Pipeline

Complete Revision

Part 4

- Deepak Poonia

(IISc Bangalore; *GATE AIR 53 & 67*)

Content of this Lecture:

1. Data Dependency & Hazards
2. Control Dependencies
3. GATE PYQs

GATE 2024 Complete Course Link:

https://www.goclasses.in/s/pages/gatecompletecourse

Instructor:

Deepak Poonia

IISc Bangalore

GATE CSE AIR 53; AIR 67; AIR 206; AIR 256

GO CLASSES

GATE 2024

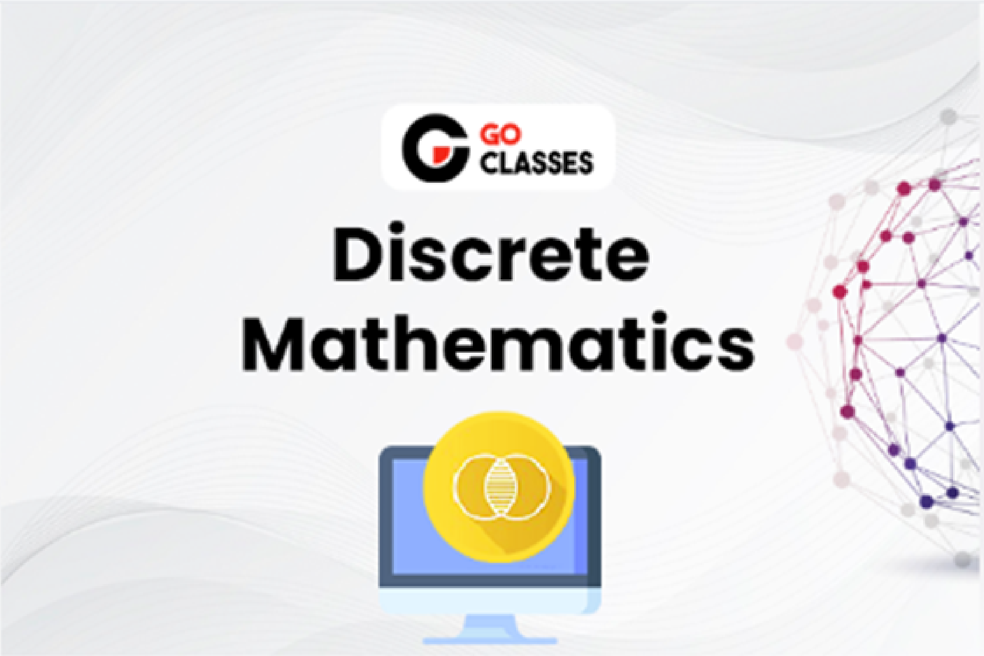
COMPLETE COURSE CS-IT

(1 YEAR)

GO CLASSES

GATE 2025
COMPLETE COURSE CS-IT

(2 YEARS)



2023 Discrete Mathematics

★ ★ ★ ★ ★ 5.0 (62 ratings)

Deepak Poonia (MTech IISc Bangal...

Free

2023 C Programming

★ ★ ★ ★ ★ 5.0 (59 ratings)

Sachin Mittal (MTech IISc Bangalor...

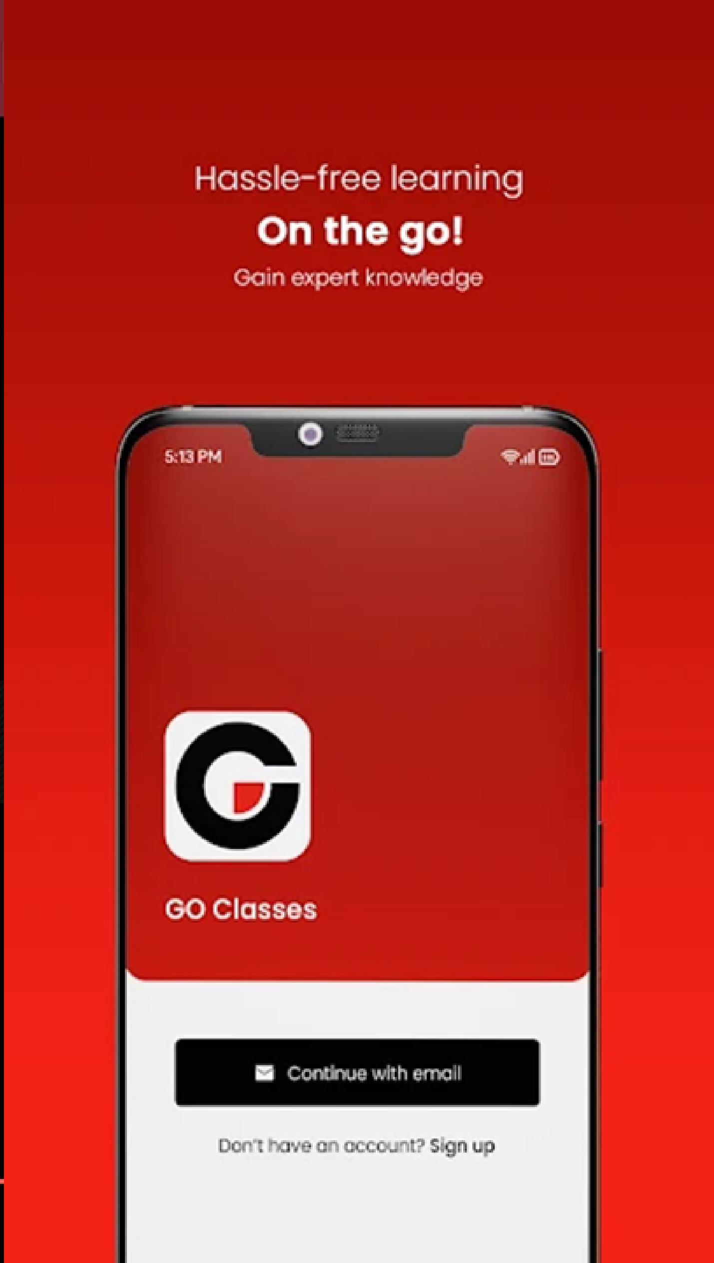
Free

On "GATE-Overflow" Website

Download the GO Classes Android App:

https://play.google.com/store/apps/details?id=com.goclasses.courses

Search "GO Classes" on Play Store.



RAR is Never a Dependency, Hence Never a Hazard.

#RAR Dep = 0

add r_1, r, r_2

sub r_3, r, r_4

any Dependency? No

Can Reorder

RAR is NOT a Dep.

Data Dep : Only Code

Data Hazard : $\boxed{\text{Code + Pipeline}}$

Data Dep :

Code
- RAW Dep / True Dep / Flow Dep
- WAW Dep / output Dep
- WAR Dep / Anti Dep

output Dep, Anti Dep → Name Dep / false Dep.

In-order pipeline:

Prog:

I_1
I_2
I_3
⋮

In-order PL:

fetch/execute in the program order.

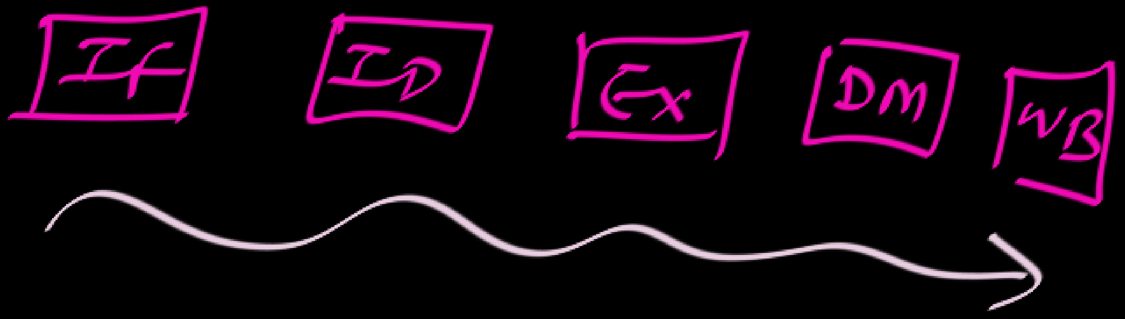
& Each instruction goes through All stages.

5-stage PL:

add r_1, r_2, r_3

Does not need Data mem. Access

BUT Still it goes Through it.



Data Dep: Only Code

Data Hazard: $\boxed{\text{Code + Pipeline}}$

Data Dep:

In In-order PL:

Code { RAW Dep → RAW Hazard may Exist.

WAW Dep, WAR Dep → Can Never be Hazard.

At the End: (IDEA)

→ Out-of-order PL

WAW Hazard

WAR Hazard

RAW Hazard

100 %

Eliminate-able

Possible

At the End.

Next:

Data Hazards Solutions

Eg: $I_1: R \leftarrow R_1 + R_2$

$I_2: R_3 \leftarrow R_4 - R_5$

$I_3: R_7 \leftarrow R + R$

RAW Dep

$I_1 - I_3$

RAW Dep = 1

Pipeline

4 stages

IF ID Ex WB

$I_1 - I_3$ RAW Dep is a Hazard ??

How to check if a <u>Data Dep</u> is a Hazard?

Just do Normal (Perfect) Execution & see if any Problems.

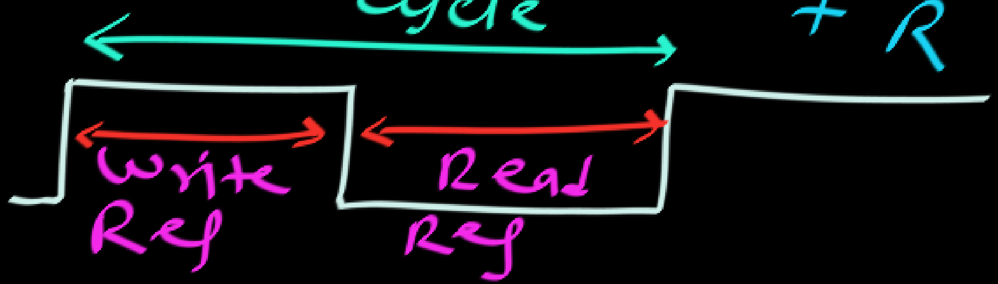
	c_1	2	3	4	5	6	7	8
I_1	IF	ID	EX	WB				
I_2		IF	ID	EX	WB			
I_3			IF	ID	EX	WB		

R correct value

Reading correct value of R

Case 1: Split phase

→ No RAW Hazard



Read WRONG Value of R

Hazard

	C 1	2	3	4	5	6	7	8
I_1	IF	ID	EX	WB				
I_2		IF	ID	EX	WB			
I_3			IF	ID	EX	WB		

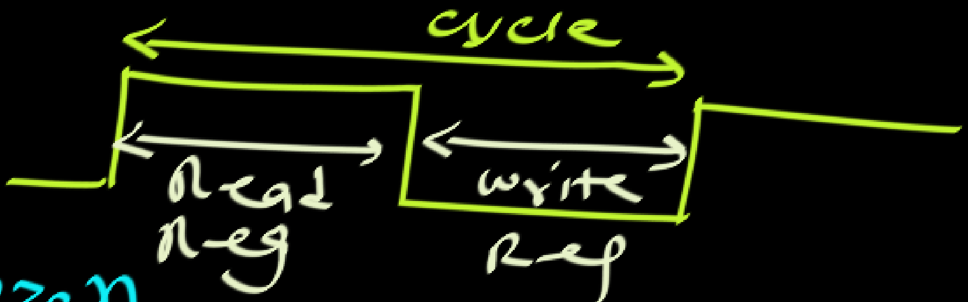
R Correct Value

Time Travel

Never Possible.

Case 2 :

RAW Hazard



There is a need to handle Hazards, So that we get Correct Execution.

Handling Data Hazards:

1. Don't do any wrong in the first place.

Don't Read wrong value at all.

Stall way ≡ Delay way

Handling Data Hazards:

2 Ractify your mistake.

Correct / fix the wrong values that you have read.

forwarding solution

Handling Data Hazards:

1. Stall/Delay/Bubble Solution

2. Forwarding Solution

Data

HANDLING OF HAZARDS

- DETECT HAZARD SITUATIONS

[redacted]

1. • STALL DEPENDENT INSTRUCTIONS
2. • FIX VALUES READ BY DEPENDENT INSTRUCTIONS

① Stall Solution:

NOT making mistake Solution

① Hardware Sol : Intuitive

Decoding Phase

OR

② Software Sol : Inserting NOP instruction

Compiler

I_1 : $r \leftarrow$ - - - -

I_2 : - - - - (independent)

I_3 : $\leftarrow r$ - - -

Classical 5-Stage PL

Assume Split Phase used

1 RAW Hazard ($I_1 - I_3$)

Using H/W Stall solution, Handle this Hazard.

I_1 : $r \leftarrow$ - - - -

I_2 : - - - - (independent)

I_3 : $\leftarrow r$ - - -

Classical 5-stage PL

Assume Split phase used

RAW Hazard :

	C1	2	3	4	5	6	7	8
I_1 :	IF	ID	Ex	MA	WB			
I_2 :								
I_3 :				IF	ID	Ex	MA	WB

RAW Hazard (Time Travel)

Stall solution : (by H/W)

Just Stall (Delay) but

read Correct Value only.

	c1	c2	3	4	5	6	7	8	9
I_1	If	ID	Ex	MA	WB				
I_2		If	ID	Ex	MA	WB			
I_3			If	ID	ID	Ex	MA	WB	

Correct r value

Stall solution

Read wrong r → So wait

Bubble (Stall) → Ex IDLE

I_1: r ← - - - -

I_2: - - - - (independent)

I_3: ← r - - -

1 RAW Hazard

Stall Solution

2 Stalls

classical 5-stage PL

Assume Reg write happens in the 2nd phase of the cycle.

	c1	2	3	4	5	6	7	8	9	...
I_1	IF	ID	Ex	MA	WB					
I_2		IF	ID	Ex	MA	WB				
I_3			IF	ID	ID	ID	Ex	MA	WB	

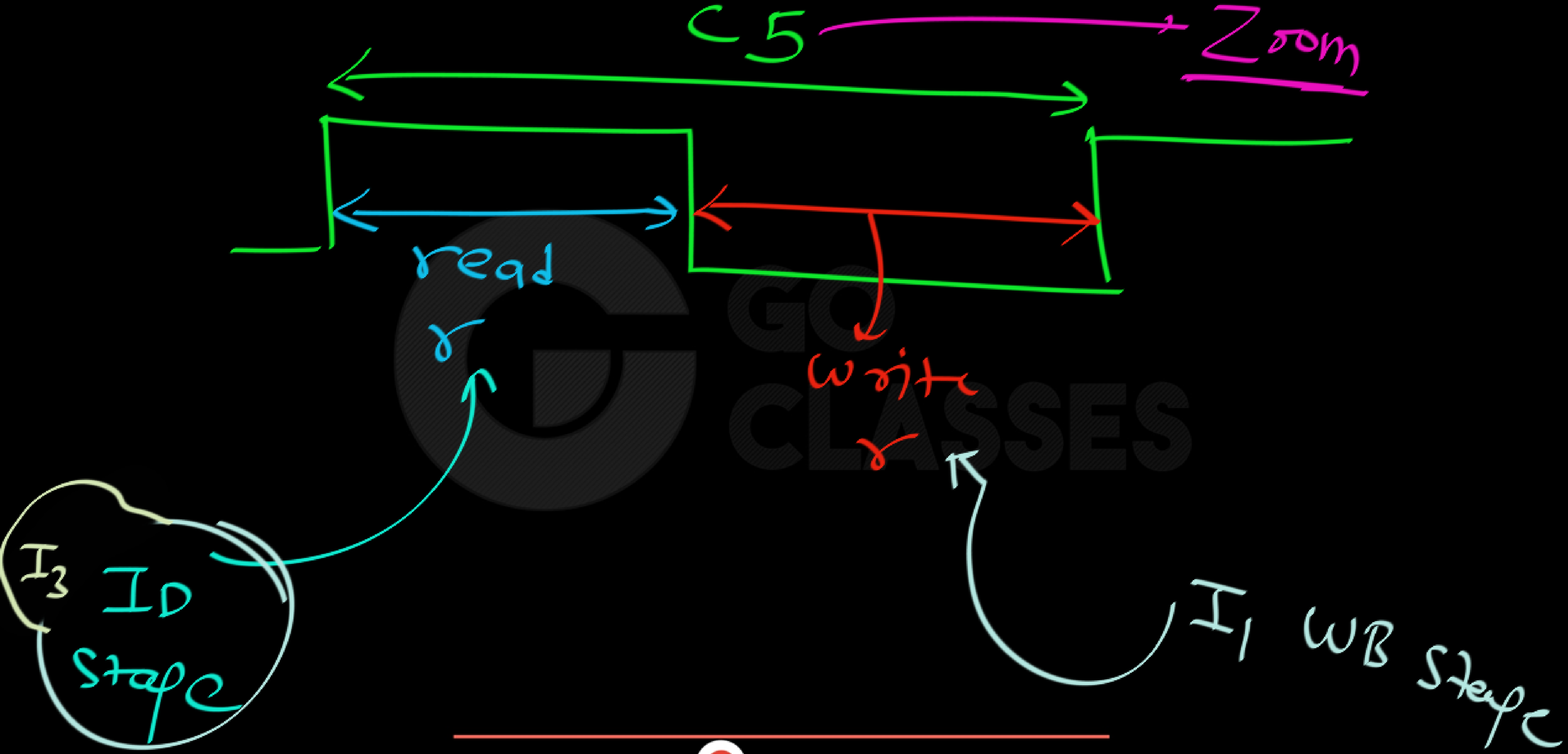
Correct r value

Reading wrong value of r

so wait

stall

stall


C5
Zoom
read
r
write
r
I3 ID stage
I1 WB stage

Stall Solution

Hardware Version

Intuitive solution

Just wait for Correct Value

Always Read Correct Value

Stall Solution

Software Solution:

Compiler

I_1: $r \leftarrow$ - - -

I_2:

I_3: $\leftarrow r$ - - -

Code

Compiler

I_1

I_2

Nop

Nop

I_3

why?

because of 2 stalls

Do Nothing

② Forwarding Solution: ✓ available

$I_1 : r \leftarrow r_1 + r_2$

$I_2 : -- \leftarrow r --$

RAW Dep

RAW Hazard in MIPS Pipeline

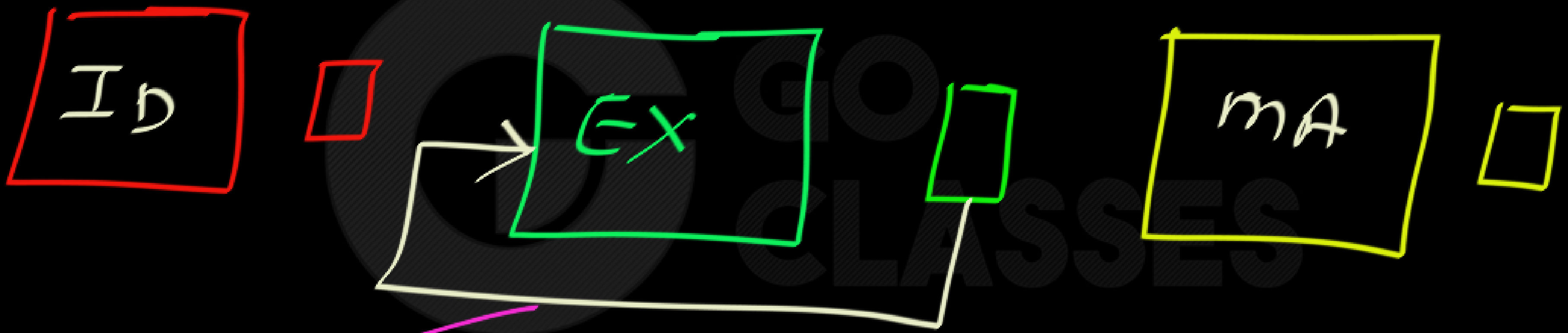
	c1	2	3	4	5	6	7
I_1	IF	ID	EX	MA	WB		
I_2		IF	ID	EX	MA	WB	

ID → EX → ✓ → MA

I_2 read wrong r

fixing the wrong value

forwarding :



forwarding :
from Ex to Ex stage

forwarding
Data path

forwarding :

If the value is available Somewhere already, then bypass/forward it the place where it is needed.

Data Hazard Solution:

1. Stall Sol
2. forwarding Sol

→ Sometimes both are used

FLUSHES, STALLS, AND FORWARDING QUIZ

FETCH	READ	ALU/BR	MEM	WR

[redacted]	[redacted]	STALL	FORWARD
[redacted]			
MUL R1, R2, R3 LW R1, 0(R1)			
ADD R1, R1, R1			

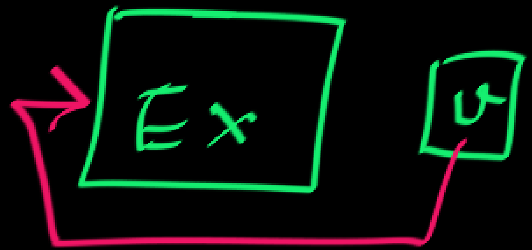
mul $R_1 \leftarrow R_2 + R_3$ → v

lw $R_1 \leftarrow 0(R_1)$

RAW Dep.

mul:

lw:



v available in Ex Buffer

c3 c4

Ex → Ex . . .

ID

lw Read wrong value of R_1

fixing the R_1

lw $r_1 \leftarrow M(2+r_1) \rightarrow v$

add $r_1 \leftarrow r_1 + r_1$

RAW Hazard

	c_1	2	3	4	5	6	7 ...
lw	IF	ID	Ex	MA	WB		
add		IF	ID	ID	Ex		

1 stall

v available

fixing r_1

Stall

ADD read wrong r_1

Ex → MA → v ✓

→ forwarding datapath

FLUSHES, STALLS, AND FORWARDING QUIZ

FETCH	READ	ALU/BR	MEM	WR

[redacted]	[redacted]	STALL	FORWARD
[redacted]			
MUL R1, R2, R3 LW R1, 0(R1)		✗	✓
ADD R1, R1, R1		✓	✓

Data Hazards Solution 1: Hardware Solution

Consider the pipeline in Figure 8.2. The data dependency just described arises when the destination of one instruction is used as a source in the next instruction. For example, the two instructions

Mul R2,R3,R4
Add R5,R4,R6

give rise to a data dependency. The result of the multiply instruction is placed into register R4, which in turn is one of the two source operands of the Add instruction. Assuming that the multiply operation takes one clock cycle to complete, execution would proceed as shown in Figure 8.6. As the Decode unit decodes the Add instruction in cycle 3, it realizes that R4 is used as a source operand. Hence, the D step of that instruction cannot be completed until the W step of the multiply instruction has been completed. Completion of step D_2 must be delayed to clock cycle 5, and is shown as step D_{2A} in the figure. Instruction I_3 is fetched in cycle 3, but its decoding must be delayed because step D_3 cannot precede D_2. Hence, pipelined execution is stalled for two cycles.

Data Hazards Solution 2: Software(Compiler) Solution

8.2.2 Handling Data Hazards in Software

In Figure 8.6, we assumed the data dependency is discovered by the hardware while the instruction is being decoded. The control hardware delays reading register R4 until cycle 5, thus introducing a 2-cycle stall unless operand forwarding is used. An alternative approach is to leave the task of detecting data dependencies and dealing with them to the software. In this case, the compiler can introduce the two-cycle delay needed between instructions I_1 and I_2 by inserting NOP (No-operation) instructions, as follows:

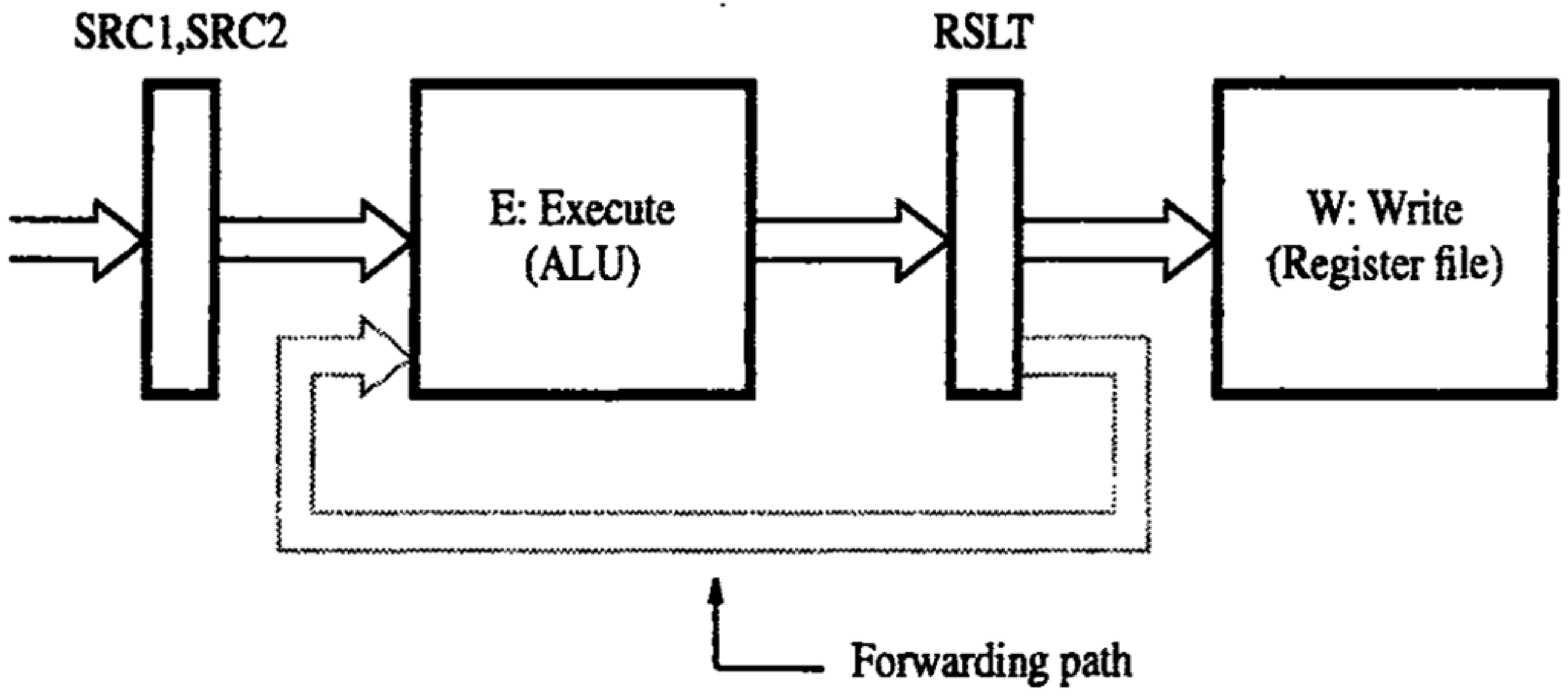
$$
\begin{array}{lll}
I_1: & \text{Mul} & \text{R2,R3,R4} \\
 & \text{NOP} & \\
 & \text{NOP} & \\
I_2: & \text{Add} & \text{R5,R4,R6}
\end{array}
$$

If the responsibility for detecting such dependencies is left entirely to the software, the compiler must insert the NOP instructions to obtain a correct result. This possibility illustrates the close link between the compiler and the hardware. A particular feature can be either implemented in hardware or left to the compiler. Leaving tasks such as inserting NOP instructions to the compiler leads to simpler hardware. Being aware of the need for a delay, the compiler can attempt to reorder instructions to perform useful tasks in the NOP slots, and thus achieve better performance. On the other hand, the

Data Hazards Solution 3:

Operand Forwarding

Hardware Solution



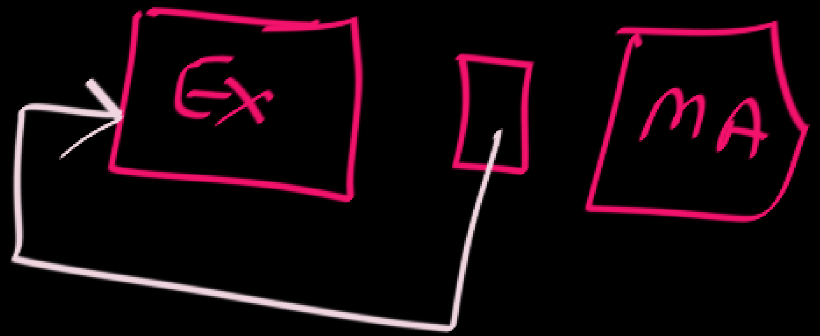
(b) Position of the source and result registers in the processor pipeline

Figure 8.7 Operand forwarding in a pipelined processor.

forwarding : $Ex \longrightarrow Ex$ forwarding

I_1 : Ex

I_2 : Ex



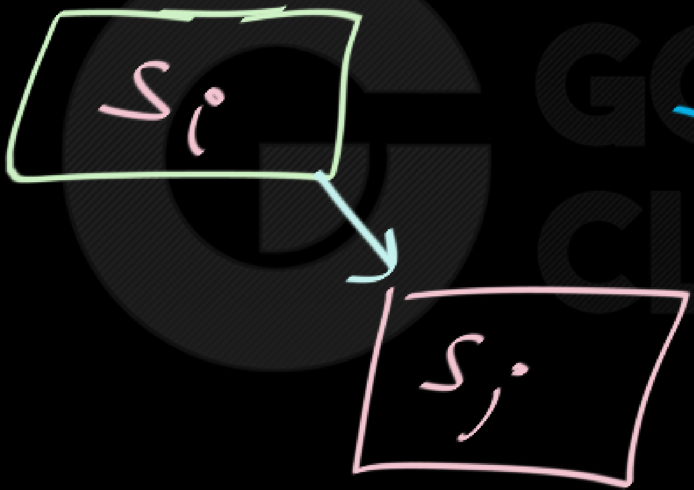
forwarding : MA $\longrightarrow$ Ex forwarding

I_1 : MA

I_2 : Ex

full forwarding : (by Default)

If value available somewhere already then forward/bypass it to the place where it is needed.



1.18.10 Pipelining: GATE CSE 2007 | Question: 37, ISRO2009-37 top

https://gateoverflow.in/1235

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 needs 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	$R2 \leftarrow R1+R0$
MUL	R4, R3, R2	$R4 \leftarrow R3*R2$
SUB	R6, R5, R4	$R6 \leftarrow R5-R4$

A. 7
B. 8
C. 10
D. 14

1.18.10 Pipelining: GATE CSE 2007 | Question: 37, ISRO2009-37 top

https://gateoverflow.in/1235

Consider a pipelined processor with the following four stages:

- IF: Instruction Fetch
- ID: Instruction Decode and Operand Fetch
- EX: Execute
- WB: Write Back

full forwarding

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 needs 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	$R2 \leftarrow R1+R0$
MUL	R4, R3, R2	$R4 \leftarrow R3*R2$
SUB	R6, R5, R4	$R6 \leftarrow R5-R4$

A. 7
B. 8
C. 10
D. 14

C1 C2 C3 C4 - - - - - -

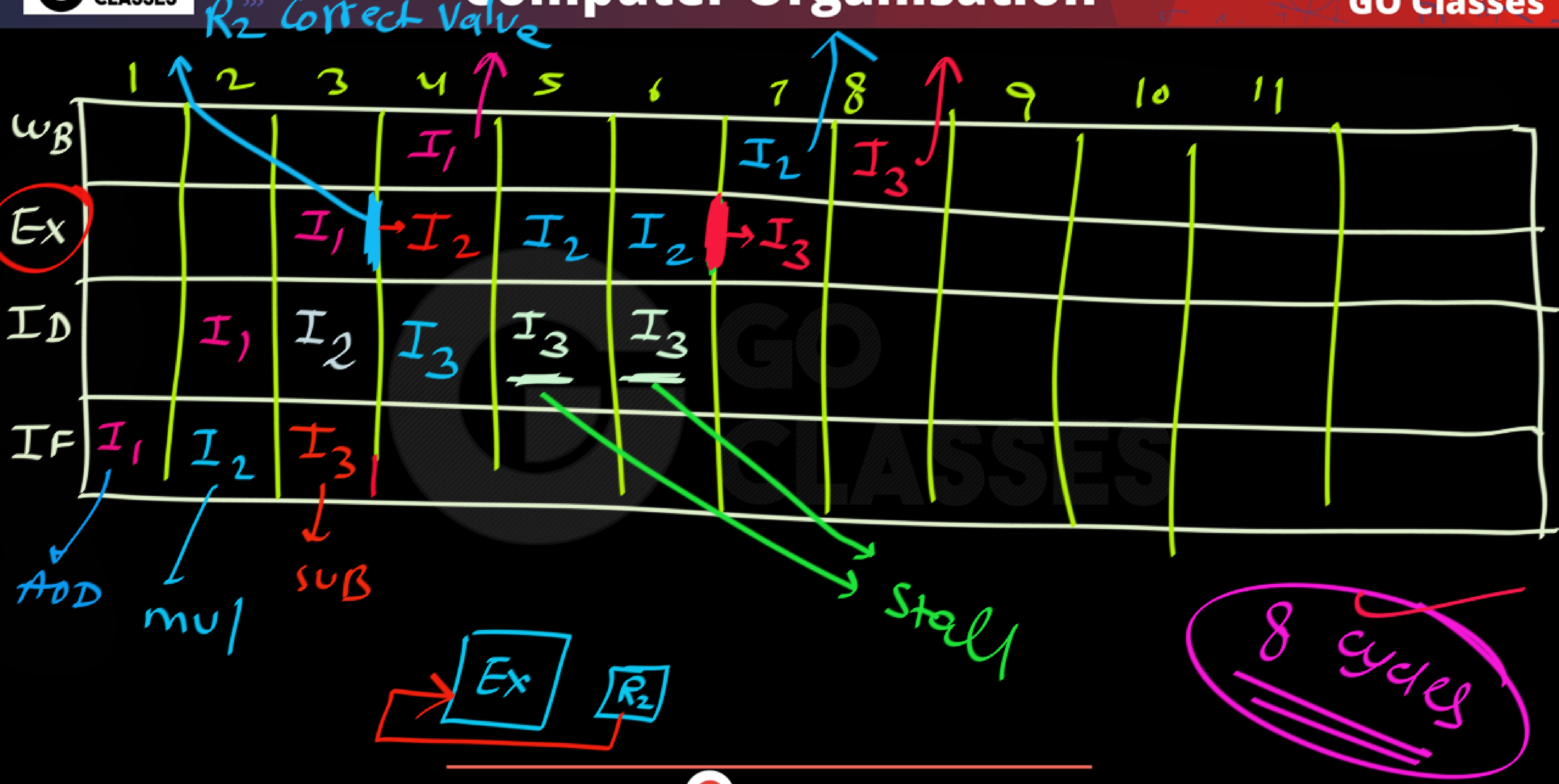
I_1 IF ID - - -

I_2 - - - -

- - -

⋮

fine BUT I prefer the Next one


R2 Correct Value
1 2 3 4 5 6 7 8 9 10 11
WB
Ex
ID
IF
I1
I2
I3
ADD
mul
SUB
Stall
Ex
R2
8 cycles

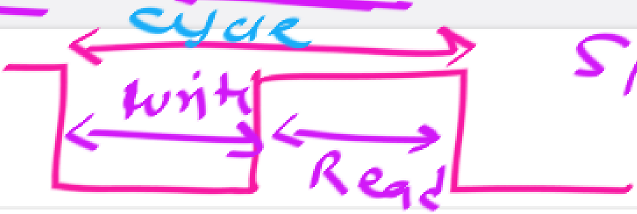
GATE Overflow Test Series | Mock GATE | Test 6 | Question: 38

Consider a 5-stage pipeline - IF (Instruction Fetch), ID (Instruction Decode and register read), EX (Execute), MEM (Memory), and WB (Write Back). All register reads take place in the second phase of a clock cycle and all register writes occur in the first phase and there is no operand forwarding in use. Consider the execution of the following instruction sequence:

- $I_1 : R_1 \leftarrow R_2 + R_3$
- $I_2 : R_3 \leftarrow R_1 - R_2$
- $I_3 : M[R_3] \leftarrow R_1$
- $I_4 : R_2 \leftarrow R_3 * R_1$

If the number of RAW (Read after write) hazards is denoted by A, WAR (Write after read) hazards by B and WAW (Write after write) hazards by C, then $2A + 3B + C =$ ________

GATE Overflow Test Series | Mock GATE | Test 6 | Question: 38



split phase

Consider a 5-stage pipeline - IF (Instruction Fetch), ID (Instruction Decode and register read), EX (Execute), MEM (Memory), and WB (Write Back). All register reads take place in the second phase of a clock cycle and all register writes occur in the first phase and there is no operand forwarding in use. Consider the execution of the following instruction sequence:

- $I_1 : R_1 \leftarrow R_2 + R_3$
- $I_2 : R_3 \leftarrow R_1 - R_2$
- $I_3 : M[R_3] \leftarrow R_1$
- $I_4 : R_2 \leftarrow R_3 * R_1$

RAW Dep : 3+1+1 = 5 ✓

WAW Dep : 0

WAR Dep : 1+1+1 = 3

[redacted]

Read : R_1, R_3

Not writing into R_3

GATE Overflow Test Series | Mock GATE | Test 6 | Question: 38

cycle

write Read

split phase

asked in CO and Architecture Jan 30, 2022 • edited Jan 30, 2022 by Lakshman Patel RJIT

160 views

2

Consider a 5-stage pipeline - IF (Instruction Fetch), ID (Instruction Decode and register read), EX (Execute), MEM (Memory), and WB (Write Back). All register reads take place in the second phase of a clock cycle and all register writes occur in the first phase and there is no operand forwarding in use. Consider the execution of the following instruction sequence:

- $I_1 : R_1 \leftarrow R_2 + R_3$
- $I_2 : R_3 \leftarrow R_1 - R_2$
- $I_3 : M[R_3] \leftarrow R_1$
- $I_4 : R_2 \leftarrow R_3 * R_1$

4 0

If the number of RAW (Read after write) hazards is denoted by A, WAR (Write after read) hazards by B and WAW (Write after write) hazards by C, then $2A + 3B + C =$ ________ 8

0

	C1	2	3	4	5	6	7	8	9	10
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		

r_1

r_3

Perfect

RAW Hazards: $I_1 - I_2$ $I_1 - I_3$ $I_2 - I_3$ $I_2 - I_4$

GATE Questions

1.18.33 Pipelining: GATE IT 2006 | Question: 78 top

https://gateoverflow.in/3622

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 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,

A. $2, 2, 4$
B. $3, 2, 3$
C. $4, 2, 2$
D. $3, 3, 2$

1.18.33 Pipelining: GATE IT 2006 | Question: 78 top

https://gateoverflow.in/3622

[redacted] are available in the registers $R0, R1, R2, R3$ and $R4$ respectively, before the execution of the instruction sequence.

WAR Dep:

$I_3 - I_4$

$I_2 - I_3$

I_1	ADD	R5, R0, R1	; R5 ← R0 + R1
I_2	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:

$I_1 - I_2$

$I_1 - I_4$ ✗

$I_2 - I_3$

$I_2 - I_5$ ✗

$I_3 - I_4$

$I_4 - I_5$

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,

✗ A. 2, 2, 4

✗ B. 3, 2, 3

C. 4, 2, 2 ✓

D. 3, 3, 2

WAW Dep: ADD-SUB, MUL-DIV

2 ✓

HW

1.18.35 Pipelining: GATE IT 2007 | Question: 6, ISRO2011-25 top

https://gateoverflow.in/3437

A processor takes 12 cycles to complete an instruction I. The corresponding pipelined processor uses 6 stages with the execution times of $3, 2, 5, 4, 6$ and 2 cycles respectively. What is the asymptotic speedup assuming that a very large number of instructions are to be executed?

A. 1.83
B. 2
C. 3
D. 6

HW

1.18.36 Pipelining: GATE IT 2008 | Question: 40 top

https://gateoverflow.in/3350

A non pipelined single cycle processor operating at 100 MHz is converted into a synchronous pipelined processor with five stages requiring 2.5 $nsec$, 1.5 $nsec$, 2 $nsec$, 1.5 $nsec$ and 2.5 $nsec$, respectively. The delay of the latches is 0.5 $nsec$. The speedup of the pipeline processor for a large number of instructions is:

A. 4.5
B. 4.0
C. 3.33
D. 3.0

gate2008-it co-and-architecture pipelining normal

HW

1.18.32 Pipelining: GATE IT 2005 | Question: 44 top

https://gateoverflow.in/3805

We have two designs $D1$ and $D2$ for a synchronous pipeline processor. $D1$ has 5 pipeline stages with execution times of 3 nsec, 2 nsec, 4 nsec, 2 nsec and 3 nsec while the design $D2$ has 8 pipeline stages each with 2 nsec execution time How much time can be saved using design $D2$ over design $D1$ for executing 100 instructions?

A. 214 nsec
B. 202 nsec
C. 86 nsec
D. −200 nsec

HW

1.18.30 Pipelining: GATE CSE 2021 Set 1 | Question: 53 top

https://gateoverflow.in/357398

A five-stage pipeline has stage delays of $150, 120, 150, 160$ and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each.

The total time to execute 100 independent instructions on this pipeline, assuming there are no pipeline stalls, is _______ nanoseconds.

1.7.1 Data Dependences: GATE CSE 2015 Set 3 | Question: 47 top

https://gateoverflow.in/8556

Consider the following code sequence having five instructions from I_1 to I_5. Each of these instructions has the following format.

OP Ri, Rj, Rk

Where operation OP is performed on contents of registers Rj and Rk and the result is stored in register Ri.

I_1: ADD R1, R2, R3

I_2: MUL R7, R1, R3

I_3: SUB R4, R1, R5

I_4: ADD R3, R2, R4

I_5: MUL R7, R8, R9

Consider the following three statements.

S1: There is an anti-dependence between instructions I_2 and I_5

S2: There is an anti-dependence between instructions I_2 and I_4

S3: Within an instruction pipeline an anti-dependence always creates one or more stalls

1.7.1 Data Dependences: GATE CSE 2015 Set 3 | Question: 47 top

https://gateoverflow.in/8556

Consider the following code sequence having five instructions from I_1 to I_5. Each of these instructions has the following format.

OP Ri, Rj, Rk

Destination

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

$I_2 - I_5$: WAW Dep

~~WAR Dep~~

WAR Dep.

Consider the following three statements.

S1: There is an anti-dependence between instructions I_2 and I_5 No

S2: There is an anti-dependence between instructions I_2 and I_4 YES

S3: Within an instruction pipeline an anti-dependence always creates one or more stalls NO

Never in In-order PL.

Pipelines (Carl Hamacher)

In-Order
Simple

No WAR
Hazard.

Superscalar
pipeline

WAR Hazard possible
BUT 100%
Eliminateable.

HW

1.18.4 Pipelining: GATE CSE 2001 | Question: 12 top

https://gateoverflow.in/753

Consider a 5−stage pipeline - IF (Instruction Fetch), ID (Instruction Decode and register read), EX (Execute), MEM (memory), and WB (Write Back). All (memory or register) reads take place in the second phase of a clock cycle and all writes occur in the first phase. Consider the execution of the following instruction sequence:

I1:	sub $r2, r3, r4$	/* $r2 \leftarrow r3 - r4$ */
I2:	sub $r4, r2, r3$	/* $r4 \leftarrow r2 - r3$ */
I3:	sw $r2, 100(r1)$	/* $M[r1 + 100] \leftarrow$ r2 */
I4:	sub $r3, r4, r2$	/* $r3 \leftarrow r4 - r2$ */

A. Show all data dependencies between the four instructions.
B. Identify the data hazards.
C. Can all hazards be avoided by forwarding in this case.

GATE CSE 1999 | Question: 13

HW

An instruction pipeline consists of 4 stages – Fetch (F), Decode field (D), Execute (E) and Result Write (W). The 5 instructions in a certain instruction sequence need these stages for the different number of clock cycles as shown by the table below

Instruction	F	D	E	W
1	1	2	1	1
2	1	2	2	1
3	2	1	3	2
4	1	3	2	1
5	1	2	1	2

Find the number of clock cycles needed to perform the 5 instructions.

GATE CSE 2002 | Question: 2.6, ISRO2008-19

asked in CO and Architecture Sep 16, 2014 • edited Jun 20, 2018 by Milicevic3306

9,414 views

HW

27

The performance of a pipelined processor suffers if:

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

gatecse-2002 co-and-architecture pipelining easy isro2008

Next Session:

WAW, WAR Hazards

& Register Renaming

Hazards vs. Dependences

dependence: fixed property of instruction stream (i.e., program)

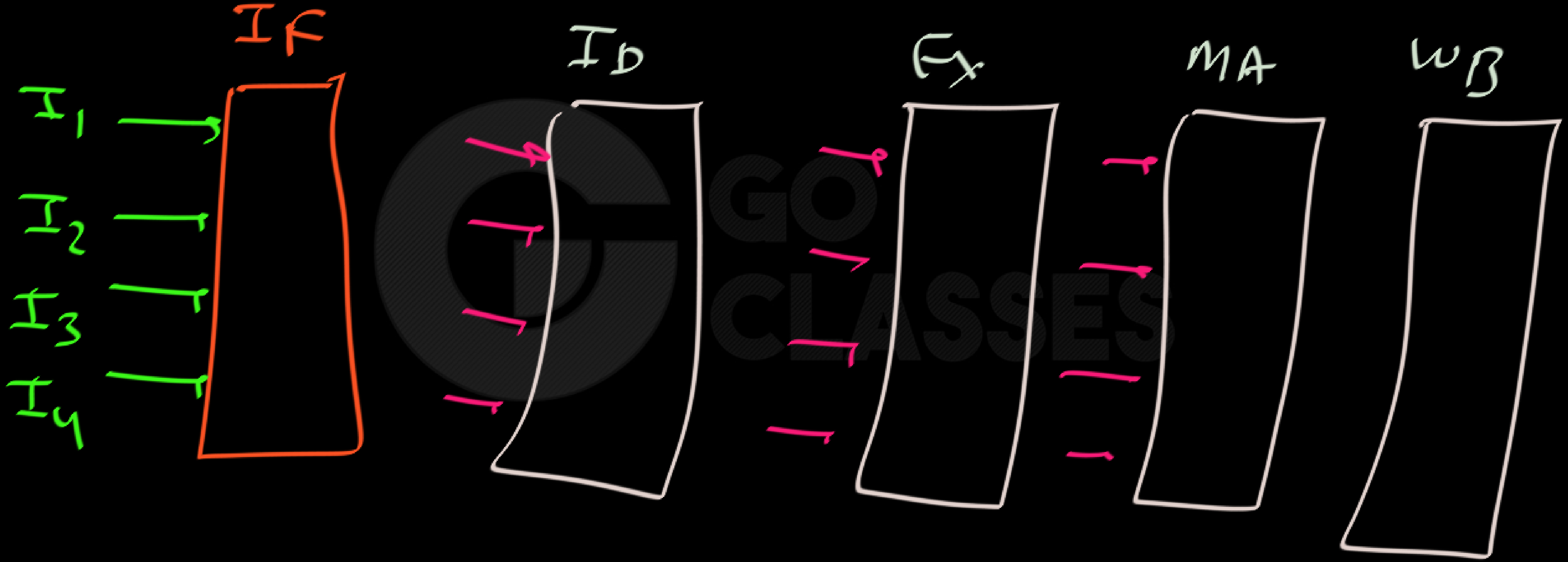
hazard: property of program *and processor organization*

- implies potential for executing things in wrong order
  - potential only exists if instructions can be simultaneously "in-flight"
  - property of dynamic distance between instrs vs. pipeline depth

For example, can have RAW dependence with or without hazard

- depends on pipeline

Superscalar System :



$I_1: R_1 \leftarrow R_2 + R_3$

$I_2: R_5 \leftarrow R_1 + R_4$

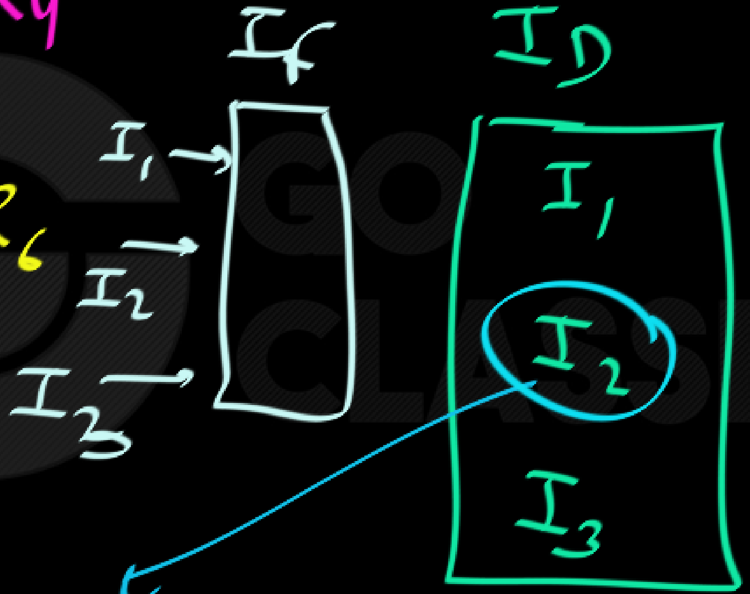
$I_3: R_5 \leftarrow R_6 + R_6$

WAW Dep $\rightarrow$ NEVER creates any stall in our simple In-order pipeline.

$I_1 : R_1 \leftarrow R_2 + R_3$ → Ex Stage

$I_2 : R_5 \leftarrow R_1 + R_4$

$I_3 : R_5 \leftarrow R_6 + R_6$



I_2 Reads wrong value of R_1 so I_2 Waits in ID

$I_1: R_1 \leftarrow R_2 + R_3$

$I_2: R_5 \leftarrow R_1 + R_4$

$I_3: R_5 \leftarrow R_6 + R_6$

R_5 of I_3

wrong

IF

I_1

I_2

I_3

ID

I_2

waiting

I_1

I_3

Out-of-order Execution

Superscalar System:

instructions

fetch is code-order

BUT may execute them out-of-order

Code:

I_1

I_2

I_3

⋮

when out-of-order (OOO) execution happens then

WAW Hazards
WAR "
RAW "
possible

when out-of-order (OOO) execution happens then

WAW Hazards
WAR "
RAW "

Can be eliminated Completely (100%)

possible

Always Real problem

Eliminate WAW, WAR 100% :

I_1 : R_1 ←

I_2 : R_1 ←

writing again in the same Reg is the Problem.

Solution: Never write in same Reg twice.

Eliminate WAW, WAR 100% :

$R_1 \leftarrow$ - - -
$R_1 \leftarrow$ - - -

Renaming $\longrightarrow$

$R_1 \leftarrow$ - - -
$R_{11} \leftarrow$ - - -

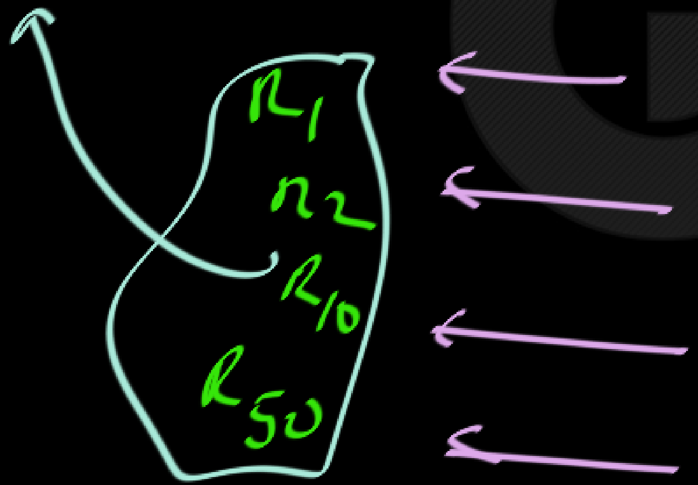
Renaming $\equiv$ $\boxed{\text{SSA in Compiler Design}}$

Renaming mechanism to Eliminate

WAW, WAR 100% from

Any pipeline :

Different Reg.



$R_{11} \leftarrow R_1 + R_1$

$R_{22} \leftarrow$

$R_{33} \leftarrow$

$\leftarrow$

No WAR
No WAW
Hazards

Done 100%

8.6 Superscalar Operation

Pipelining makes it possible to execute instructions concurrently. Several instructions are present in the pipeline at the same time, but they are in different stages of their execution. While one instruction is performing an ALU operation, another instruction is being decoded and yet another is being fetched from the memory. Instructions enter the pipeline in strict program order. In the absence of hazards, one instruction enters the pipeline and one instruction completes execution in each clock cycle. This means that the maximum throughput of a pipelined processor is one instruction per clock cycle.

A more aggressive approach is to equip the processor with multiple processing units to handle several instructions in parallel in each processing stage. With this arrangement, several instructions start execution in the same clock cycle, and the processor is said to use *multiple-issue*. Such processors are capable of achieving an instruction execution throughput of more than one instruction per cycle. They are known as *superscalar* processors. Many modern high-performance processors use this approach.

8.6.1 Out-of-Order Execution

In Figure 8.20, instructions are dispatched in the same order as they appear in the program. However, their execution is completed out of order. Does this lead to any problems? We have already discussed the issues arising from dependencies among

GO Classes Test Series 2023 | Mock GATE | Test 8 | Question: 64

asked in CO and Architecture 3 days ago • edited 2 days ago by Lakshman Patel RJIT

243 views

5

Consider the following instruction sequence for a hypothetical RISC processor.

T. $R1 \leftarrow R2 + R3$
U. $R4 \leftarrow R5 + R6$
V. $R5 \leftarrow R7 + R8$
W. $R9 \leftarrow R5 + R1$
X. $R10 \leftarrow R4 + R1$
Y. $R11 \leftarrow R10 + R1$
Z. $R9 \leftarrow R1 + R4$

Which of the following is a possible legal execution order for the instructions on an out-of-order processor without register renaming?

A. T, U, X, V, W, Z, Y
B. T, U, X, V, Z, W, Y
C. T, V, U, X, W, Y, Z
D. U, T, V, Y, X, W, Z

GO Classes Test Series 2023 | Mock GATE | Test 8 | Question: 63

asked in CO and Architecture 3 days ago • edited 2 days ago by Lakshman Patel RJIT

265 views

5

A classical RISC five-stage pipeline is depicted below:

instruction fetch	register fetch	execute	memory access	register write back

The pipeline has the following characteristics.

- The register file is written at the end of the clock cycle.
- The pipeline does not have other bypassing/forwarding hardware.

Consider the following sequence of assembly instructions:

```
#1: ADD R1, R2, R3
#2: SUB R4, R1, R5
#3: AND R6, R1, R7
#4: OR R8, R1, R9
#5: XOR R10, R1, R11
```

In all the above instructions, the destination register is the first (leftmost) register.
A read-after-write(RAW) data dependency occurs when a later instruction requires an input value that is set by an earlier instruction. A RAW data hazard occurs when one instruction writes a value into a register that will be used as input by a later instruction, but that value does not actually appear in the register by the cycle on which the later instruction attempts to read it.
Note that a RAW data hazard always implies a RAW data dependency, but some RAW data dependencies do not imply a RAW data hazard.
Identify the data hazards that would prevent the given sequence of instructions from executing correctly on the given hardware design above, unless the compiler inserted one or more nop instructions for correct execution.
Which of the following statements is correct?

A. Every RAW dependency in the given program is a RAW data hazard for the given pipeline.
B. The number of RAW dependencies in the given program is 4 but only 1 of them is a RAW data hazard for the given pipeline.
C. The number of RAW dependencies in the given program is 4 but only 3 of them are RAW data hazards for the given pipeline.
D. The number of RAW dependencies in the given program is 1 and it is a RAW data hazard for the given pipeline.

goclasses2023-mockgate-8 | goclasses | co-and-architecture | pipelining | 2-marks

Next Session:

Control Hazards