COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_06

Vijay Agarwal sir

01 Pipelining Hazards

ET in Pipeline

ET in Non Pipeline.

Speed up Factor, Throughput

Uniform Delay

Non Uniform Delay.

$\boxed{CPI = 1}$

How to set CPI.

Pipeline Hazard | Dependency

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

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.

In the Question
Any Number of
Stage Given.

3 Stage
4 Stage
5 Stage
6 Stage
7 Stage

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

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

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

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 $\boxed{CPI \neq 1.}$
- 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

↓

Create Due to Resource Conflict

Solution

Resource Duplication & Renaming

Cache Memory
- I-Cache [Code Memory]
- D-Cache [Data Memory]

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 ; $r_4 \leftarrow r_1 * r_5$

when Next Instn use the Result of Previous Instruction as a operand.

Data Dependency

Solution of Data Dependency.

1. Hardware Interlock.
2. 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 $r_3, r_0, r_2 : r_3 \leftarrow r_0 * r_2$

① True Data Dep. {Adjacent Instn}

② Data Dep.

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

$I_3 - I_1$
$I_4 - I_1$ } Data Dep.

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

H.W

[GATE-2002 : 2 Marks]

A. the pipeline stages have different delays

B. consecutive instructions are dependent on each other

C. the pipeline stages share hardware resources

D. All of the above

MCQ

1.w

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.

H.W

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

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

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.

H.W

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 13 Ans.

[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

H.W

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

Data Dependency / Data Hazards.

1. RAW [Read After Write] / True Data Dependency
2. WAR [Write After Read] / Anti Dependency
3. WAW [Write After Write] / Output [Write] Dependency.

These words Actual execution sequence, if Not follow in that order then Respective Hazard will be Create.

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 Dependency.

$(I_i)(I_i)$ I_1 ADD r_0 r_1 r_2 ; $r_0 \leftarrow r_1 + r_2$

OR

$(I_{i+1})(I_j)$ I_2 ADD r_4 r_0 r_3 ; $r_4 \leftarrow r_0 + r_3$

when Next Instn (I_2 or I_j or I_{i+1}) tries to use the Result of Previous Instn (I_1 or I_i) as a operand. ⇒ then RAW Hazard created.

(B/c Not follow the sequence (Read After Write))
I_j I_i

↳ Solution is operand forwarding
H/W Interlock.

Types of Data Hazard

② WAR (Write After Read) / ANTI - Dependency

I_i (OP) (I_i) I_1 : $R_1 \in R_2 * R_3$

I_{i+1} (I_j) I_2 : $R_2 \in R_4 + R_5$

When Next Instruction tries (I_2) to write the value before Previous Instruction (I_1) Read it.

So

Wrong Value is Read by the Instruction [Not Actual Value that was Needed]

Why? Due to Any Reson Suppose I_1 taking more time (multiply)

So I_2 tries to write the value of Register R_2

if I_2 execute first (Before I_1)

Assume initially $R_2 = 500$

When I_2 execute $R_2 \Rightarrow 10$ Now When I_1 Read $R_2 = 10$ Wrong DATA

Types of Data Hazard

② WAR (Write After Read) / ANTI - Dependency

I_i (I_i) I_1 : $R_1 \leftarrow R_2 * R_3$

OR

I_{i+1} (I_j) I_2 : $R_2 \leftarrow R_4 + R_5$

Why? Due to Any Reson suppose I_1 taking more time (multiply)

So I_2 tries to write the value of Register R_2

if I_2 execute first (Before I_1)

So Here I_2 tries to write (update) the value of Register R_2, before Instruction I_1 Read the value of Register R_2.

Assume initially $R_2 = 500$

When I_2 execute $R_2 \Rightarrow 10$ Now When I_1 Read $R_2 = 10$ wrong DATA

Types of Data Hazard

③ WAW [Write After Write] | Output | Write Dependency

I_i OR I_j OR I_{i+1}

$I_1 : R_1 \in R_2 * R_3$

$I_2 : R_1 \in R_4 + R_5$

Assume
If I_2 execute before the I_1 then it update the Register(R_1) Value before I_1 update it.

Output Dependency is created when the Next Instruction (I_2 or I_j or I_{i+1}) tries to write before the Previous Instruction (I_i or I_1) write it.

Types of Data Hazard

③ WAW [Write After Write] | Output | Write Dependency

I_i I_j OR OR I_{i+1} I_j

$I_1 : R_1 \leftarrow R_2 * R_3$

$I_2 : R_1 \leftarrow R_4 + R_5$

Assume
If I_2 execute before the I_1 then it update the Register(R_1) Value before I_1 Update it.

In this Here I_2 update the Value of Register R_1 before I_1 Update the Value of Register R_1

So wrong Updation (Not getting Correct Update).

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.

The Solution of ANTI [WAR] & output [WAW] Dependency is Register Renaming.

Register Renaming: Use the temporary Storage 'Registers' to Store the out of order Instruction Result.

(Use temporary Reg. & Rename it)

After the Proper execution, then Temporary Register Data Store in the Register file.

Q WHEN & HOW ANTI (OR) output Dependency is Created ?

Sol^n:- (i) If the Previous Instruction taking Very-High, More time Compare to Next Instruction [Uneven Delay].

(ii) When there is a Data Dependency b/w 2 Instruction & to Minimize the Stalls we trying to Reschedule the Instruction Execution sequence. (out of order execution)

Types of Data Hazard

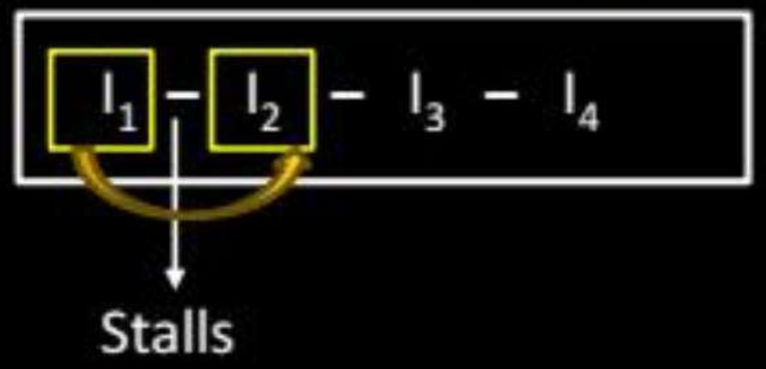
- CPU always execute a sequence the program from top to bottom in a sequence called as **"in-order"** execution.

- 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

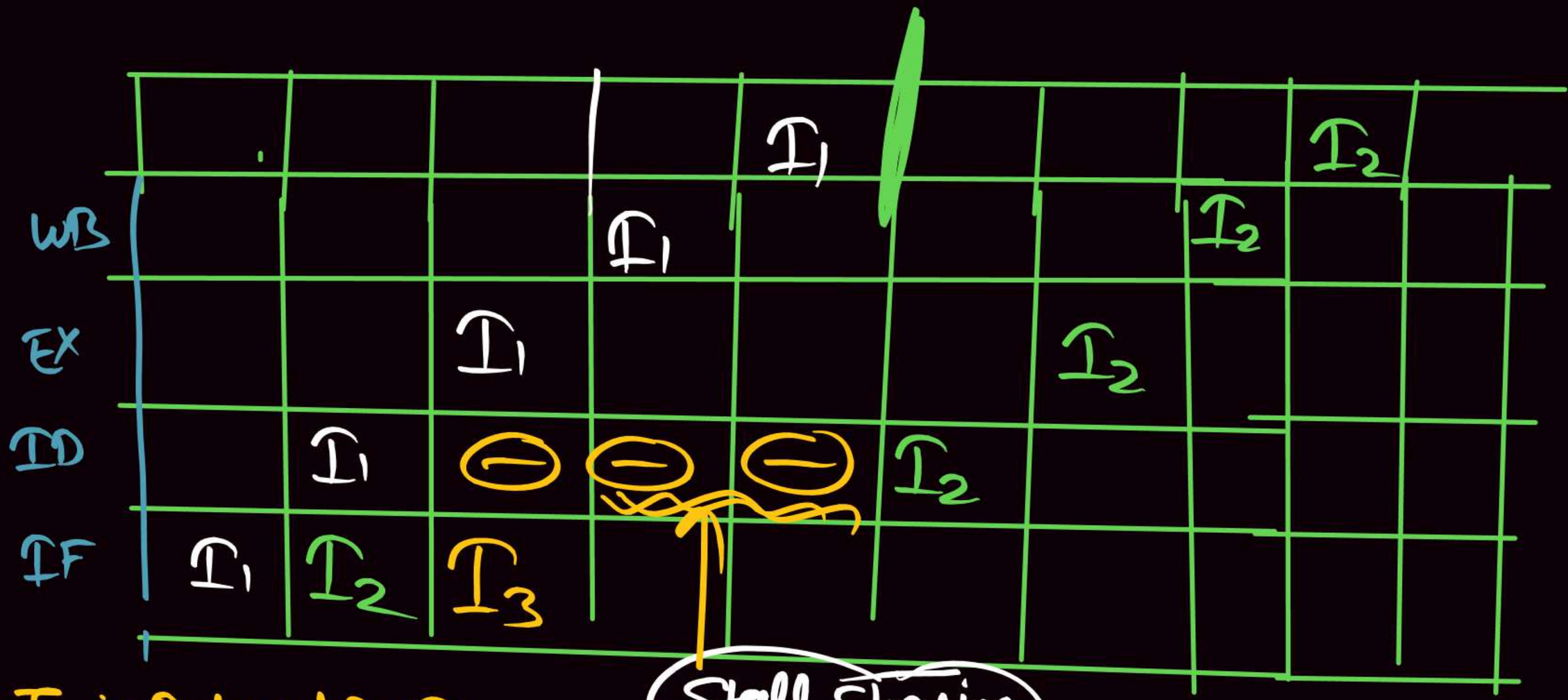
In- order execution sequence



$I_1 - I_2 - I_3 - I_4$

I_2 Depand on I_1

So Stalls.



Stall Sharing

I_2 is Depend on I_1 so stalls

But this Stalls shared by I_3 & I_4 (Even they are Not Dependent)

I_1	IF	ID	EX	MA	WB						
I_2		IF	⊖	⊖	⊖	ID	EX				
I_3			IF	⊖	⊖	⊖	ID	EX			

Stall Sharing

these stalls share by I_3 & I_4.

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}.

$I_1 - I_3 - I_4 - I_2$

- 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]

↓

AFTER

↓

Delay

↓

Stall

↓

Hazard

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

Reorder Execution

$I_1 - I_3 - I_4 - I_2$

Out of order Exec.

$I_1 - I_3 - I_4 - I_2$

[WAR] ANTI Dep : $I_3 - I_2$

[WAW] OUTPUT Dep : $I_4 - I_2$

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].

Note

ANTI & OUTPUT Dependency Not Create a Stalls, But Wrong Reading & wrong value Updation Problem occur.

Types of Data Hazard

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

Temporary Reg

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.

I_1

$I_3 \longrightarrow$ Store in

$I_4 \longrightarrow$ Temporary Register

I_2

When I_2 finish then Temp. Register Update into Respective Register.

Types of Data Hazard

Register Re-naming:

Execution

$I_1 \rightarrow r_0$

$I_3 \rightarrow$
$I_4 \rightarrow$ Re-order buffer (Temp Reg.)

$I_2 \rightarrow r_3$ (Status 1)

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

So Data Safe

(No Loss of DATA)

Types of Data Hazard

Hazards:

1. Hazard is a delay.
2. Delay creates stalls in the pipeline
3. Dependency problem causes hazards. These are three kinds
   I. Structural hazards
   II. Data hazards
   III. Control hazards
3. Data hazard is further divided into three types based on the reader and write operation sequence-
   (i) RAW [Read - after -write] hazard
   (ii) WAR [write - after -Read] hazard
   (iii) WAW [write - after -write] hazard

Types of Data Hazard

- RAW hazard is created when the instruction 'J' tires to reads the data before instruction 'I' writes it. [True data dependency]

- WAR hazard is created when the instruction 'J' tires to write the data before instruction 'I' reads it [Anti data dependency]

- WAR hazard is created when the instruction 'J' tires to write the data before instruction 'I' writes it- [output data Dependency]

Q. Consider the following code sequence having five instruction I1 to I5. Each of these instructions has the following format.

OP Ri, Rj, Rk

Where operation OP is performed on contents of registers Rj and Rk and the result is stored in register Ri.

I_1: ADD R1, R2, R3
I_2: MUL R7, R1, R3
I_3: SUB R4, R1, R5
I_4: ADD R3, R2, R4
I_5: MUL R7, R8, R9

S1 ANTI I_2 & I_5

S2 ANTI I_2 & I_4

S3 ANTI Dep always create Stalls

$I_2 - I_5$ output Dep.

$I_2 - I_4$ ANTI

Consider the following three statements:

S_1: There is an anti-dependence between instruction I_2 and I_5.

S_2: There is an anti-dependence between instructions I_2 and I_4.

S_3: Within an instruction pipeline an anti-dependence always creates one or more stalls.

Which one of above statements is/are correct?

[GATE-2015 (Set-3): 2 Marks]

A. Only S1 is true

B. Only S2 is true

Ans (B)

C. Only S1 and S3 are true

D. Only S2 and S3 are true

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 variable 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.

I_1	ADD	R5, R0, R1	; R5 ← R0 + R1
I_2	MUL	R6, R2, R5	; R6 ← R2 * R5
I_3	SUB	R5, R3, R6	; R5 ← R3 − R6
I_4	DIV	R6, R5, R4	; R6 ← R5 /R4
I_5	STORE	R6, X	; X ← R6

WAR
$I_3 - I_2$
$I_4 - I_3$

WAW
$I_3 - I_1$
$I_4 - I_2$

RAW : 4
WAR : 2
WAW

The number of Read-After-Write (RAW) dependencies, Write-After-Read (WAR) dependencies, and Write-After-Write (WAW) dependencies in the sequence of instructions are, respectively,

[GATE-2006: 2 Marks]

RAW: True DATA Dep.

(a) 2, 2, 4

(b) 3, 2, 3

(c) 4, 2, 2

(d) 3, 3, 2

RAW : 4

[ANTI] WAR : 2

[output] WAW : 2

Ans [c].

Control Dependency

Control Dependency is Created in the Pipeline Due to

- Branch Instruction
- function Call
- Program Control

OR

Transfer of Control Instr (TOC Instn)

- Unconditional TOC
- Conditional TOC.

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the $\boxed{\text{last Stage}}$ (5th Stage)

Assume

$(S_1\ S_2\ S_3\ S_4,\ S_5)$

1000	I_1	
⇒ 1001	I_2 → JMP 2000	unconditional TOC (Branch Instn)
1002	I_3	
1003	I_4	
⋮		
2000	BI_1	
2001	BI_2	

	1	2	3	4	5	6	7
I_1	S_1	S_2	S_3	S_4	S_5		
I_2		S_1	S_2	S_3	S_4	S_5	
I_3			S_1	S_2	S_3	S_4	
I_4				S_1	S_2	S_3	
I_5					S_1	S_2	
I_6						S_1	BI_1

4 Stalls

I_3, I_4, I_5 & I_6 Unwanted

& BI_1 is the next executable Instruction.

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the [LAST Stage]. 5th Stage.

→ 4 STALLS

1000 I_1

1001 I_2 → JMP 2000

1002 I_3

1003 I_4

⋮

2000 BI_1

2001 BI_2

		1	2	3	4	5	6	7	8	9	10	11
S5	WB					I_1	I_2					
S4	MA				I_1	I_2	I_3					
S3	EX			I_1	I_2	I_3	I_4			BI_1		
S2	ID		I_1	I_2	I_3	I_4	I_5		BI_1			
S1	IF	I_1	I_2	I_3	I_4	I_5	I_6	BI_1				

4 cycle 4 STALL's

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the (4th Stage) → 3 STALL's

1000 I_1
1001 I_2 → JMP 2000
1002 I_3
1003 I_4
⋮
2000 BI_1
2001 BI_2

		1	2	3	4	5	6	7	8	9	10	11
S5	WB					I_1						
S4	MA				I_1	I_2				BI_1		
S3	EX			I_1	I_2	I_3			BI_1			
S2	ID		I_1	I_2	I_3	I_4		BI_1				
S1	IF	I_1	I_2	I_3	I_4	I_5	BI_1					

3 STALL's

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the (3rd Stage) → 2 STALL'S

1000 I_1
1001 I_2 → JMP 2000
1002 I_3
1003 I_4
⋮
2000 BI_1
2001 BI_2

		1	2	3	4	5	6	7	8	9	10
S5	WB					I_1				BI_1	
S4	MA				I_1				BI_1		
S3	EX			I_1	I_2			BI_1			
S2	ID		I_1	I_2	I_3		BI_1				
S1	IF	I_1	I_2	I_3	I_3	BI_1					

2 STALL'S

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the (2nd Stage) → 1 Stalls

1000 I_1
1001 I_2 → JMP 2000
1002 I_3
1003 I_4
⋮
2000 BI_1
2001 BI_2

		1	2	3	4	5	6	7	8	9	10
S_5	WB					I_1			BI_1		
S_4	MA				I_1			BI_1			
S_3	EX			I_1			BI_1				
S_2	ID		I_1	I_2		BI_1					
S_1	IF	I_1	I_2	I_3	BI_1						

1 STALLS

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the (2nd Stage) → 1 stalls

1000 I_1

1001 I_2 → JMP 2000

1002 I_3

1003 I_4

⋮

2000 BI_1

2001 BI_2

	1	2	3	4	5	6	7	8	9	10
I_1	IF	ID	EX	MA	WB					
I_2		IF	ID	EX	MA	WB				
I_3			~~IF~~ I_3							
BI_1				BI_1 IF						

Control Dependency:

Assume 5 Stage Pipeline & I_2 is the Branch Instn (TOC Instn) & target address of Branch Instn is available in the <u>last stage</u>

1000 I_1

1001 I_2 → $\boxed{\text{JMP 2000}}$

1002 I_3

1003 I_4

⋮

2000 BI_1

2001 BI_2

If Target Address of Branch Instn is available at 'k' stages.

then

$$\boxed{\#\text{Stalls from Branch} = k-1}$$

Control Dependency/Control Hazards

$$\boxed{\text{Ideal CPI} = 1}$$

But Due to Control Dependency.

$$\text{Avg Inst}^n \text{ ET} = 1 + \#\text{Inst}^n \text{ which Create Control Hazard} \times \#\text{Stalls from the Branch.}$$

$$\text{Avg Inst}^n \text{ ET} = 1 + \left(\text{Branch Inst}^n \text{ freq} \times \#\text{Stalls Due to Branch Inst}^n\right)$$

Control Dependency.

- The Control is Created in the pipeline During the Branch (TOC) Instruction, so Program execution may be changed.
  - In this the Next Sequential Instruction may be wanted (or) Unwanted its Depand the output of Decoding of the Previous Instruction.
  - If the Previous decode as Simple (Data transfer, Airthmatic operation) then the Next sequential Instruction become wanted.
  - If the Previous Inst^n decode as Branch Inst^n (or) Uncondition Toc (or) Conditional Toc with true Condition then Next sequential Instruction become Unwanted.

for in Previous example

when I_1 execute then I_2 is wanted

But when

I_2 execute, I_3. I_4 Unwanted Instn

Bc2 After I_2, BI_1 will be execute.

Control Dependency.

- In the Pipeline overlapping execution, ie Before Decode the Previous Instn Next Sequential Instn inserted into the Pipeline.

- If the Next sequential Instn is unwanted then we have to Remove the unwanted Instn from the Pipeline.

- The Process of Removing the unwanted instruction flush (OR) Freeze.

Control Dependency.

The Number of Stall's [Extra Cycle] Create Due to Branch Instn is called **Branch Penalty.**

Note: This Branch Penalty Value Depand on outcome of target Address stage.

Note: generally Target Address is available in decode Phase, But Condition Checking [Evaluation] Done in Execution Phase.

Control Dependency.

If target Address is available at then end 'k^{th}' Stages then

Branch Penalty (B.P) or #Stalls [Extra Cycle] $= k - 1$

$\frac{\#\text{Stalls}}{\text{Instn}} = \frac{\#\text{Branch}}{\text{Instn}} \times \text{Stalls}$

Branch Frequency Means # of Branch type Instn.

$$\text{Branch Penalty} = \left(\begin{array}{c}\text{At what stage} \\ \text{Target address} \\ \text{is available}\end{array}\right) - 1$$

Control Dependency.

$$\boxed{\text{Avg CPI} = 1}$$

But Due to Branch Hazards

$$\boxed{\text{Avg CPI} = 1 + \#\text{Branch type Instr}^n \times \#\text{Stalls (B.P)}}$$

Control Dependency.

Q) We have 100 Instruction, out of 100 ⇒ 30 Instn are Branch type which takes 3 STALL cycle then Calculate the Avg CPI?

Sol^n

$$\boxed{\#\text{Stalls} = \#\text{of Branch Instn} \times \text{Stall Due to Branch Instn}}$$

$$\#\text{Stalls} = 30 \times 3 = 90 \text{ Stalls.}$$

$$ET = 1 \times 100 + 90$$
$$= 190$$

$$\text{Avg} = \frac{190}{100} = 1.9$$

Control Dependency. 30%

Q) We have 100 Instruction, out of 100 ⇒ 30 Instn are Branch type which takes 3 STALL cycle then Calculate the Avg CPI?

Sol^n

Branch frequency = 30%

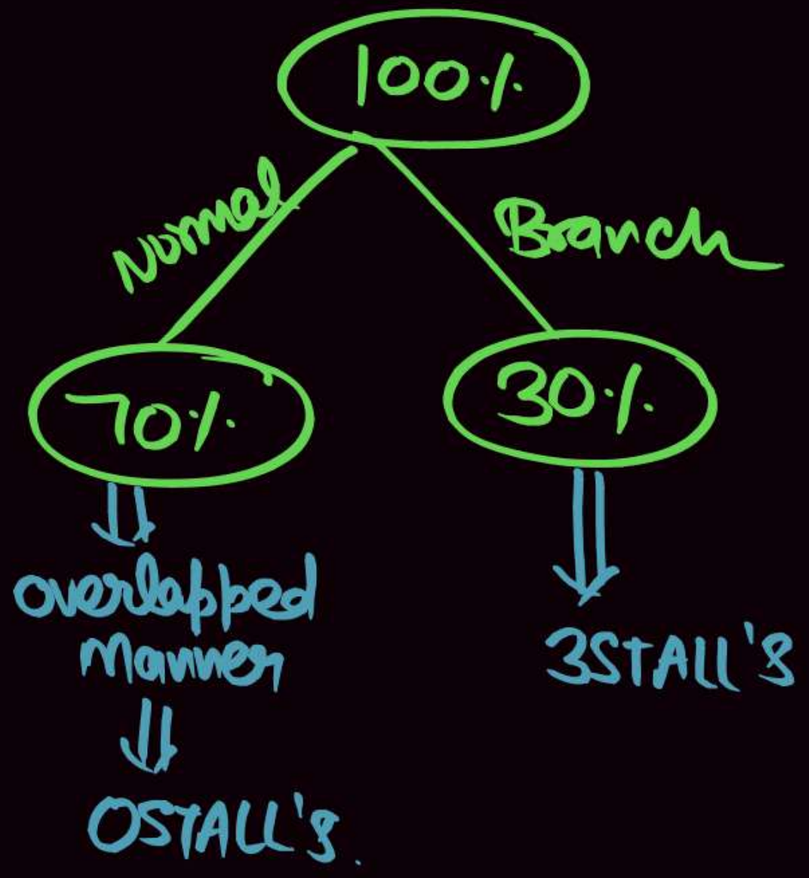
Branch penalty (# stalls) = 3

$$\text{Avg CPI} = [1 + .30 \times 3]$$

$$\Rightarrow [1 + 0.9]$$

$$\text{Avg CPI} = 1.9 \quad \underline{\text{Ans}}$$

Control Dependency.



$$\#\text{Stalls}/\text{Instn} = .70 \times 0 + .30 \times 3$$

$$\boxed{\#\text{Stalls}/\text{Instn} = 0.9}$$

Assume [20ns]

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

$$\Rightarrow [1 + 0.9] \times 20\text{ns}$$

$$= 1.9 \times 20 = 38\text{ nsec}$$ Ans

Control Dependency.

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

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/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

Without Any Hazards

$$\boxed{\text{Avg Instn } ET_{PIPE} = CPI \times \text{Cycle time.}}$$

$$\boxed{CPI = 1}$$

But Due to Control Hazard (Branch Instn) Stalls also Created

Now $\boxed{CPI \neq 1}$

Now $\boxed{CPI = 1 + \#\text{Stalls}/\text{Instn}}$

$$\boxed{\text{Avg Instn } ET_{PIPE} = \left(1 + \#\text{Stalls}/\text{Instn}\right) \times \text{Cycle time.}}$$

Speed Up factor with Stalls

Performance analysis with stalls :

$$S = \frac{\text{Avg. Instruction } ET_{nonpipe}}{\text{Avg. Instruction } ET_{nonpipe}} \quad \frac{ET_{NP}}{ET_{PIPE}}$$

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

Speed Up factor with Stalls

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

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

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

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

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

(Perfectly Balanced / Uniform Delay)

Q) Consider a pipeline 5 Stage, each Stage takes 20nsec then what is speed up Factor when very large No. of Instn (Ideal Case)

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

$$\frac{CPI_{NONPIPE} \times \cancel{\text{Cycle time}}}{CPI_{PIPE} \times \cancel{\text{Cycle time}}}$$

$$S = \frac{5 \times \cancel{20}}{\cancel{20}}$$

$$\boxed{S = 5}$$

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

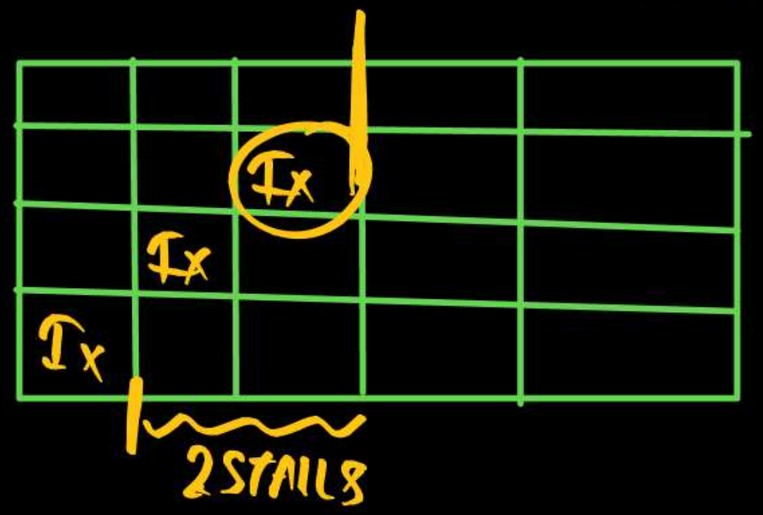
[GATE-2006 : 2 Marks]

(A) 1.0 second

(B) 1.2 second

(C) 1.4 second

(D) 1.6 second



Soln

DATA Given

Program having $= 10^9$ Instn

Clock freq $= 1GHz$. $\Rightarrow$ Cycle time $= \frac{1}{10^9}$ sec $=$ 1 nsec

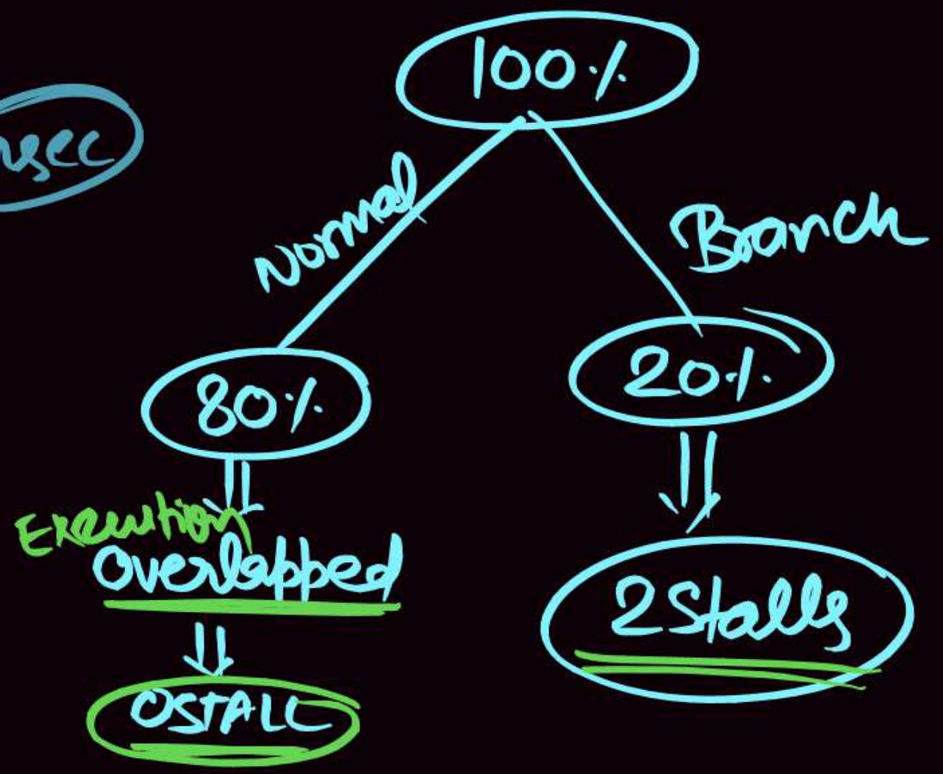
Branch Instn freq $= 20\%$

Target Address available at 3rd Stage.

(#Stalls) Branch Penalty $= \underset{k-1}{3-1}$

#Stalls $\boxed{B.P = 2}$

B.I.F $= 20\%$



$\frac{\#Stalls}{Instn} = .80 \times 0 + .20 \times 2$

$\boxed{\#Stalls/Instn = 0.4}$

(1 Instn ET)

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

$\Rightarrow (1+0.4) \times 1\,nsec$ or $1.4\,nsec$.

$$\boxed{\text{Avg Instn ET} = 1.4 \times 10^{-9}\ sec}$$

Program having 10^9 Instn

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

$= \underline{1.4\ sec}$ 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]

Ans(4)

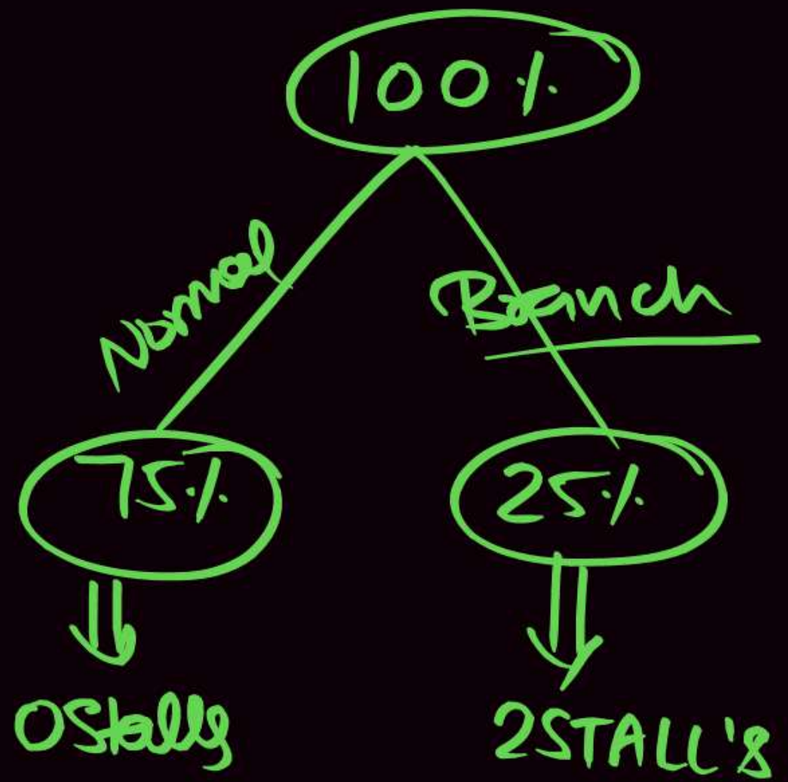
$$S = \frac{ET_{NONPIPE}}{ET_{PIPE}}$$

Assume cycle time is t_c.

$$S = \frac{CPI \times \text{Cycle time}}{(1 + \#Stalls/Instn) \times \text{cycle time}}$$

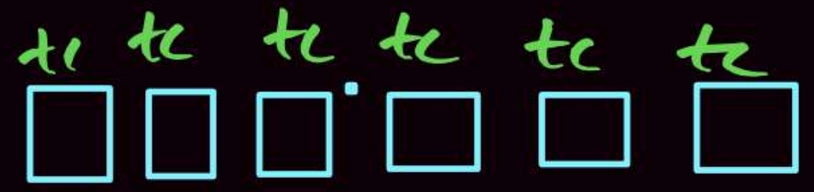
$$= \frac{6 \times \cancel{t_c}}{(1 + 0.5) * \cancel{t_c}}$$

$$= 6/1.5 \quad \boxed{S = 4}$$



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

$$\boxed{\#Stalls/Instn = 0.5}$$



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

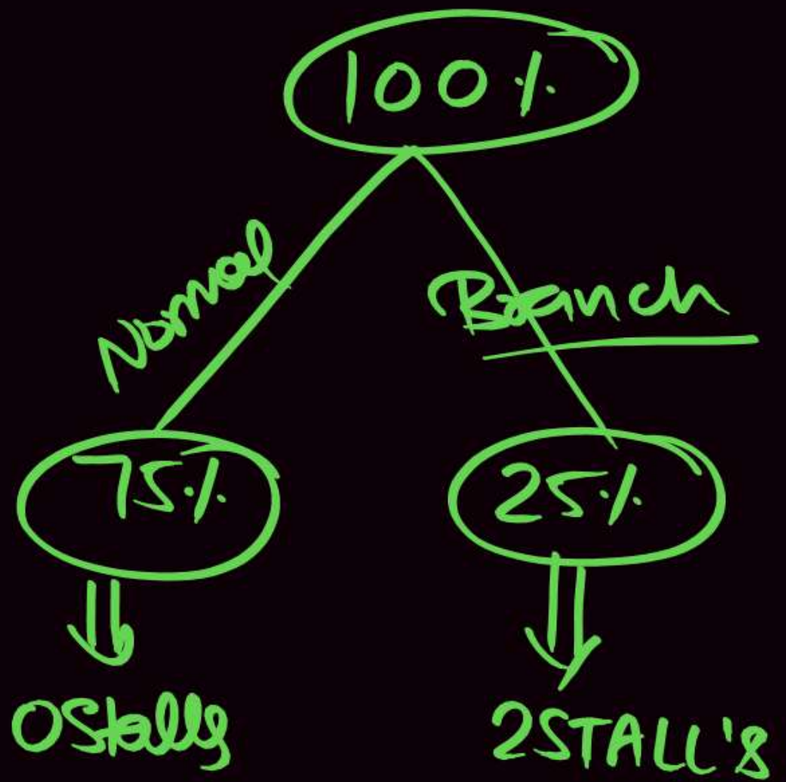
$ET_{NP} = 6tc$

Assume cycle time is tc.

$$S = \frac{ET_{NONPIPE}}{(1 + \#Stalls/Instn) \times \text{cycle time}}$$

$$= \frac{6 \times \cancel{tc}}{(1 + 0.5) * \cancel{tc}}$$

$$= 6/1.5 \quad \boxed{S = 4}$$



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

$$\boxed{\#Stalls/Instn = 0.5}$$

Q. 6

H.W

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

14.10

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]

MCQ

H.W

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

H.W

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

Any Doubt ?

THANK YOU!