COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_08

Vijay Agarwal sir

01 Pipelining Hazards

① Structural Hazards	Resource Conflict	Solution Resource Replication Renaming.
② Data Hazards	Data	H/w Interlock & OPERAND Forwarding
③ Control Hazards.	Branch Instⁿ (TOC Instⁿ) Program Control Instⁿ	H/w → Branch Prediction S/w → Delay Branch

Pipeline Hazards

Occur when the Pipeline, or some portion of the pipeline must stall Because conditions Do not permit Continued execution

There are three Types of hazards:

1. Resource
2. Data
3. Control

Also referred to as a Pipeline bubble

In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. *Resource conflicts* caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. *Data dependency* conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. *Branch difficulties* arise from branch and other instructions that change the value of PC.

Hazards/Dependencies In the pipeline

- Dependency is a major problem in the pipeline, causes extra cycle.
- Cycle in the pipeline without new input is called as extra cycle. Also named as "Stall".
- When stall in present in the pipeline then CPI $\neq 1$.
- There are 3 kinds as dependencies possible in the pipeline-
  - I. Structural dependency/ Structural Hazards
  - II. Data dependency/ Data Hazards
  - III. Control dependency/ Control Hazards

Structural Dependency

Resource Conflict

Soln → Resource Replicition (Using Additional Resource)

→ Renaming.

Data Dependency

I_1 : ADD r_1 r_2 r_3 ; $r_1 \leftarrow r_2 + r_3$

I_2 : MUL r_4 r_1 r_5 ; $r_4 \leftarrow r_1 * r_5$

Data Dep. → H/w Interlock
Data Dep. → OPERAND Frowarding | By Passing | Short circuit

ANTI (WAR) Dep.
OUTPUT (write) Dep
} Solution Register Renaming

Q. The performance of a pipelined processor suffer if

[GATE-2002 : 2 Marks]

↳ Control

A. the pipeline stages have different delays

↳ Data Dep.

B. consecutive instructions are dependent on each other

↳ Structural Dep.

C. the pipeline stages share hardware resources

D. All of the above

Q. For a pipelined CPU with a single ALU, consider the following situations

1. The $j^{th} + 1^{st}$ instruction uses the result of the j^{th} instruction as an operand → True Data Dep.
2. The execution of a conditional jump instruction → Control Dep.
3. The j^{th} and $j^{th} + 1^{st}$ instructions require the ALU at the same time → Structural Dep.

Which of the above can cause a hazard?

[GATE-2003 : 1 Marks]

(A) 1 and 2 only

(B) 2 and 3 only

(C) 3 only

(D) All the three

Q.

A pipelined processor uses a 4-stage instruction pipeline with the following stages: Instruction fetch (IF), Instruction decode (ID), Execute (EX) and Writeback (WB). The arithmetic operations as well as the load and store operations are carried out in the EX stage. The sequence of instructions corresponding to the statement $X = (S - R * (P + Q))/T$ is given below. The values of variables P, Q, R, S and T are available in the registers R0, R1, R2, R3 and R4 respectively, before the execution of the instruction sequence.

```
ADD      R5, R0, R1      ;  R5 ← R0 + R1
MUL      R6, R2, R5      ;  R6 ← R2 * R5
SUB      R5, R3, R6      ;  R5 ← R3 – R6
DIV      R6, R5, R4      ;  R6 ← R5 /R4
STORE    R6, X           ;  X ← R6
```

The IF, ID and WB stages take 1 clock cycle each. The EX stage takes 1 clock cycle each for the ADD, SUB and STORE operations, and 3 clock cycles each for MUL and DIV operations. Operand forwarding from the EX stage to the ID stage is used. The number of clock cycles required to complete the sequence of instructions is

[GATE-2006: 2 Marks]

A 10

B 12

C 14

D 16

Q. Consider the sequence of machine instructions given below:

```
MUL     R5, R0, R1
DIV     R6, R2, R3
ADD     R7, R5, R6
SUB     R8, R7, R4
```

In the above sequence, R0 to R8 are general purpose registers. In the instructions shown, the first register stores the result of the operation performed on the second and the third registers. This sequence of instructions is to be executed in a pipelined instruction processor with the following 4 stages: (1) Instruction Fetch and Decode (IF), (2) Operand Fetch (OF), (3) Perform Operation (PO) and (4) Write back the Result (WB). The PO state takes 1 clock cycle for ADD or SUB instruction. 3 clock cycles for MUL instruction and 5 clock cycles for DIV instruction. The pipelined processor uses operand forwarding from the PO stage to the OF stage. The number of clock cycles taken for the execution of the above sequence of instructions is _____.

[GATE-2015 (Set-2): 2 Marks]

NAT

The instruction pipeline of a RISC processor has the following stages. Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Writeback (WB) The IF, ID, OF and WB stages take 1 clock cycle each for every instruction. Consider a sequence of 100 instructions , In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycles each, and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data hazards and no control hazards.

The number of clock cycles required for completion of execution of the sequence of instructions is _____.

[GATE-2018-CS: 2M]

MCQ

Consider an instruction pipeline with five stages without any branch prediction: Fetch Instruction (FI), Decode Instruction (DI), Fetch Operand (FO), Execute Instruction (EI) and Write Operand (WO). The stage delays for FI, DI, FO, EI and WO are 5 ns, 7 ns, 10 ns, 8 ns and 6 ns, respectively. There are intermediate storage buffers after each stage and the delay of each buffer is 1 ns. A program consisting of 12 instruction I_1, I_2, I_3, ...I_{12} is executed in this pipelined processor. Instruction I_4 is the only branch instruction and its branch target is I_9. If the branch is taken during the execution of this program, the time (in ns) needed to complete the program is

[GATE-2013-CS: 2M]

A 132

B 165

C 176

D 328

Types of Data Hazard

- Read after write (RAW), or true dependency
  - An instruction modifies a register or memory location.
  - Succeeding instruction reads data in memory or register location.
  - Hazard occurs if the read takes place before write operation is complete.
- Write after read (WAR), or antidependency
  - An instruction reads a register or memory location.
  - Succeeding instruction writes to the location.
  - Hazard occurs if the write operation completes before the read operation takes place.
- Write after write (WAW), or Write/output dependency
  - Two instruction both write to the same location.
  - Hazard occurs if the write operations take place in the reverse order of the intended sequence.

Types of Data Hazard

- CPU always execute a sequence the program from top to bottom in a sequence called as **"in-order"** execution.
- In this execution sequence if any instruction is dependent then the remaining instruction are also sharing the stall cycles even, they are independent.

Ex-: Code

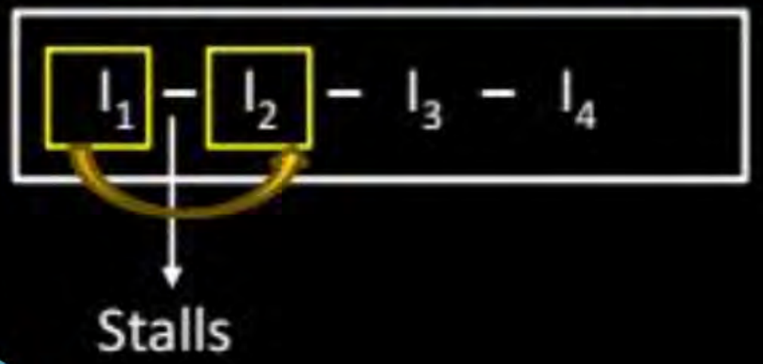
I_1: Add r_0, r_1, r_2

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



Types of Data Hazard

- In the above execution sequence 'I_2' is data dependent on "I_1" So, I_2 will be waiting until the 'I_1' execution is completed when the hardware does not support operand forwarding.
  This waiting creates stall in the pipeline this stall are also shared by the I_3 and I_4 instruction even they are independent.

- To utilize this point instruction scheduling concept states that insert the independent instruction first.
  It causes out of order execution {Re-order execution}.

- Out of order execution sequence is $I_1 - I_3 - I_4 - I_2$
- Out of order execution create two more dependencies in the pipeline ..
  (i) Anti data dependency
  (ii) output/Write data dependency

Types of Data Hazard

#	Instruction "J"		Instruction "I"
	Read	– Before –	write [True Data Loss]
	Write	– Before –	Read [ANTI Data loss]
	Write	– Before –	write [O/P DATA Loss]

Types of Data Hazard

Anti Dependency will be occurred when the instruction 'J' try to write the data before instruction 'I' reads it.

Example - I3 executed before I_2 so I_3 modified the register (r_4) before I_2 reads It, therefore I_2 incorrectly read the new value from (r_4) [Data loss].

Output dependency will be occurred when the instruction "J" tired to write the data before instruction 'I' writes.

Ex- I_4 executes before I_2 so, I_4 modifies the register (r_3) before I_2 writes it therefore destination is incorrectly updated with old value [data loss].

Types of Data Hazard

Note: To handle the above dependency problem hardware technique is used i.e., register renaming.

This technique states that, use the re-order buffer to store the out-of-order instruction output later update the register file with a re-order buffer content after the completion of a dependent instruction therefore data is safe.

Types of Data Hazard

Register Re-naming:

Execution

$I_1 \rightarrow r_0$

$I_3 \rightarrow$
$I_4 \rightarrow$
} Re-order buffer

Other Temp. Register.

$I_2 \rightarrow r_3$ (Status 1)

Re-order buffer $\rightarrow$ Reg. file $\{R_4, R_3\}$

(R_3, R_4).

↳ finish then Temp Reg Data Stored in Register file

Control Hazard

- Also known as a branch hazard
- Occurs when the pipeline makes the wrong decision on a branch prediction.
- Brings instructions into the pipeline that must subsequently be discarded
- Dealing with Branches:
  - Multiple Streams
  - Prefetch branch target
  - Loop buffer
  - Branch prediction
  - Delayed branch

Control Dependency :

↳ Control Dep. is created in the Pipeline Due to

- Branch Instn
- Program Control
- TOC Instruction
  - UnConditional TOC
  - Conditional TOC.

Control Dependency/Control Hazards

Control Dependency:

Control dependency problem, will be occurred in the pipeline, when the Branch Instruction, Function call, program Control/transfer of control instruction are executed.

Consider the program code-

1000 : I_1 ↓ Falling /Fall through path
1001 : I_2
1002 : I_3 (JMP 2000)
1003 : I_4
... ...
2000 BI_1 ↓ Taken path
2001 BI_2 ↓

Expected O/P sequence: $I_1 - I_2 - I_3 - BI_1 - BI_2$

Control Dependency :

Target Address is available at the end of k Stage

Branch Penalty = k − 1

Topic : Dependencies In he pipeline

- Analysis:

Target address availability with respect to question

1. Stage number given then

   Branch penalty = stage number –1

2. Stage name given then

   Branch penalty = [Respective name stage position] –1

3. Until Instruction execution is completed /All Instruction proceed with all stage then

   Branch penalty = [Last stage number] –1

Control Dependency/Control Hazards

PC: 1000	C1	C2	C3	C4	C5	C6	
I_1	IF PC: 1001	ID	EX	MA	WB		
I_2		IF PC: 1002	ID	EX	MA	WB	
I_3			IF PC: 1003	ID 2000 PC: 1004	EX	MA	
I_4				IF 2000 PC: 1004	ID	EX	
B I_1					IF PC: 2000	ID	
B I_2						IF PC: 2000	

∴ Actual O/P sequence: $I_1 - I_2 - I_3 - I_4 - BI_1 - BI_2$

Control Dependency/Control Hazards

In the above execution sequence, un-wanted instruction is executed in the program so, program functionality is unsafe. This kind of disturbance in pipeline is called control dependency.

To handle the above problem, we need for stop the unwanted instruction fetch. For this purpose "NOP" : Instruction is inserted after the jump instruction. This process is called as freeze or flush & that is described below_____.

NOP: opcode: [Delay instruction, no operation]

Control Dependency.

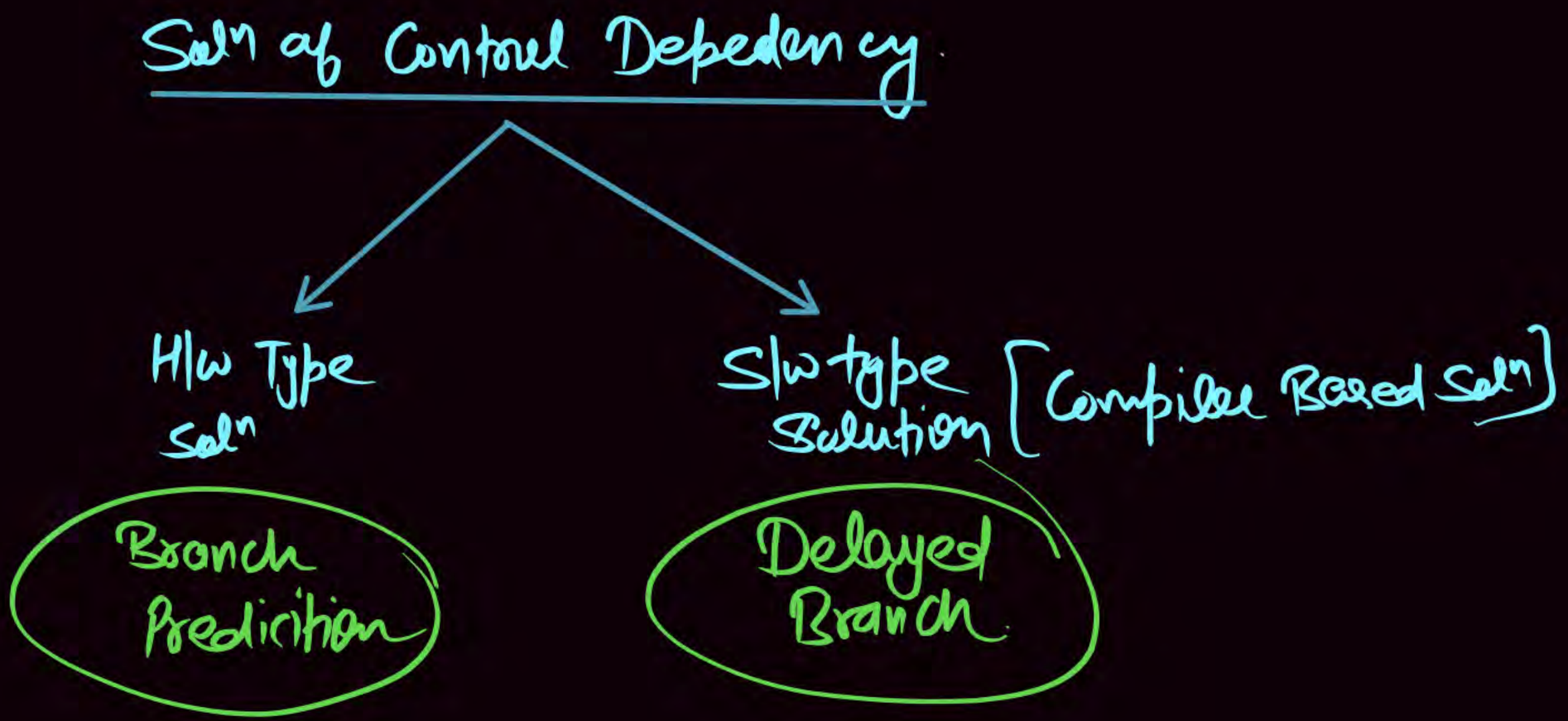
1. Flushing
2. Stalling

Soln
- A. Branch Prediction
- B. Delayed Branch
- C. Nop Instruction

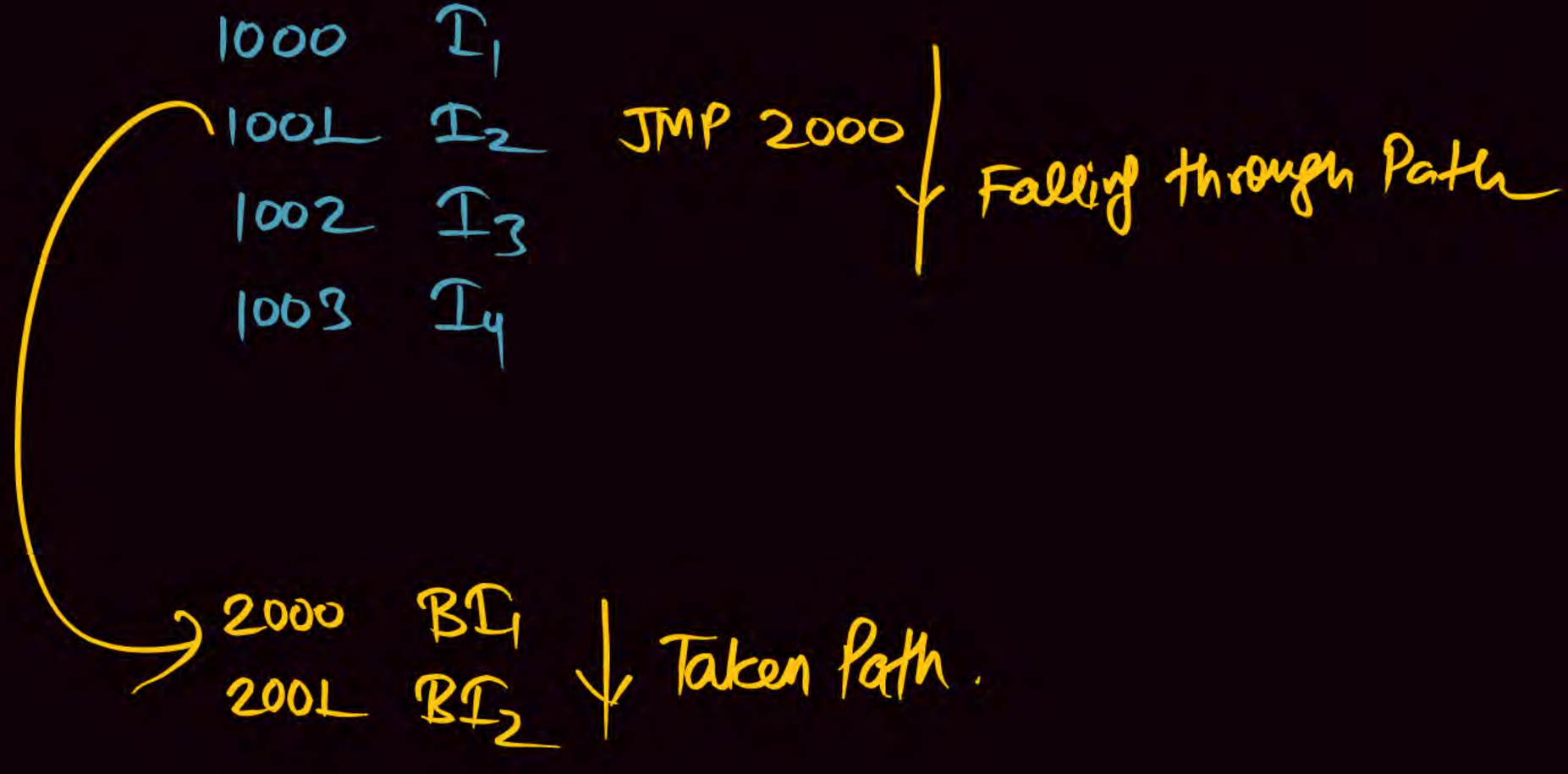
Control Dependency.

Stalling : Design a pipeline in such a Manner, if it Detect the Branch Instruction, then we have to Stall (Stop Fetching the Next Sequential Instruction into the pipeline) Untill Branch Instr Evaluation [Execution] Not Completed.

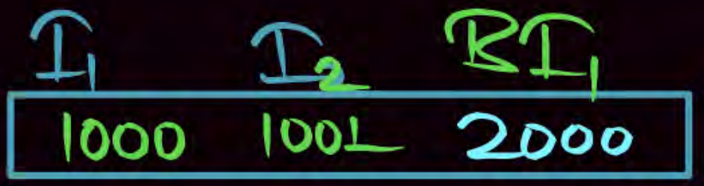
Control Dependency.



Control Dependency.



Control Dependency.



Branch Instn

(i) Unconditional : JMP x

Conditional : JNZ [Jump On Not Zero]
JZ [Jump on zero]
BNE [Branch on Not equal]
etc.

Branch Prediction: (Guess)

In this we Predicit Branch is taken (or) Branch is Not taken & If Predicition is Correct then Accordingly Instruction will execute.

(i) Branch is Always Taken : (In this we assume every time Branch is taken, so in this Next Instruction is Target Address Instruction for execution.

(ii) Branch is Never Taken : In this we Assume every time Branch is Not taken, so in this Next Instruction is the Next Sequential Instruction for execution.

Note: If Predicition goes wrong then we have to subber from stalls.

∴

① Branch is always Taken : (1001 - 2000 - 2001 - 2002 .)

↓

if guess [Predicition] wrong then Subbel from Stalls.

↓

If it is Normal Instruction [Non Inst Branch]

then Predicition feil & Subbel from Stalls.

② Branch is Never Taken :

1000, 1001, 1002, 1003, 1004 . . .

In this type Branch Instn Not coming. Branch is Not taken
Guess: (Non Branch Instn coming)
So Next Sequential Instruction inserted into Pipeline.

wrong ✗ If we getting Branch Instruction (Prediction wrong)
then we have to Suffer from Stalls

these Two are Static Branch Prediction Method.

Dynamic Prediction.

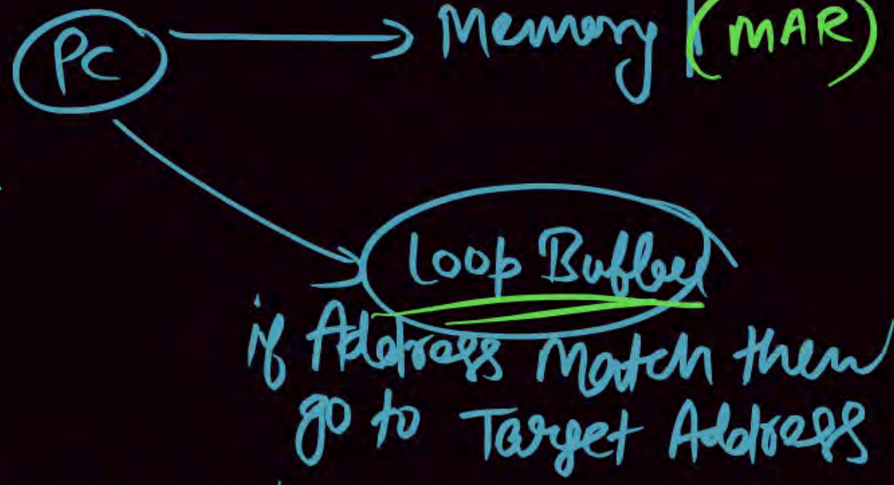
(i) bit Implemation.

(ii) Branch Target Buffer | Branch Loop Buffer | Branch Prediction Buffer.

Branch Loop Buffer : It is a High Speed Buffer, mainted in the Instruction Fetch Stage of Pipeline.

its Just kind a Database

Branch Instn Address	Target Address
1001	2000

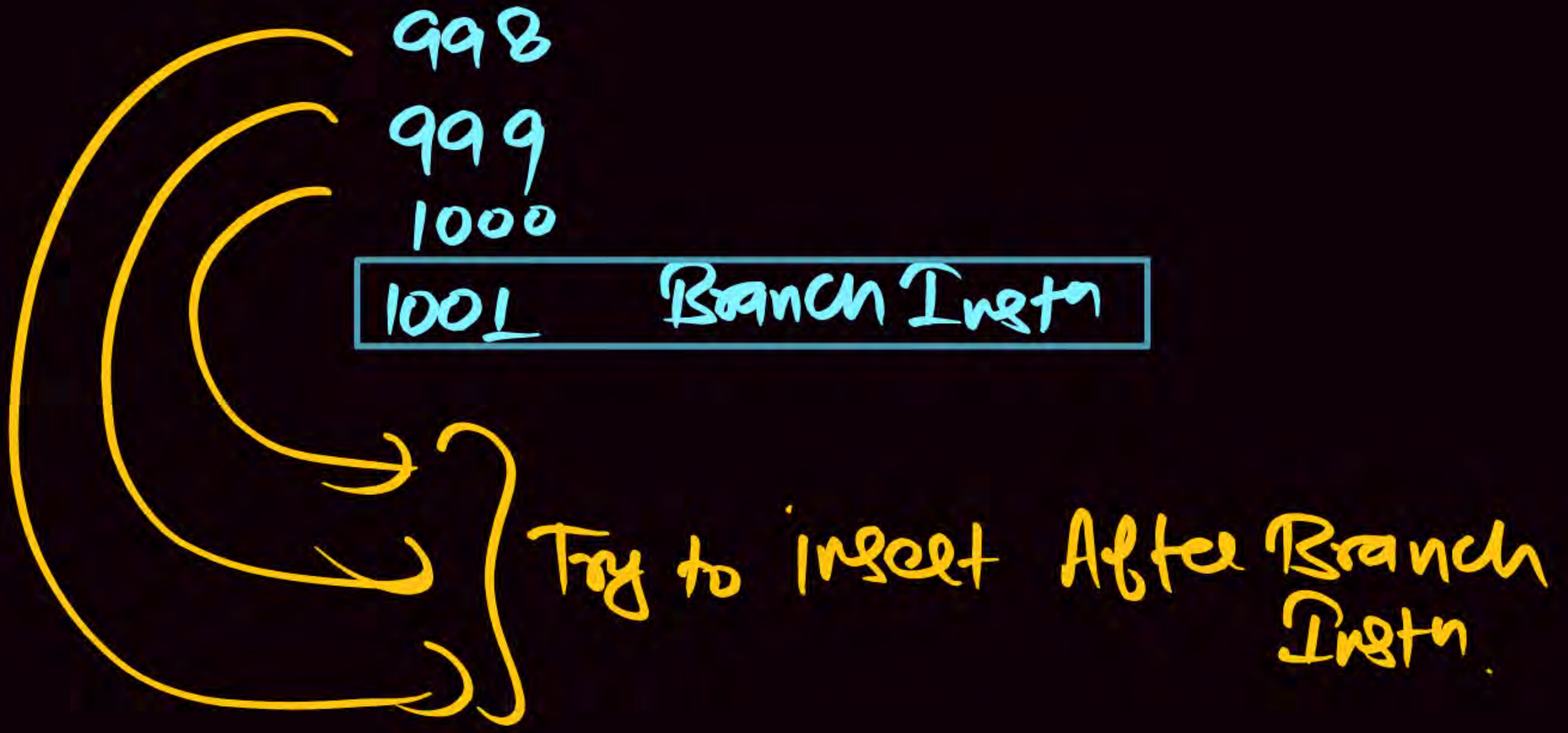


4
When the Next time this type of Instn Come then if Match with the address Directly getting the target Address.

Delayed Branch: It is a Compile technique [Slw Sw] in which trying to Rearrange the Instruction if possible.

If Not Possible to Rearrange then Insert 'NOP Instruction' After the Branch Instruction.

NOP Instruction: are the Instruction they Fetch & execute but Not Update Any Registed (or) Not effect our Desired output.


998
999
1000
1001 Branch Instn
Try to insert After Branch Instn.

Rearrange the Instn

Delay Slot

→ After the Branch Instruction

Independent Instruction. (which effect Prog Execution & output.)

RISC :: Target Address

2nd Stalls

Delay Slot = 2 - 1 = 1 Stall

1 Delay Slot

'K' Stage Target Address

Delay slot : (K-1) Delay slot.

Control Dependency/Control Hazards

Code: [with flush operation]:

1000	:	I_1
1001	:	I_2
1002	:	I_3 (JMP .2000)
1003	:	NOP
1004	:	I_4
.....		.....
2000	:	B I_1
2001	:	B I_2

Topic : Dependencies In he pipeline

PC: 1000	C1	C2	C3	C4	C5	C6
I_1	IF PC: 1001	ID	EX	MA	WB	
I_2		IF PC: 1002	ID	EX	MA	WB
NOP			IF PC: 1003	Unconditional TOC ID 2000 PC: ~~1004~~	EX	MA
I_4				IF 2000 PC: ~~1004~~	ID	EX
BI_1					IF PC: 2001	ID
BI_2						IF PC: 2002

∴ Actual O/P sequence: $I_1 - I_2 - I_3 -$ **NOP** $- BI_1 - BI_2$

Topic : Dependencies In he pipeline

Flush operation creates stall in the pipeline. Number of stalls created in the pipeline due to a branch instruction is called as “branch penalty”.

It is depending on the availability of a “Target Address” in the pipeline i.e.

Branch penalty = [At what stage target address available in the pipeline] –1

NOTE : RISC pipeline branch penalty is always one because target address is available in the second stage of a pipeline.

Topic : Dependencies In he pipeline

Number of stall created from the branches during the program execution is calculated as :

Stall = [Branch frequency] * [Branch penalty]

number stalls from Branches = Number of Branch * instruction in the prog. Number of Stalls/ Branch

Topic : Dependencies In he pipeline

- Analysis:

  Target address availability with respect to question

  1. Stage number given then

     Branch penalty = stage number –1

  2. Stage number given then

     Branch penalty = [Respective name stage position] –1

  3. Until Instruction execution is completed /All Instruction proceed with all stage then

     Branch penalty = [Last stage number] –1

Topic : Dependencies In he pipeline

NOTE: To minimize the control dependency stalls in the pipeline, hardware technique is used i.e., branch predication buffer [Branch target buffer or loop buffer]

It is a high-speed buffer present in the IF used to hold the predicted target address.

address is present in first address then stall is not created.

When the target address is present in first address then stall is not created. [Not practically present].

Topic : Dependencies In he pipeline

NOTE: When the question states that pipeline with branch prediction ,then assume that target address is available in the 1st stage so, Branch penalty '0'.

Other wise [pipeline without branch prediction] assume that target address is not present in the 1st stage. So, penalty depends on the target address availability in the pipeline.

Topic : Dependencies In he pipeline

Note: To minimize the control dependency stalls software technique is used i.e., Delayed Branch.

- Delayed Branch: It is a complier technique so, Complier rearranges the code, if possible, to arrange or substitute the non instruction after the branch instruction if not possible to rearrange to preserve the execution path.

Topic : Dependencies In he pipeline

Code:

1000	:	I_1
1001	:	I_2
1002	:	I_3 (JMP, 2000)
1003	:	I_4
.....		.....
2000		BI_1
2001		BI_2
.....		.....

Expected o/p seq.

$I_1 - I_2 - I_3 - BI_1 - BI_2$

Actual o/p Seq.

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

Topic : Dependencies In he pipeline

#

'Compiler'

'Re-arrangement'

Code:

System Generated Address :

I_1

I_3 (JMP 2000)

I_2

I_4

.....

BI_1

BI_2

.....

Topic : Dependencies In the pipeline

PC: I_1	C1	C2	C3	C4	C5
I_1	IF PC: I_3	ID	EX	MA	WB
I_3		IF PC: I_2	ID Unconditiona l TOC	EA	MA
I_2			IF BI_1 PC : BI_1	ID	EX
BI_1				IF PC : BI_2	ID
BI_2					IF PC : BI_3

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

Topic : Dependencies In he pipeline

'Compiler'

'Non- substitution'

Code:

System Generated address:

I_2

I_3 (JMP BI_1)

NOP

I_4

.....

BI_1

BI_2

.....

Topic : Dependencies In the pipeline

PC: I_1	C1	C2	C3	C4	C5	C6
I_1	IF PC: I_2	ID	EX	MA	WB	
I_3		IF PC: I_3	ID	EX	MA	WB
I_3			IF PC: NOP	ID Unconditional NOP. TOC	EX	MA
NOP				IF PC: BI_1	ID	EX
BI_1					IF PC: BI_2	ID
BI_2						IF PC: BI_2

$\therefore$ Actual O/P sequence: $I_1 - I_2 - I_3$ –NOP Stall – $BI_1 - BI_2$

Control Dependency

Generally CPI = 1

Avg Instruction ET = CPI X Cycle time

But Due to Control Hazard CPI ≠ 1

$$CPI = (1 + \#Stalls/Instn)$$ (per)

$$\#Stall\ Per\ Instn = Branch\ Instn\ Frequency \times Branch\ Penalty$$

$$Avg\ Instn\ ET = \left(1 + \#Stalls/Instn\right) \times Cycle\ time$$

$$Branch\ Penalty = (Target\ Address\ Stage) - 1$$

In Pipeline

$$\text{Avg Instn ET}_{\text{PIPE}} = (1 + \#\text{Stalls/Instn}) \times \text{Cycle time}$$

Speed Up factor with Stalls

Performance analysis with stalls :

$$S = \frac{\text{Avg. Instruction ET}_{\text{nonpipe}}}{\text{Avg. Instruction ET}_{\text{pipe}}}$$

$$S = \frac{ET_{NP}}{ET_{P}}$$

$$S = \frac{\text{CPI}_{\text{nonpipe}} \times \text{cycle time}_{\text{non pipe}}}{\text{CPI pipe} \times \text{cycle time}_{\text{pipe}}}$$

Speed Up factor with Stalls

Ideal CPI of pipeline is always '1' due to a dependency problem some stalls are created in the pipeline.

$$S = \frac{\text{CPI}_{\text{nonpipe}} \times \text{cycle time}_{\text{nonpipe}}}{(1+\text{number of stalls /Instruction})(\text{cycle time}_{\text{pipe}})}$$

When the pipeline stages are perfectly balanced then 1 task execution time in the non-pipeline is also equal to number of stages in the pipeline i.e.....

$t_n = k.\ t_p$

under this condition

$$S = \frac{K.\ t_p}{(1 + \text{number of stalls /Instruction})(\text{cycle time}_{\text{pipe}})}$$

$$S = \frac{(\text{Pipeline Depth})K}{(1+\text{number of stalls /Instruction})}$$

$t_n = 2+2+2+2 = 8$

t_n

$8 = 4 \times 2\text{nsec}$

$t_n = 8\text{nsec}$

Q

A CPU has five-stages pipeline and runs at 1 GHz frequency. Instruction fetch happens in the first stage of the pipeline. A conditional branch instruction computes the target address and evaluates the condition in the 'third stage' of the pipeline. The processor stops fetching new instructions following a conditional branch until the branch outcome is known. A program executes 10^9 instructions out of which 20% are conditional branches. If each instruction takes one cycle to complete on average, then total execution time of the program is

[GATE-2006 : 2 Marks]

A 1.0 second

B 1.2 second

C 1.4 second

D 1.6 second

Frequency = 1GHz ⇒ Cycle time = $\frac{1}{1GHz}$ sec = 1nsec

5 Stage Pipeline

Total = 10^9 Instn.

Target Address = 3rd Stage

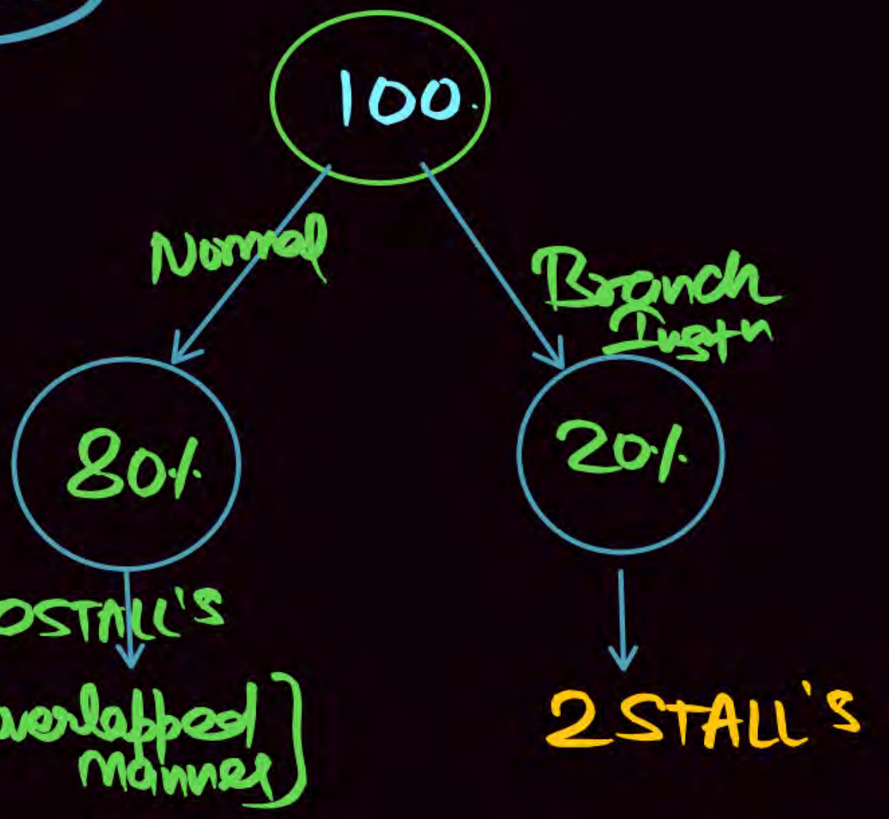
Best Approach

Branch Penalty = 3 − 1 = (2) Stalls.

Avg Instn ET = (1 + #Stall/Instn) × Cycle time

⇒ (1 + 0.4) × 1nsec

Avg Instn ET = 1.4 nsec



#Stalls/Instn = .80X0 + .20X2

#Stall/Instn = 0.4.

Program Contain = 10^9 Instn

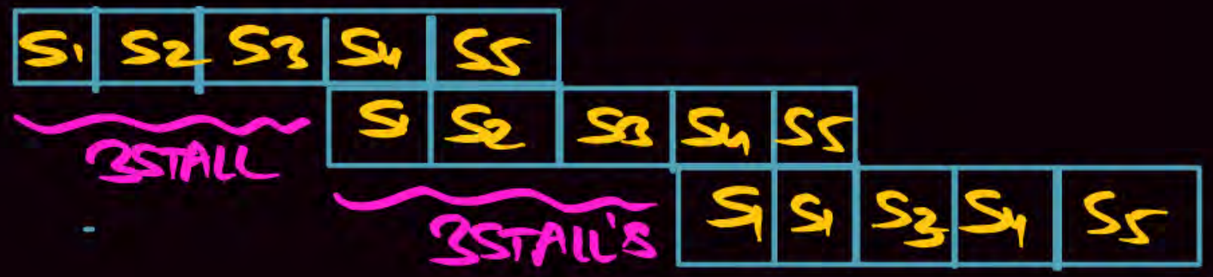
Prog. ET = $1.4 \times 10^{-9} \times 10^9$

= 1.4 sec Avg

Another Approach.

5Stage Pipeline.

3rd STAGE



$$\text{Avg CPI} = .80 \times 1 + .20 \times 3$$

$$= 0.8 + 0.6$$

$$\boxed{\text{Avg CPI} = 1.4 \text{ Cycle}}$$

$$\text{Prog ET} = 1.4 \times 10^{9} \times 10^{9}$$

$$= 1.4 \text{ Sec}$$

$$\text{Avg Instn ET} = 1.4 \times 1\text{nsec}$$

$$= 1.4 \text{ nsec}$$

Q Consider a 5-stage instruction pipeline, where all stage are perfectly balanced. & have a cycle time of 20ns. Target address of branch Instruction is available at the end of the execution(last Stage), there are 30% instruction.

(i)What is Average Instruction Execution time(ignore the fact some are conditional)? 44 nsec

(ii) What is the speed up factor? 2.27 nsec.

(iii) Among the branch instruction 30% are conditional & 70% of them does not satisfy the condition then What is Average Instruction Execution time ?

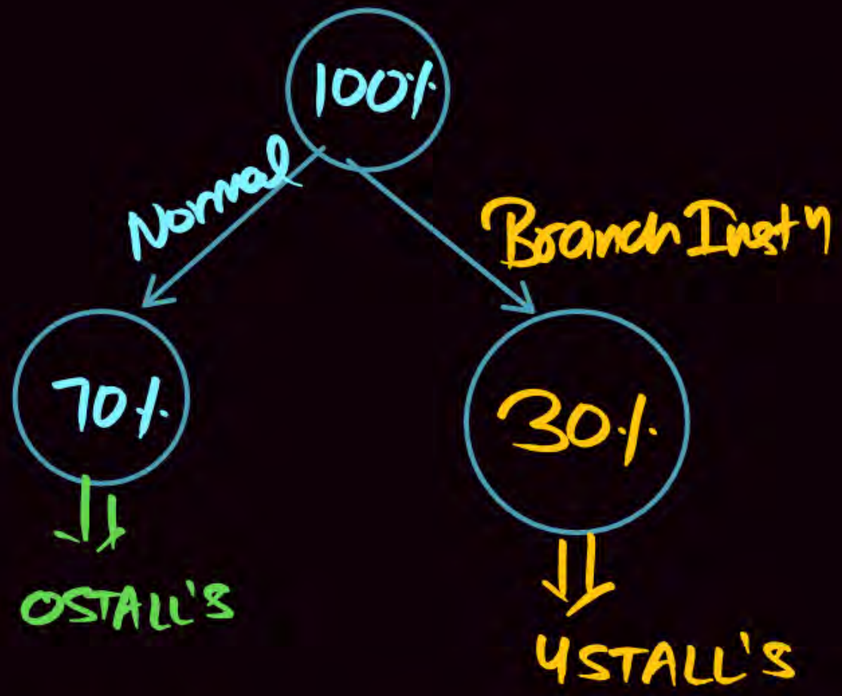
(1) # Stage = 5

Cycle time = 20 nsec

30% Branch Instn.

Target Address is Available at the end of LAST Stage (5th Stage)

Branch Penalty = 5 − 1 = (4)



(1) Avg Instn ET_{PIPE} = (1 + #Stalls/Instn) × cycle time

$\Rightarrow (1 + 1.2) \times 20$

$= 2.2 \times 20$

(1) $ET_{PIPE} = 44$ nsec Ans

#Stalls/Instn = .70 × 0 + .30 × 4

#Stalls/Instn = 1.2

9 (ii) Speed up Factor $= \frac{ET_{NP}}{ET_{PIPE}}$

$\Rightarrow \frac{5 \times 20 \text{ nsec}}{(1 + \#\text{Stalls/Instn}) \times \text{Cycle}}$.

$\Rightarrow \frac{5 \times 20}{(1+1.2) \times 20}$

$= \frac{5}{2.2}$

(ii) $\boxed{S = 2.27 \text{ Ans}}$

Perfectly balance

$\Rightarrow \frac{5}{(1 + \#\text{Stalls/Instn})}$

$= \frac{5}{1+1.2} = 2.27$

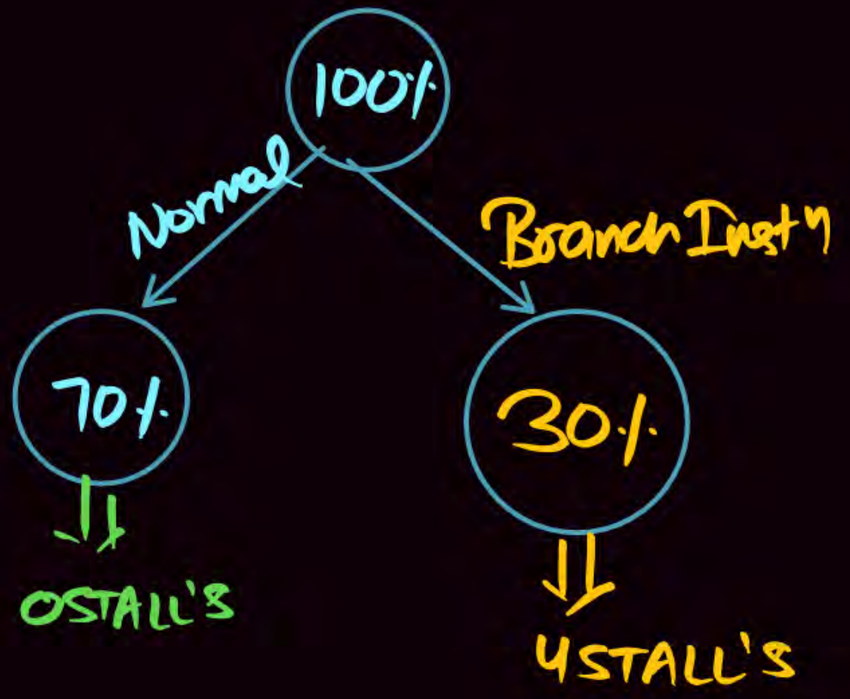
(3) # Stage = 5

Cycle time = 20 nsec

30% Branch Instn.

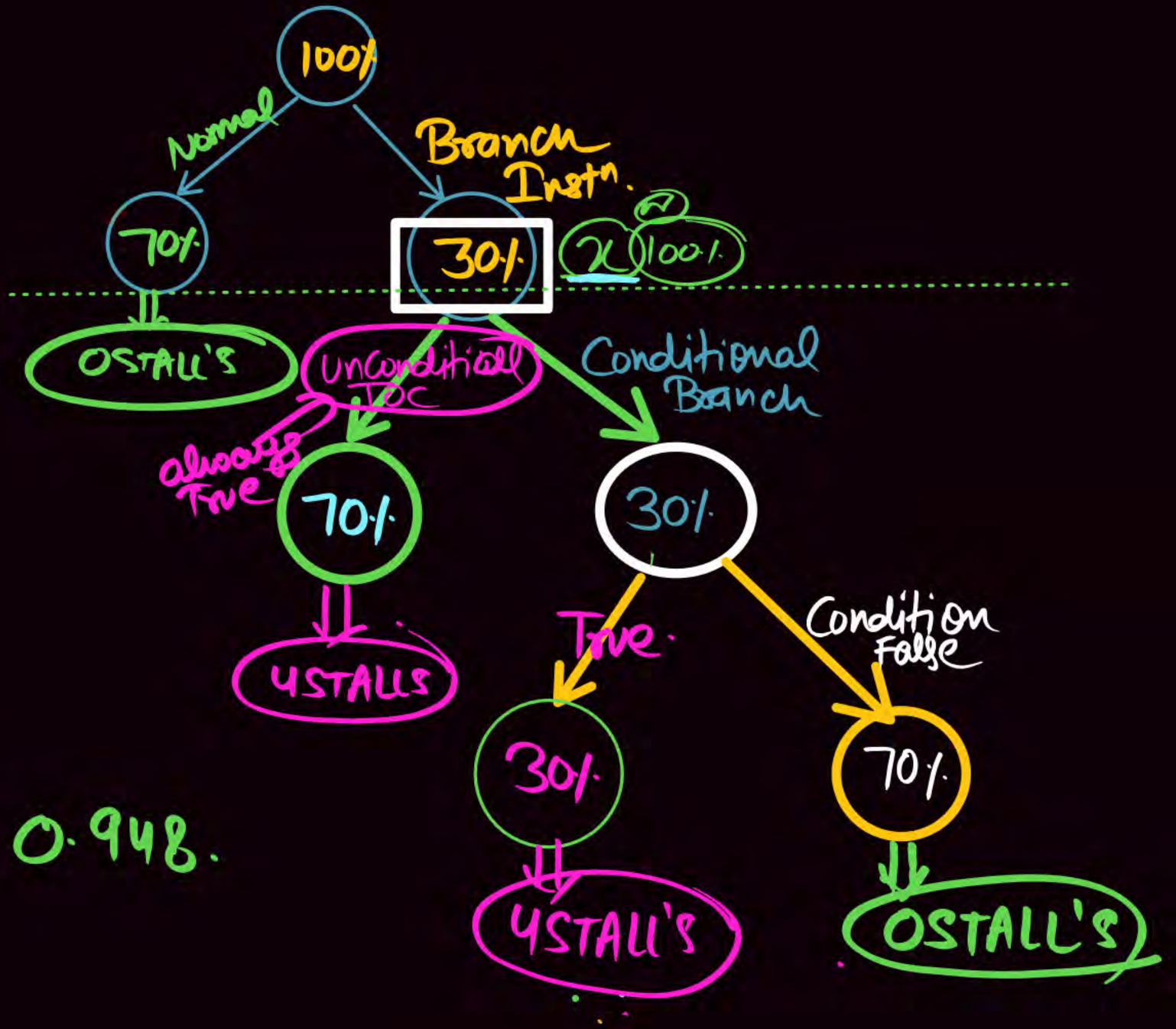
Target Address is Available at the end of LAST Stage (5th Stage)

Branch Penalty = 5 − 1 = (4)



Now Complete for 3rd Question Diagram

(iii)



#Stalls/Instn =

- 70 X 0 +
- 30 X . 70 X 4 +
- 30 X . 30 X . 30 X 4 +
- 30 X . 30 X . 70 X 0 = 0.948.

Avg Instn ET = (1 + #Stalls/Instn) x cycle time

$\Rightarrow (1 + 0.948) \times 20$

$= 1.948 \times 20$ ns

ET = 38.96 nsec Ans

Q

Consider a 6-stage instruction pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6-stage pipeline, the speedup achieved with respect to non-pipelined execution if 25% of the instructions incur 2 pipeline stall cycles is _____.

[GATE-2014(Set-1) : 2 Marks]

k=6 Stage

Assume time = tc.

$$tn = 6tc$$

$$tn = k * tp$$

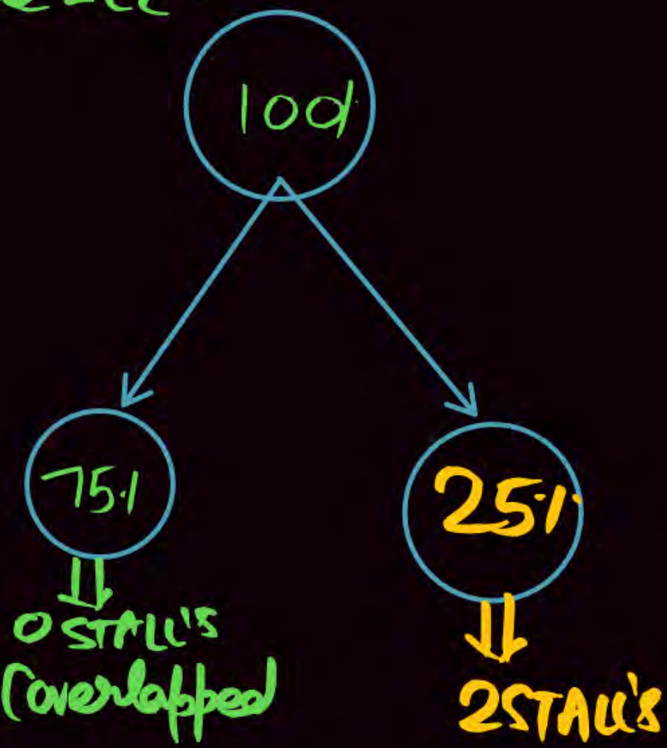
$$S = \frac{ET_{NONPIPE}}{ET_{PIPE}}$$

$$\Rightarrow \frac{6 * tc}{(1 + \#Stalls/Instn) \times Cycle\ time}$$

$$\Rightarrow \frac{6 * tc}{(1 + 0.5) * tc}$$

$$= 6/1.5 = 4$$

S = 4



$$\#Stall/Instn = .75 \times 0 + .25 \times 2$$

$$\#Stalls/Instn = 0.5$$

Q. 6

HW

An instruction pipeline has five stages, namely, instruction fetch (IF), instruction decode and register fetch (ID/RF), instruction execution (EX), memory access (MEM), and register writeback (WB) with stage latencies 1ns, 2.2ns, 2ns, 1ns. and 0.75ns, respectively (ns stands for nanoseconds). To gain in terms of frequency, the designers have decided to split the ID/RF stage into three stages (ID. RF1, RF2) each of latency 2.2/3 ns. Also, the EX stage is split into two stages (EX1, EX2) each of latency 1ns. The new design has a total of eight pipeline stages. A program has 20% branch instructions which execute in the EX stage and produce the next instruction pointer at the end of the EX stage in the old design and at the end of the EX2 stage in the new design. The IF stage stalls after fetching a branch instruction until the next instruction pointer is computed. All instructions other than the branch instructions have an average CPI of one in both the designs. The execution times of this program on the old and the new design are P and Q nanoseconds, respectively. The value of P/Q is _____.

[GATE-2014 (Set-3): 2 Marks]

NAT

Consider a non-pipelined processor operating at 2.5 GHz. It takes 5 clock cycles to complete an instruction. You are going to make a 5-stage pipeline out of this processor. Overheads associated with pipelining force you to operate the pipelined processor at 2 GHz. In a given program, assume that 30% are memory instructions, 60% are ALU instructions and the rest are branch instructions. 5% of the memory instructions cause stalls of 50 clock cycles each due to cache misses and 50% of the branch instructions cause stalls of 2 cycles each. Assume that there are no stalls associated with the execution of ALU instructions. For this program, the speedup achieved by the pipelined processor over the non-pipelined processor (round off to 2 decimal places) is 2.16 Ans

[GATE-2020-CS: 2M]

Pipeline Cycletime $= \frac{1}{2GHz} = 0.5$ nsec

#Stalls/Instn =

$.30 \times 0.05 \times 50$

$+ .60 \times 0$

$+ .10 \times .50 \times 2$

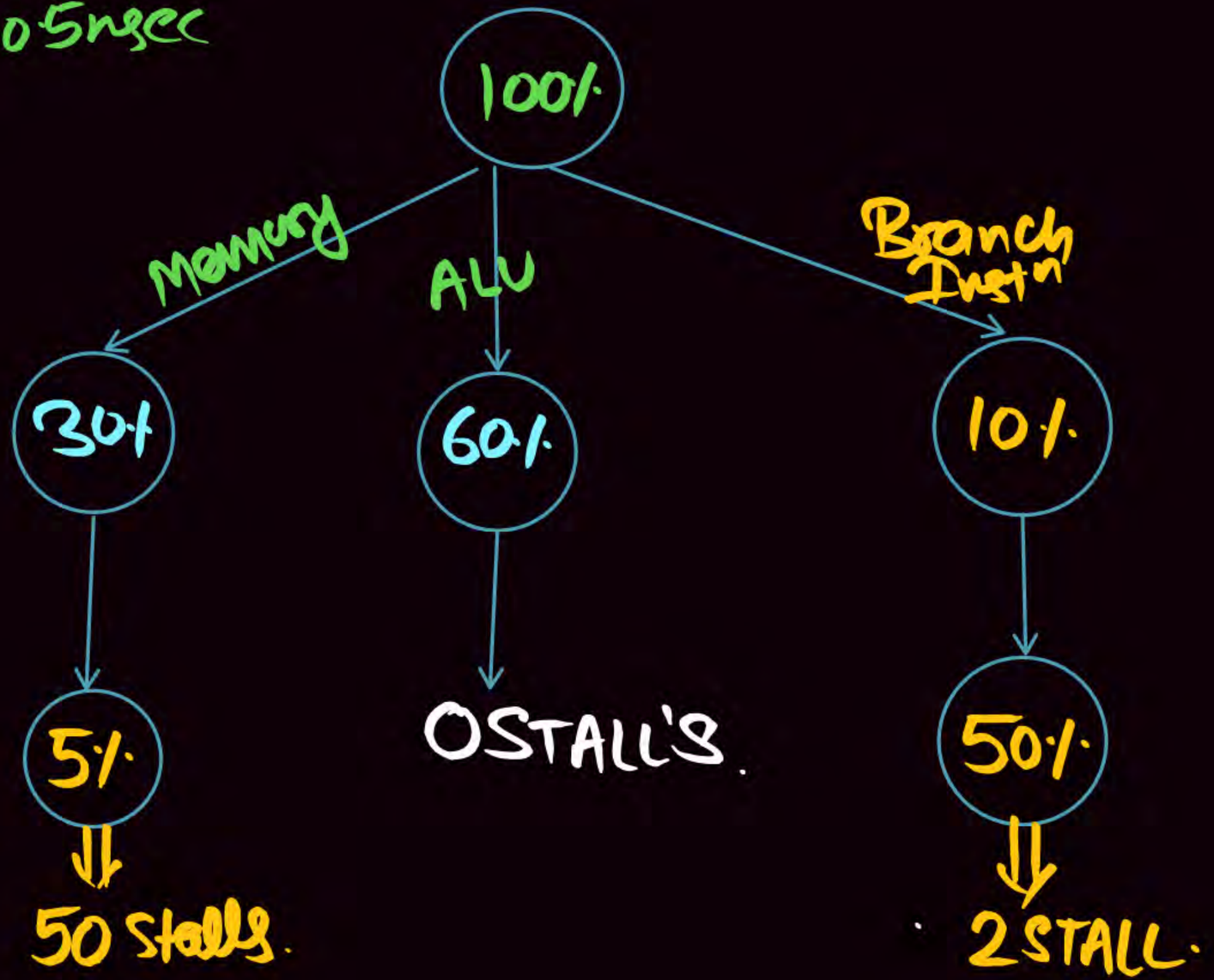
#Stall/Instn = 0.85

Avg Instn ET = $(1 + \#Stall/Instn) \times$ Cycle time

$\Rightarrow (1 + 0.85) \times 0.5$

$= 1.85 \times 0.5$

ETPIPE = 0.925 nsec



Non pipeline

5 clock cycle

2.5 GHz

Cycletime $= \frac{1}{2.5G}$ sec $=$ 0.4 nsec

$ET_{Nonpipe} = 5 \times 0.4$ nsec

$\boxed{ET_{NP} = 2 \text{ nsec}}$

$$S = \frac{ET_{NP}}{ET_{PIPE}} = \frac{2}{0.925}$$

$$\boxed{S = 2.16}$$ Ans

MCQ

Register renaming is done in pipelined processors

[GATE-2012-CS: 1M]

A. As an alternative to register allocation at compile time

B. For efficient access to function parameters and local variables

C. To handle certain kinds of hazards

D. As part of address translation

MCQ

Delayed branching can help in the handling of control hazards.
The following code is to run on a pipelined processor with one branch delay slot:

I1: ADD R2 ← R7 + R8
I2: SUB R4 ← R5 - R6x
I3: ADD R1 ← R2 + R3
I4: STORE Memory[R4] ← R1
BRANCH to Label if /R1 = = 0

Which of the instruction I1, I2, I3 or I4 can legitimately occupy the delay slot without any other program modification?

[GATE-2008-CS: 2M]

A I1

B I2

C I3

D I4

Ans (D)

(a) I_1 : $R_2 \leftarrow R_7 + R_8$

the o/p of I_1 (R_2 Value) is used in the I_3 as a Input So I_1 X

I_1 Can not Insert in o Delay Slot.

(b) I_2 : $R_4 \leftarrow R_5 - R_6$.

this R_4 is using in M[R_4] storing the Result

So I_2 Can not Insert in Delay Slot.

(c) I_3 : $R_1 \leftarrow R_2 + R_3$.

This R_1 Value using in the I_4 & Condition checking as well.

So I_3 Can not insert in Delay Slot

(d) I_4 ✓

Can Insert in Delay Slot Bc2

No Changes in Program.

THANK YOU!