COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_07

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

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.

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

Structural Dependency

Resource Conflict.

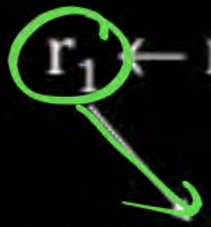
Solution → Resource Replication
(Use Additional H/w)

→ Divide the memory → I-Cache [CM]
→ D-Cache [DM].

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 ; $r_4 \leftarrow r_1 * r_5$



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

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

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

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

Data Dependency

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

i.e. $I_3 - I_1$

$I_4 - I_1$

Q. Consider a pipelined processor with the following four stages

IF : Instruction Fetch

ID : Instruction Decode and Operand Fetch

EX : Execute

WB : Write Back

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

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

[GATE-2007 : 2 Marks]

A 7

B 8

C 10

D 14

Q. The performance of a pipelined processor suffer if

[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]

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 219. Ans

[GATE-2018-CS: 2M]

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

Type of Data Dependency.

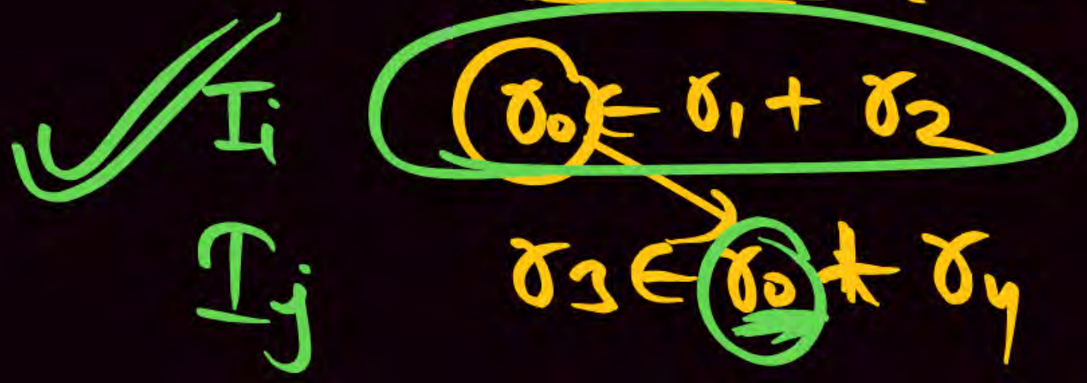
① RAW (Read After Write) / True Data Dependency
   (Ij, Ii)

② Name [False Dep.]
   WAR (Write After Read) / ANTI Dependency

③ Name [False Dep.]
   WAW (Write After Write) / output / write Dependency.

Note: These words are actual Execution Sequence. if Not follow in that order then Problem will be Created.

RAW (Read 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.

Types of Data Hazard

① RAW (Read After Write) | True Data Dependency.

I_1 I_i $R_0 \leftarrow R_1 + R_2$

I_2 I_j $R_3 \leftarrow R_0 + R_4$

When the Next Instruction (I_2, I_j or I_{i+1}) try to Use the Result of Previous Instruction (I_1, I_i or I_i) as a operand. (if Not follow the sequence I_j then I_i) then Problem Will be Created.

↳ Soln
- Hardware Interlock
- Operand Forwarding | Bypassing | short circuit
- Insert Nop Oper^n.

Types of Data Hazard

② WAR (Write After Read) | ANTI Dependency

I_i I_i I_1 $R_0 \in R_1 * R_2$

@ @

I_{i+1} I_j I_2 $R_1 \in R_3 + R_4$

When Next Instruction (I_2) tries to Write the Data Before Previous Inst^n (I_i) Read it

Now R_1 value Read by I_1 is Incorrect value.

WHY? Assume Suppose I_1 takes More & More time as Compare to I_2.

then I_2 execute first & Update (R_1 value)

Initial $R_1 = 500$

Any How But if I_2 execute (write) it Update the value of Register R_1.

$R_1 = 10$ By I_2.

Types of Data Hazard

② WAR (Write After Read) | ANTI Dependency.

I_i I_i or I_{i+1} I_j I_1 or I_2

$R_0 \in R_1 * R_2$

$R_1 \in R_3 + R_4$

Here I_2 tries to Update the Value of Register R_1, before Read this Value (R_1) By I_1. So I_1 Read Incorrect Value.

WHY? Assume Suppose I_1 takes More & More time as Compare to I_2.

then I_2 execute first & Update (R_1 Value)

Initial $R_1 = 500$

Anyhow But if I_2 execute (write) it Update the Value of Register R_1.

$R_1 = 10$ By I_2.

Types of Data Hazard

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

I_i (or) I_{i+1} I_1 : $R_2 \leftarrow R_1 * R_0$

I_j I_2 : $R_2 \leftarrow R_3 + R_4$

Assume I_1 taking more & more time compare to I_2 so I_2 execute Firstly so it update the Register value R_2.

When the next Instruction (I_2 | I_j | I_{i+1}) tries to write the value, before the Previous Instn (I_1 | I_i) write it.

Types of Data Hazard

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

I_i or I_{i+1} : I_1 : $R_2 \leftarrow R_1 * R_0$

I_j : I_2 : $R_2 \leftarrow R_3 + R_4$

Assume I_1 taking more & more time compare to I_2 so I_2 execute Firstly so it update the Register value R_2.

Here I_2 is executed before the I_1 so I_2 update the value of Register R_2 before I_1 Updation.

So Incorrect (wrong Updation).

I_1: $R_1 = 50,000$

I_2: $R_2 = 25$ lakh

But I_2 then I_1 then 50,000

25 lakh

Types of Data Hazard

Note

ANTI & OUTPUT Dependency does not Create Any Stalls But in ANTI & OUTPUT we are getting the Problem of Not Reading | Updating the Correct Value.

The Solution of ANTI & OUTPUT Dependency is 'Register Renaming'

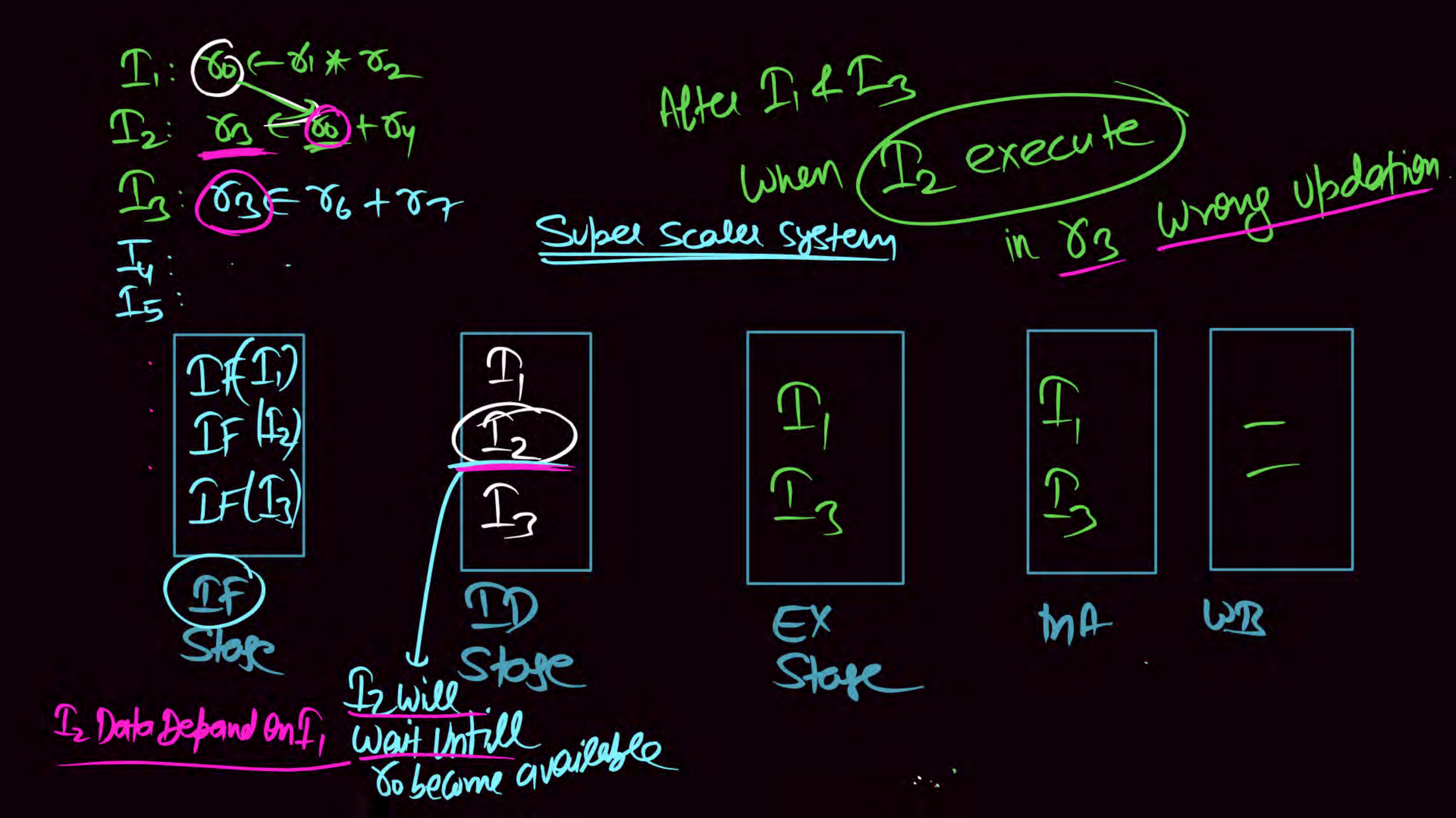
Register Renaming: Use Some Temporary Storage Register to Store the Result of the Out of Order Instruction output.

Q WHEN & HOW ANTI Dep. & OUTPUT Dep. Create Problem ?

Soln (i) If the Previous Instruction taking More & More time (Very High time) Compare to Next Instruction. [UnEven Delay (Multiple pipe)]

(ii) When there is True Data Dependency between 2 Instruction then to Minimize the Stalls. We trying to Reschedule the Instruction [out of Order Instruction execution]

(iii) In Super Scaler System, Multiple Instruction Inserted in the Stages So Next Instn if Read/write the value that was used in Previous Instr then create a Problem...


I1 : r0 ← r1 * r2
I2 : r3 ← r0 + r4
I3 : r3 ← r6 + r7
I4 :
I5 :
After I1 & I3
When I2 execute
in r3 Wrong Updation
Super scaler system
IF(I1)
IF(I2)
IF(I3)
IF Stage
I1
I2
I3
ID Stage
I1
I3
EX Stage
I1
I3
MA
WB
I2 Data Depand on I1
I2 will Wait Untill r0 become available

I_1
I_2
I_3
I_4

] Sequential order [Inorder execution].

Types of Data Hazard

- CPU always execute a sequence the program from top to bottom in a sequence called as **"in-order"** execution.
- In this execution sequence if any instruction is dependent then the remaining instruction are also sharing the stall cycles even, they are independent.

I_2 Depand on I_1 Data

Ex-: Code

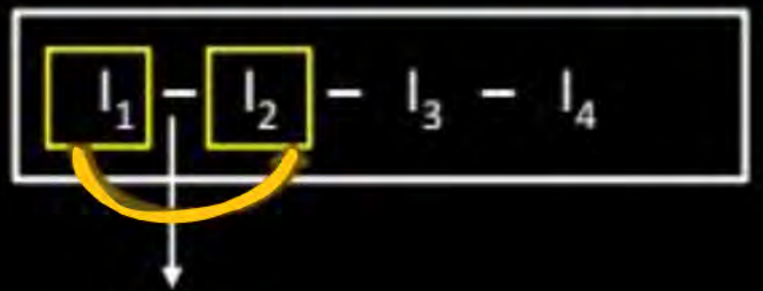
I_1: Add r_0, r_1, r_2

I_2: sub r_3, r_0, r_4

I_3: MUL r_4, r_5, r_6

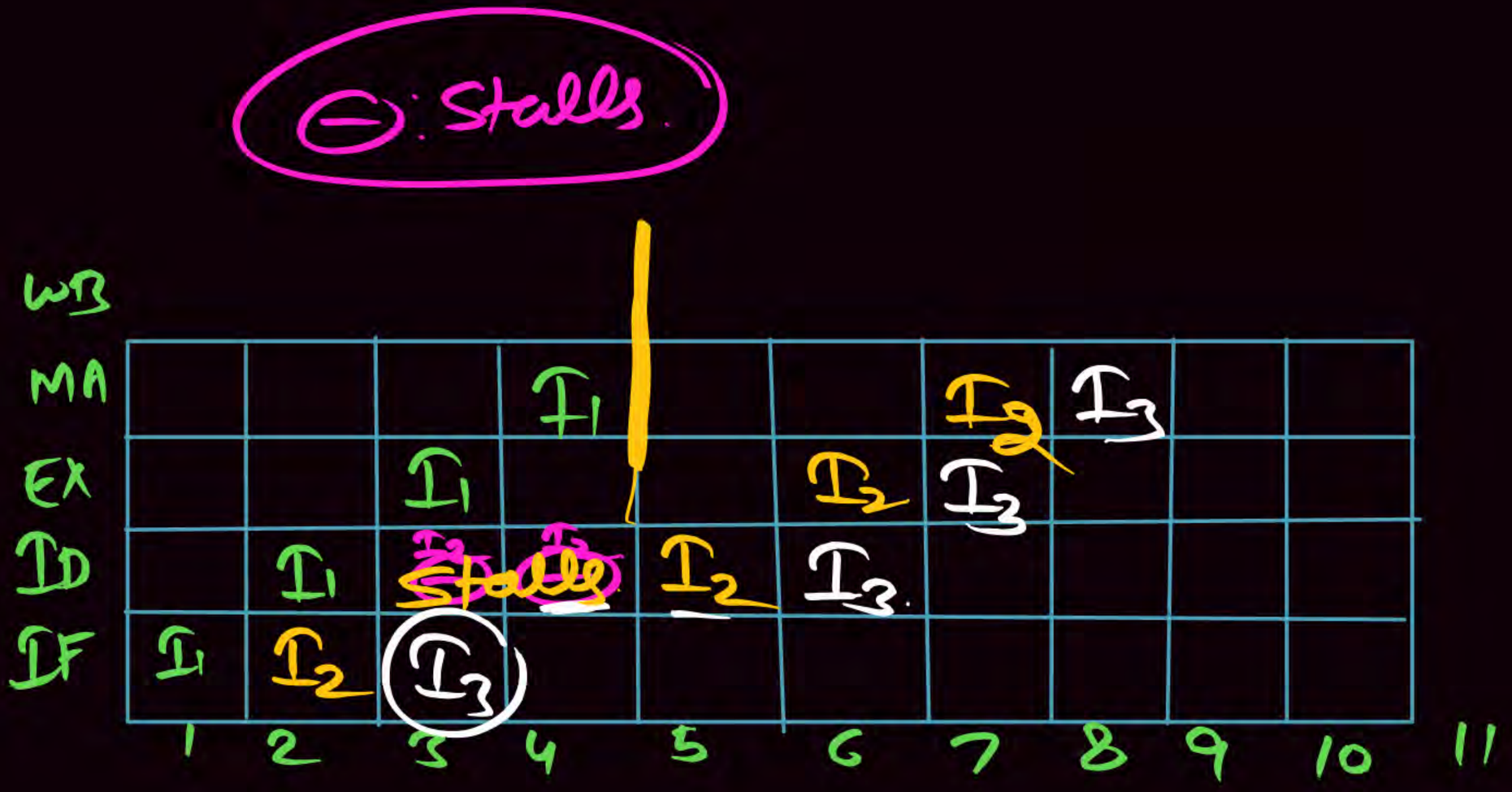
I_4: DIV r_3, r_7, r_8

In- order execution sequence



Reschedule (Out of order execution)

I_1
I_3
I_4
I_2


Stalls
WB
MA
EX
ID
IF
I1
I2
I3
1
2
3
4
5
6
7
8
9
10
11

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]

$I_1 - I_3 - I_4 - I_2$

I_1 : Add r_0 r_1 r_2 ; $r_0 \in r_1 + r_2$

I_3 : MUL r_4 r_5 r_6 ; $r_4 \in r_5 * r_6$

I_4 : DIV r_3 r_7 r_8 ; $r_3 \in r_7 / r_8$

I_2 : SUB r_3 r_0 r_4 ; $r_3 \in r_0 - r_4$

① ANTI Dep $(I_3 - I_2)$

② OUTPUT Dep. $(I_4 - I_2$

Types of Data Hazard

Anti Dependency will be occurred when the instruction 'J' try to write the data before instruction 'I' reads it.

Example - I3 executed before I_2 so I_3 modified the register (r_4) before I_2 reads It, therefore I_2 incorrectly read the new value from (r_4) [Data loss].

Output dependency will be occurred when the instruction "J" tired to write the data before instruction 'I' writes.

Ex- I_4 executes before I_2 so, I_4 modifies the register (r_3) before I_2 writes it therefore destination is incorrectly updated with old value [data loss].

Types of Data Hazard

Soln

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 (Temp. Register)

$I_2 \rightarrow r_3$ (Status 1)

Re-order buffer → Reg. file $\{R_4, R_3\}$] So Data Safe.

After completion. Content Update

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

$R_i \in R_j \circledcirc R_k$.

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

$S_1 \times ANTI(I_2 - I_5)$

$S_2 \checkmark ANTI(I_2 - I_4)$

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

Control Dependency.

Control Dependency is created in the Pipeline Due to

- Branch Instruction
- Program Control
- function Call
- Transfer of Control Instn (TOC Instn)
  - Unconditional TOC
  - Conditional TOC.

Control Hazard

- Also known as a branch hazard
- Occurs when the pipeline makes the wrong decision on a branch prediction.
- Brings instructions into the pipeline that must subsequently be discarded
- Dealing with Branches:
  - Multiple Streams
  - Prefetch branch target
  - Loop buffer
  - Branch prediction
  - Delayed branch

Control Dependency :

Assume 5 Stage (S1, S2, S3, S4, S5) pipeline & I_2 is the Branch Instruction. & Assume the target Address of the Branch Instruction is available at the end of 'LAST Stage' (5 Stage)

I_2 is a Branch Instn

1000	I_1	(Unconditional Tor) Branch Instn
1001	I_2	JMP 2000
1002	I_3	
1003	I_4	
1004	I_5	
1005	I_6	
1006	I_7	
2000	BI_1	
2001	BI_2	

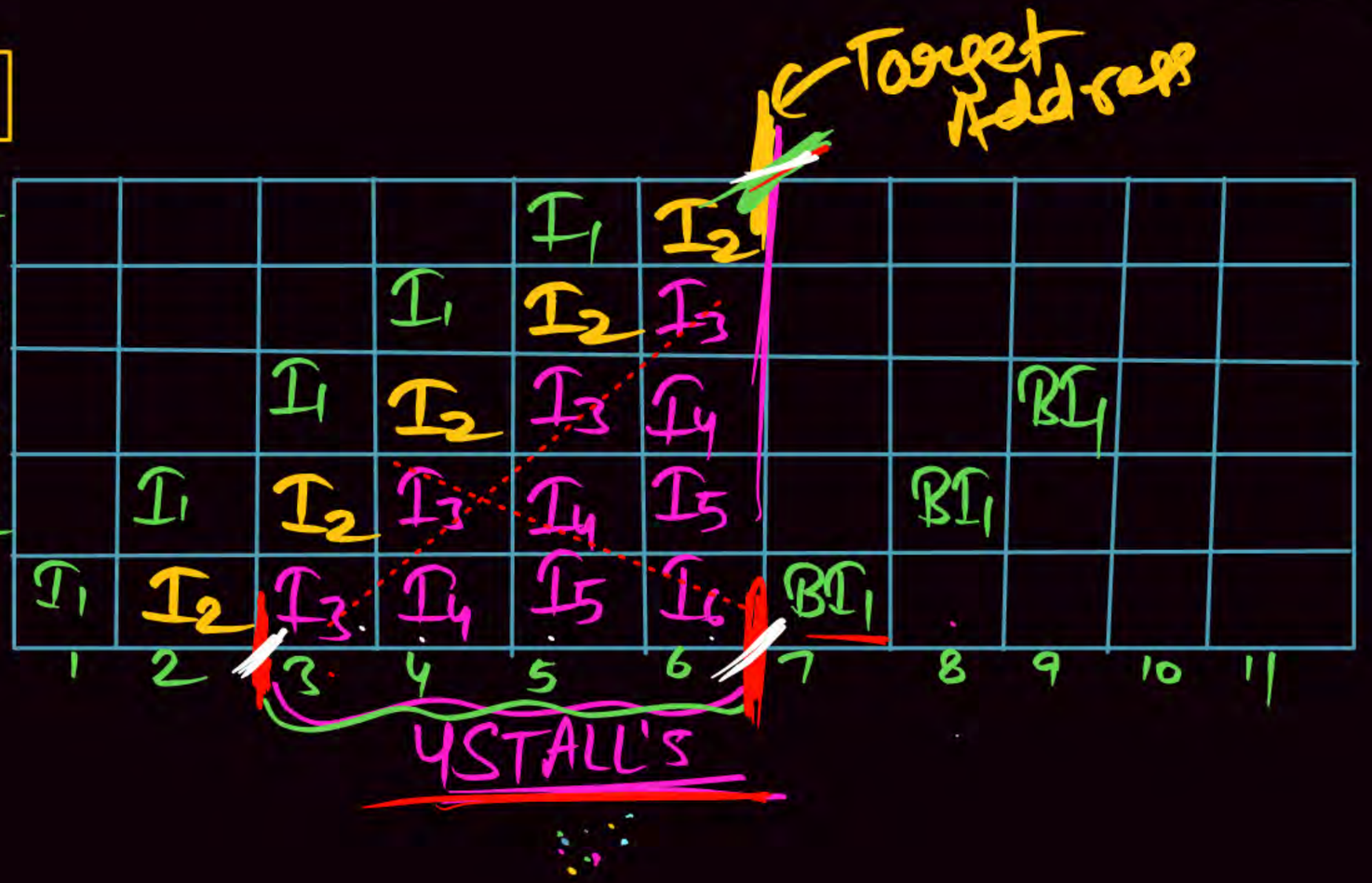
I_3 I_4 I_5 & I_6 is unwanted Instn.

BI_1 is the wanted Instn

Control Dependency :

Assume 5 Stage (S_1, S_2, S_3, S_4, S_5) Pipeline & I_2 is the Branch Instruction. & Assume the target Address of the Branch Instruction is available at the end of LAST Stage (5th Stage)

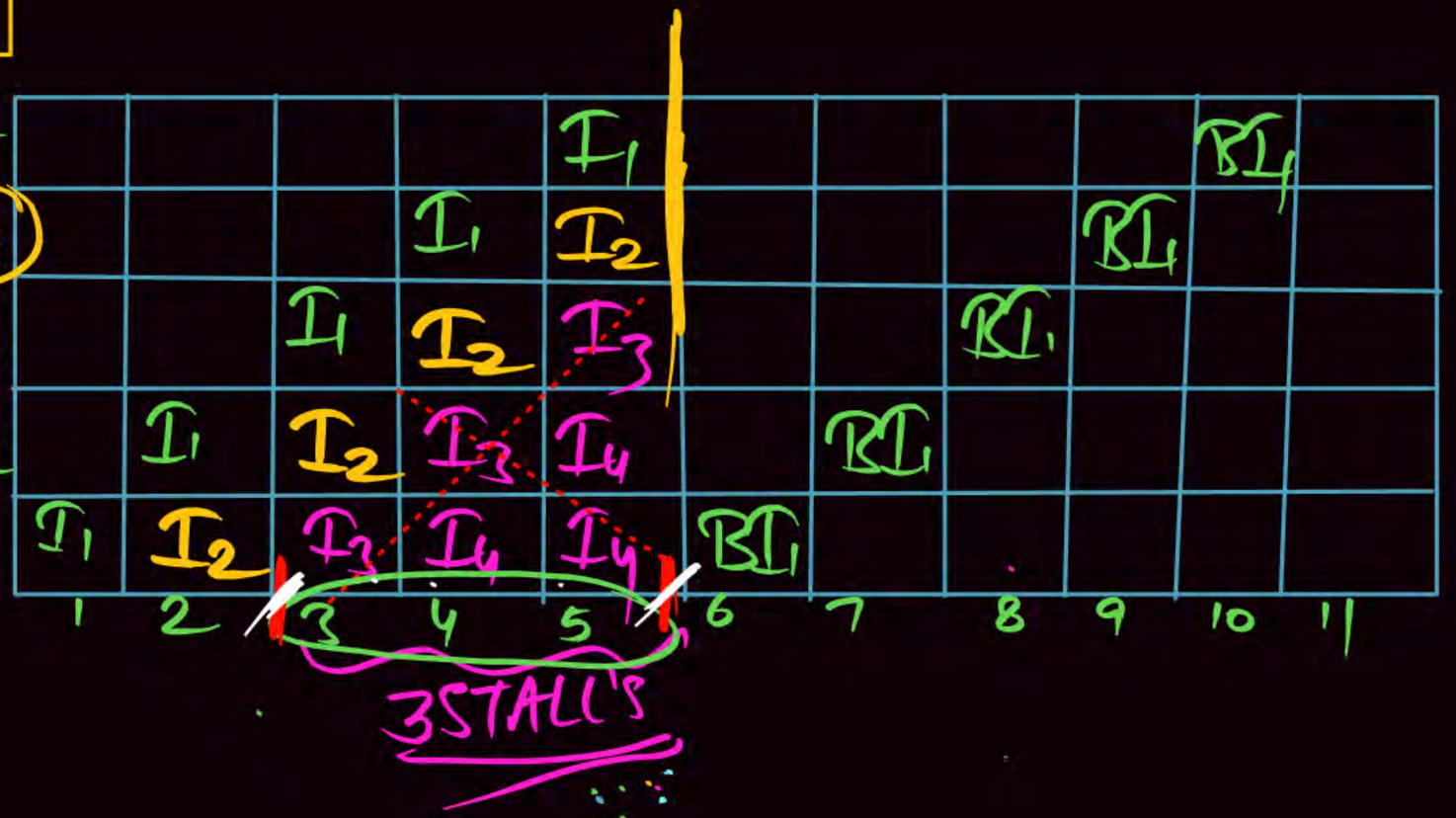
1000	I_1	
1001	I_2	JMP 2000
1002	I_3	
1003	I_4	
1004	I_5	
1005	I_6	
1006	I_7	
2000	BI_1	
2001	BI_2	



Control Dependency :

Assume 5 Stage (S1, S2, S3, S4, S5) pipeline & I_2 is the Branch Instruction. & Assume the target Address of the Branch Instruction is available at the end of 4th Stage.

1000	I_1	
1001	I_2	JMP 2000
1002	I_3	
1003	I_4	
1004	I_5	
1005	I_6	
1006	I_7	
2000	BI_1	
2001	BI_2	



Control Dependency :

Assume 5 Stage (CS1 S2 S3 S4 S5) Pipeline & I_2 is the Branch Instruction. & Assume the target Address of the Branch Instruction is available at the end of 3rd Stage :

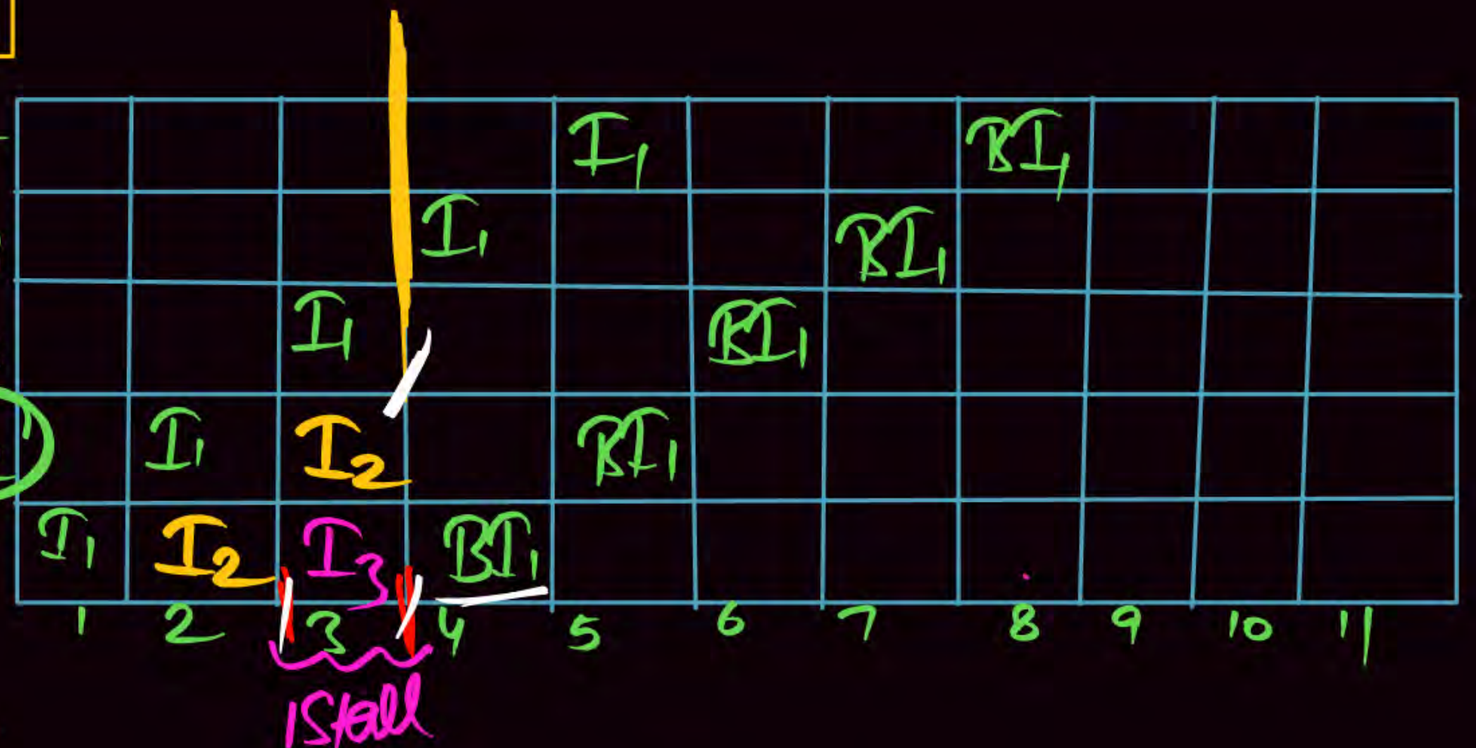
1000	I_1	
1001	I_2	JMP 2000
1002	I_3	
1003	I_4	
1004	I_5	
1005	I_6	
1006	I_7	
2000	BI_1	
2001	BI_2	

(WB) S_5
(MA) S_4
(EX) S_3
(ID) S_2
(IF) S_1

Control Dependency :

Assume 5 Stage (CS1 S2 S3 S4 S5) pipeline & I_2 is the Branch Instruction. & Assume the target Address of the Branch Instruction is available at the end of 2nd Stage.

1000	I_1	
1001	I_2	JMP 2000
1002	I_3	
1003	I_4	
1004	I_5	
1005	I_6	
1006	I_7	
2000	BI_1	
2001	BI_2	



Control Dependency :

Assume 5 Stage (CS, SF, SR, SE, SS) pipeline & I_2 is the Branch Instruction. & Assume the target Address of the Branch Instruction is available at the end of 'k stage'.

1000	I_1	
1001	I_2	JMP 2000
1002	I_3	
1003	I_4	
1004	I_5	
1005	I_6	
1006	I_7	
2000	BI_1	
2001	BI_2	

If the Target Address of Branch Instruction is available at 'k^{th}' Stages

#Stalls from Branch Instr = k − 1

Control Dependency/Control Hazards

Generally
Avg ET

Ideal $\boxed{CPI = 1}$

But Due to Control Hazard / Control Dep.

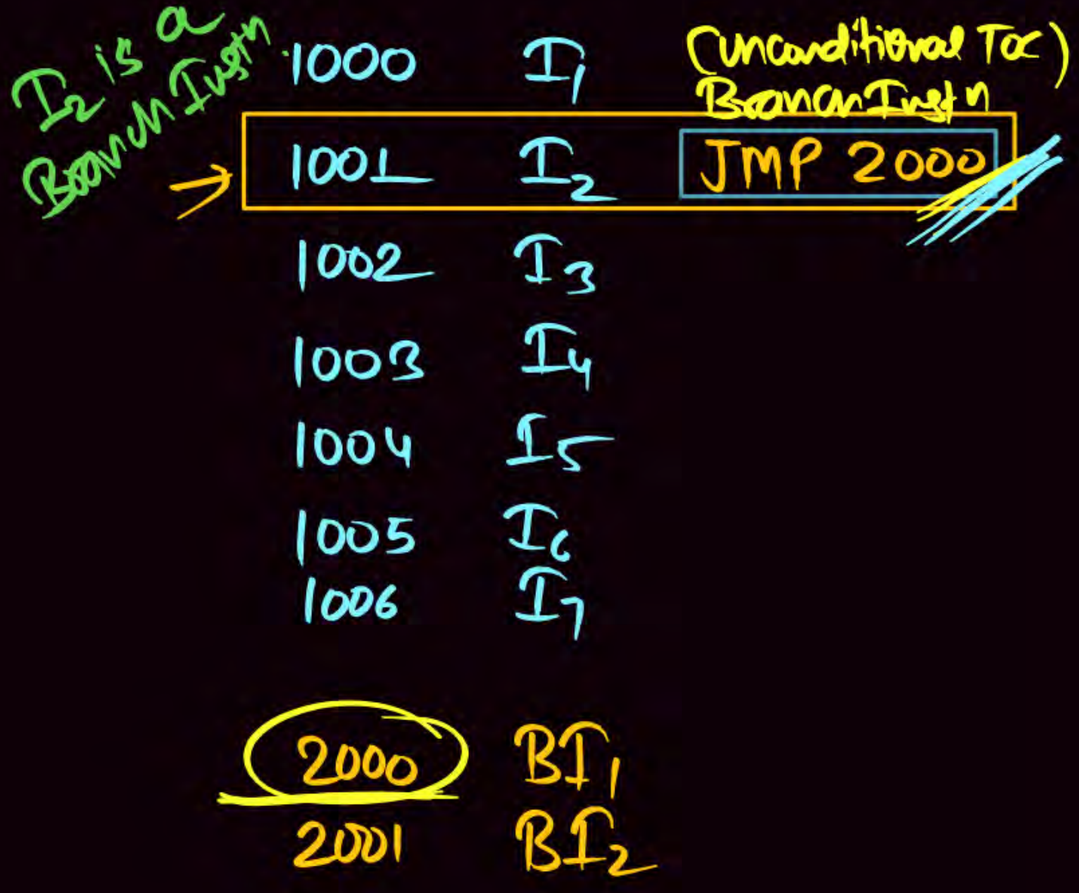
$$\text{Avg Instn ET} = 1 + \#\text{Instn which Create Stalls} \times \#\text{Stalls from Branch}$$

$$\text{Avg Instn ET} = 1 + \left(\text{Branch Instn Frequency} \times \#\text{Stalls Due to Due Instn}\right)$$

Control Dependency :

- Control Dependency is created in the pipeline Due to execution of Branch (TOC Instn) So Program execution will be changed
- In this the Next Sequential Instruction become Wanted (or) unwanted, its Depends on the output Previous Instruction decoding.
- If the Previous Instruction decode the Instruction as Simple Instn (Airthmatic, Data transfer etc) then the Next Instruction become Wanted Instruction
- If the Previous Instn decode the Instruction as Branch Instn. Unconditional TOC, Conditional TOC with True then Next Seq. Instn become Unwanted.

Control Dependency :



ID Decode as Simple (ALU, Data transfer Instn) then I2 is wanted.

When I1 execute then I2 is Wanted Instruction.

But when I2 execute (Decode) then I3, I4 ... is a Unwanted Instruction

BCZ I2 Decode as Uncondition Toc (JMP 2000)

So BI1 is the Now Wanted Instruction.

Control Dependency :

- In the pipeline overlapping Execution Happens, that means before Decoding the Previous Instⁿ, Next Sequential Instⁿ Inserted into Pipeline
- If the Next Instruction is Unwanted, then we have to Remove | Flush the Unwanted Instⁿ from the Pipe.

The Process of Removing the Unwanted Instruction is Called Flush / Freeze.

Control Dependency :

- The Number of stalls created Due to Branch operation is called **Branch Penalty**.
- This Branch Penalty Value Depends on the Out Come of the target Address.

(Note) In the RISC Pipeline Branch Penalty Value is '1' Because Target Address is available at the end of 2nd Stage.

$$\text{Branch Penalty} = K-1 \Rightarrow 2-1 = 1$$

Control Dependency :

Note

Generally Target address is available in decoding stage/phase but Condition checking [Evaluation] done in execution Phase/Stage.

Note

If Target is available at 'k' stage.

$$\text{Branch Penalty} = K - 1$$

$$\text{Branch Penalty} = \left(\begin{array}{l}\text{At what Stage} \\ \text{Target Address Available}\end{array}\right) - 1$$

Control Dependency :

Branch Instn Freq. = # Branch Instn.

Avg $CPI = 1$

But Due to Control Hazards

$$CPI = 1 + \left(\text{Branch Instn freq.} \times \#\text{Stalls Due to Branch} \right)$$

Control Dependency :

Q. Suppose we have 100 Instruction. Out of 100 Instruction 30 Instn are Branch type, which Create 3 STALL Cycle then Avg CPI ?

Total # Stalls	=	Number of Branch Instn	X	# Stalls Due to Branch Instn

100 Insty

$\Rightarrow 30 \times 3$

$ET = 100 \times 1 = 100$

 # Stalls = 90.

Total for 100 Instn $= 100 + 90$

$= 190$

Avg CPI $= \frac{190}{100} = 1.9$ Ans

Control Dependency :

Q. Suppose we have 100 Instruction. Out of 100 Instruction 30 Instn are Branch type, which Create 3 STALL Cycle then Avg CPI ?

$$\text{Avg Instn ET} = 1 + \left(\text{Branch Instn Frequency} \times \text{\# Stalls Due to Branch Instn}\right)$$

$$\text{Avg Instn ET} = 1 + (.30 \times 3)$$

$$\Rightarrow 1 + 0.9$$

$$= 1.9 \text{ Avg}$$

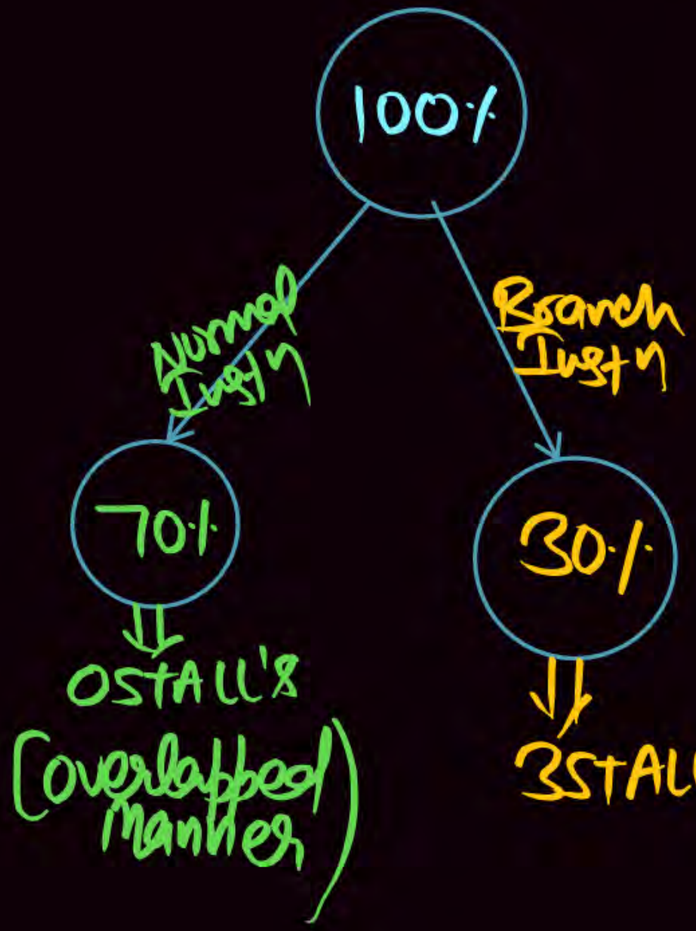
100 ⇒ 30 Branch Instn

30% Branch

3 STALL's

Control Dependency :

Q) Suppose we have 100 Instruction. Out of 100 Instruction 30 Instn are Branch type, which Create 3 STALL Cycle then Avg CPI ?



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

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

$$\text{Avg Instn ET} = [1 + \#\text{Stalls/Instn}]$$

$$\Rightarrow [1 + 0.9]$$

$$\boxed{\text{Avg Instn ET} = 1.9}$$ Ans

THANK YOU!