COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_06

Vijay Agarwal sir

01 Pipelining Hazards

1. Structural Hazards
2. Data Hazards
3. 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.

Stages can varies given in the Question

4 Stage

5 Stage

6 Stage

etc.

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:

1. Resource
2. Data
3. Control

Also referred to as a Pipeline bubble

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

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

Hazards/Dependencies In the pipeline

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

Structural Dependency

↓↓

Created Due to 'Resource Conflict'.

Resource : May be Registers, Memory or Functional Unit.

Structural Dependency

Soln of Structure Dep.

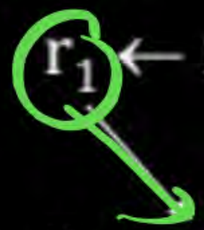
① Resource Replicition
(Use Additional Resource)

② Divide the Memory → CM/Im (Code Instruction Memory)
→ Dm (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 (I_j) using the Result of Previous Instn (I_i), as a operand.

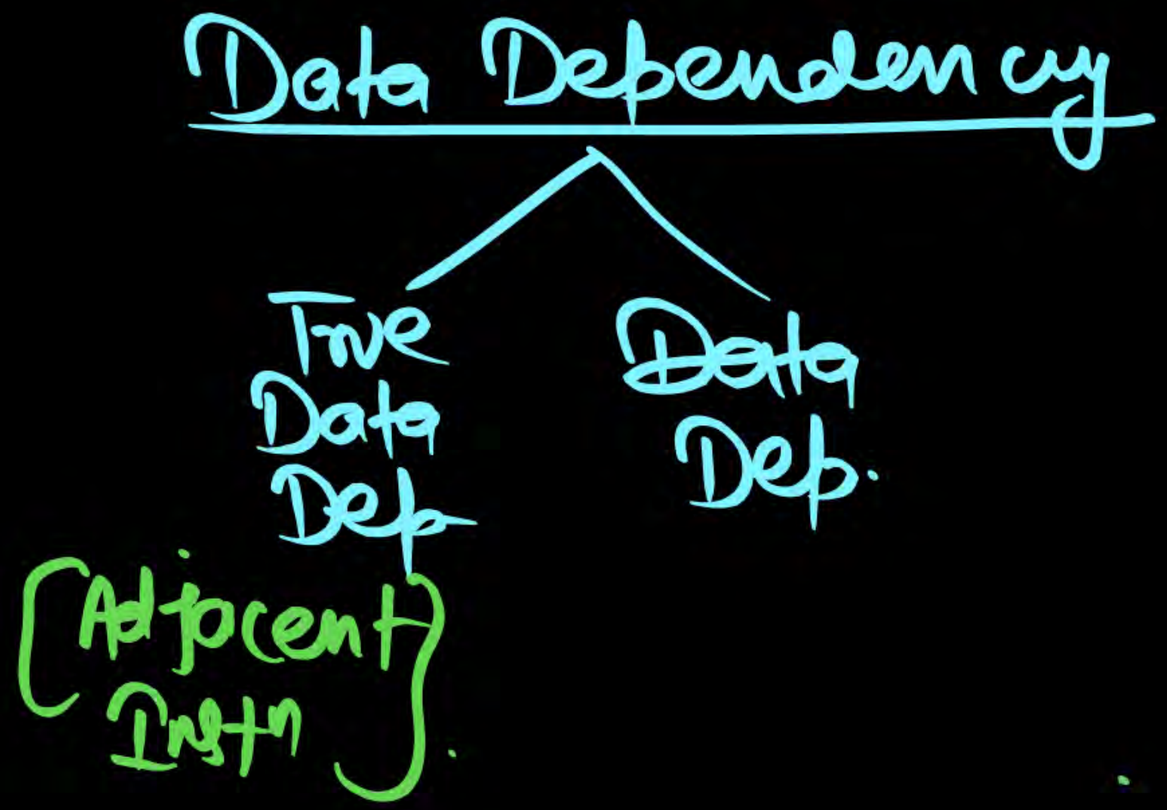
Data Dependency

1. RAW Data Dep. (True Data Dep.)
2. ANTI Dep. (WAR Dep.)
3. OUTPUT/Write Dep. (WAW Dep.)

Data Hazard

A data hazards is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason. Or Any condition that cause the pipeline to stalls is called as hazards.

Data Dependency



Data Dependency

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

I_1: Add r0, r1, r2 ($r_0 \leftarrow r_1 + r_2$)
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. $I_3 - I_1$

$I_4 - I_1$

W: Write
R: Read

Data Dep.

1. RAW Dep. (DATA Dep.) | Flow Dep
2. WAR Dep. (ANTI Dep)
3. WAW Dep. (OUTPUT | write Dep.)

] Name & False Dep.

Data/RAW Dependency

RAW [Read After Write].

Data dependency: creates in the pipeline when an instruction Ij (Next Instruction) depends on the result of a Ii (previous instruction),.

I_1: Add r0, r1, r2 I_i $\quad \gamma_0 \in \gamma_1 + \gamma_2$

I_2: Sub r3, r0, r2 I_j $\quad \gamma_3 \in \gamma_0 * \gamma_2$

RAW [Read After Write]

1. Data Dependency /RAW Dependency

1. Read after write (RAW), or Data dependency

- An instruction modifies a register or memory location.
- Succeeding instruction reads data in memory or register location

I_1: Add r0 , r1, r2
I_2: Sub r3 , r0, r2

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

I_j $\quad \gamma_3 \in \gamma_0 * \gamma_2$

RAW [Read After Write]

2. Anti Dependency /WAR Dependency

(2) Write after read (WAR), or anti dependency :

- An instruction reads a register or memory location.
- Succeeding instruction writes to the location.

I_1: MUL r3 , r1, r2
I_2: ADD r1 , r4, r5

I_i $\quad \gamma_3 \in \gamma_1 * \gamma_2$

I_j $\quad \gamma_1 \in \gamma_4 + \gamma_5$

Uneven Delay — WAR [Write After Read]

3. Output Dependency /WAW Dependency

③ Write after write (WAW), or Write/output dependency:

- Two instruction both write to the same location.

I_1: MUL r3 , r1, r2
I_2: ADD r3 , r4, r5

I_i (R) I_1 : $\gamma_3 \in \gamma_1 * \gamma_2$
I_j I_2 : $\gamma_3 \in \gamma_4 + \gamma_5$

WAW [Write After Write].

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.

Data/RAW Dependency

Data dependency: creates in the pipeline when an instruction Ij (Next Instruction) depends on the result of a Ii (previous instruction),.

I_1: Add r0 , r1, r2
I_2: Sub r3 , r0, r2

Data Hazard

A data hazards is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.

Data Dep.

Data Dependency & Hazards

5 stage (IF, ID, EX, MEM, WB)

eg1. Consider the following program segment, executed on a RISC pipeline:

Q.1 **How** Many RAW(Data Dependency) Dependency?

$r_0 \leftarrow r_1 + r_2$

I_1: Add r0, r1, r2
I_2: Sub r3, r0, r2
I_3: MUL r4, r0, r5
I_4: DIV r0, r6, r7
I_5: MUL r8, r0, r9

Dep.

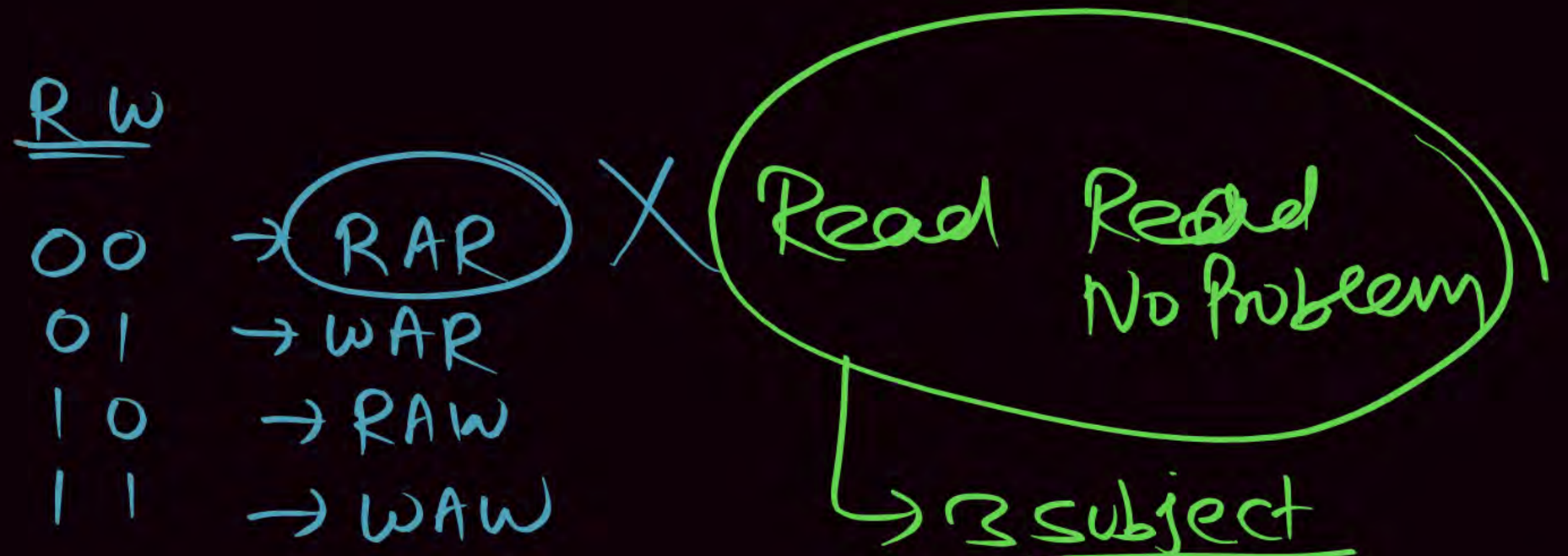
$I_2 - I_1$
$I_3 - I_1$
$I_5 - I_4$

OR

$I_1 - I_2$
$I_1 - I_3$
$I_4 - I_5$

$I_5 - I_1$ X

Q. How to check RAW Dep. creates Hazard or Not ?



DBMS ⇒ RR No Problem

OS ⇒ Reader ✓ Not Conflict Left Reader ✓

COA: Read & Read No Problem

Data Dependency & Hazards

eg2. Consider the following program segment, executed on a RISC pipeline:

Q.1 **Check** Here RAW(Data Dependency) Dependency Create RAW Hazards?

$r_0 \leftarrow r_1 + r_2$

I_1: Add r0, r1, r2
I_2: Sub r3, r4, r5
I_3: MUL r6, r0, r7
I_4: Add r_7 r_5 r_8
I_5: ADD r_9 r_0 r_{10}

RAW Dep. : 2 Data Dep.

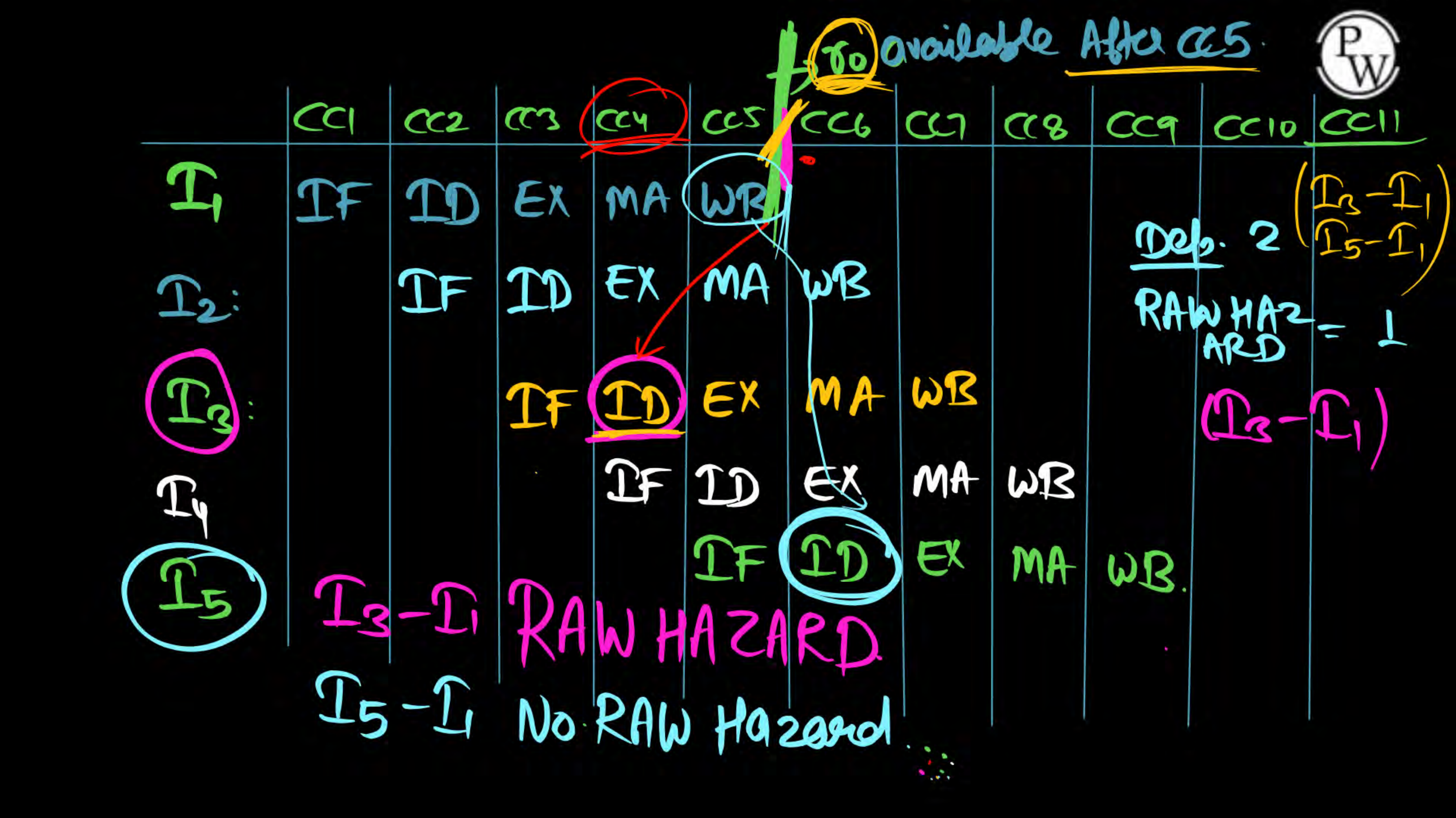
- $I_1 - I_3$ ✓
- $I_1 - I_5$ ×

(or) $I_3 - I_1$, $I_5 - I_1$

RAW Hazard : $I_3 - I_1$

Q How to CHECK RAW Dep. Create RAW Hazards or Not ?

Soln Pipeline Execution then Check if that Result is Required earlier or Not by Next Instruction.


R0 available After CC5
CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8 CC9 CC10 CC11
I1 IF ID EX MA WB
I2: IF ID EX MA WB
I3: IF ID EX MA WB
I4 IF ID EX MA WB
I5 IF ID EX MA WB
Dep. 2 (I3-I1, I5-I1)
RAW HAZARD = 1
(I3-I1)
I3-I1 RAW HAZARD
I5-I1 No RAW Hazard

Note

Dependency may or may Creates Hazards.

Data Dependency & Hazards

eg3. Consider the following program segment, executed on a RISC pipeline:

Q.1 How Many RAW(Data Dependency) ?

Q.2 How Many RAW Hazards ?

$r_0 \leftarrow r_1 + r_2$

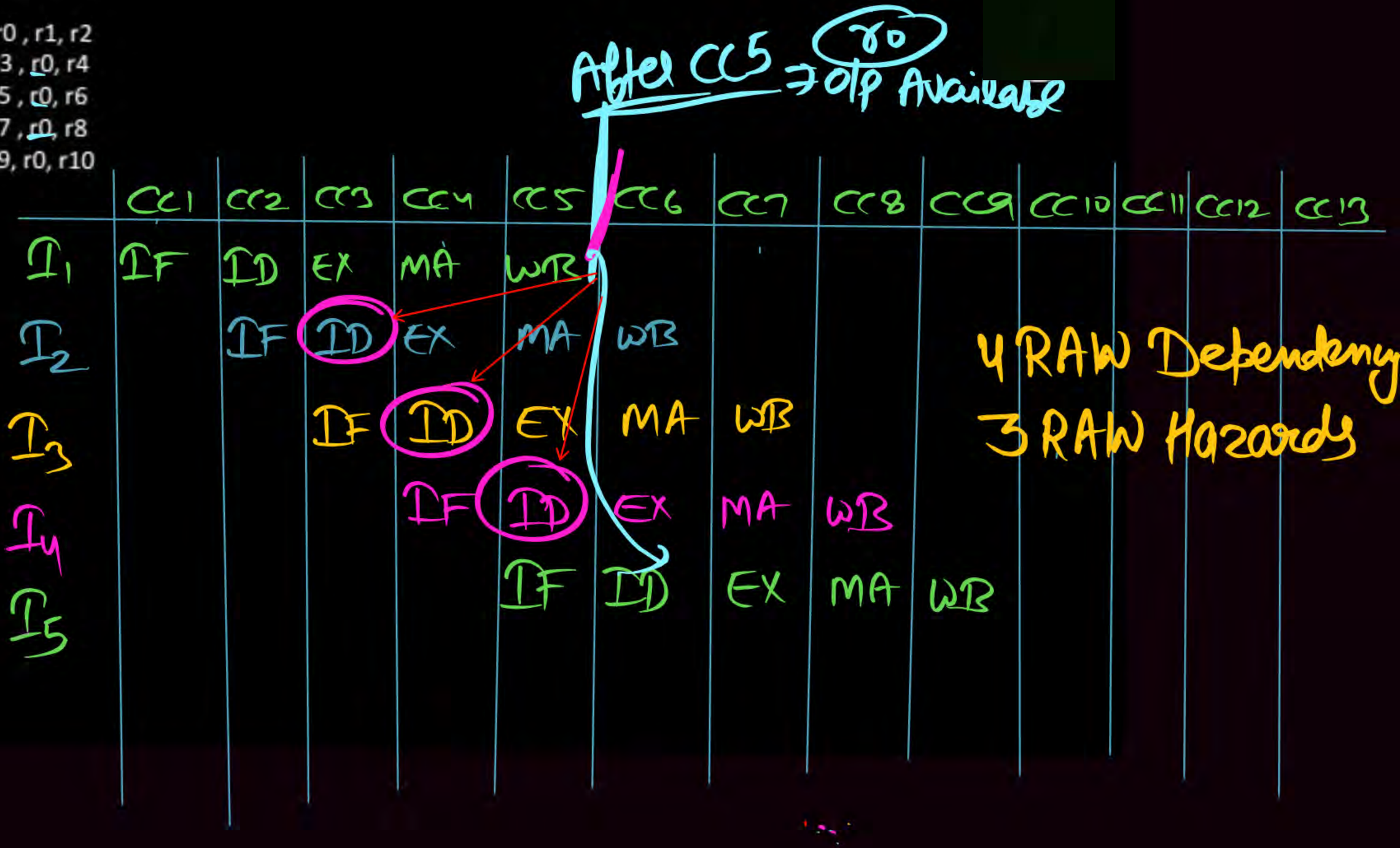
I_1: Add r0 , r1, r2
I_2: Sub r3 , r0, r4
I_3: MUL r5 , r0, r6
I_4: XOR r7 , r0, r8
I_5: MUL r9, r0, r10

Data (RAW) Dep.

$I_2 - I_1$
$I_3 - I_1$
$I_4 - I_1$
$I_5 - I_1$

4 RAW Dep.
3 RAW Hazards.

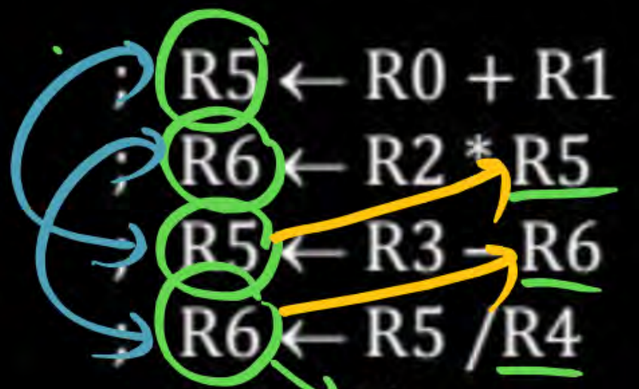
I_1: Add r0 , r1, r2
I_2: Sub r3 , r0, r4
I_3: MUL r5 , r0, r6
I_4: XOR r7 , r0, r8
I_5: MUL r9, r0, r10



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



RAW Dep. 4 (I_2-I_1, I_3-I_2, I_4-I_3, I_5-I_4)

WAR Dep : 2 (I_3-I_2, I_4-I_3)

WAW Dep : 2 (I_3-I_1, I_4-I_2)

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

Data Dependency.

Data Dependency

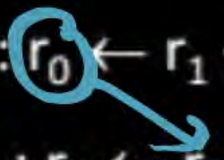
Data dependency will be occurred when the instruction "J" try to read the data before instruction "I" writes it.

"Read – Before – write"

Ex.-

I_1: 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

- 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].

In Non bipeline : Non overlapping Execution
No Problem in Data Dep.

Data Dependency

- 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].

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

Read – Before write

Old (wrong)Data Access {Data loss}

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_2	r_0 (I_1)	–	0	r_2 = value EX- Stage not alocated

"0" execution yet not complete
"1" execution complete


Dependency ?
Domain & Range

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 :

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	ID	ID r_0 = value	EX
I_3			IF	IF	IF	IF	ID
I_4				≡	≡	≡	IF

3 STALL'S

Read –After Write

7 cycles – 4 i/ps = 3 stalls

Solution of Data Dependency.

- ① Sw Soln (Compiler Based)
  - NOP Instruction.
- ② Hw type Soln
  - Hw Interlock
  - operand Forwarding / Bypassing / Short circuit.

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.

Solution of Data Dependency

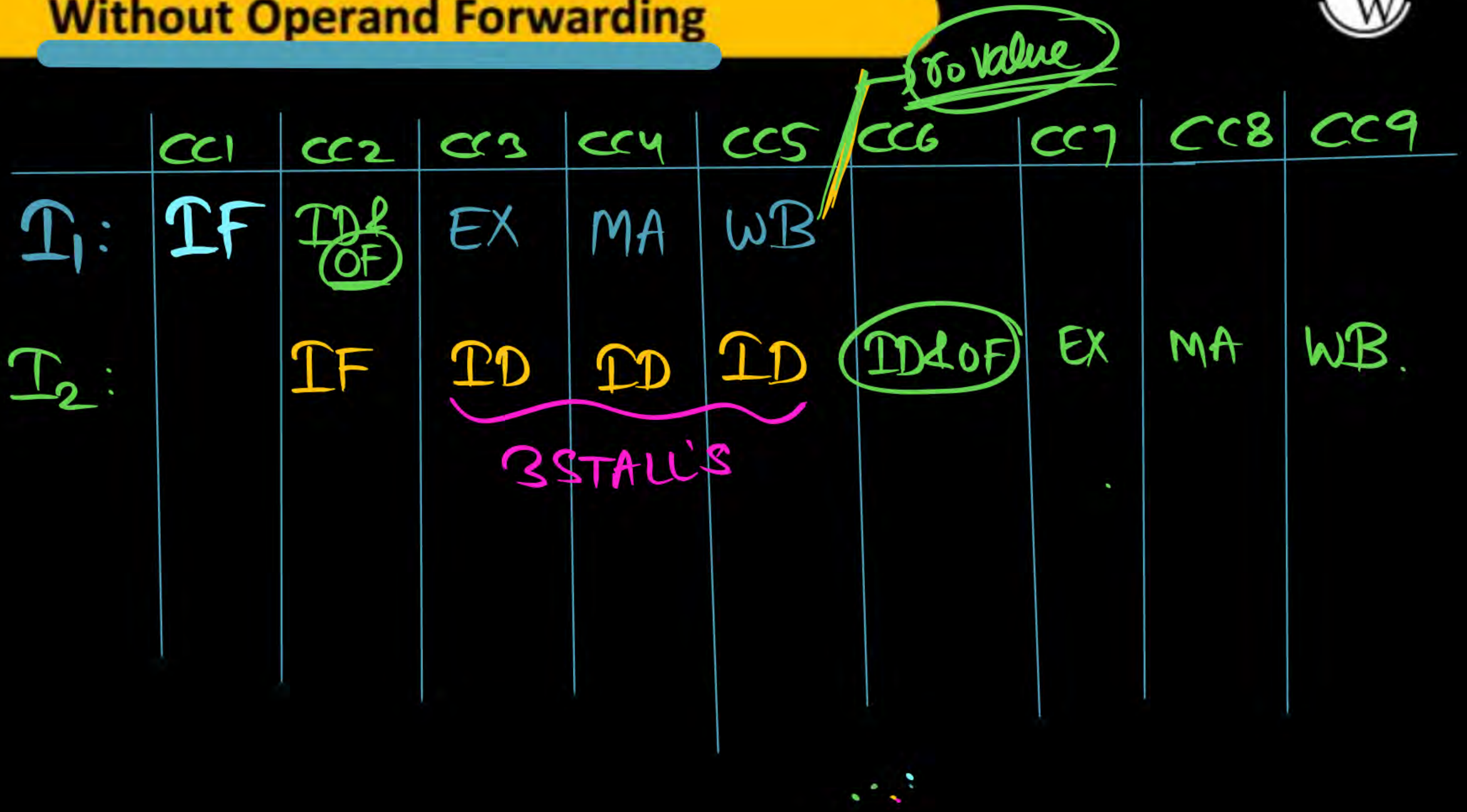
- The minimize the data dependency stall in the pipeline hardware technique is used i.e. operand forwarding. Also named as **By-passing or Short –circuit**
- This mechanism state that use the buffer between the stage to hold the intermediate result so that, dependent instruction will be accessing the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

I_1 : ADD γ_0 γ_1 γ_2 ; $\gamma_0 \in \gamma_1 + \gamma_2$

I_2 : MUL γ_3 γ_0 γ_4 ; $\gamma_3 \in \gamma_0 * \gamma_4$

I_2 Data Dependant on I_1.

Without Operand Forwarding



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

OPERAND - forwarding.

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?

EX

1 I_1
3 I_2
1 I_3

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

without operand Forwarding

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
I_1:	IF	ID	EX	WB									
I_2		IF	ID	ID	ID	EX	EX	EX	WB				
I_3			IF	ID	ID	ID	ID	ID	ID	ID	EX	WB.	

Without operand Forwarding = 12.

without operand Forwarding

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
I_1:	IF	ID	EX	WB									
I_2		IF	////	////	ID	EX	EX	EX	WB				
I_3			IF	//	//	//	//	//	//	ID	EX	WB	

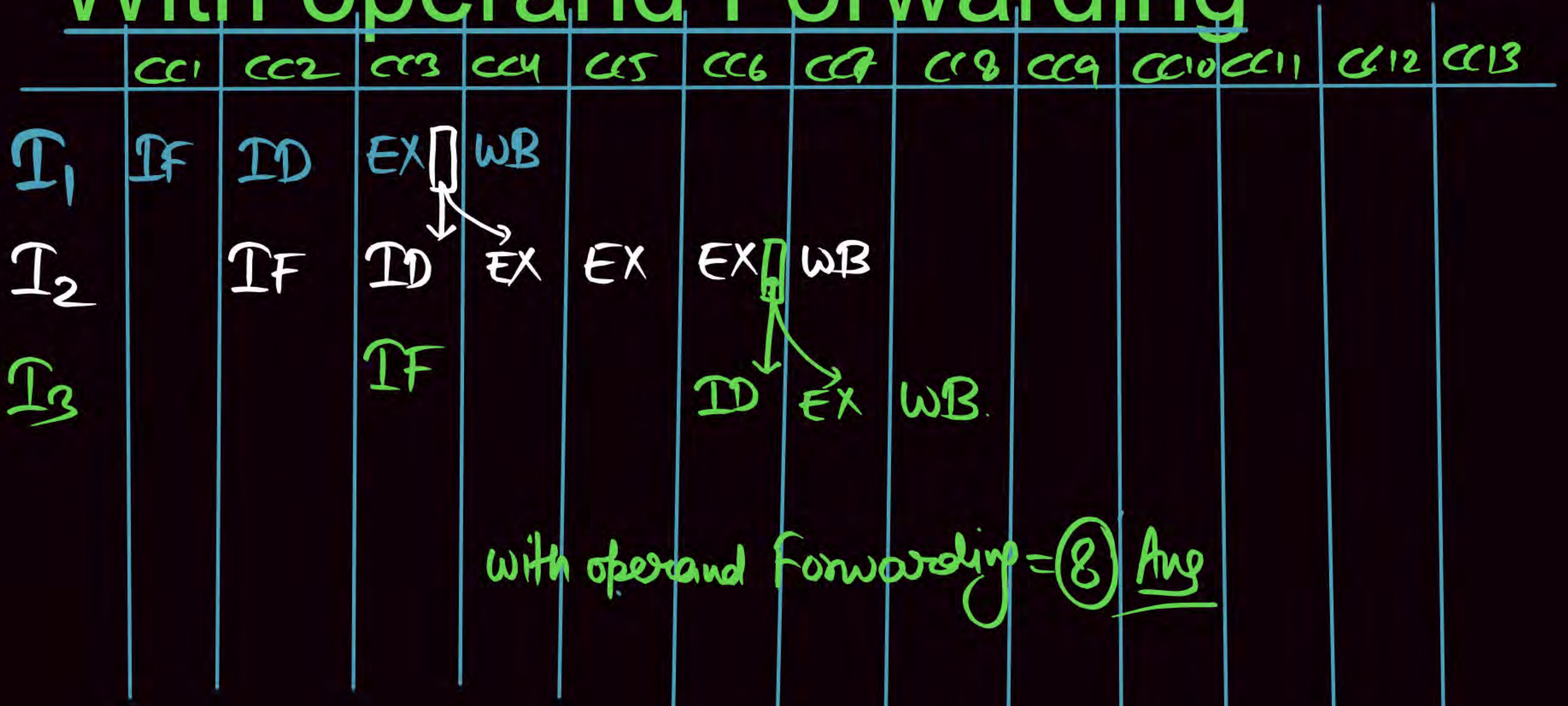
Without operand Forwarding = 12

without operand Forwarding :

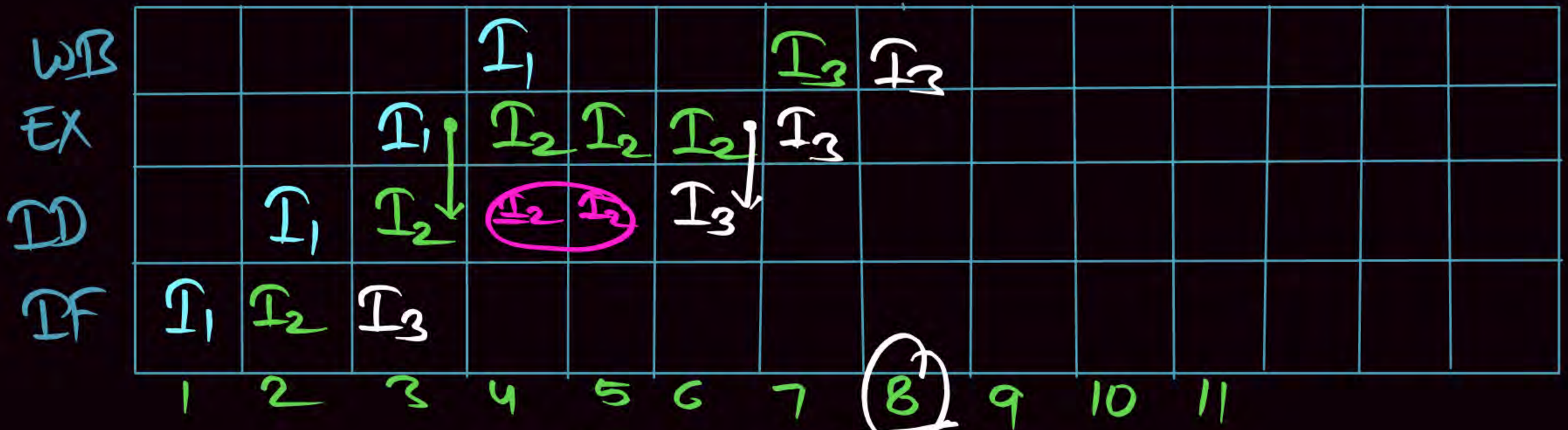
Without operand Forwarding 12

With operand forwarding.

With operand Forwarding



with operand Forwarding



With operand Forwarding = 8.

OR Another Approach

Without timing Diagram.

(Not Preferable).

By Default In Pipeline CPI = 1

Restriction
Only One type of Dep. Then this method work.

But Some Instructions takes More Cycle [Extra cycle] in Some Stage.

$$\text{Total ET} = ET_{\substack{\text{PIPE} \\ \text{Normal}}} + \text{Stalls [Extra cycle]}$$

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?

$k=4$ 4 Stage

$n=3$ 3 Instruction (I_1, I_2, I_3).

IF : 1

ID : 1

EX : 3 (2 Extra Cycle [Stalls])

WB : 1

Stalls = 2 Cycle

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

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

$= 6$ Cycle

Total ET = ET_{PIPE} + Stalls

$\Rightarrow 6+2$

$= 8$ Cycle Ans

$n=3,\ k=4.$

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

I_1 I_2 I_3

[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

H.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. 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

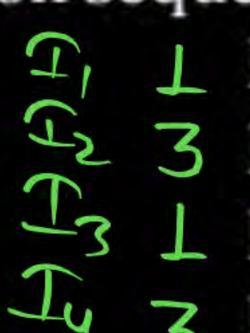
(B) 2 and 3 only

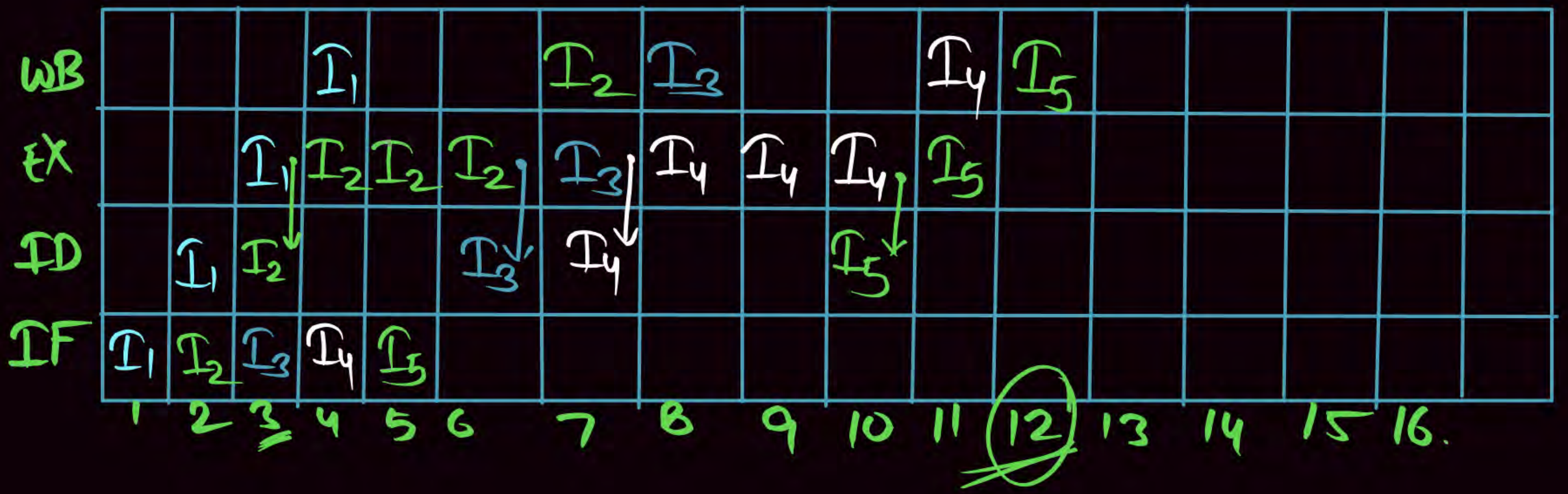
(C) 3 only

(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	1	R5, R0, R1	; R5 ← R0 + R1
MUL	3	R6, R2, R5	; R6 ← R2 * R5
SUB	1	R5, R3, R6	; R5 ← R3 − R6
DIV	3	R6, R5, R4	; R6 ← R5 /R4
STORE	1	R6, X	; X ← R6




WB
EX
ID
IF
I1
I2
I3
I4
I5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16.

Another Approach

#Instⁿ = 5 (n = 5)

Stage = 4 (k = 4)

		Extra
MUL :	3	2
DIV :	3	2
Total Extra (Stalls)		4

IF |
ID |
EX |
WB |

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

$\Rightarrow [4 + 5 - 1)]$

$= 8$ cycle

Total ET $= ET_{PIPE} + Stalls$

$\Rightarrow 8 + 4$

$= 12$ cycle

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

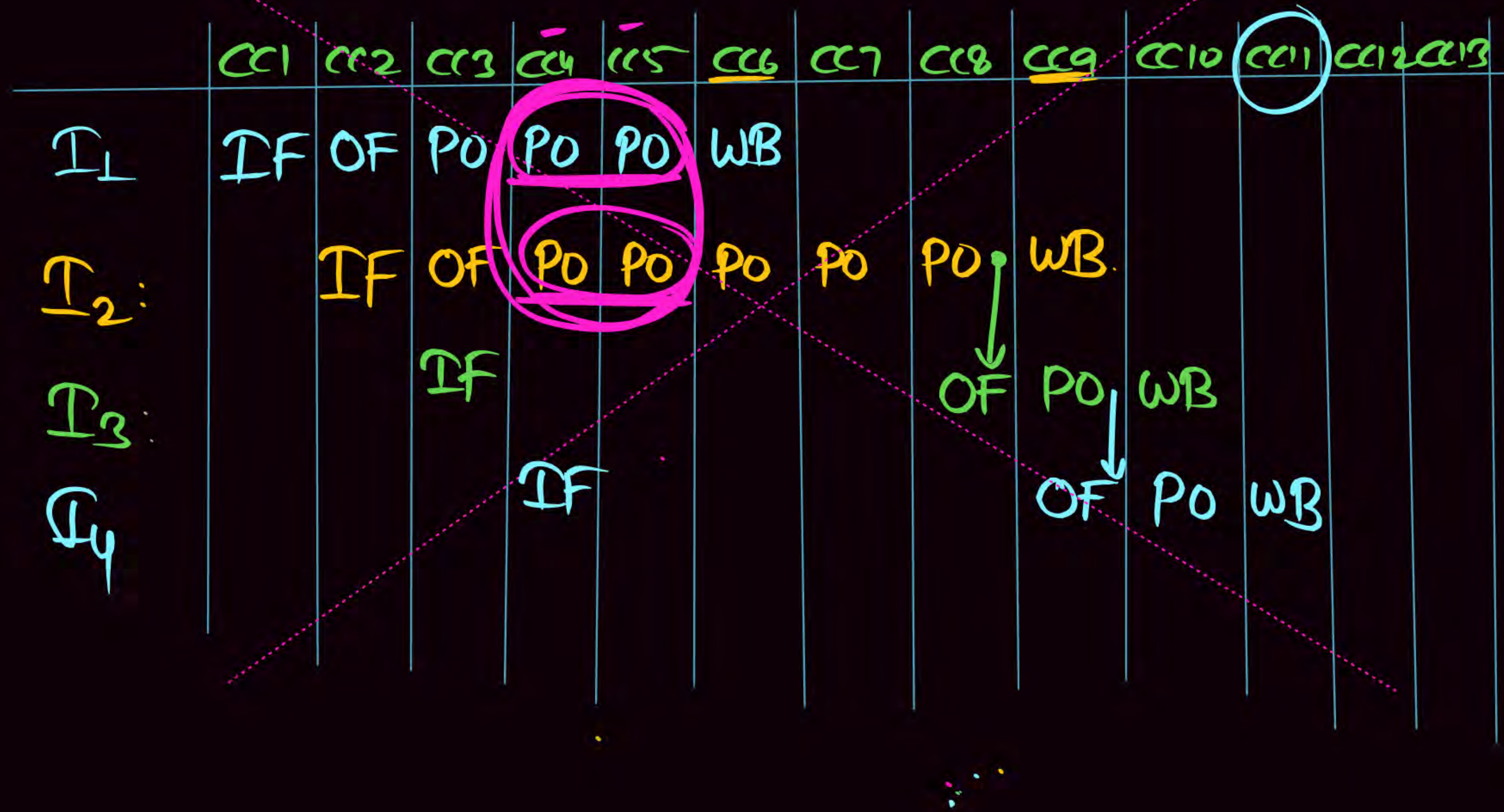
Q. Consider the sequence of machine instructions given below:

MUL	I_1	R5, R0, R1	$R_5 \leftarrow R_0 * R_1$
DIV	I_2	R6, R2, R3	$R_6 \leftarrow R_2 / R_3$
ADD	I_3	R7, R5, R6	$R_7 \leftarrow R_5 + R_6$
SUB	I_4	R8, R7, R4	$R_8 \leftarrow R_7 - R_4$

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 cycle

IF
OF
PO
WB

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


CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8 CC9 CC10 CC11 CC12 CC13
I1 IF OF PO PO PO WB
I2: IF OF PO PO PO PO PO WB
I3 IF OF PO WB
I4 IF OF PO WB


WB
PO
OF
IF
I1
I2
I3
I4
1 2 3 4 5 6 7 8 9 10 11 12 13
13 cycle

By Another Approach

$n = 4$ ($I_1\ I_2\ I_3\ I_4$)

$k = 4$ (4 stage)

IF	1
OF	1
PO	1
WB	1

$$ET_{PIPE} = [k + (n-1)]\,tp$$

$$\Rightarrow [4 + (4-1)]\ \text{cycle}$$

$$ET_{PIPE} = 7\ \text{cycle}$$

CPI = 1

		Extra
MUL	3	2
DIV	5	4
ADD	1	0
SUB	0	0

6 Stalls

$$ET = ET_{PIPE} + \text{Stalls}$$

$$\Rightarrow 7 + 6$$

$$= 13\ \text{cycle}$$ Ans

$\therefore$

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]

$k=5,\ n=100$

IF : 1
ID : 1
OF : 1
PO : (1)
WB : 1

		(Stalls) Extra
PO : 40 Instⁿ :	3	2
35 Instⁿ :	2	1
25 Instⁿ :	1	0

Total Stalls $\Rightarrow 40\times 2 + 35\times 1$

$\Rightarrow 80 + 35$

$= 115$ Stalls.

$n=100,\ k=5$

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

$\Rightarrow [5+(100-1)]$

$= 104$ cycle

Total ET = ET_{PIPE} + Stalls

$\Rightarrow 104 + 115$

$= 219$ Ans

k=5, n=100

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

4

PO : 40 Instn : 3
35 Instn : 2
25 Instr : 1

PO stage
= 40×3 + 35×2 + 25×1
= 120 + 70 + 25
= 215 cycle

Total ET = 4 + PO Stage Total

⇒ 4 + 215

⇒ 219 cycle Ans

∴

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

THANK YOU!