Address availability :

a) ~~Available~~ Stage Number given then

(stalls) 'Branch Penality' = Stage number - 1.

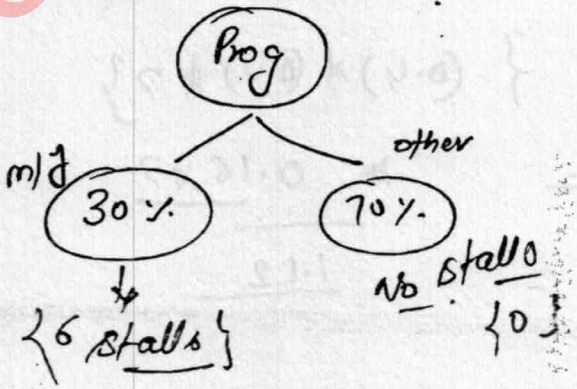
b.) Stage Name given then Branch Penality.

'Branch Penality' = corresponding Stage No - 1

c.) All Instn Proceed through all stages / (or) until the Instn is completed then

'Branch Penality' = last Stage Number - 1.

Que :- Consider 8-stage Pipeline operated with a 3 ns clock which allows all the Instn except memory Instn. Penality of a m/y Instn is 6 cycles program contain 30% memory instructions. what is the average Instn execution time ?

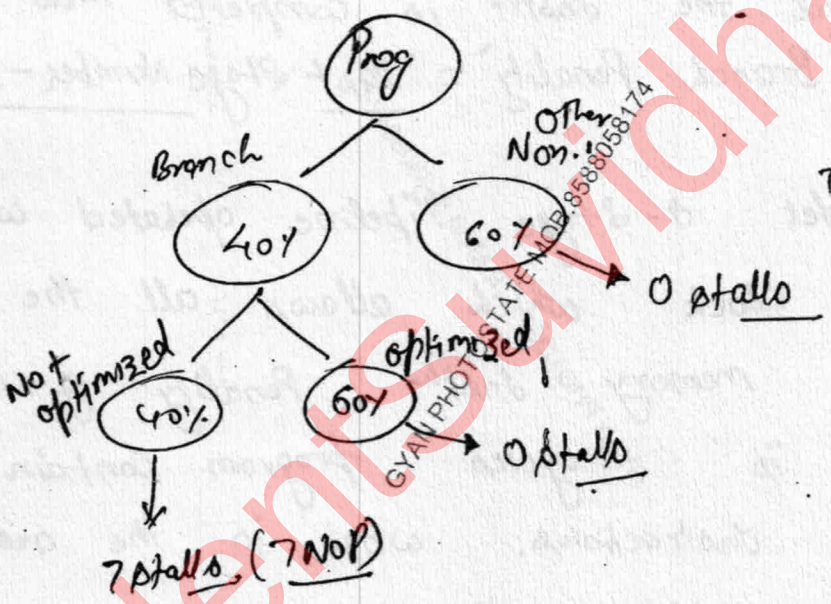


$$\#\ \text{stalls/Instn} = 0.3 * 6 = 1.8$$

$$\text{Avg Instn}_{\text{pipe}} = (1 + 1.8) * 3 = 2.8 \times 3 = 8.4$$

Que:- Consider a Hypothetical Pipeline which allows all the Instn except Branch Instn. To optimize the Branch Instn Delay Branch is used in the Pipeline design this technique substitutes the 7 NOP Instn after the Jump If not possible to Re-Arrange. Program Contain 40% Branch Instn among them 60% of Instn are optimized what is the average Instn execution time. when the pipeline is operated with 250 MHz clk.

Soln



$$T = \frac{1}{250\,MHz} = \frac{1000}{250} = 4\,ns$$

$$\#\,stalls/Instn = \{(0.4) * (0.4) * 7\}$$

$$= 0.16 \times 7$$

$$= 1.12$$

$$Avg\ Time_{Pipe} = (1 + 1.12) \times 4$$

$$= 2.12 \times 4$$

$$= 8.48\ ns$$

Que:- Consider 6 stage pipeline used to execute the Program code size of 20 Instns (I_1 to I_{20}). Stages are perfectly balanced with 2ns clock. All the Instns are proceed through all the stages in the pipeline I_{12} Instn is a unconditional Jump transfers the control to I_{19} Instn. what is the program execution time.

Soln:-

6 Stage Pipeline

I_1 to I_{20}

Target addr. = I_{12}

Penalty = 12 − 1 = 11

12 to 19

tp = 2ns

$T_{in} = \frac{20}{6} = 3.5$

$[(3.5 * 11)]$

$= 38.5$

Pipe = $(1 + 38.5) * 2$

$= 48.8 * 2$

$= 97.0$

1 Instn

12 to 19 Stages

$[11 + 7 + 6] * 2ns$

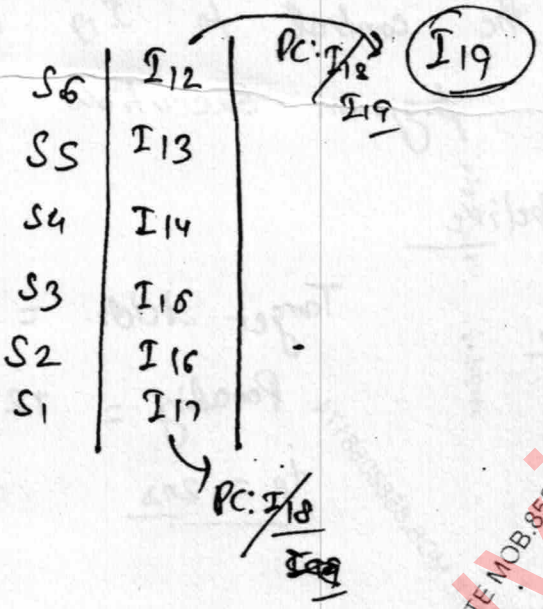
↑ Stalls

$= 48 ns$

Program size = 20 Instr
= {I_1 to I_{20}}

T.A. Present in the last stage.

6 Stages pp



I_{18} will never enter to Pipeline.

k = 6
n = 19
tp = 2ns

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

$= (6+19-1)\,2ns = (24)\times 2 = 48\,ns$

Que:- Consider the following code running on a Pipelined Processor:

I_1 : $r_0 \leftarrow r_1 + r_2$

I_2 : $r_1 \leftarrow r_0 - r_2$

I_3 : $r_2 \leftarrow r_1 * r_0$

I_4 : $r_0 \leftarrow r_2 / r_1$

I_5 : $r_2 \leftarrow r_0 - r_1$

I_6 : $m[x] \leftarrow r_2$

How many RAW, WAR, WAW Hazards are possible in the pipeline?

1. True data dependency = 5 → (RAW)
2. O/P data dependency = 2 → (WAW)
3. ANTI data dependency = 8 → (WAR).

RAW [Adjacent]
[IN-out]

$I_2 - I_1$ [r_0]
$I_3 - I_2$ [r_1]
$I_4 - I_3$ [r_2]
$I_5 - I_4$ (r_0)
$I_6 - I_5$ [r_2]

(5)

WAR
[out-IN]

$I_2 - I_1$ [r_1]
$I_3 - I_1$ [r_2]
$I_3 - I_2$ [r_2]
$I_4 - I_2$ [r_0]
$I_4 - I_3$ [r_0]
$I_5 - I_1$ [r_2]
$I_5 - I_2$ [r_2]
$I_5 - I_4$ [r_2]

(8)

WAW
[out-out]

$I_4 - I_1$ [r_0]
$I_5 - I_3$ [r_2]

(2)