COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_05

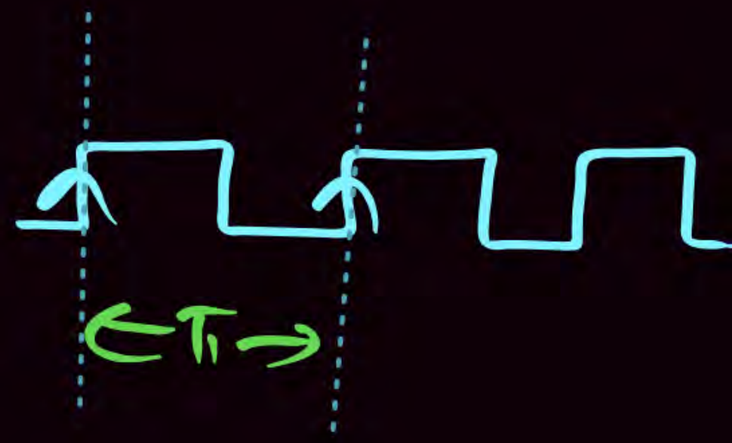
Vijay Agarwal sir

01 Pipelining Hazards

PIPELINE Concept

Execution time

Performance Evaluation



Efficiency & Throughput

Uniform Delay & Non Uniform Delay.

CPI (Cycle Per Instruction)

How to set this CPI in Uniform & Non Uniform Delay Pipeline.

RISC | MIPS | Standard | Classical Pipeline.

RISC/MIPS

PIPELINE Hazards.

1. Structural Hazards
2. Data Hazards.
3. Control Hazards

V.V.V.V.Imp

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 .

Decoder 2

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.

Registe write [write into the Register].

Different - Different Stage

2 Stage

4 Stage

5 Stage

6 Stage etc.

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

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:

1. Resource
2. Data
3. Control

Also referred to as a Pipeline bubble

Hazards.

↳ Stalls | Extra cycle | Bubbles

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 | Hazards.

CC: Clock cycle

→ Resource Conflict

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	MEM	ID	ALU	MEM	WB.				
I_2		MEM	ID	ALU	MEM	WB			
I_3			MEM	ID	ALU	MEM	WB		
I_4				MEM					

keep I_4 into waiting.

But this waiting creates

Stalls | Extra cycle | bubbles in the pipeline.

Structural Dependency | Hazards.

CC: Clock cycle

→ Resource Conflict

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	MEM	ID	ALU	MEM	WB.				
I_2		MEM	ID	ALU	MEM	WB			
I_3			MEM	ID	ALU	MEM	WB		
I_4				MEM	MEM	MEM	MEM		

3 Stalls

Soln of Structural Dep: Renaming.

I_1: Accessing the memory for Data Store.

I_4: Accessing the memory for Instruction Fetch

MEM
- I-Cache [Instn/Code memory] [IM/CM]
- D-Cache [Data Memory] [DM].

Structural Dependency | Hazards.

CC: Clock cycle

→ Resource Conflict

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	CM	ID	ALU	DM	WB.				
I_2		CM	ID	ALU	DM	WB			
I_3			CM	ID	ALU	DM	WB		
I_4				CM	ID	ALU	DM	WB	

No Stalls.

Data Dependency

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

(or) I_j I_{i+1} I_2 : MUL r_3 r_0 r_4 : $r_3 \leftarrow r_0 \times r_4$

I_2 Depend on I_1.

Because I_2 is Using the Result of I_1 as a operand.

2 Data Dependency

I_1 : LW r_0 $100[r_1]$; $r_0 \leftarrow M[100 + r_1]$.

I_2 : MUL r_3 r_0 r_4 : $r_3 \leftarrow r_0 * r_4$

2 Data Dependency

I_1 : SUB r_0 r_1 r_2 : $r_0 \leftarrow r_1 - r_2$

I_2 : MUL r_3 r_0 r_4 : $r_3 \leftarrow r_0 * r_4$

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 ; $r_4 \leftarrow r_1 * r_5$

Instruction I_2 is Data Dependant on Instruction I_1.

Because Instn I_2 using the Result of Instruction I_1, as operand.

Data Dependency

Data Dependency is Created in the Pipeline when Next (I_j or I_{i+1} or Next Instⁿ) Instruction Using the Result of (Previous | I_i Instruction)

Concept of

Data Dependency & DATA Hazards.

Dependency

1. Data Dependency | True | Flow Dependency.
2. ANTI Dependency
3. Output | Write Dependency.

Hazards.

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.

Hazard is Condition which create stalls. → Stalls | Extra Cycle | Bubbles.

called

Data Hazards:

R: Read
W: Write.

1. RAW (Read After Write)
2. WAR (Write After Read)
3. WAW (Write After Write)

Data Dependency

1. Data Dependency | True | Flow Dependency.
   ↳ RAW Hazards.
2. ANTI Dependency → WAR Hazards.
3. Output | Write Dependency → WAW Hazards.

Important Point

Note: Every Dependency Not Creates Stalls.

Note: ANTI & OUTPUT Dependency Never Creates Stalls.

Note: Data Dependency May (or) May Not Creates Stalls | Bubbles | Extra Cycle [RAW Hazards].

Difference b/w DATA Dependency & DATA Hazards.

Q.1 WHAT is Dependency ?

Q.2 WHAT is Hazards ?

Q. Dependency Concept

$OP \quad r_i \; r_j \; r_k : \quad r_i \leftarrow r_j \; \underset{\text{operation}}{\odot} \; r_k$

$\boxed{r_0 \leftarrow r_1 + r_2}$

Q.1

I_1 : ADD r_0 r_1 r_2

I_2 : MUL r_3 r_0 r_4

I_3 DIV r_5 r_0 r_6

I_4 MUL r_7 r_0 r_8

I_5 ADD r_9 r_0 r_{10}

I_6 MUL r_{11} r_0 r_{12}

Soln

Data Dep.

1. $I_2 - I_1$
2. $I_3 - I_1$
3. $I_4 - I_1$
4. $I_5 - I_1$
5. $I_6 - I_1$

5 Data Dep.

Stalls.

Q Is Data Dependency Always Create RAW Hazards?

↳ Now RAW Hazard Concept Along with Data Dep.

$OP \quad r_i \quad r_j \quad r_k : \quad r_i \leftarrow r_j \overset{\text{operation}}{\odot} r_k$

$\boxed{r_0 \leftarrow r_1 + r_2}$

Q.2

I_1 : ADD r_0 r_1 r_2

I_2 : MUL r_3 r_{13} r_4

I_3 DIV r_5 r_{14} r_6

I_4 MUL r_7 r_{15} r_8

I_5 ADD r_9 r_0 r_{10}

I_6 MUL r_{11} r_0 r_{12}

Soln 2

Data Dep.

$I_5 - I_1$
$I_6 - I_1$

2 Data Dependency

Instruction I_6 Depends on I_1 so its Data Dep. b/w I_6 & I_1 & $I_5 - I_1$

IF (Instn Fetch)

ID (Operand Fetch)

EX

Load & Store → MA

WB.

Operand Fetch ⇒ ID

Now in Q.2 there is a2 Data Dependency b/w (I_5 & I_1) & (I_6-I_4) Now check it creates RAW Hazards (Stalls) or Not ?

Stall 2

RISC : 5 Stage IF, ID, EX, MA, WB.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
I_1	IF	ID	EX	MA	WB								
I_2		IF	ID	EX	MA	WB							
I_3			IF	ID	EX	MA	WB						
I_4				IF	ID	EX	MA	WB					
I_5					IF	ID	EX	MA	WB				
I_6						IF	ID	EX	MA	WB			

80

- At CC5 [Clock Cycle 5] we getting the value of Register r_0 of Instruction I_1.
  (operand)
- After CC5, if r_0 is Need then there is No Hazards. (No Stalls).

In Q.2 Data Dependency b/w $I_5 - I_1$
$I_6 - I_1$

↳ 2 Data Dependency.

But this 2 Dep Not Creates Stalls | Bubbles

↳ 0 RAW Hazards

(Not Every Data Dep. Create RAW Hazards).

Soln 2

1. 2 Data Dep.
2. 0 RAW Hazards.

Data Dependency.

1. True Data Dep.
   (Adjacent Instr Dep which creates Stalls.

2. Data Dependency
   ↳ which may or may Not create Stalls.

Data Dependency

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

$r_0 \leftarrow r_1 + r_2$

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

Non- Adjacent dependency

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

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

$OP \quad r_i \quad r_j \quad r_k : \quad r_i \leftarrow r_j \odot r_k$ (operation)

$\boxed{r_0 \leftarrow r_1 + r_2}$

I_1 :	ADD	r_0	r_1	r_2
I_2 :	MUL	r_3	r_0	r_4
I_3	DIV	r_5	r_{14}	r_6
I_4	MUL	r_7	r_{15}	r_8
I_5	ADD	r_9	r_0	r_{10}
I_6	MUL	r_{11}	r_0	r_{12}

Total 3 Data Dep.

$I_2 - I_1$

$I_5 - I_1$

$I_6 - I_1$

True Data Dep.

① True Data Dep. $I_2 - I_1$

② DATA Dependency.

$I_5 - I_1$

$I_6 - I_1$

Data Dependency

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

i.e. $I_3 - I_1$ ✗

$I_4 - I_1$ ✗

Data

I_i

$\{$

I_j

Data Dependency & Hazards

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

Q) Find How Many Data Dependency ?

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

$I_5 - I_4$

But Not $I_5 - I_1$ X

Data Dep.

$I_i \rightarrow I_j$

Data Dep.

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

OR

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

Q How to check that Dependency Creates Hazards or Not ?

Sol^n Execution the Instruction in a Normal Pipeline manner if there is a Stalls | Bubble | Extra Cycle occur then Data Dep. Creates Stalls.

Q.4

I_1 : ADD r_0 r_1 r_2;		$r_0 \leftarrow r_1 + r_2$
I_2 : MUL r_3 r_0 r_4;		$r_3 \leftarrow r_0 * r_4$

Now Check this Dep (I_2 - I_1) Creates Hazards or Not ?

Stalling

Default

RISC : 5 Stage IF, ID, EX, MA, WB.

r_0 value.

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

3 STALL'S

⇓ Correct value

At CC3 I_2 Read wrong value of r_0

At CC4 I_2 Read wrong value of r_0

At CC5 I_2 Read wrong value of r_0

Bcz r_0 Available at end of CC5

So 3 Extra cycle / stalls / Bubbles.

Interesting Concept

CC5 ⇒ Clock cycle 5 the value of r_0 getting [Write Back into Register]

Split Phase

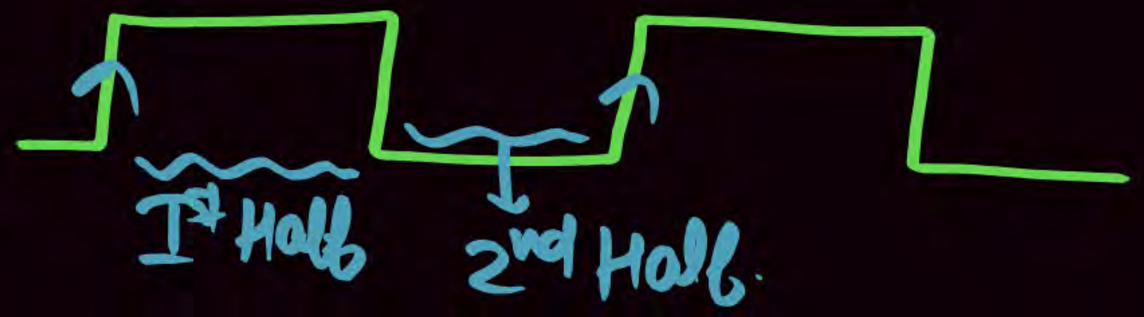
Clock Cycle

Part I — I^st Half

Part II — 2nd Half

Split Phase:

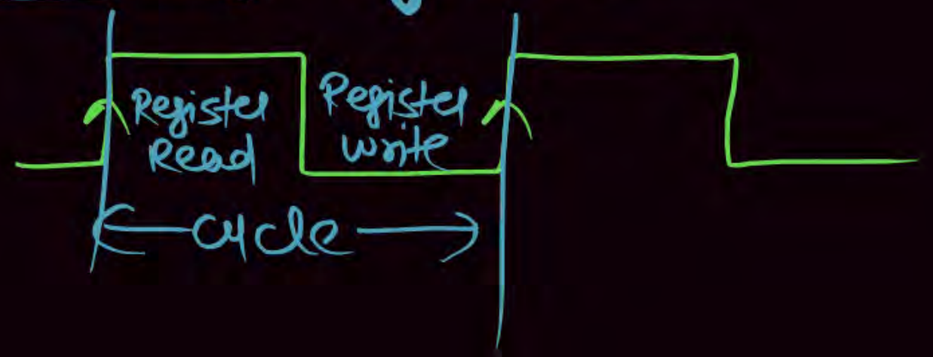
WB



① Case

1st Half : Reg. Read

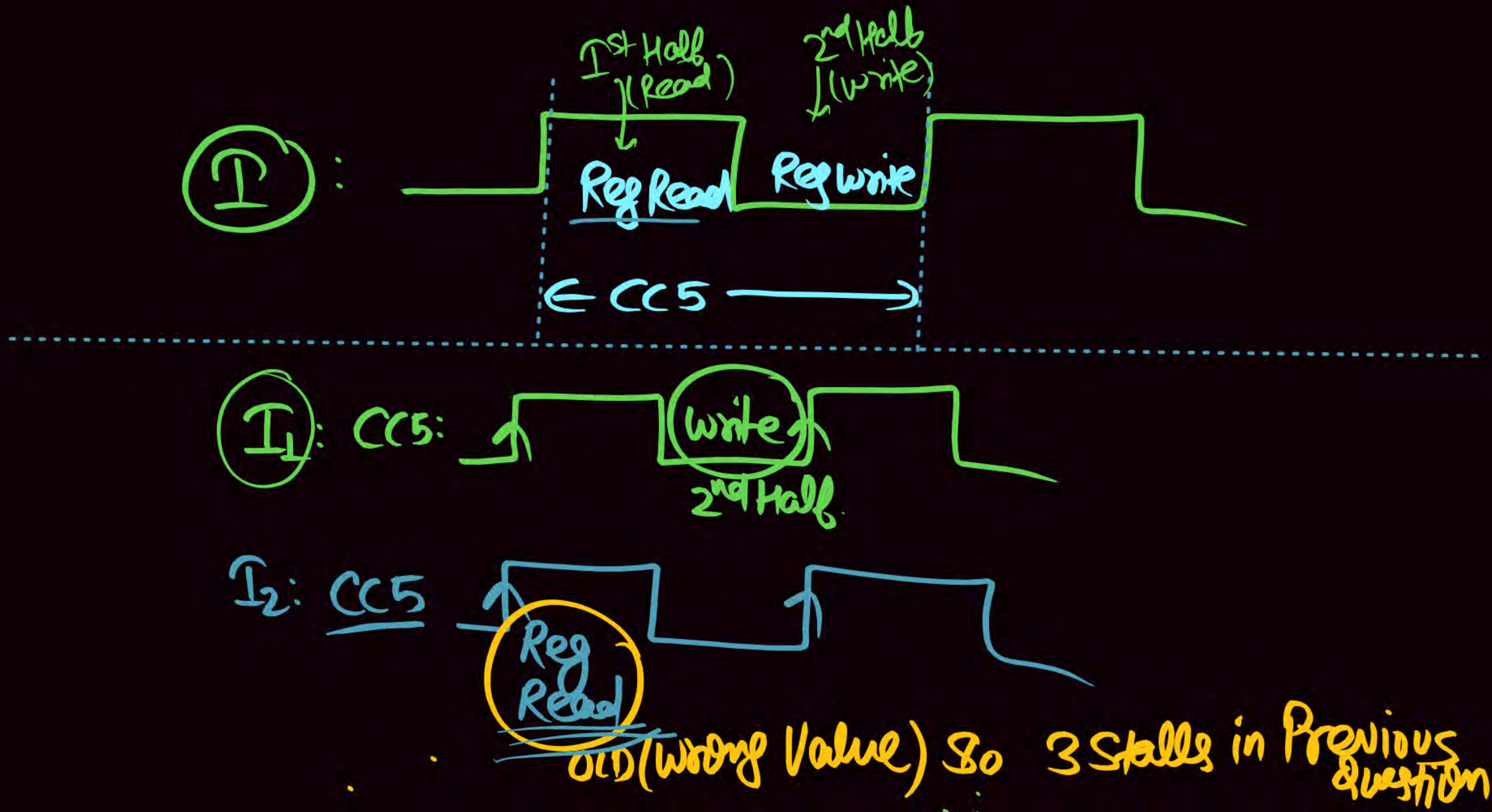
2nd Half : Reg write.



② CASE II

1st Half: Register write

2nd Half : Register Read.


I
1st Half (Read)
2nd Half (write)
Reg Read
Reg write
CC5
I1: CC5:
write
2nd Half
I2: CC5
Reg Read
OLD (wrong Value) So 3 Stalls in Previous Question.

Solmy (I)

Read | Write

RISC : 5 Stage IF, ID, EX, MA, WB.

r_0 value.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
I_1	IF	ID	EX	MA	WB.								
I_2		IF	ID	ID	ID	ID ✓ ⇓ Correct value	EX	MA	WB.				

3 STALL'S

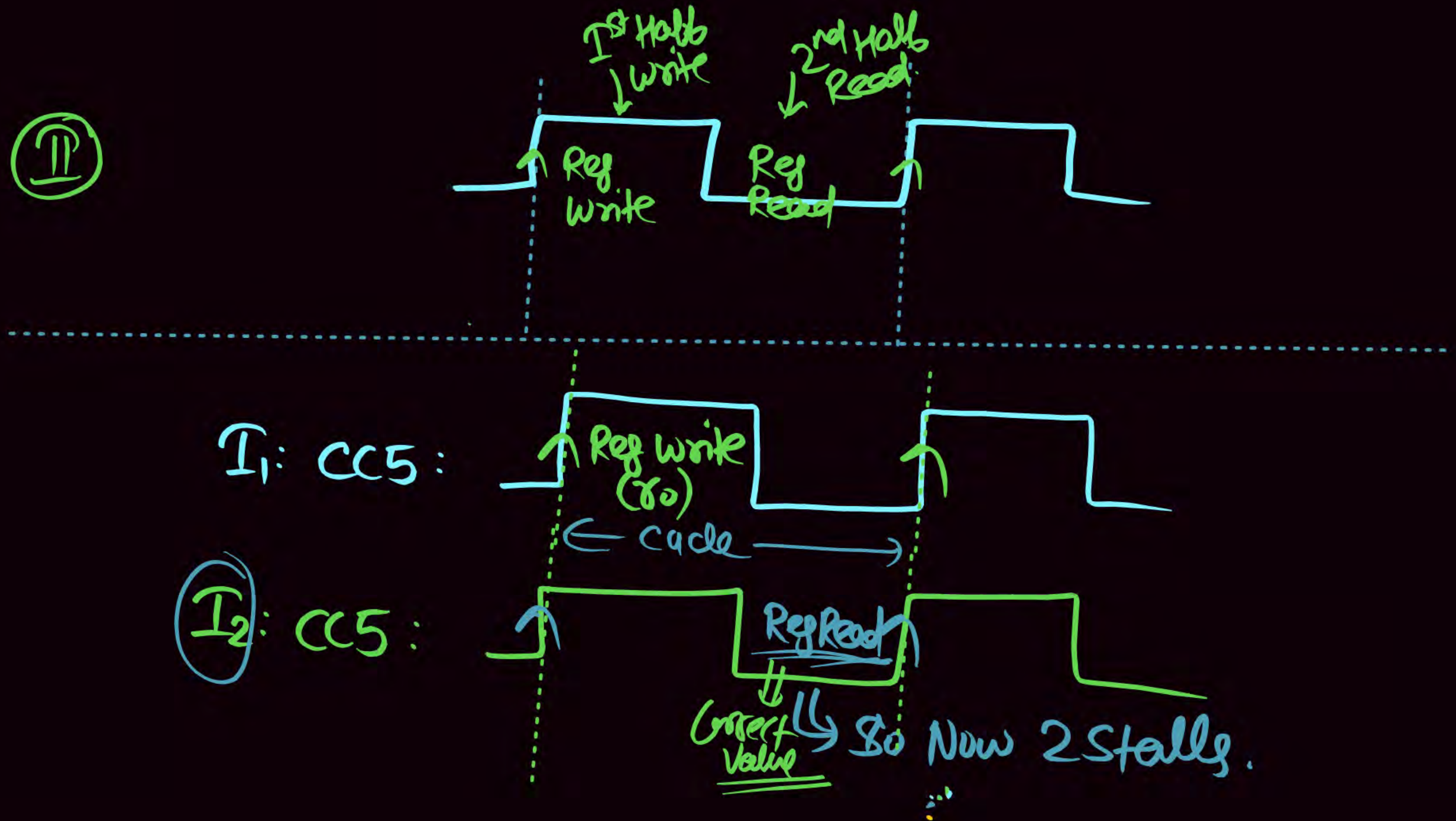
At CC3 I_2 Read wrong value of r_0

At CC4 I_2 Read wrong value of r_0

At CC5 I_2 Read wrong value of r_0

Bcz r_0 Available at end of CC5

So 3 Extra cycle / stalls / Bubbles.


II
1st Half write
2nd Half Read
Reg Write
Reg Read
I1: CC5:
Reg write (80)
← cycle →
I2: CC5:
Reg Read
Correct Value
So Now 2 Stalls.

Now in IInd Case

in which Reg Write in first Half
Reg Read in Second Half
of the Any Clock Cycle.

II Case

Reg write — Reg Read

RISC : 5 Stage IF, ID, EX, MA, WB.

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

r_0

2 STALL'S.

At CC3 I_2 Read OLD value of r_0

At CC4 I_2 Read OLD (wrong) Value of r_0

At CC5 (2nd Half) of cycle I_2 Read Correct Value of r_0

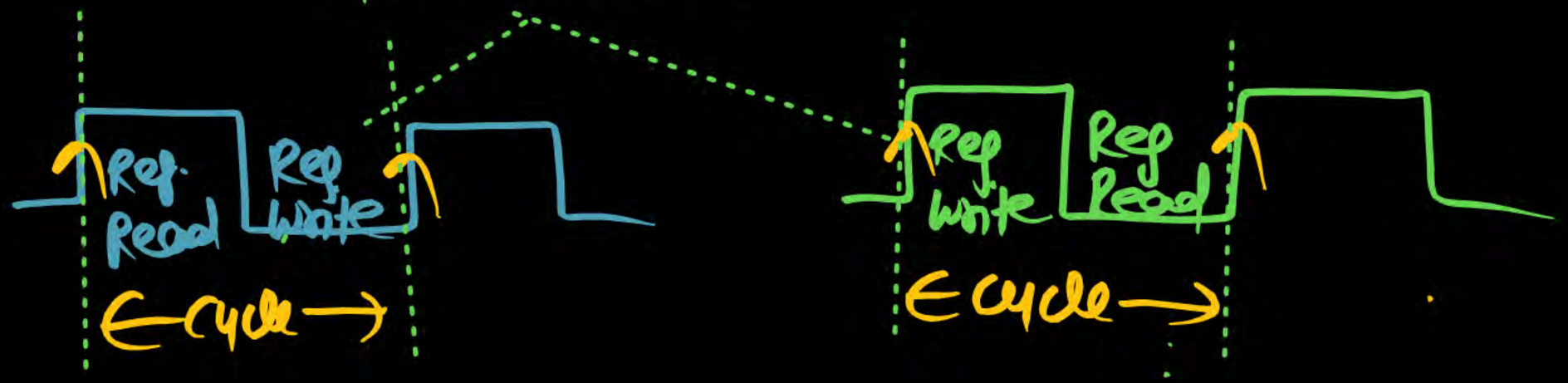
So 2 Stalls

RISC : 5 Stage IF, ID, EX, MA, WB.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13

Till Now we Learn

- Data Dep.
- Data Hazard (RAW Hazard)
- Diff b/w Data Dep & RAW Hazard.
- Split Phase



Data Dependency

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

$\boxed{\text{"Read} - \text{Before} - \text{write"}}$

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].

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].

D S1 S2

$I_1: r_0 \leftarrow r_1 + r_2$

$I_2: r_3 \leftarrow r_0 * r_2$

	CC1	CC2	CC3	CC4	CC5
I_1	IF	ID	EX	MA	WB
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

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 :

Like 3 Stalls | Bubble | Extra Cycle.

Stalling

Default

RISC : 5 Stage IF, ID, EX, MA, WB.

r_0 value.

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

3 STALL'S

⇓ Correct value

At CC3 I_2 Read wrong value of r_0

At CC4 I_2 Read wrong value of r_0

At CC5 I_2 Read wrong value of r_0

Bcz r_0 Available at end of CC5

So 3 Extra cycle / stalls / Bubbles.

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

3STALL'S

Read –After Write

7 cycles – 4 i/ps = 3 stalls

Solution of Data Dependency (RAW Hazard)

- ① Hardware Solution
- ② Software Solution [Compiler Based]
  - NOP Instruction inserted.

    'No operation'

Solnn

Default

RISC : 5 Stage IF, ID, EX, MA, WB.

No value.

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

3 STALL'S

⇓ Correct value

3STALL'S ⇒ After I_1

Insert 3 NOP Instruction.

Hardware Sol^n :

1. Hardware Interlock.
2. Operand Forwarding | By passing | Short circuit.

V.V.V.V.V.V.V Imp

Solution Of Data Dependency

① 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.

② Imp 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.

$I_1 : r_0 \leftarrow r_1 + r_2$

$I_2 : r_3 \leftarrow r_0 * r_4$

Without Operand Forwarding

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	
I_1	IF	ID	EX	MA	WB			
I_2:		IF	ID	ID	ID	ID	EX	WB

3 Stalls

$I_1 : r_0 \leftarrow r_1 + r_2$

$I_2 : r_3 \leftarrow r_0 * r_4$

With Operand Forwarding

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1	IF	ID	EX	MA	WB		
I_2		IF	ID	EX	MA	WB	

Bubble

If Here wrong (OLD) value of r_0 then in CC4

Correct value of r_0 using

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.1 Consider a pipelined processor with the following four stages

IF : Instruction Fetch

ID : Instruction Decode and Operand Fetch

EX : Execute

WB : Write Back

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

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

[GATE-2007 : 2 Marks]

(A) 7

(B) 8

(C) 10

(D) 14

Q. The performance of a pipelined processor suffer if

[GATE-2002 : 2 Marks]

(A) the pipeline stages have different delays

(B) consecutive instructions are dependent on each other

(C) the pipeline stages share hardware resources

(D) All of the above

MCQ Q.2

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 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. 3 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. 4

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!