COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_05

Vijay Agarwal sir

01 Pipelining Hazards

PIPELINE Concept

& Execution time.

& Speed up Factor, throughput

→ Without Stalls | Extra cycle | Bubbles

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.

May be we have.

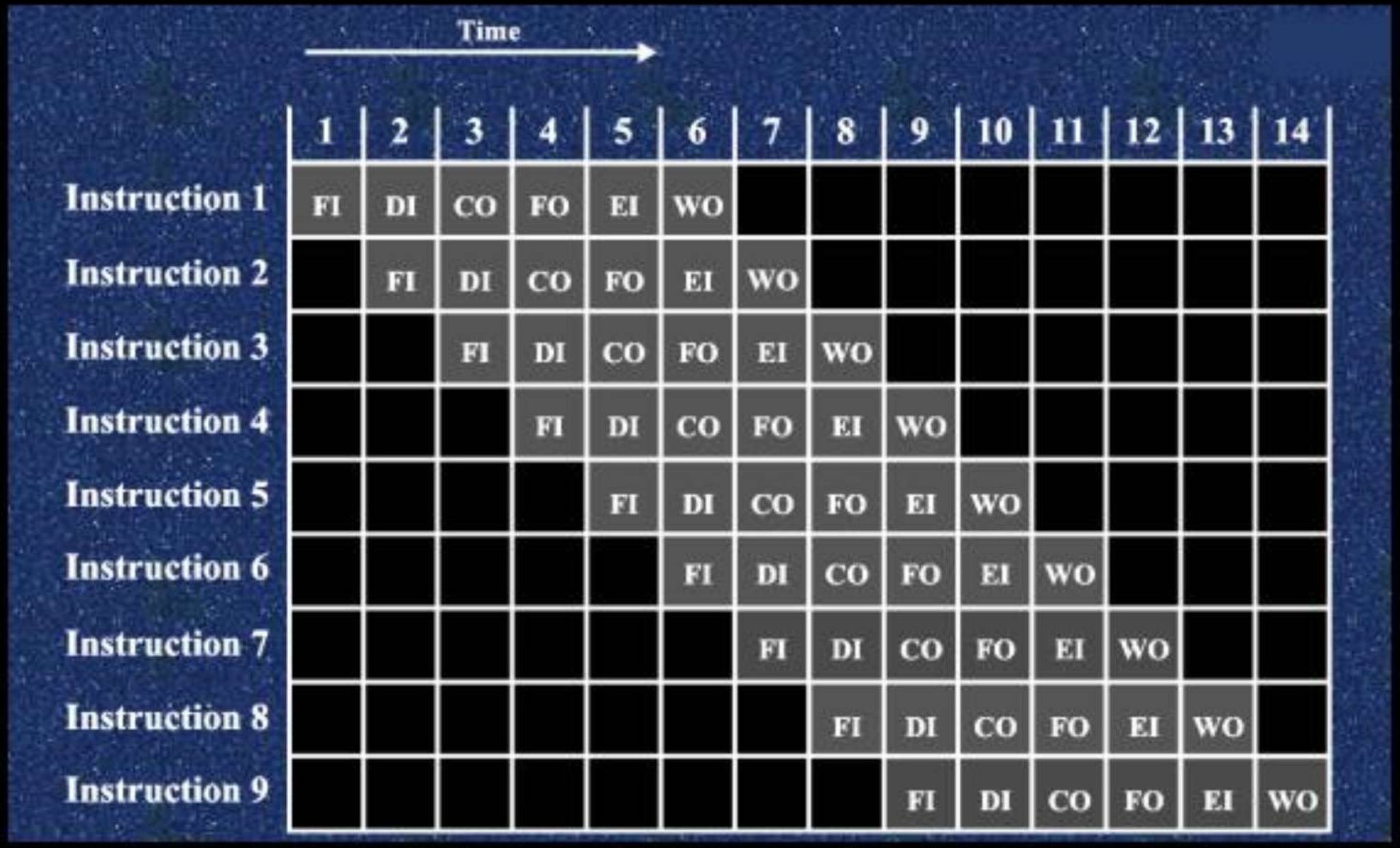
- 4 Stage
- 5 Stage
- 6 Stage
- 7 Stage

(OR) (or) (or)

Given in the Question.

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

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

① Structural Dependency.

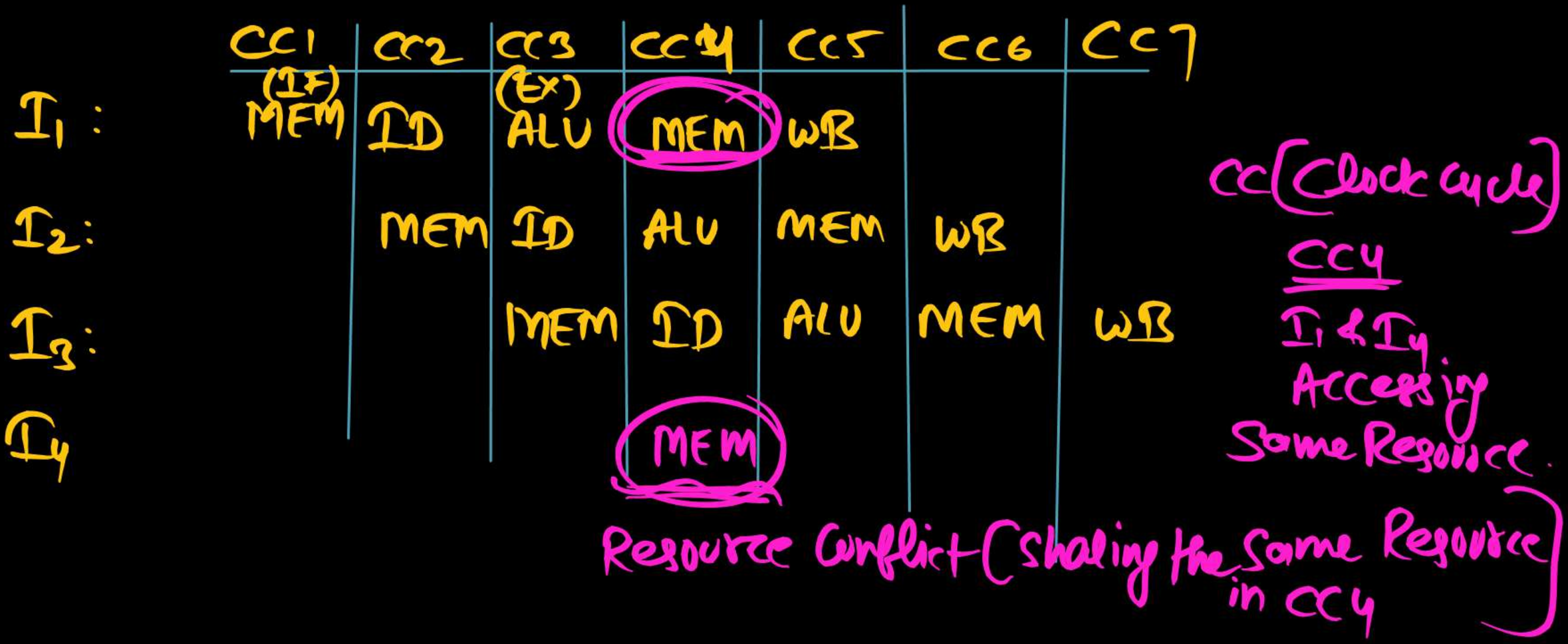
Structural Dependency

- When 2 Stages Require the Same Resource at the same time [Same cycle].

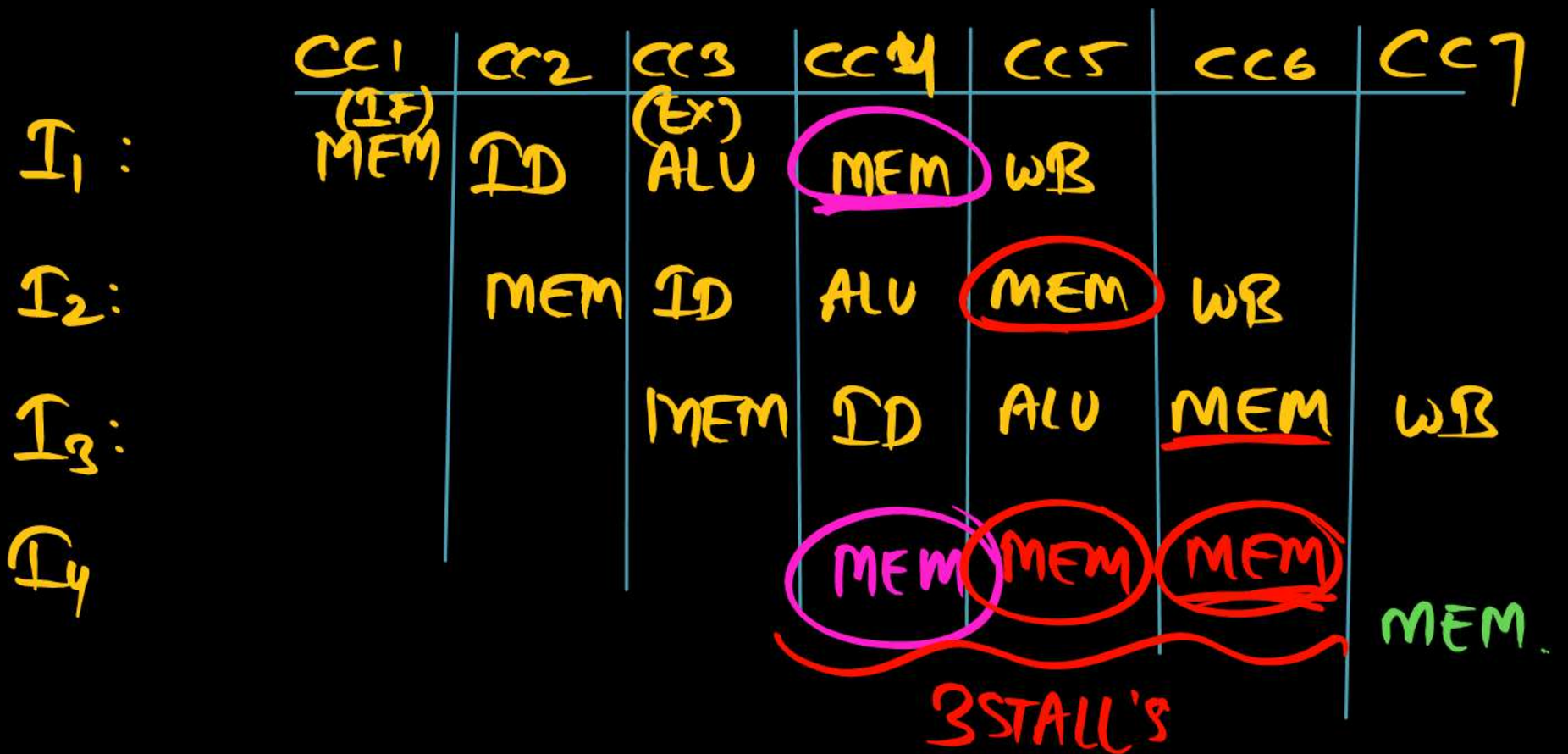
- Due to Resource Conflict.

Resource May ALU, Functional Unit @ others.

Structural Dependency



Structural Dependency



Structural Dependency

Solution of Structural Dependency.

- Resource Replication.
- Divide the Memory into Indepent Module.

Cache
- I-Cache (Instruction Cache) [CODE Memory [CM]] → for Instruction
- D-Cache [Data Cache] [Data Memory (DM)] → for DATA

Structural Dependency

- Solution:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
I_1:	CM	ID	ALU (Ex)	DM	WB			
I_2:		CM	ID	ALU	DM	WB		
I_3:			CM	ID	ALU	DM	WB	
I_4				CM	ID	ALU	DM	WB

No Stall.

② DATA Dependency.

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 ; $r_4 \leftarrow r_1 * r_5$

I_2 is trying to using the Result of I_1 as a operand.

So create Data Dependency.

- Data Dependency : is created in the Pipeline, when Next Instruction $[I_{i+1}, I_2, I_j]$ tries to Use the Result of Previous Instruction $[I_i, I_i, I_i]$ as a operand | DATA.

Data Dependency

Assume 5 stage

IF & ID

OF

EX

MA

WB

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_L:	IF	OF	EX	MA	WB				
I_2:		IF				OF	EX	MA	WB.

Data Dependency

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

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

Here I_2 tries to Read the value of Register r_0 before I_1 write it.

Data Dependency

Not create Problem in Non Pipeline

- When the above instruction are executed in a Non-pipelined system then data dependency condition does not- occur because "I_2" executing after 'I_1' So, I_2 reads the register (r_o) data after I_1 writes it [Read-After- write].

Because In Non Pipelining 'Non overlapping execution'.

ie In Non Pipelining New Input Insert Only After Completion of Old input.

→ But in Pipelining Overlapping execution So Data Dep. Creates a Problem in Pipelining.

Data Dependency

- But When the above instruction are executed in a 'pipeline system' then data dependency condition will be occurred because (I_2) will be executing along with (I_1) so (I_2) reads the (r_0) data before I_1 writes it. Therefore, I_2 incorrectly get the old value from (r_0) [Data loss].

$R_0 \leftarrow R_1 + R_3$

$R_3 \leftarrow R_0 * R_2$

Destination

Source.

	CC1	CC2	CC3	CC4	CC5
I_1	IF	ID&OF	EX	MA	WB
I_2		IF	ID&OF So: r_0, r_2 D: r_3 r_0 = value r_2 = value		
I_3			IF		
I_4					

After CC5 I_1 OR

OLD Value / Incorrect Value

Read – Before write

Old Data Access

{Data loss}

- When we tries to Read the Value ab γ_0 (by I_2) before I_1 Write it then I_2 Reading the OLD Value (or) Incorrect Value.

- So for getting the Correct Value Instruction I_2 (I_{i+1}, I_j) must Wait Untill the Completion of Instruction I_1 $[I_i]$.

Data Dependency

Assume 5 stage

IF & ID

OF

EX

MA

WB

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1:	IF	OF	EX	MA	WB				
I_2:		IF	//////	//////	//////	OF	EX	MA	WB.

Stalls

This waiting creates Extra cycle/stalls/bubbles in the Pipeline.

Q. How to check there is a Dependency between the Consecutive Instruction ?

How to Detect Dep btw I_1 & I_2

OR

$(I_i$ & $I_{i+1})$

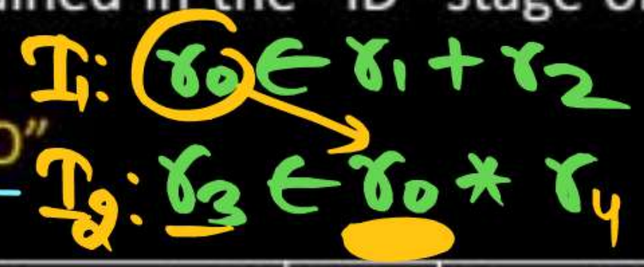
OR

$(I_i$ & $I_j)$

Soln → P.T.O. (Next Page)

Data Dependency

- To handle above problem, some logic will be maintained in the "ID" stage of pipeline to stop the accessing of a dependent data. Structure of a logic is as follows: "TOMASULO ALGO"



S/No	Function Unit	Destination	Independent Source 1	Independent Source 2	Dependent Source 1	Dependent Source 2	Status	Reg. file access
I_1	ADD	R_0	r_1	r_2	–	–	0	r_1 = value r_2 = value EX- Stage Allocated
I_2	MUL	r_3	-	r_4	r_0 (I_1)	–	0	r_2 = value EX- Stage not alocated

"0" execution yet not complete

"1" execution complete

I_i: $\gamma_0 \in \gamma_1 + \gamma_2$

I_j: $\gamma_3 \in \gamma_0 * \gamma_4$

Domain : Source operand

Range : (Destination)
(in which trying to write)

If $Range(I_i) \cap Domain(I_j) \neq \phi$

[Something Common]

then Data Dep.

eg.

I_1 I_i: $\gamma_0 \in \gamma_1 + \gamma_2$

I_2 (I_{i+1}) I_j: $\gamma_3 \in \gamma_0 * \gamma_4$

I. Domain: Source operand
(Trying to Read)

II. Range: (Destination)
(trying to write)

Domain (I_i)	Range (I_i)
γ_1, γ_2	γ_0

Domain (I_{j+1} @ I_j)	Range (I_{i+1} (I_j))
γ_0 γ_4	γ_3

If Range(I_i) $\cap$ Domain(I_j) $\neq \phi$

(γ_0) $\cap$ (γ_0, γ_4)

(γ_0) Having Common.

then Data Dep.

[Something Common] → then Dep.

So Data Dependency is there?

So.

Data Dependency

- By using the above logic, we need to stop the access of dependent data unit the data becomes available.

  This waiting creates Extra Cycle/Bubble/stall in the pipeline that is described below :

Data Dependency

Assume 5 stage

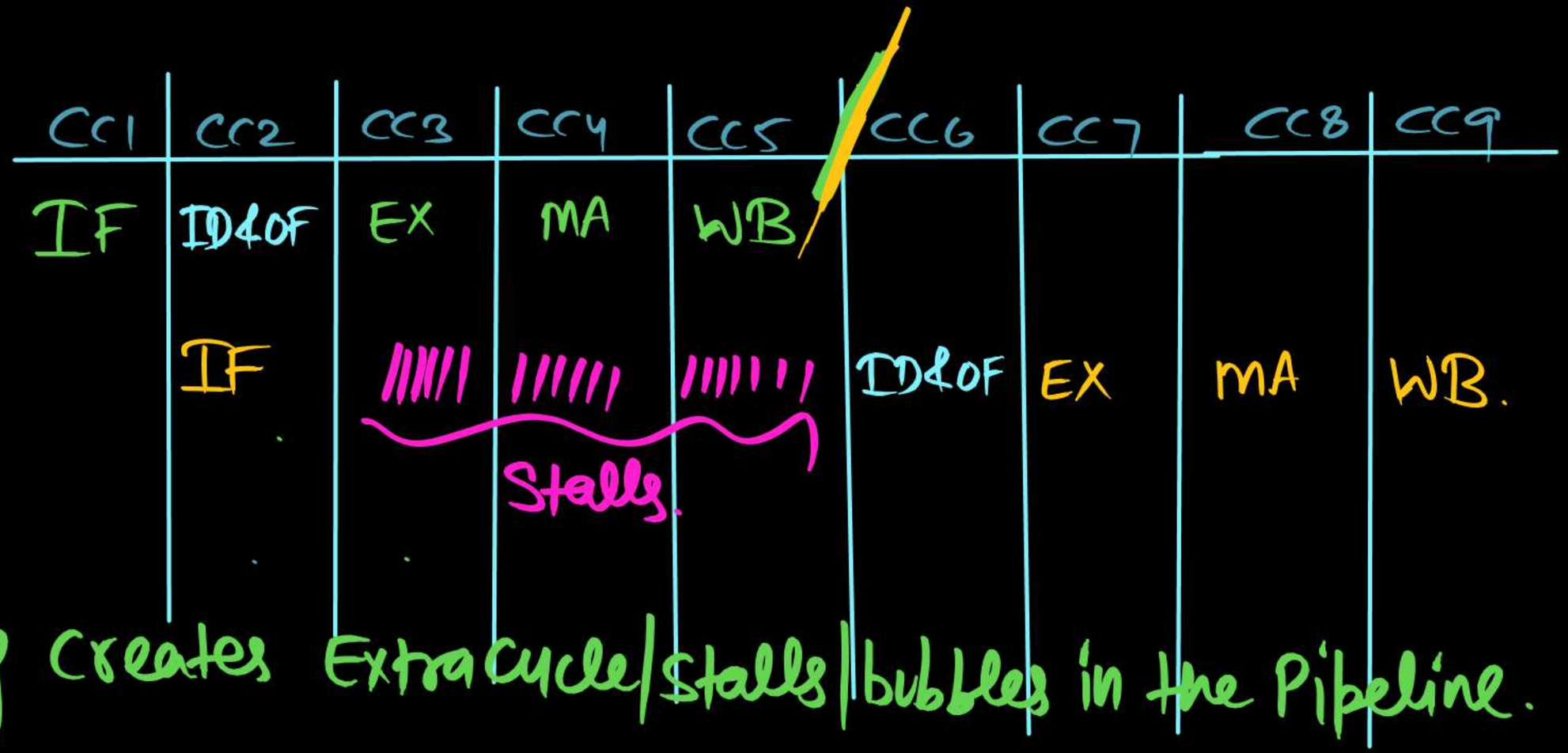
IF & ID

OF

EX

MA

WB



This waiting creates Extra Cycle/Stalls/bubbles in the Pipeline.

Topic : Dependencies In the pipeline

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1	IF	ID	EX	MA	WB r_0 = updated Status =1		
I_2		IF	ID S: r_0, r_2 (D) (I_0) r_2 = value	ID STALL'S	ID	ID r_0 = value	EX
I_3			IF	IF	IF	IF	ID
I_4				≡	≡	≡	IF

Read –After Write

7 cycles – 4 i/ps = 3 stalls

Solution Of Data Dependency

Hardware Interlocks

The most straightforward method is to insert hardware interlocks. An interlock is a circuit that detects instructions whose source operands are destinations of instructions further up in the pipeline. : Lock ALL the Stage For Dependent & further Instn.

Operand Forwarding

Another technique called operand forwarding uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. For example, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed as a source in the next instruction, it passes the result directly into the ALU input, bypassing the register file.

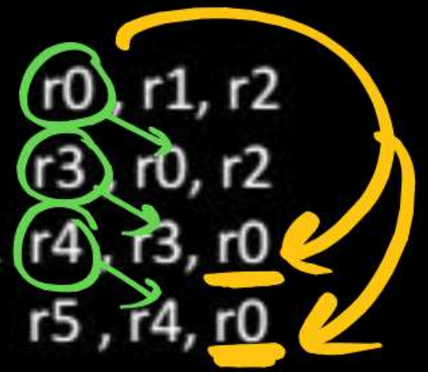
Solution of Data Dependency

- The minimize the data dependency stall in the pipeline hardware technique is used i.e. operand forwarding. Also named as **By-passing or Short –circuit**
- This mechanism state that use the buffer between the stage to hold the intermediate result so that, dependent instruction will be accessing the new data from the buffer before update register file.

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

① True Data Dep.

② 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. $I_3 - I_1$

$I_4 - I_1$

$I_3 - I_1$

$I_4 - I_1$.

Data Dependency (without operand forwarding)

Assume 5 stage

IF & ID

OF

EX

MA

WB

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1:	IF	ID & OF	EX	MA	WB				
I_2:		IF	//////	//////	//////	ID & OF	EX	MA	WB.

Stalls.

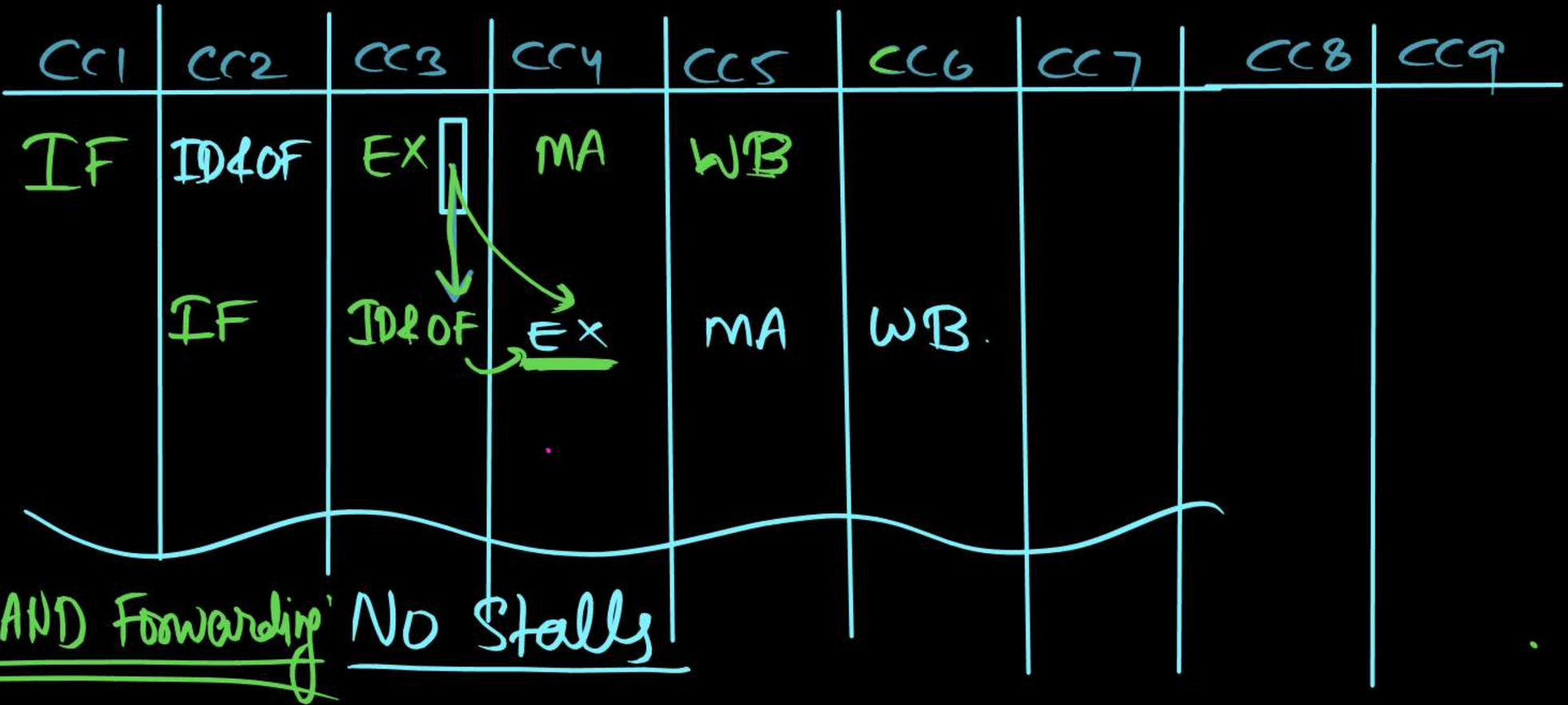
This waiting creates Extra cycle/stalls/bubbles in the Pipeline.

Data Dependency

(with operand forwarding)

Assume 5 stage

IF
ID & OF
EX
MA
WB



'OPERAND Fowarding' No Stalls

With Operand Forwarding

	CC1	CC2	CC3	CC4	CC5	CC6	
I_1	IF	ID	EX	MA	WB r_0= update		
I_2		IF	ID (I_1)	EX	MA	WB	
I_3			IF	ID (I_2) ,(I)	EX	MA	
I_4				IF	r_0 = value ID (I_1, I_3)	EX	
I_5					IF	ID	
I_6						IF	

6 cycles – 6 i/p's = 0 stalls

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?

IF
ID/OF
EX →
WB

(i) Without operand Forwarding.

1	ADD	R2	R1	R0	R2 ← R1 + R0
3	MUL	R4,	R3,	R2	R4 ← R3 * R2
1	SUB	R6,	R5,	R4	R6 ← R5 − R4

Ans(12) without operand forwarding.

[GATE-2007 : 2 Marks]

A 7
B 8
C 10
D 14

Other Approach (ii) Without operand Forwarding.

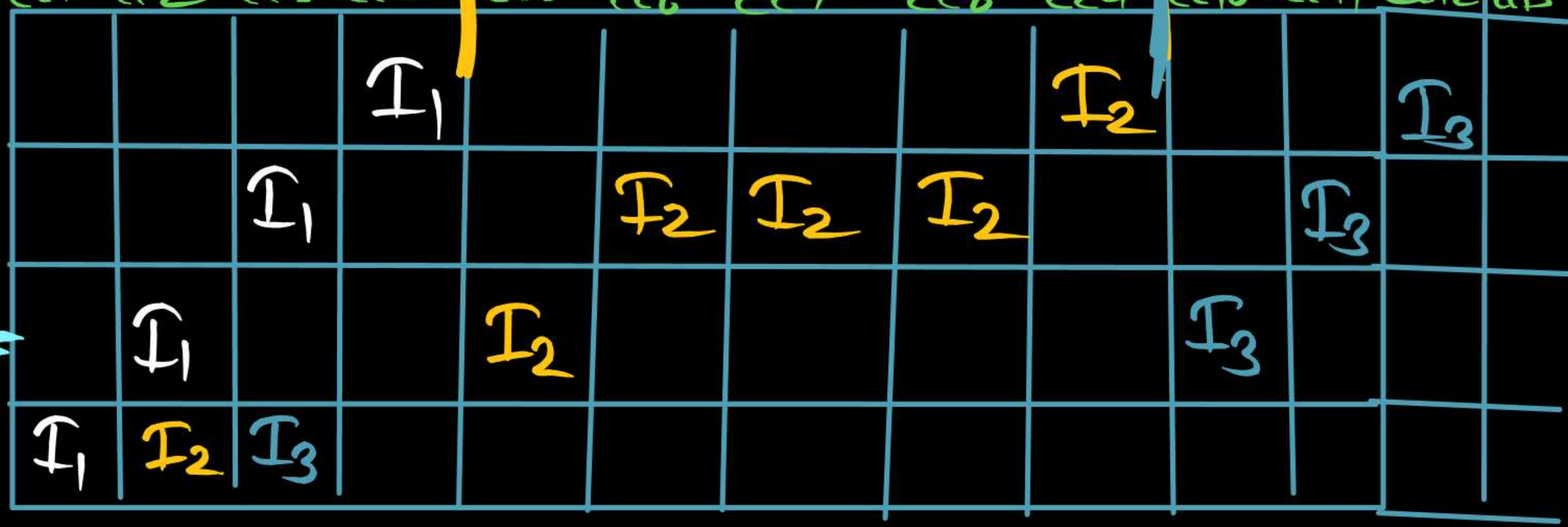
IF
ID/OF I_1 1
EX → I_2 3
WB
I_3 1

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

Ans(12) Without operand forwarding.

[GATE-2007 : 2 Marks]

A 7
B 8
C 10
D 14



(I) With operand forwarding.

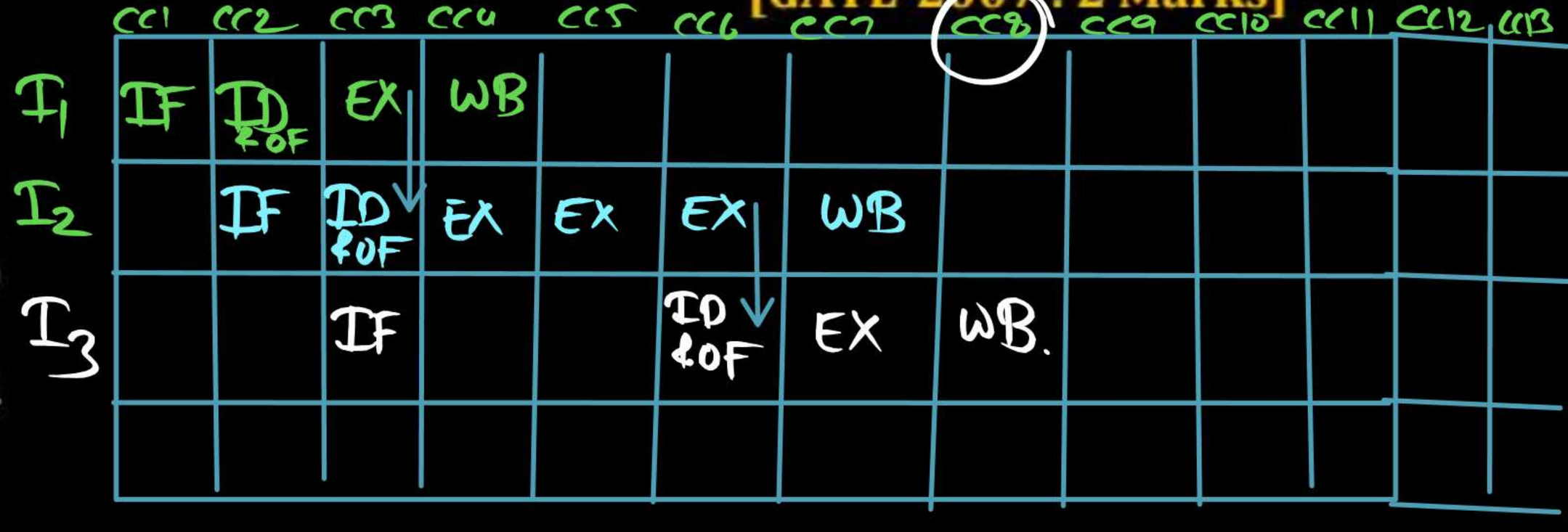
IF
ID&OF → 1
EX → 3
WB → 1

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

With operand forwarding (8).

[GATE-2007 : 2 Marks]

A 7
B 8
C 10
D 14



IF
ID/OF
EX
WB

other Approach

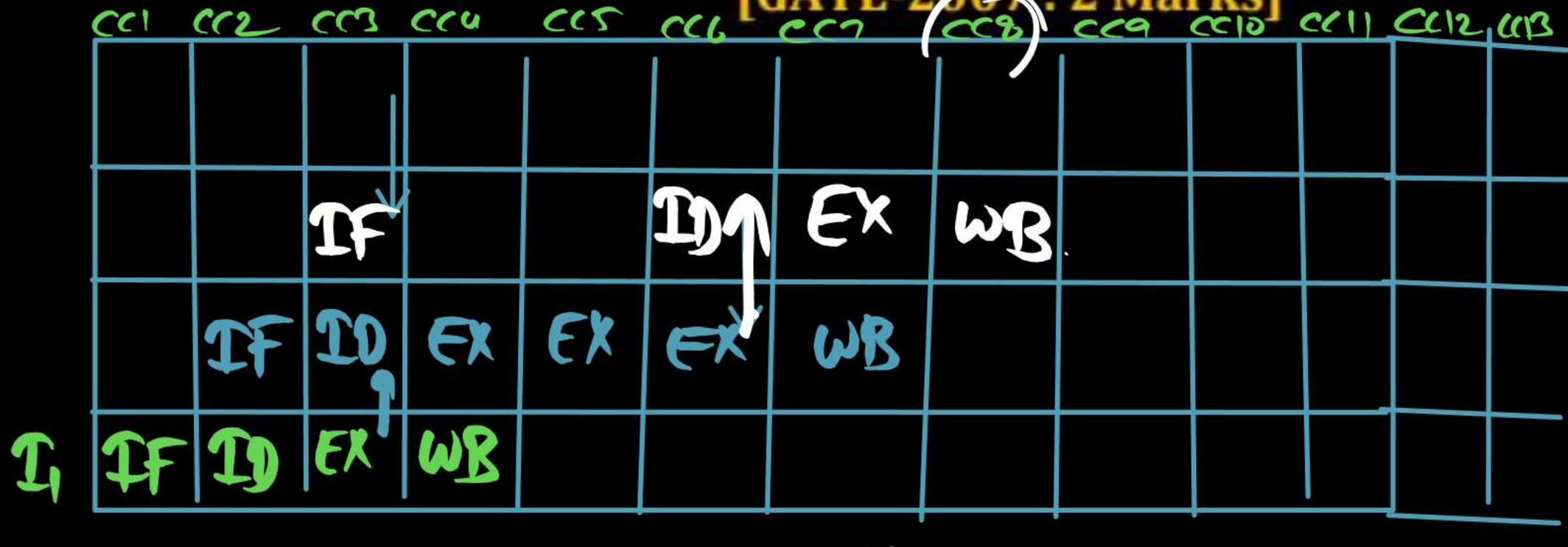
(I) With operand forwarding.

1	ADD	R2	R1	R0	R2 ← R1 + R0
3	MUL	R4,	R3,	R2	R4 ← R3 * R2
1	SUB	R6,	R5,	R4	R6 ← R5 − R4

With operand forwarding (8).

[GATE-2007 : 2 Marks]

(A) 7
(B) 8
(C) 10
(D) 14



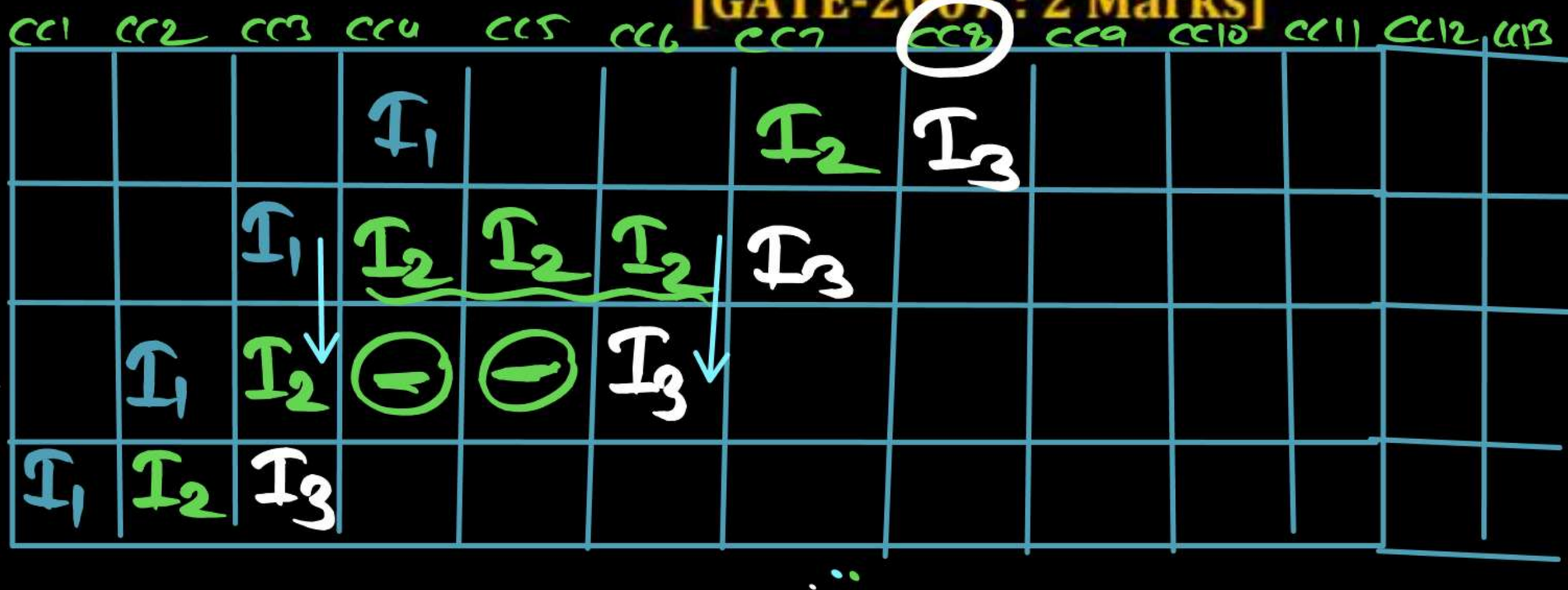
Another Approach (I) With operand forwarding

IF 1
IDOF 1
EX → 3
WB 1

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



By other Approach.

In Pipeline $\boxed{CPI = 1}$

But Some Instr take More Cycle in the Stage.

So we Count Extra Cycle [Stalls]

$$\text{Total Cycle} = ET_{PIPE} + \text{Extra Cycle [Stalls]}$$

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?

IF : 1

ID : 1

EX : 1 (for ADD & SUB)

WB : 1

3 For MUL.

2 extra cycle (Stalls) taken.

So

CPI = 1

n (# Instn) = 3

k (# Stages) = 4

Normally

ET_{PIPE} without stalls $= [k + (n-1)]$ cycle

$\Rightarrow [4 + (3-1)]$

ET_{PIPE} without stalls = 6 cycle

Extra cycle (Stalls) = 2 cycle.

Total cycle = 8 cycle

Ans

IF
ID/OF
EX → 3
WB

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]

IF : 1
ID : 1
EX : ADD & SUB & MUL
① 3
WB : 1

$n =$

(A) 7
(B) 8
(C) 10
(D) 14

IF
ID/OF
EX →
WB

1	ADD	R2	R1	R0	R2 ← R1 + R0
3	MUL	R4,	R3,	R2	R4 ← R3 * R2
1	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

B. consecutive instructions are dependent on each other

C. the pipeline stages share hardware resources

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
2. The execution of a conditional jump instruction
3. The j^{th} 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. 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]

I_1 MUL $R_5 \leftarrow R_0 * R_1$ ⟶ Postage 3

I_2 DIV $R_6 \leftarrow R_2 / R_3$ ⟶ 5

I_3 ADD $R_7 \leftarrow R_5 + R_6$ ⟶ 1

I_4 SUB $R_8 \leftarrow R_7 - R_4$ ⟶ 1

I_2 is Not Depant on I_1

IF (Fetch & Decode)

OF (operand Fetch)

PO (Perform operation)

WB (write Back)

IF : 1
OF : 1
WB : 1

	Total Cycle	Extra
MUL	3	2
DIV	5	4

6 Extra Cycle

$k = 4$ (Stages)

$n = 4$ (Instn)

ET_{PIPE} without stalls $= [k + (n-1)]$ cycle

$= (4 + 4 - 1)$

$= 7$ cycle

Total cycle $= 7 + 6 = 13$ cycle.

Note: This Method will Work when only One Stage taking Different Delay & One type of Dependency is there.

I_1 MUL $R_5 \leftarrow R_0 * R_1$ ⟶ Postage 3

I_2 ON $R_6 \leftarrow R_2 / R_3$ ⟶ 5

I_3 ADD $R_7 \leftarrow R_5 + R_6$ ⟶ 1

I_4 SUB $R_8 \leftarrow R_7 - R_4$ ⟶ 1

I_2 is Not Defpant on I_1

IF (Fetch & Decode)

OF (operand Fetch)

PO (Perform operation)

WB (write Back)

	1	2	3	4	5	6	7	8	9	10	11	12	13
WB						I_1					I_2	I_3	I_4
PO			I_1	I_1	I_1	I_2	I_2	I_2	I_2	I_2	I_3	I_4	
OF		I_1	I_2							I_3	I_4		
IF	I_1	I_2	I_3	I_4									

I_1 MUL $R_5 \leftarrow R_0 * R_1$ → 3

I_2 DIV $R_6 \leftarrow R_2 / R_3$ → 5 (2 NUDep.)

I_3 ADD $R_7 \leftarrow R_5 + R_6$ → 1

I_4 SUB $R_8 \leftarrow R_7 - R_4$ → 1

Postage

IF (Fetch & Decode)

OF (operand Fetch)

PO (Perform operation)

WB (Write Back)

I_2 is Not Dep. on I_1

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
I_1	IF	OF	PO	PO	PO	WB							
I_2 (Not Dep. on I_1)		IF	OF	PO	PO	PO	PO	PO	WB				
I_3			IF					OF	PO	WB			
I_4				IF					OF	PO	WB		

Sometimes Mistake Done by the Students.

I_1 MUL $R_5 \leftarrow R_0 * R_1$ → 3

I_2 DIV $R_6 \leftarrow R_2 / R_3$ → 5 (2 NO Dep.)

I_3 ADD $R_7 \leftarrow R_5 + R_6$ → 1

I_4 SUB $R_8 \leftarrow R_7 - R_4$ → 1

IF (Fetch & Decode)

OF (operand Fetch)

PO (Perform operation)

WB (Write Back)

I_2 is Not Dep. on I_1

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
I_1	IF	OF	PO	PO	PO	WB							
I_2 (Not Dep. I_1)		IF	OF			PO	PO	PO	PO	PO	WB		
I_3			IF							OF	PO	WB	
I_4				IF							OF	PO	WB

13 cycle

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]

CPI = 1

IF : 1
ID : 1
OF : 1
PO
WB : 1

40 → 3, 35 → 2, 25 → 1

Extra Cycle ⇒ 40×2 = 80
+35×1 +35
+25×0 +0
115

n = 100
k = 5

ET = [k + (n-1)]
⇒ 5 + (100-1)

without stalls = 104 cycle

Total cycle = 104 + 115
= 219 cycle.

IF : 1
ID : 1
OF : 1
PO
WB : 1

CPI = 1

40 → 3, 35 → 2, 25 → 1

Extra Cycle ⇒ 40X2 = 80
+35X1 = +35
+25X0 = +0
115

n = 100
k = 5

ET = [k + (n-1)]
⇒ 5 + (100-1)
Without stalls = 104 cycle

Total cycle = 104 + 115
= 219 Cycle.

DATA Dependency / Hazard.

1. RAW (Read After Write): True Data Dep.
2. WAR (Write After Read): ANTI Dep.
3. WAW (Write After Write): Output/Write Dep.

DATA Dependency / Hazard.

1. RAW (Read After Write): True Data Dep.
2. WAR (Write After Read): ANTI Dep.
3. WAW (Write After Write): Output/Write Dep.

Note

Here RAW, WAR, WAW means Execution Sequence ie RAW (Read, After Write).

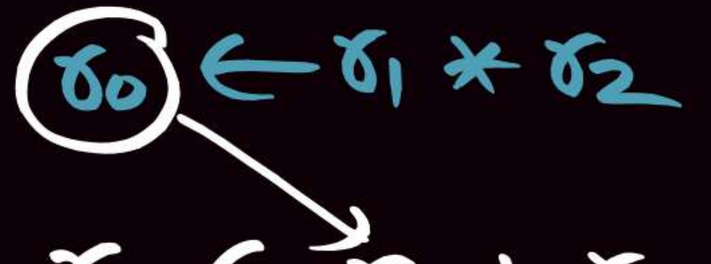
if Follow the execution sequence then No Problem.

But if Not Follow then these Hazard Create.

RAW [Read After Write] | True Data Dep.

$I_1 (I_i)$: MUL $r_0 \leftarrow r_1 * r_2$

$I_2 (I_j)$: ADD $r_4 \leftarrow r_0 + r_3$

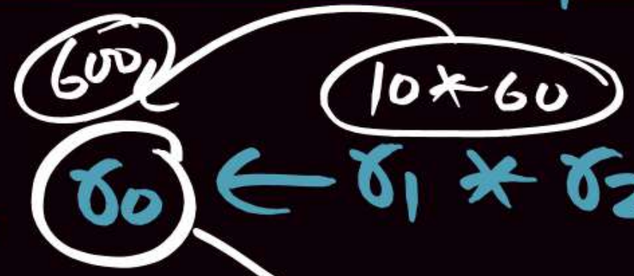


If first $I_1 (I_i)$ (Write) execute then $I_2 (I_j)$ (Read) execute. [ie Read After Write] — I_j Read After I_i Write.

then No Hazards (No Problem)

But When $I_j (I_2)$ tries to Read before $I_i (I_1)$ write then Hazards

RAW [Read After Write]. | True Data Dep.



Initial value $r_0 = 25$

$r_1 = 10$

$r_2 = 60$

If I_2 tries to Read r_0 before I_1 Update [write]

I_2 Read r_0 value $= 25$ — OLD value / Incorrect value

But actually r_0 value $= 600$

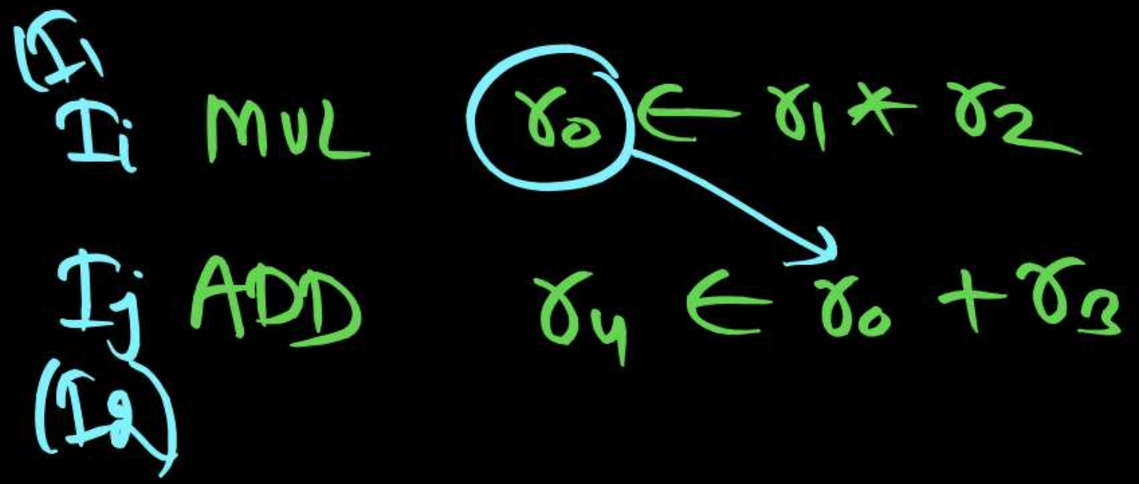
} Hazards

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

RAW



Types of Data Hazard

WAR | ANTI Dep. [Write, After Read]

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

(I_j) I_2 $\quad R_1 \in R_4 + R_5$

If $I_2(I_j)$ Tries to write the Data before $I_i(I_1)$ Read it.

Types of Data Hazard

WAW [Write After Write]

(Ii) I_1 $\quad R_3 \leftarrow R_1 * R_2$

(Ij) I_2 $\quad R_3 \leftarrow R_4 + R_5$

If Ij tries to write before Ii write it then Output Dep.

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

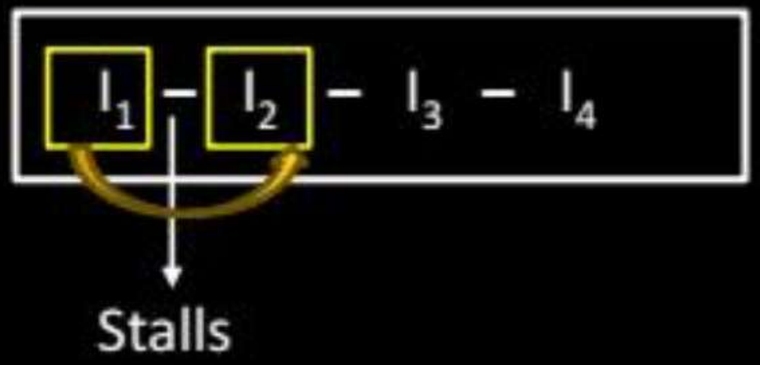
I_1: Add r_0, r_1, r_2

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

$I_2 \rightarrow r_3$ (Status 1)

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

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) WAR [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

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

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.

```
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 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]

(a) 2, 2, 4
(b) 3, 2, 3
(c) 4, 2, 2
(d) 3, 3, 2

THANK YOU!