COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_07

Vijay Agarwal sir

01 Pipelining Hazards

Pipelining Hazards

1. Structural Dep. | Structural Hazards
2. Data Dependency | Data Hazards
3. Control Dependency | Control Hazards

RISC PIPELINE

In The RISC Pipeline 5 Stages:

1. Instruction Fetch {IF Stage}
2. Instruction Decode{ID Stage}
3. Execute {EX Stage}
4. Memory Access {MA Stage}
5. Write Back {WB Stage}

RISC PIPELINE

1. Instruction Fetch {IF Stage}: In this stage Instruction is fetched from Memory.

2. Instruction Decode{ID Stage} : In this Stage 2 operation are performed:

   (i) Decode the instruction

   (ii) Operand loading(fetching) from the register file .

This stage also contain comparator circuit to evaluate the branch condition.

RISC PIPELINE

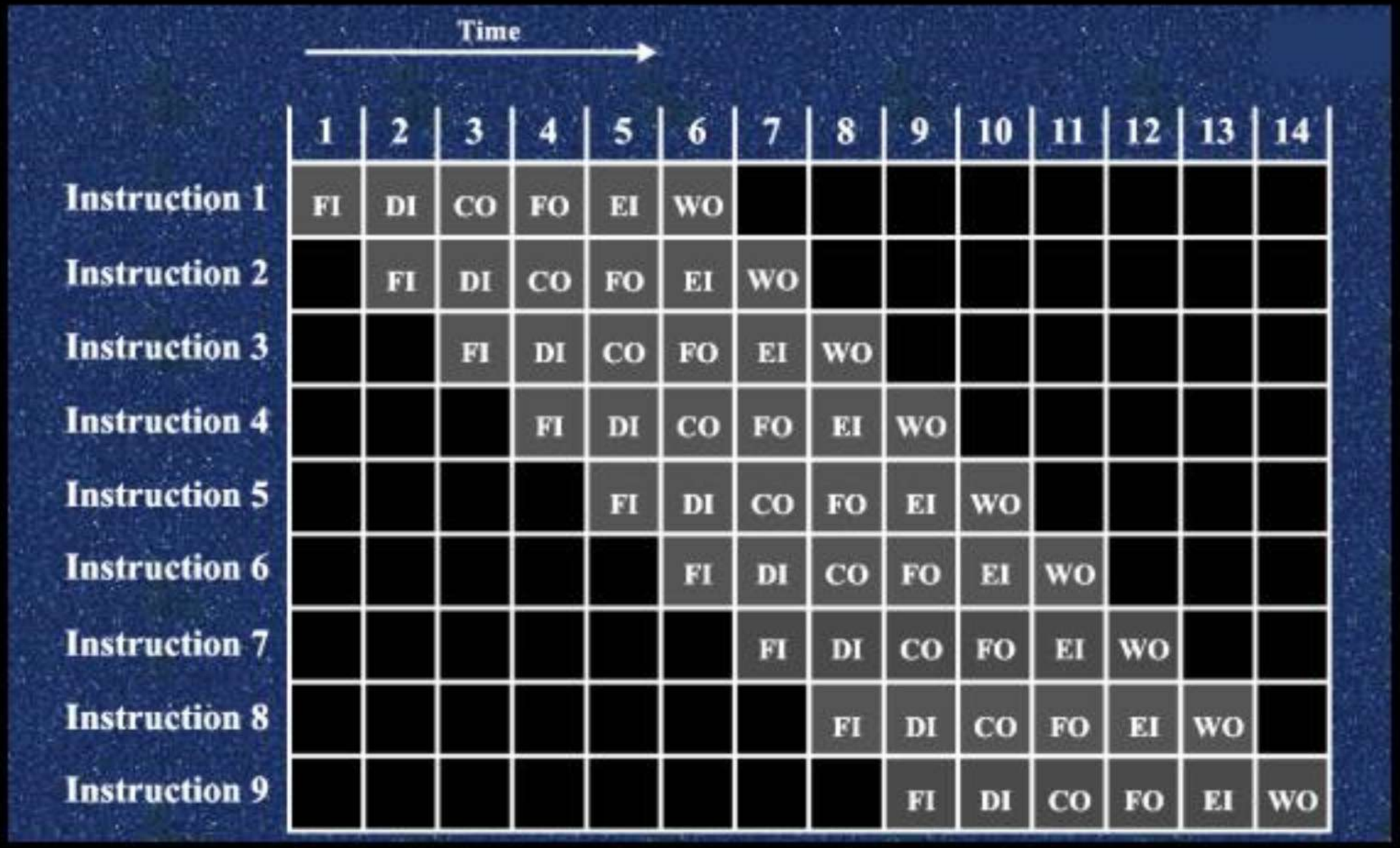
3. Execute {EX Stage}: In this stage Data Processing (ALU Operations) are performed

4. Memory Access {MA Stage} : In this Stage Operand (Data) will be accessed from memory(load or store).

5. Write Back {WB Stage} : In this stage Register write (Operand storing into reg. file) operation performed.

Additional Stages

- **Fetch Instruction (FI)**
  - Read the next expected Instruction into a buffer.
- **Decode Instruction (DI)**
  - Determine the opcode and the operand specifiers.
- **Calculate operands(CO)**
  - Calculate the effective address of each source operand.
  - This may involve displacement, register indirect or other forms of address calculations.
- **Fetch Operands(FO)**
  - Fetch each operand from memory.
  - Operands in register need not be fetched.
- **Executed Instruction(EI)**
  - Perform the indicated operation and store the result, if any, in the specified destination operand location
- **Write Operand(WO)**
  - Store the result in memory



Timing Diagram for Instruction pipeline operation

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:

- Resource
- Data
- Control

Also referred to as a Pipeline bubble | Stalls / Extra Cycle.

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

Due to Resource Conflict

Solution

Resource Replication | Duplication

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 Dependency

① RAW (True Data Dep.)

Solution of True Data Dep. (i) Hardware Interlock

(ii) Operand forwarding / Bypassing / short circuiting.

Data Dependency

Data dependency will be occurred when the instruction "J" try to read the data before instruction "I" writes it.

"Read – Before – write"

Ex.-

I_1: Add $r_0, r_1, r_2 : r_0 \leftarrow r_1 + r_2$

I_2: MUL $r_3, r_0, r_2 : r_3 \leftarrow r_0 * r_2$

Data Dependency

eg. Consider the following program segment, executed on a RISC pipeline with a operand forwarding technique.

I_1: Add r0 , r1, r2
I_2: Sub r3 , r0, r2
I_3: MUL r4 , r3, r0
I_4: DIV r5 , r4, r0

Non- Adjacent dependency

Note: (1) Adjacent data dependency is named as true data dependency i.e.

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

Data Dependency

(2) Non-Adjacent data dependency is named as data dependency only.

i.e. $I_3 - I_1$

$I_4 - I_1$

Q. Consider a pipelined processor with the following four stages

IF : Instruction Fetch

ID : Instruction Decode and Operand Fetch

EX : Execute

WB : Write Back

The IF, ID and WB stages take one clock cycle each to complete the operation. The number of clock cycles for the EX stage depends on the instruction. The ADD and SUB instructions need 1 clock cycle and the MUL instruction need 3 clock cycles in the EX stage. Operand forwarding is used in the pipelined processor. What is the number of clock cycles taken to complete the following sequence of instructions?

ADD	R2	R1	R0	R2 ← R1 + R0
MUL	R4,	R3,	R2	R4 ← R3 * R2
SUB	R6,	R5,	R4	R6 ← R5 –R4

[GATE-2007 : 2 Marks]

A 7

B 8

C 10

D 14

Q. The performance of a pipelined processor suffer if

[GATE-2002 : 2 Marks]

A. the pipeline stages have different delays

↳ Control Dep.

B. consecutive instructions are dependent on each other

↳ Data Dep.

C. the pipeline stages share hardware resources

↳ Structural Dep.

D. All of the above

MCQ

A 5-stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Write Operand (WO) stage. The IF, ID, OF and WO stage take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instructions ,3 clock cycles for MUL instruction, and 6 clock cycles for DIV instruction respectively. Operand forwarding is used in the pipeline. What is the number of clock cycles needed to execute the following sequence of instructions?

Instruction	Meaning of Instruction
I_0: MUL R_2, R_0, R_1	$R_2 \leftarrow R_0 * R_1$
I_1: DIV R_5, R_3, R_4	$R_5 \leftarrow R_3 / R_4$
I_2: ADD R_2, R_5, R_2	$R_2 \leftarrow R_5 + R_2$
I_3: SUB R_5, R_2, R_6	$R_5 \leftarrow R_2 - R_6$

[GATE-2010-CS: 2M]

A 13

B 15

C 17

D 19

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 → Data Dep./Data Hazards.
2. The execution of a conditional jump instruction → Control Hazard.
3. The j^{th} and $j^{th} + 1^{st}$ instructions require the ALU at the same time → Structural Dep/Str. Hazards.

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

4 - 1 = 3

FI DI FO EX WO

5 7 10 8 6

$I_1\ I_2\ I_3\ I_4\ I_5\ I_6\ I_7\ I_8\ I_9\ I_{10}\ I_{11}\ I_{12}$

out of 12 we execute = 8 Instn

Bubble Delay = 1

$t_p = 10 + 1$

$t_p = 11$ nsec

$ET_{PIPE} = (k + (n-1))$ cycle

$\Rightarrow (5 + 8 - 1) = 12$ cycle.

Extra Cycle (Stalls) = 3 Stalls.

15 Cycle

Cycle time = 11ns

$15 \times 11 = 165$ nsec

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

① RAW [Read After Write] True Data Dep.

② WAR [Write After Read] ANTI Dep.

③ WAW [Write After Write] Output/write Dependency.

Types of Data Hazard

WAR, WAW — Does not Create Any Stall (Extra Cycle)

→ Solution is Register Renaming.

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 Hazards

in Pipeline

Ideal CPI = 1

Avg CPI = 1

How Avg CPI = 1 ?

$ET_{PIPE} = [k + (n-1)]$ cycle

$Avg\ ET_{PIPE} = \frac{[k + (n-1)]\text{ cycle}}{n} = \frac{k - 1 + n}{n}$

for very # Instn

Ignore k-1

$= 1$ CPI = 1

Control Dependency/Control Hazards

Due to Control Hazard Extra Cycle | Stall Created.

$$\text{Branch Penalty} \left(\begin{matrix}\#\text{Stalls} \\ (\text{Extra Cycle})\end{matrix}\right) = k - 1$$

$$\#\text{Stalls Due to Branch Instr}^{n} = \#\text{of Branch type Instr}^{n} \times \#\text{Stalls}.$$

Control Dependency/Control Hazards

Stall Per Instn Due to Control Hazard

$$\#\text{Stalls/Instn} = \text{Branch Penalty} \times \text{Branch Instn Frequency}$$

Avg CPI = 1 + Rut Due to Control Hazards

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

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: $I_1 - I_2 - I_3 - BI_1 - BI_2$

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

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	

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

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

Speed Up factor with Stalls

Performance analysis with stalls :

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

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

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

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

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

Control Hazards | Dependency.

- (1) Flushing
- (2) Stalling :
- (3) Rearrange | Delay Branch
- (4) NOP Instruction
- (5) Branch Prediction.

Stalling :: Design a pipeline in such a manner if it Detect the Branch Instruction then we have to Stall.

(Stop the Next (Renaming) instruction into pipeline Untill the Branch Instruction Execution Not Completed.

Solution of Control Dependency.

1000 I_1

1001 I_2 JMP 2000

1002 I_3

I_1 I_2 ~~I_3~~

1000 – 1001 – ~~1002~~

1000 – 1001 – 2000

I_1 I_2 BI_1

2000 BI_1

2001 BI_1

Branch Predicition: (Guess)

In this we Predicit a Branch is taken (OR) Not Taken, if Predicition is Correct then Accordingly Instn will be execute.

Design the Processor

(i) **Branch is always Taken** (In this we Assume every time Branch taken So Next Instn is target Adding Instn)

(ii) **Branch is Never Taken**: (In this always Next Sequential Instn in the Execution)

Note: If Predicition goes Wrong then Have to Suffer from Stalls

(i) Branch is always taken: (1001 - 2000 - 2001 - 2002 ...)

↓ Predicition wrong.

↳ If its Normal Instn then Predicition wrong & Suffer Stalls

If we get the Normal (Non-Branch) Instruction then Suffer from Stalls.

(ii) Branch Never taken. [eg I_1 1000, I_2 1001, I_3 1002, I_4 1003, I_5 1004, ... Execution

In this type of Processor we Predicit Branch Instn Not taken & in this always Next Sequential [Consective] Instn inserted

Wrong

But if we getting Branch Instn (Prediction Wrong) then Suffer from Stalls.

Note

But these 2 Approach are Static [at Design Time] we Fixed the Action.

(iii) Dynamic Poredicition (① bit Implemantation)

② Branch Target Bubbles | Branch Loop Bubbles)

Branch Loop Buffer:

It is the High Speed Buffer maintained in the Instruction Fetch of the Pipeline.

Just a kind of Database.

Branch Instn Address	Target Address
1001	2000

When Next time this type of Instn we get the Directly getting the Target Address.

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'.

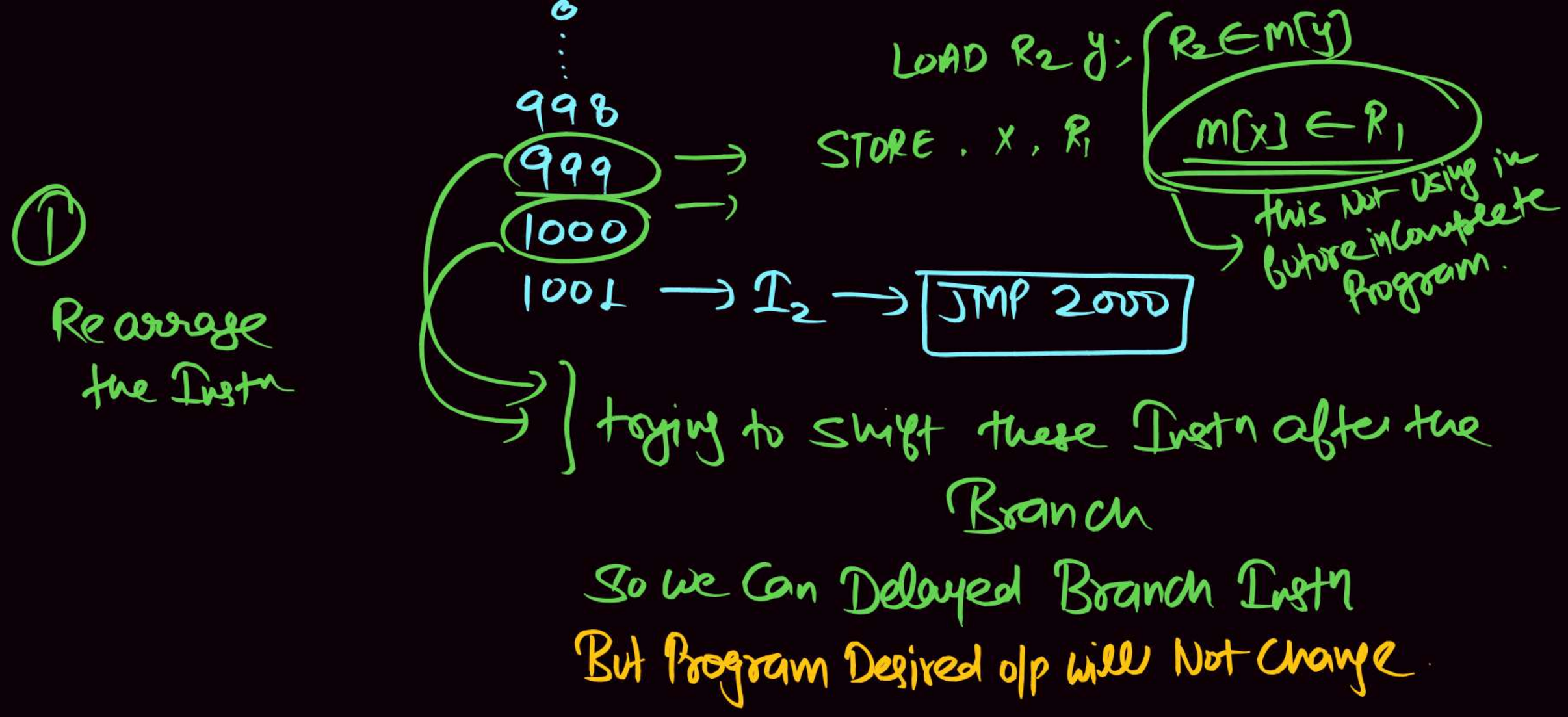
Other wise [pipeline without branch prediction] assume that target address is not present in the 1st stage. So, penalty depends on the target address availability in the pipeline.

Delayed Branch: It is a Compiler Solution (Slw type Solution)

In which Compiler try to Rearrange the Instruction if Possible.

(OR) If Not Possible to Rearrange then Insert the NOP operation [No-operation] After the Branch Instruction.

NOP operation/Instruction: are the Instruction then Fetch & execute but Not effect/Update the any Register (or) Not effect the Program Desired output.



So we Can Delayed Branch Instn

But Program Desired o/p will Not Change.

Topic : Dependencies In he pipeline

#

'Compiler'

'Re-arrangement'

Code:

System Generated Address :

I_1

I_3 (JMP 2000)

I_2

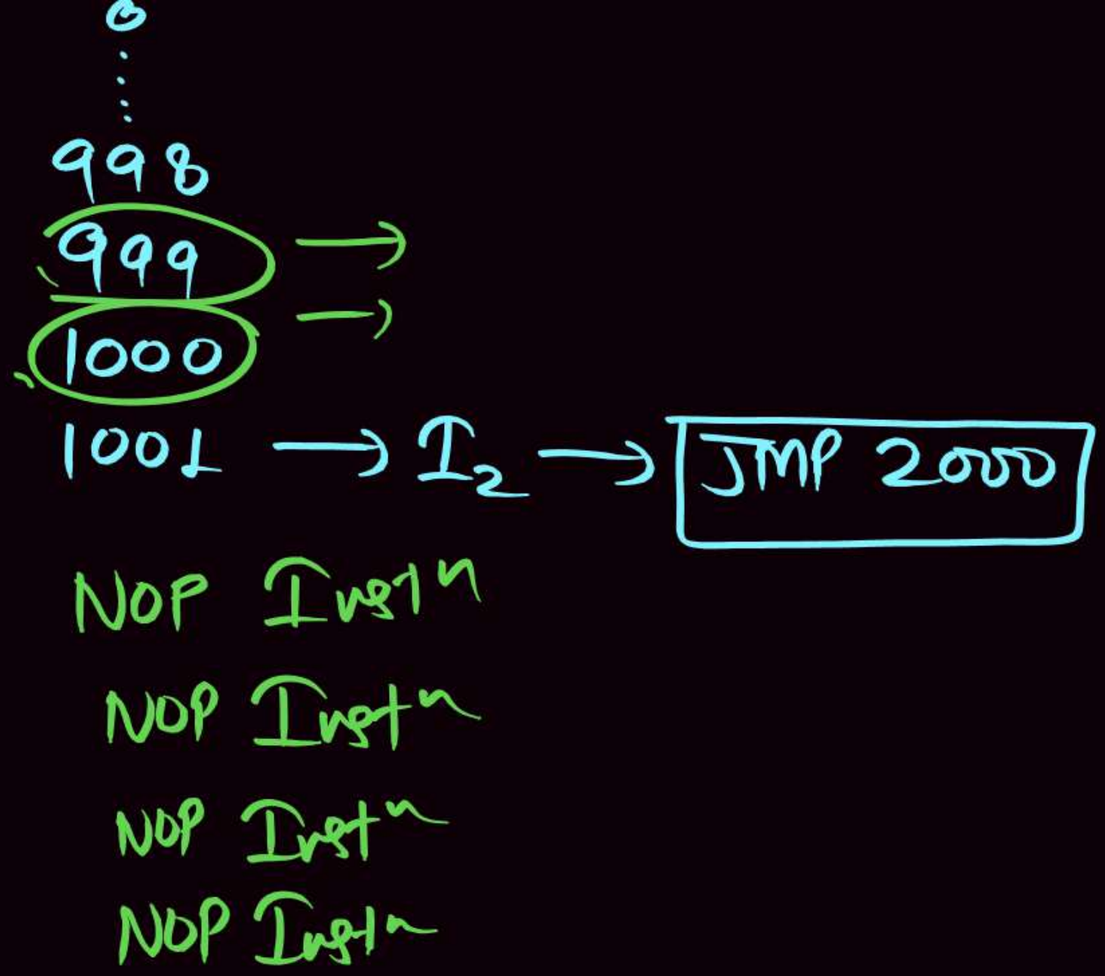
I_4

.....

BI_1

BI_2

.....



(2)
Insert
NOP operation.

Topic : Dependencies In he pipeline

'Compiler'

'Nop- substitution'

Code:

System Generated address:

I_2

I_3 (JMP BI_1)

NOP

I_4

.....

BI_1

BI_2

.....

Q

Cycle time = 1nsec

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

Solved by 2 Approach.

Cycle time = 1 nsec.

Branch Instn freq = 20%

Target address = 3rd stage

Program Contain = 10^9 Instn

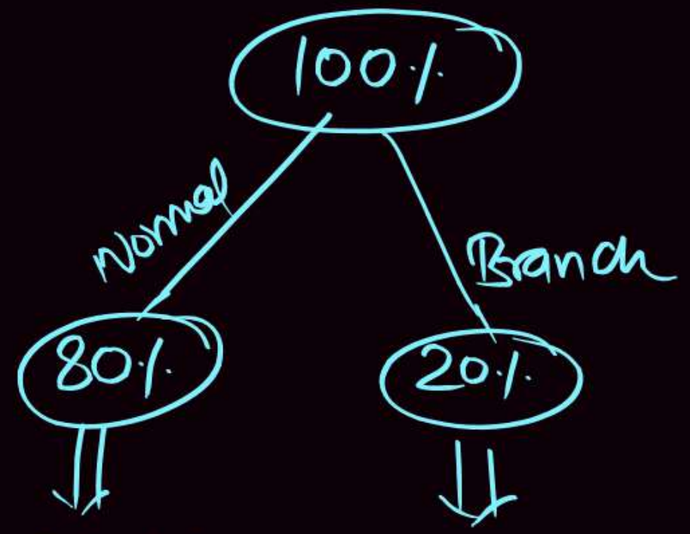
BEST Approach

Branch Penalty = 3 − 1 = (2)

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

⇒ (1 + 0.4) × 1 nsec = 1.4 nsec

Program ET = $10^9 \times 1.4 \times 10^{-9}$ = (1.4 sec) Ans



#Stalls/Instn = .80 × 0 + .20 × 2

#Stall/Instn = 0.4

Cycle time = 1 nsec.

Branch Instn freq = 20%

Target address = 3rd stage.

Program contain = 10^9 Instn

Branch Penalty = 3 − 1 = 2

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

⇒ (1 + 0.4) × 1 nsec = 1.4 nsec

Program ET = 10^9 × 1.4 × 10^{-9}

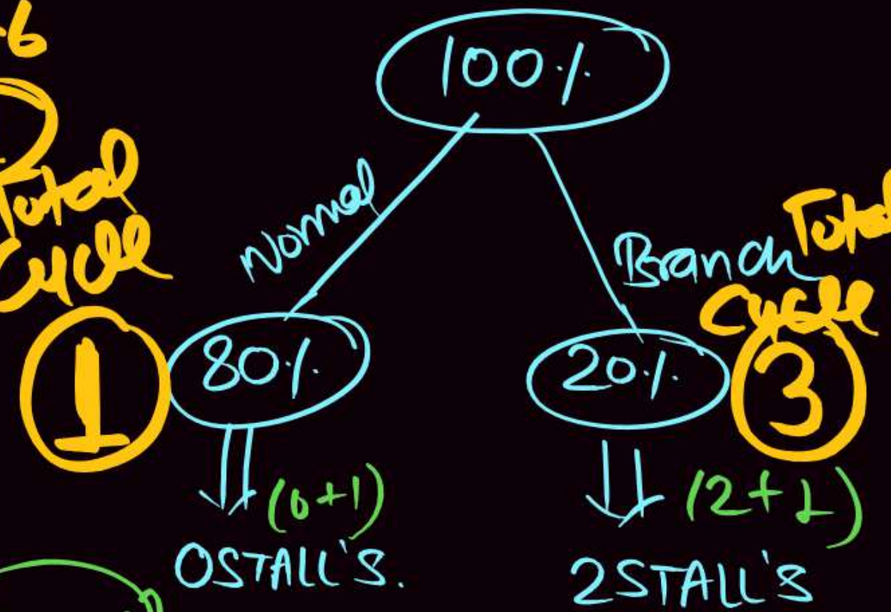
= 1.4 sec Ans

.80 × 1 + .20 × 3

0.8 + 0.6

= 1.4 cycle

Total cycle



BEST Approach

In this Approach [1 + Extra cycle Stalls]

CPI

Control Hazard

#Stalls/Instn = .80 × 0 + .20 × 2

#Stall/Instn = 0.4

Extra cycle/Instn

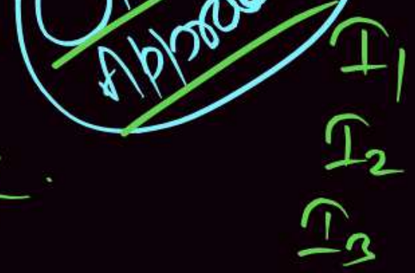
Cycle time = 1 nsec.

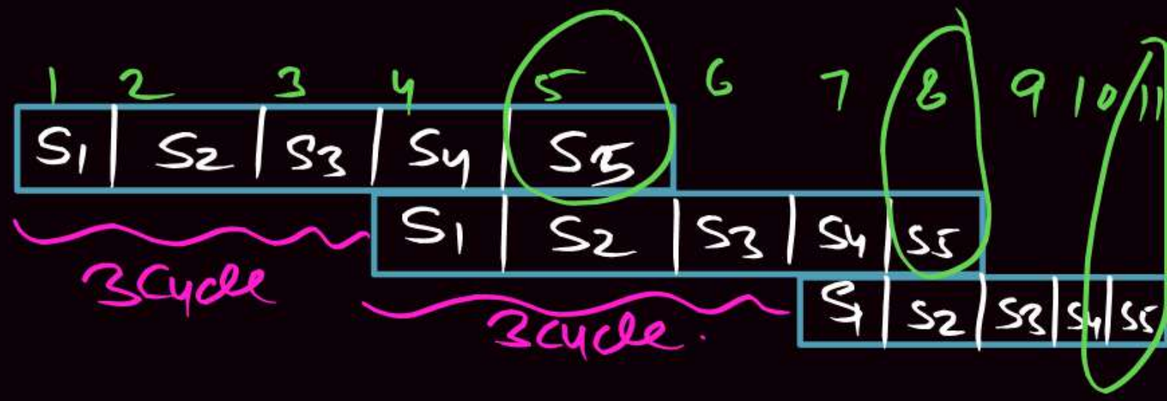
Branch Instn freq = 20%

Target address = 3rd stage.

Program contain = 10^9 Instn

Other Approach





CPI = 3 for Branch Instn } forcefully 3rd cycle 1 Instn o/p getting

Avg CPI = (1 + .20 × 2) (Extra Cycle)

= 1 + 0.4

= 1.4 cycle

Avg CPI = .80 × 1 + .20 × 3

= 0.8 + 0.6 = 1.4 Cycle

Q Consider a 5-stage instruction pipeline, where all stages 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)? Ans) 44 nsec.

(ii) What is the speed up factor?

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

k = 5 Stage

Cycle time = 20nsec

Target Address = at the last Stage [5th Stage]

Branch Instn Frequency = 30%

Branch Penalty = Target Address Stage − 1

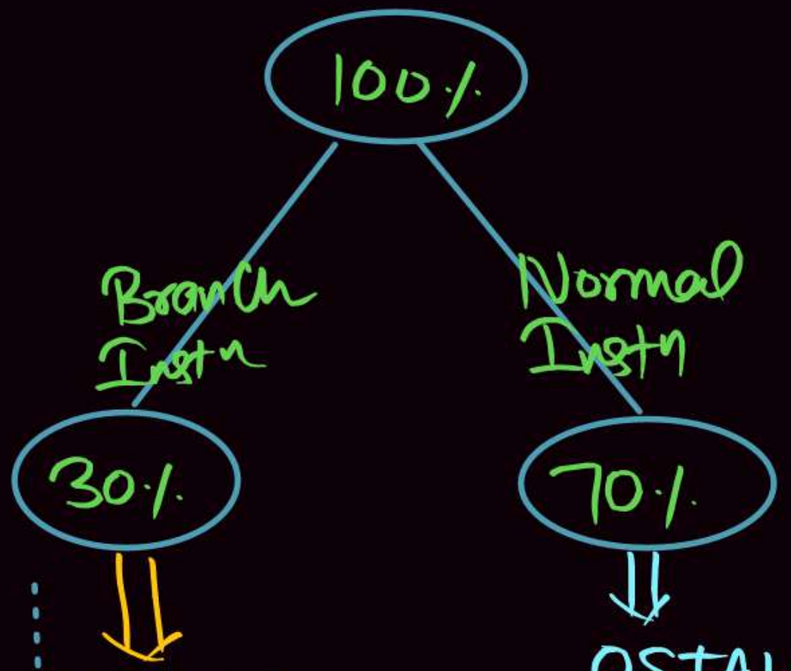
Branch Penalty = 5 − 1

Branch Penalty = 4

Avg Instn ET PIPE = [1 + #Stalls/Instn] × Cycle time

= [1 + 1.2] × 20 ⇒ 2.2 × 20

(i) Avg Instn ET PIPE = 44nsec Avg



#Stalls/Instn = .30 × 4 + .70 × 0

= 1.2

Soln 2

$$\text{SPEED UP FACTOR} = \frac{ET_{NONPIPE}}{ET_{PIPE}}$$

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

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

$$= \frac{5}{2.2} = 2.27 \text{ Ans}$$

OR

$$S = \frac{(\#\text{Stage})}{\text{PIPELINE Depth}} \Big/ (1 + \#\text{Stalls/Instn})$$

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

$$= \frac{5}{2.2}$$

$$S = 2.27 \text{ Ans}$$

Conditional Branch

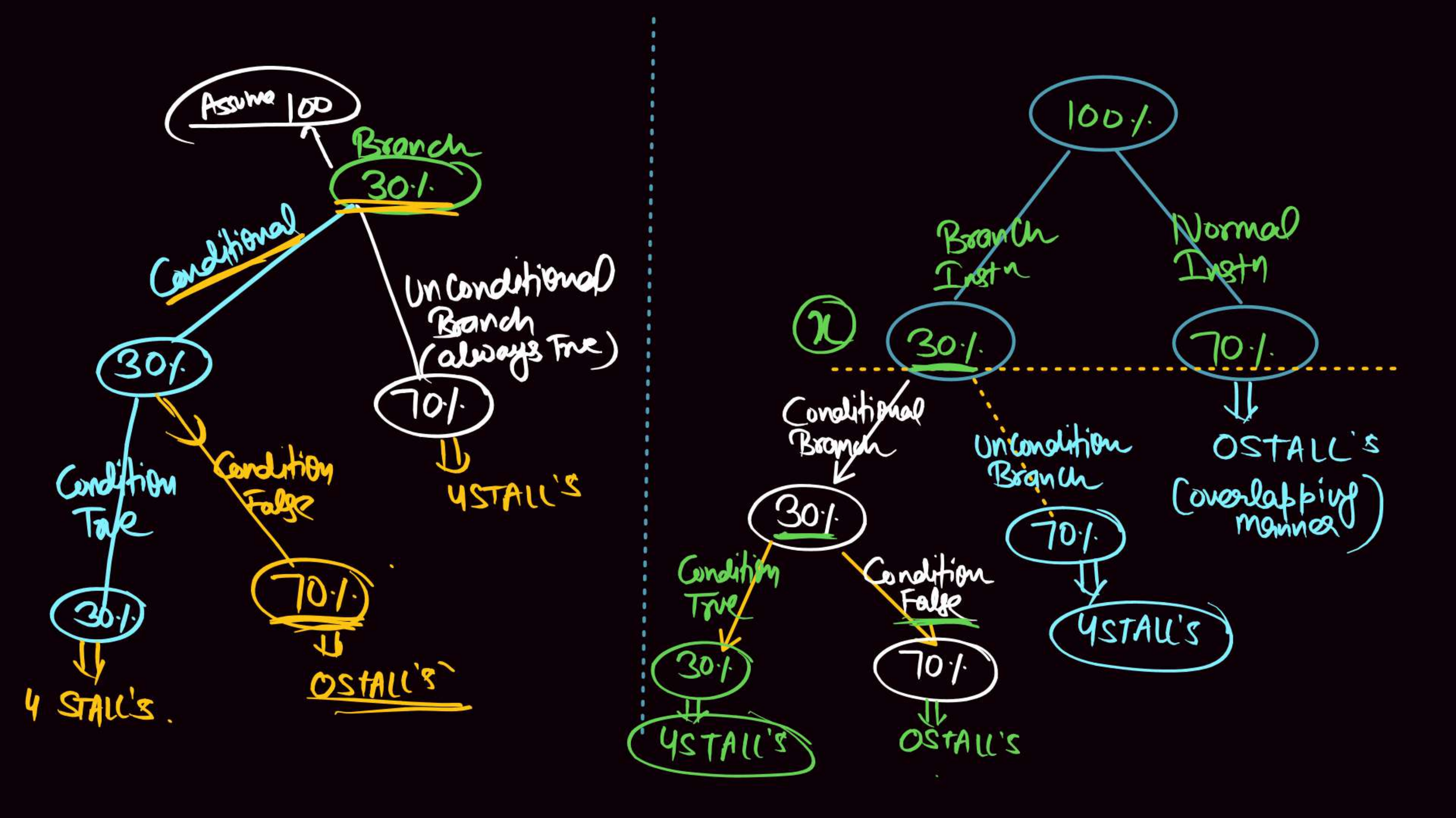
JNZ (Jump on Not Zero)

BZ (Branch on Zero)

↳ When Only the value '0' then Jump (Branch taken).

UnConditional Branch

JMP 4000 — No Condition ⇓ Always True


Assume 100
Branch
30%
Conditional
Unconditional Branch (always True)
30%
70%
Condition True
Condition False
4STALL's
30%
70%
4 STALL's.
OSTALL's
100%
Branch Instn
Normal Instn
30%
70%
OSTALL's (overlapping manner)
Conditional Branch
Unconditional Branch
30%
70%
Condition True
Condition False
4STALL's
30%
70%
4STALL's
OSTALL's

$$\#Stalls/Instn = .30 \times .30 \times .30 \times 4\,Stall + .30 \times .30 \times .70 \times 0 + .30 \times .70 \times 4\,STALL's + .70 \times 0$$

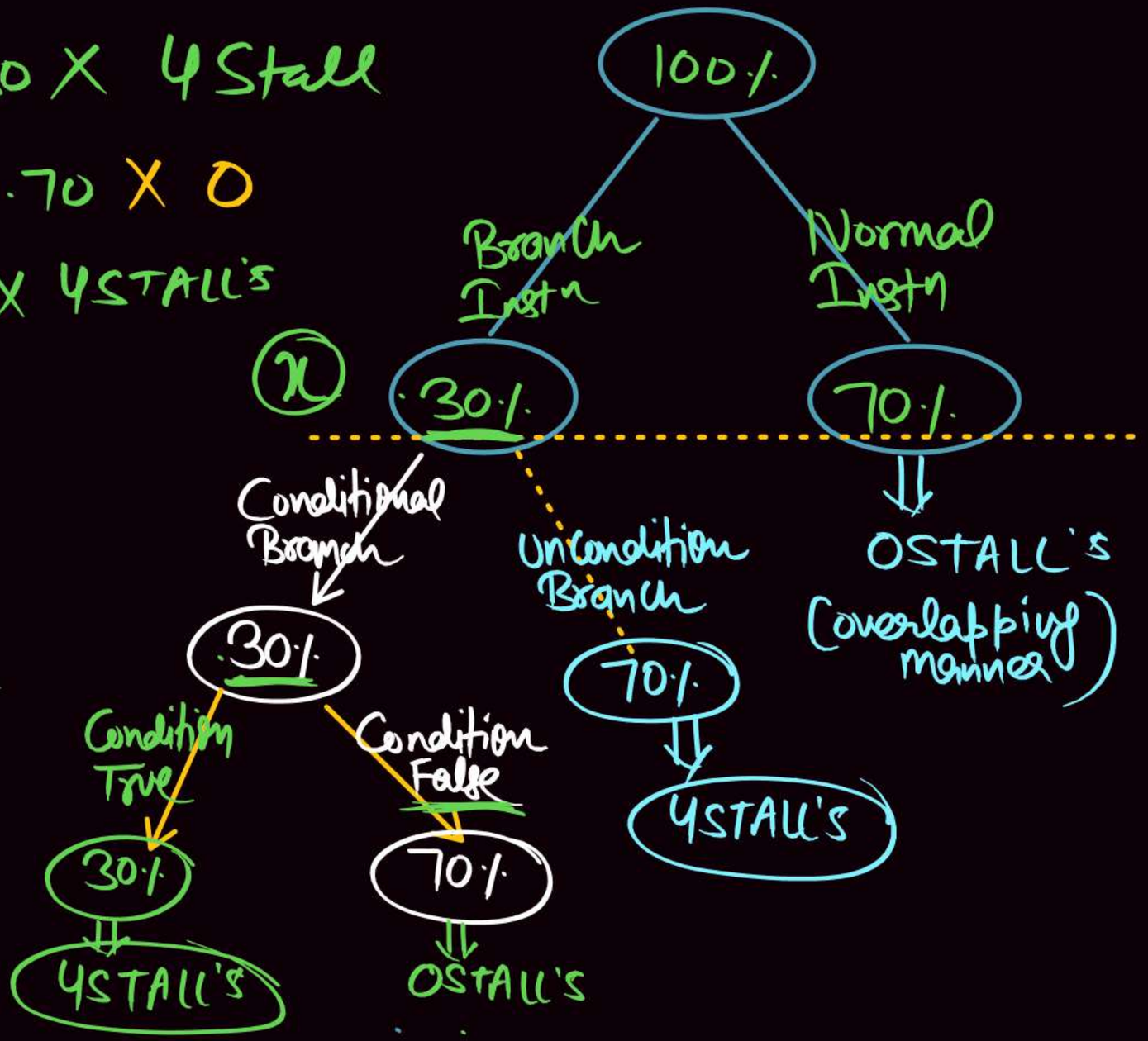
$$\#Stalls/Instn = 0.948$$

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

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

$$= 1.948 \times 20\,ns$$

$$= 38.96\ nsec$$ Avg



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]

Q. 6 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]

①	②	③	④	⑤
IF	ID/RF	EX	MA	WB
1ns	2.2ns	2ns	1ns	0.75ns

max

$t_p = 2.2$ nsec

Branch Instn Freq $= 20\%$

In OLD Pipeline Target Address at EX [3rd stage]

Branch Penalty $= 3 - 1 = 2$

#Stalls/Instn $= .20 \times 2 = 0.4$

Avg Instn ET [P] $= (1 + \#\text{Stalls/Instn}) \times \text{Cycle time}$

$\Rightarrow (1 + 0.4) \times 2.2$ ns $\Rightarrow 1.4 \times 2.2$

$P = 3.08$ nsec

①	②	③	④	⑤	⑥	⑦	⑧
IF	ID	RF1	RF_2	EX1	EX2	MA	WB
1nsec	2.2/3	2.2/3	2.2/3	1ns	1ns	1ns	0.75 ns

Branch Instn Freq $= 20\%$

$t_p = 1$ nsec

6th stage.

In the New Pipeline Target Address in EX2 Stage.

Branch Penalty $= 6 - 1 = 5$

#Stalls/Instn $= .20 \times 5 = 1$

Avg Instn ET [Q] $= (1 + \#\text{Stalls/Instn}) \times \text{Cycle time}$

$\Rightarrow (1 + 1) \times 1$ nsec

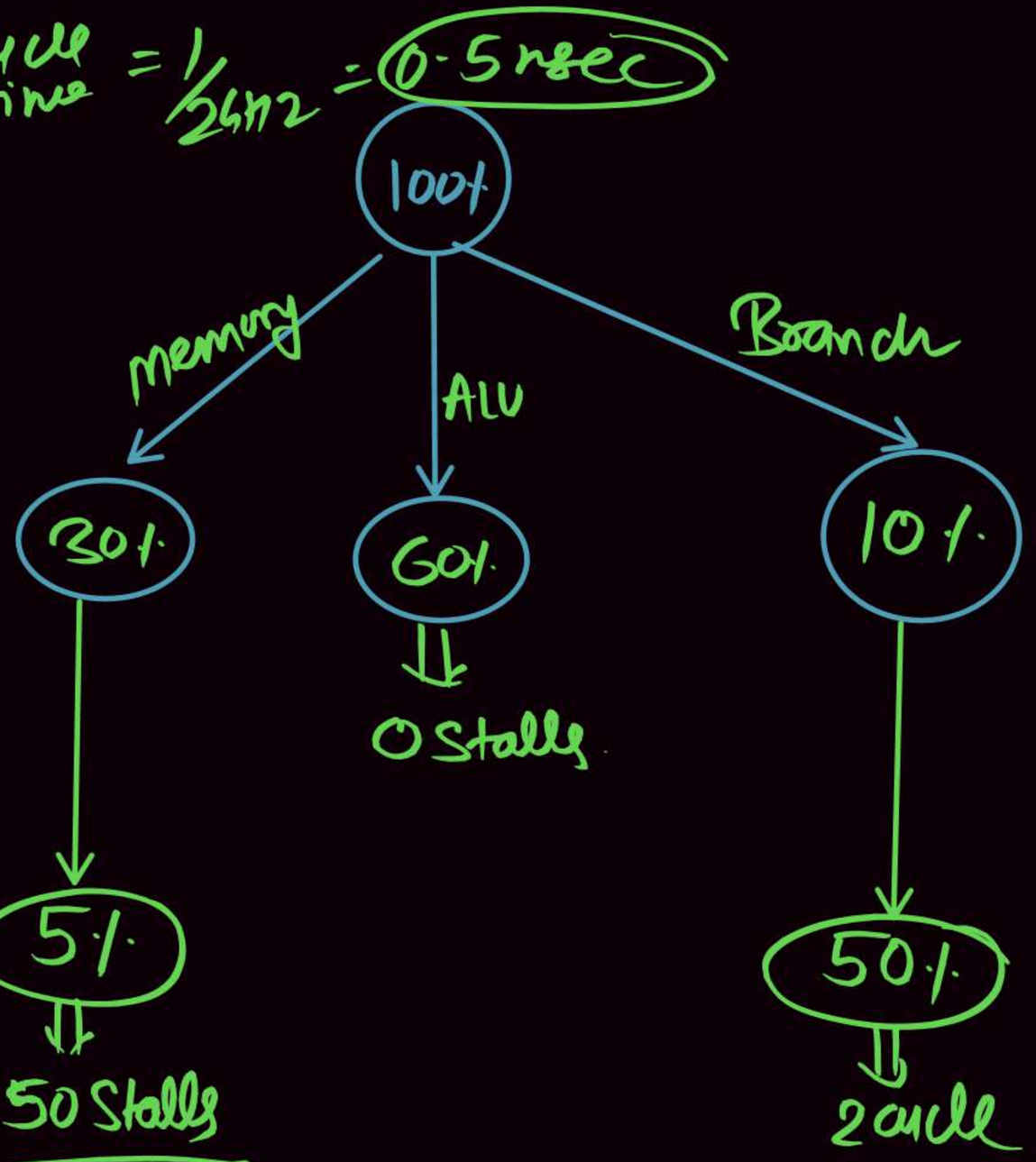
$Q = 2$ ns

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 _____

[GATE-2020-CS: 2M]

Cycle time $= \frac{1}{2\text{GHz}} = 0.5$ nsec



$\#\text{Stalls/Instn} = .30 \times .05 \times 50 + .60 \times 0 + .10 \times .50 \times 2$

$\#\text{Stalls/Instn} = 0.85$

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

$= (1 + 0.85) \times 0.5\text{ns}$

$= 1.85 \times 0.5$

$ET_{PIPE} = 0.925$ nsec

Non Pipeline Processor

2.5 GHz

$$\text{Cycle time} = \frac{1}{2.5 \times 10^9} = 0.4 \times 10^{-9}$$

$$= 0.4 \text{ nsec}$$

$$CPI_{NONPIPE} = 5$$

$$ET_{NonPIPE} = CPI_{NONPIPE} \times \text{Cycletime}$$

$$= 5 \times 0.4$$

$$= 2 \text{ nsec}.$$

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

$$= \frac{2}{0.925} = 2.16$$

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 - R6
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?

A I1

B I2

[GATE-2008-CS: 2M]

C I3

D I4

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

Here R_2 Register

this R_2 (I_1 output) Used in I_3 Instn

So I_1 Can not insert After Branch.

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

this R_4 (I_2 o/p) using in I_4

Storing $[R_4] \leftarrow$ Register Content.

So I_2 Cannot insert After Branch.

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

this R_1 Content Stored in Memory $[R_4] \leftarrow R_1$ by (I_4) & Used in Condition Checking ($R_1 == 0$)

(d) Store $M[R_4] \leftarrow R_1$

So I_4 Can be Insert After Branch. (Shift)

THANK YOU!