COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_04

Vijay Agarwal sir

01 Pipelining Hazards

PIPELINE CONCEPT

ET in PIPELINE & NON PIPELINE

SPEED UP FACTOR

Throughput & effiecieny.

CPI = 1 Concept

↳ cycle Per Instruction

How to set this CPI in Uniform & Non Uniform Delay

Practice Question

15- GATE QUESTION

NIC EXAM Question.

Pipelining Strategy

To apply this concept
To instruction
Execution we must
Recognize that an
Instruction has a
Number of stages

Similar to the use of
An assembly line in a
Manufacturing plant

New inputs are
Accepted at one end
Before previously
Accepted inputs
Appear as output at
The other end

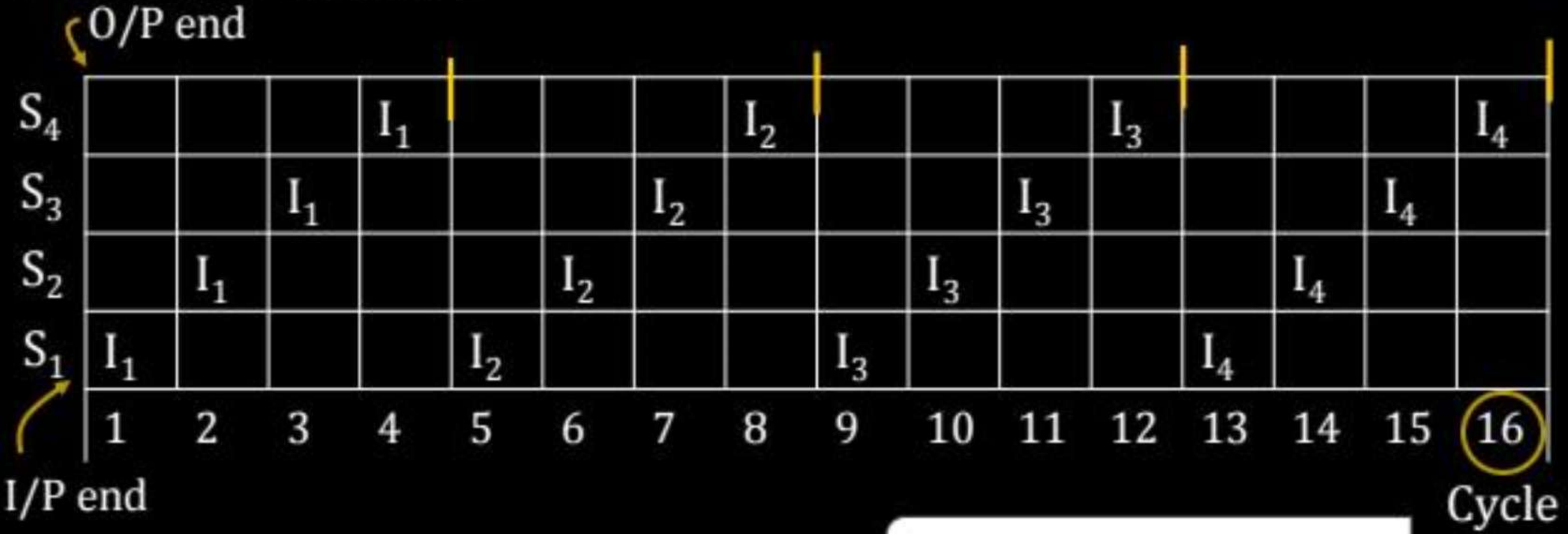
PIPELINE

- Pipelining is a mechanism which is used to improve the performance of the system in which task (Instruction) are executed in overlapping manner.
- Pipelining is a decomposition technique that means the problem is divided into sub problem & Assign the sub problem to the pipes then operate the pipe under the same clock.

Let us consider 4 segment pipeline used to execute 4 instruction the execution sequence as:

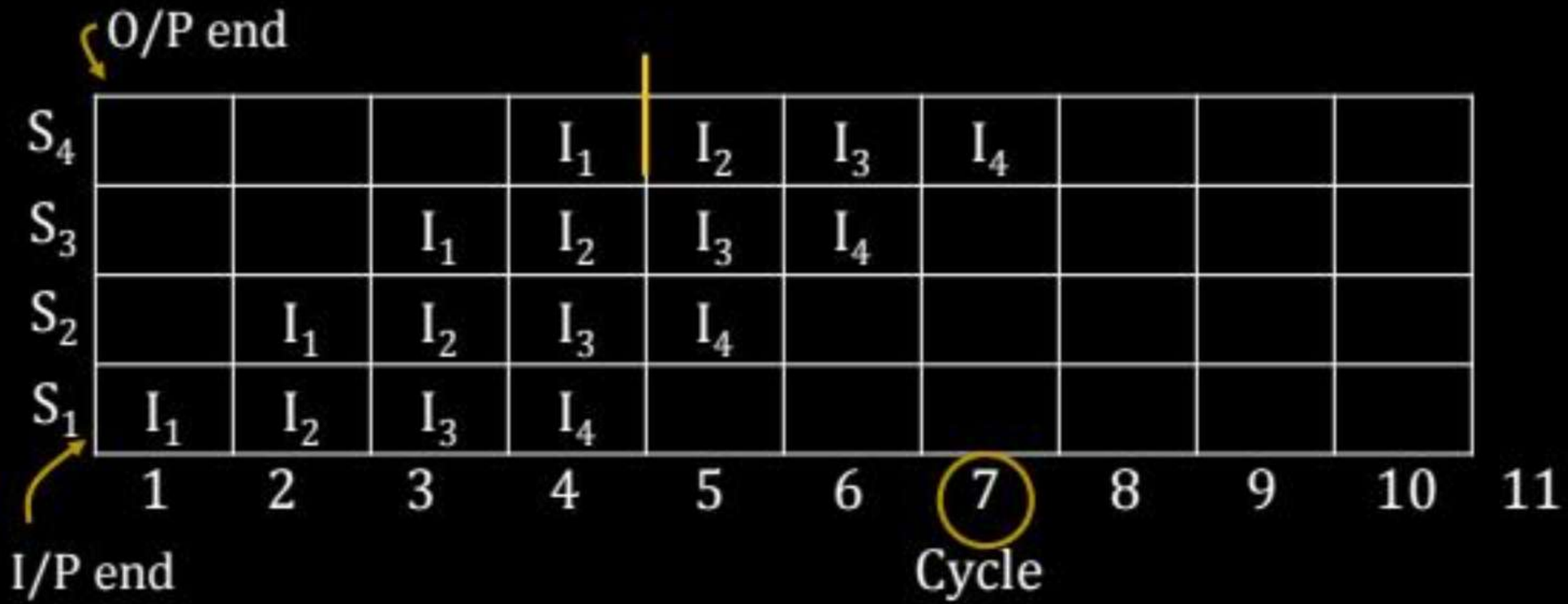
Segment/stages = $[S_1\ S_2\ S_3\ S_4]$

Instruction: $[I_1\ I_2\ I_3\ I_4]$



$n = 4,\ t_n = 4$, Non pipeline

Non-PIPELINE



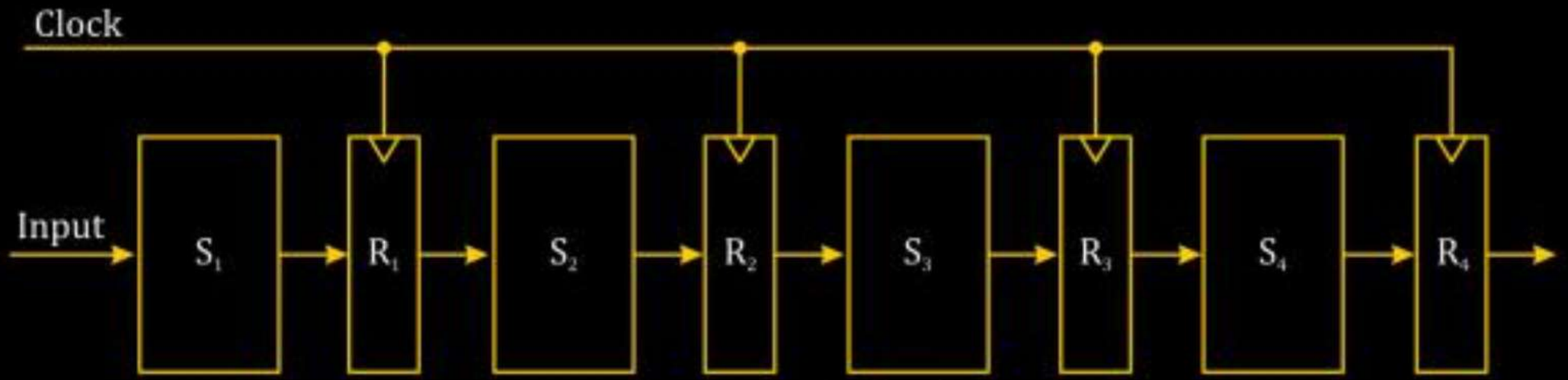
PIPELINE

$k = 4$

$n = 4$

PIPELINE

PIPELINE Design



Four-segment pipeline.

PIPELINE HAZARD's

CPI = 1

1000 I_1 | IF | ID | EX | MA | WB → JMP 1050

1001 I_2

⋮

1050 BI_1

RISC PIPELINE

In The RISC Pipeline 5 Stages:

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

Note If 'N Stage' entire CPU is divided into 'N functional Unit'.

RISC

1. IF
2. ID : (Decode, Operand Loaded from Registers)
3. EX : Data Processing

   MA : Memory Access

   WB : Write Back.

RISC PIPELINE

1. Instruction Fetch {IF Stage}: In this stage Instruction is fetched from Memory. (MEM to CPU(IR))

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

   (i) Decode the instruction

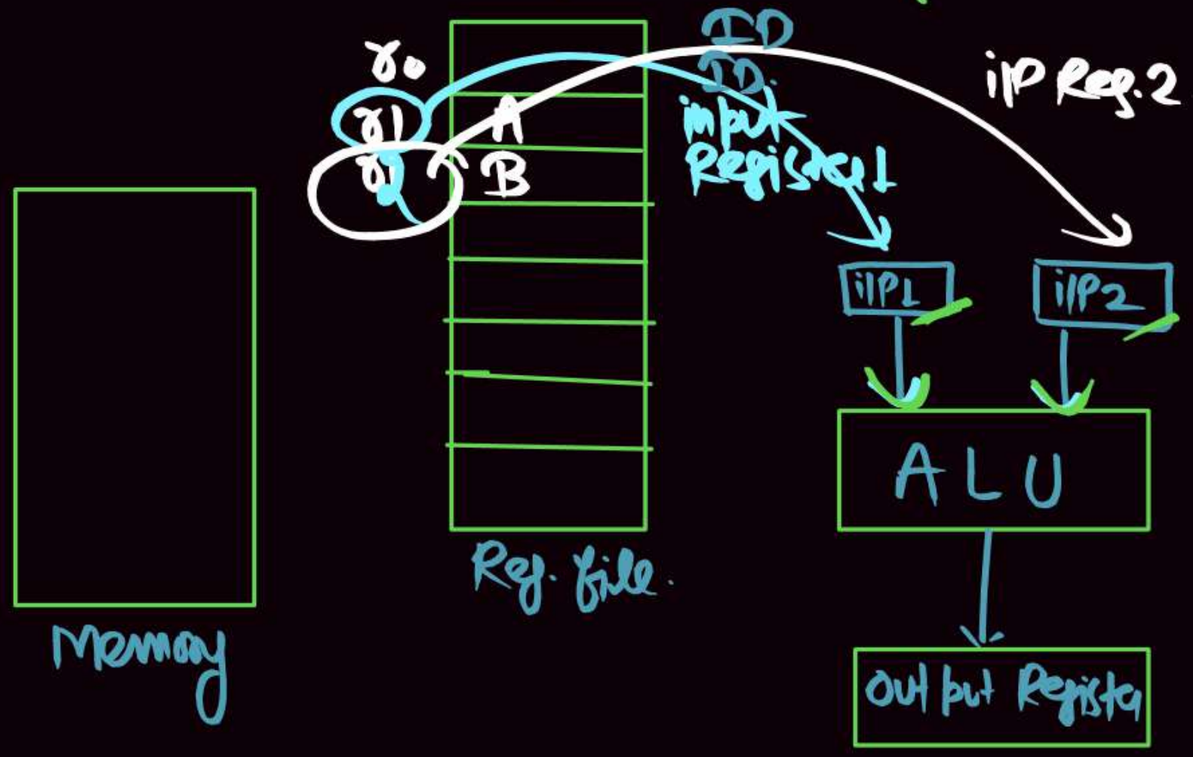
   (ii) Operand loading(fetching) from the register file.

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

Q.1

How the operand is available in the beggining of the Instruction ?

Directly in the Register file,



RISC

C ← A + B.

I1: LOAD γ_1 A

I2: LOAD γ_2 B

I3: ADD γ_3 γ_1 γ_2

I4: STORE C, γ_3

M[C] ← γ_3.

In ID Phase.

Soln) In the ID Stage we Fetch [Load] Operand from the 'Register file' [Not from memory], Because Previous Instruction already stored that operand in the Register file.

I_1 & I_2.
R_1 & R_2 Store 'A' & 'B'

Eg) I_3: ADD r_3, r_1 r_2

Here from the Register file r_1, r_2 Register Content Load into the input Register 1 & 2 for the ALU operation.

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 RISC 5 Stage

IF : [Mem to CPU]

ID : Decode

EX : ALU.

MA : Mem.

WB : WB.

But

In the GATE Question

Some times · 4 Stage,

· 5 Stage

· 6 Stage

are given

Consider :

① IF

② ID

③ OF [operand Fetch]

④ EXCPO (Perform operation)

⑤ MA

⑥ WB.

① Sometimes

IF & ID into 1 Stage.

② Sometimes

ID & OF into 1 Stage.

Additional Stages

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

Timing Diagram for Instruction pipeline operation

Extra cycle
(Stalls, Bubbles)

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

I_2 : $\gamma_4 \in \gamma_0 * \gamma_3$

I_2 Depand on I_1
ie I_2 use the Result of I_1
as a operand.
So I_2 must wait untill
Completion of I_1
This Waiting create 'extra cycle' 'Stalls'

Pipeline Hazards

Dependency in the Pipeline.

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

There are three Types of hazards:

1. Resource
2. Data
3. Control

Also referred to as a Pipeline bubble

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

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

Hazards/Dependencies In the pipeline

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

① Structural Dep. [Resource Conflict]

② Data Dep. [Due to operand]

③ Control Dep. [Branch / TOC Instn]

Structural Dependency

Structural Dependency is created when 2 or More Phase Require the same Resource, so that they are Not Able to Run (Process/Execute) simultaneously.

- Structural Dependency is created in the Pipeline Due to Resource Conflict.
- Resource may be Register, ALU or functional Unit etc.

Structural Dependency

CC: Clock Cycle.

	CC1	CC2	CC3	CC4	CC5
I_1	MEM	ID	ALU	MEM	WB
I_2		MEM	ID	ALU	MEM
I_3			MEM	ID	ALU
I_4				MEM	

Diagram (I)

Resource Conflict

- In CC4 Both I_1 & I_4 Accessing the Same Resource [memory] at the same time.
- This situation is called Resource Conflict & No One (I_1 & I_4) go for Execution..

- Here Both I_1 & I_4 Can not go for Execution.

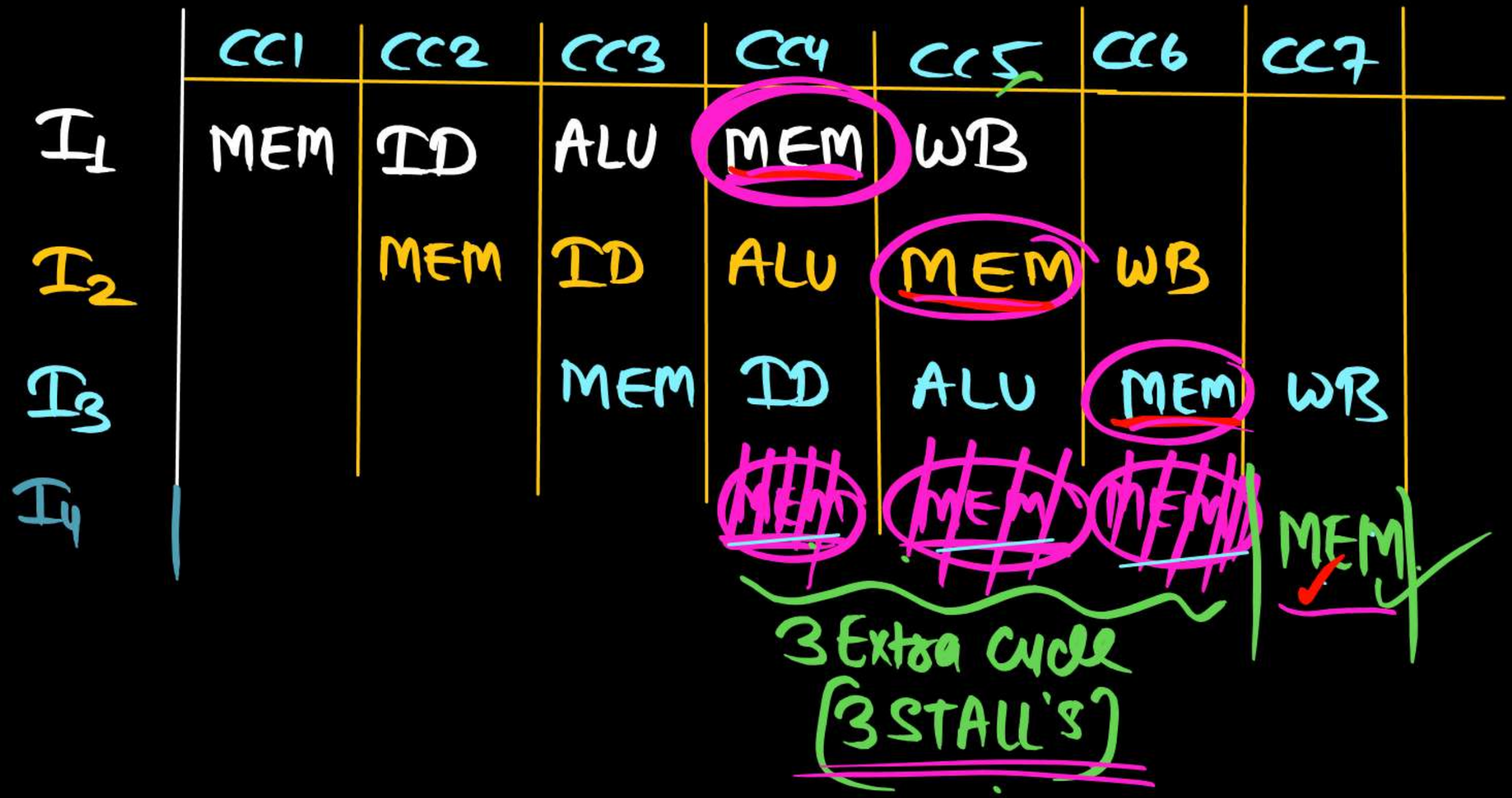
So keep I_4 into the Waiting Untill the Resource become Available.

This Waiting Creates an Extra Cycle [Stalls] in the Pipeline.

∴

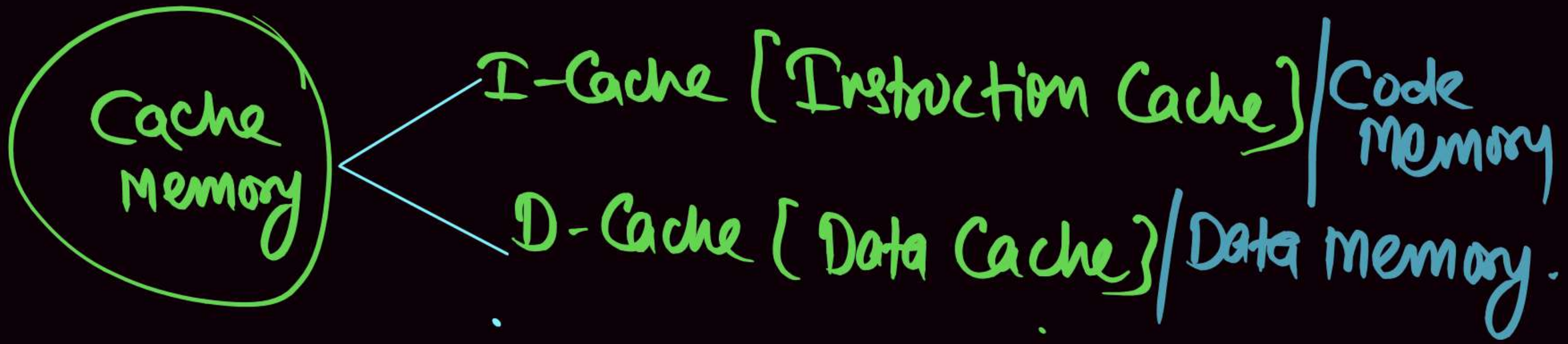
Structural Dependency

CC: Clock Cycle.



In this I_1 is Accessing Memory for Operand (Data) Store

I_4 is Accessing Memory to Fetch the Instruction

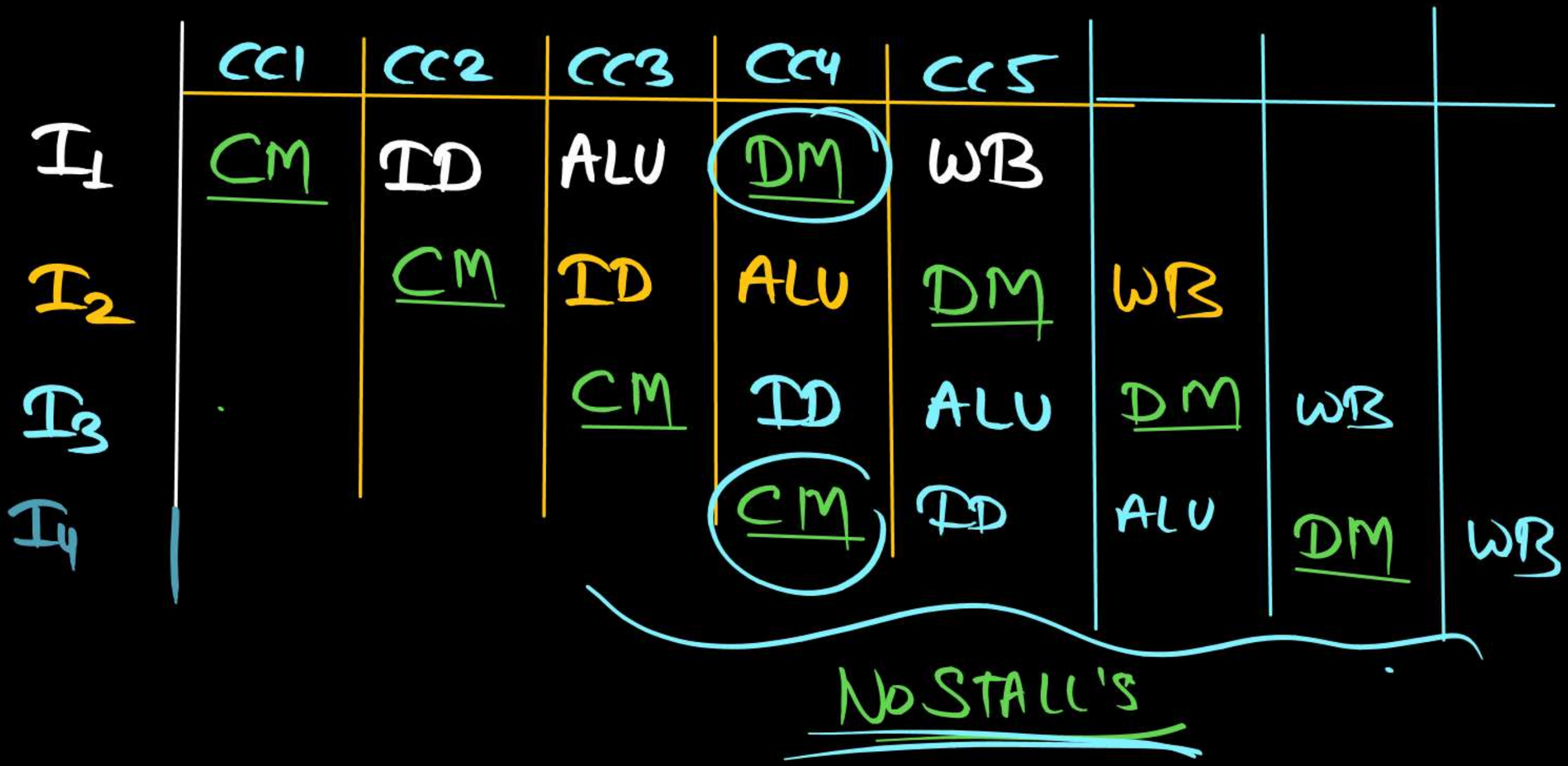


To Minimize the Stalls Due to Structural Dependency One Hardware Mechanism is Used Called as Renaming.

Renaming M/c ism State Divide the Memory into Indepent Module to Store the Instruction & Data Separately Called [I-Cache/CODE Memory [CM] & D-Cache/Data Memory [DM].

Structural Dependency

CC: Clock Cycle.



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 Data Dependant on I_1.

Data Dependency is created in the Pipeline when Next Instruction $[I_2 / I_{i+1} / I_j]$ use the Result of Previous Instruction $[I_1 / I_i / I_i]$ as operand / As a Input.

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 ; $R_4 \leftarrow R_1 * R_5$

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	
I_1:	IF	ID	OF	EX	MA	WB				
I_2:		IF	ID	STALL'S	STALL'S	STALL'S	OF	EX	MA	WR.

Data Dependency

- In this I_2 is Data Dependant on I_1
- So I_2 will wait Untill the output of I_1 (γ_1 value) is available.
- This waiting creates an extra cycle [Stalls] in the pipeline.

Solution of DATA Dep.

To Minimize the Stall in the Pipeline

Due to Data Dependency H/w M/c ism

is used Called ① OPERAND Forwarding / Short Circuit / By passing

② Hardware Interlock

OPERAND FORWARDING: Access the Value From Register before Update into Register file.

Data Dependency

OPERAND Forwarding

I_1 : ADD γ_1 γ_2 γ_3 ; $\gamma_1 \leftarrow \gamma_2 + \gamma_3$

I_2 MUL γ_4 γ_1 γ_5 ; $\gamma_4 \leftarrow \gamma_1 * \gamma_5$

	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		

No 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.

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.

Data Dependency

Data Hazard

(Read After Write) RAW (Data Dep.)

Write After Read WAR (Anti Dep.)

(Write After Write) WAW (Output Dep.)

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
EX
WB

EX

Without operand forwarding.

I_1	ADD	R2	R1	R0	R2 ← R1 + R0
I_2	MUL	R4,	R3,	R2	R4 ← R3 * R2
I_3	SUB	R6,	R5,	R4	R6 ← R5 − R4

Without operand forwarding : 12

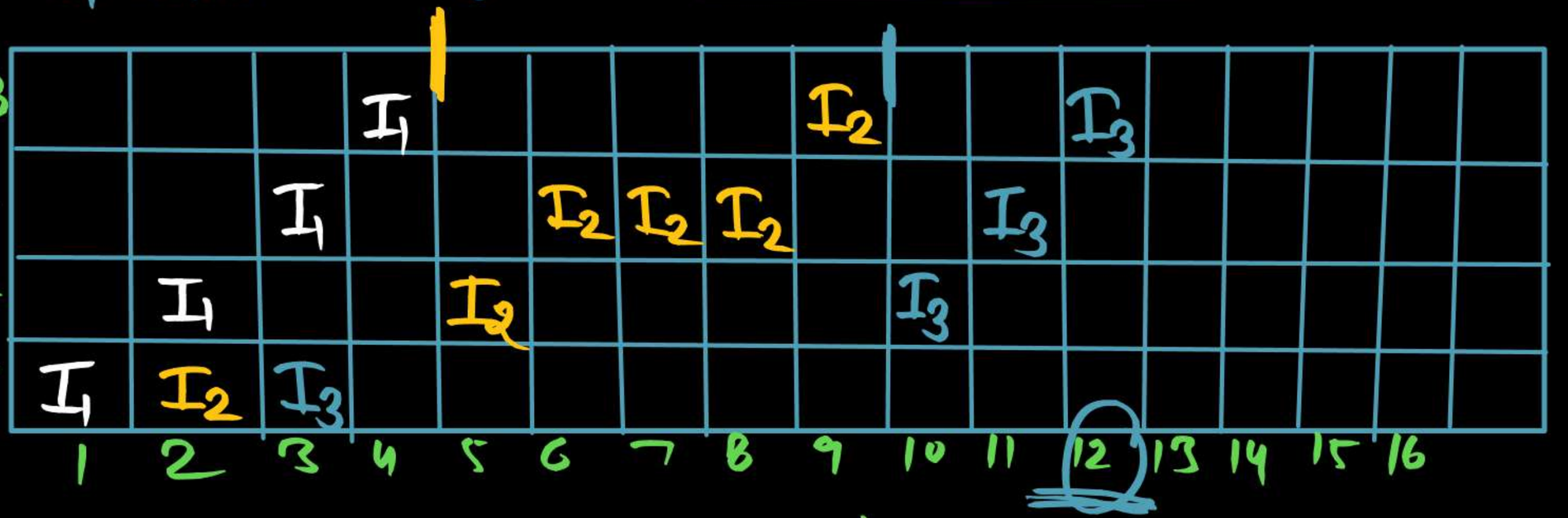
[GATE-2007 : 2 Marks]

A) 7

B) 8

C) 10

D) 14



IF
ID
EX
WB

EX

With operand forwarding.

1	I_1	ADD	R2	R1	R0	R2 ← R1 + R0
3	I_2	MUL	R4,	R3,	R2	R4 ← R3 * R2
1	I_3	SUB	R6,	R5,	R4	R6 ← R5 − R4

With operand Forwarding.

[GATE-2007 : 2 Marks]

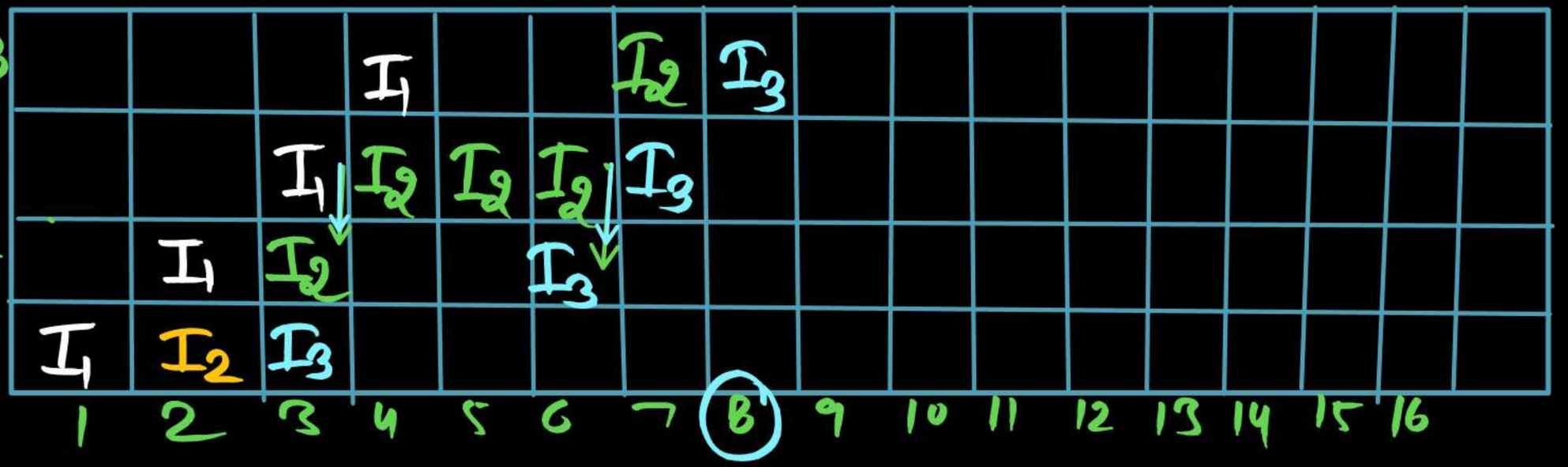
(8) Ans

(A) 7

(B) 8

(C) 10

(D) 14



IF
ID
EX
WB

EX				
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.8 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]

THANK YOU!