COMPUTER SCIENCE

Computer Organization and Architecture

Instruction Pipelining

Lecture_04

Vijay Agarwal sir

01 Pipelining Hazards

Pipeline

- Pipeline Concept
- Pipeline Design
- Execution Time in pipeline
- Performance Gain (Speed up Factor)
- Efficiency & Throughput.

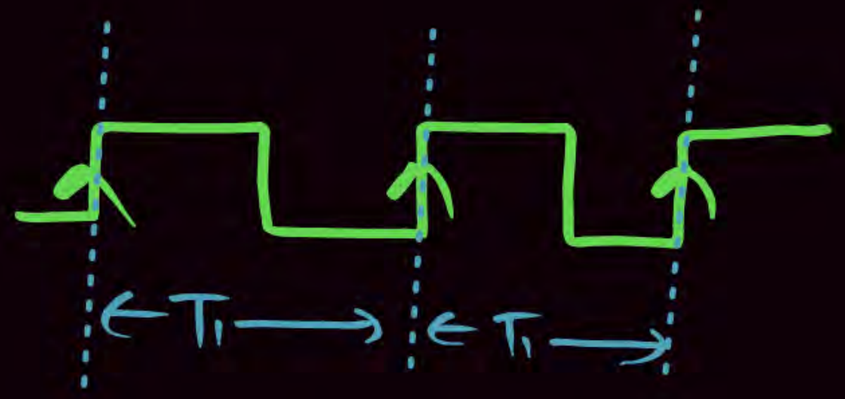
Practice Question & GATE PYQ.

How to Construct Pipeline.

CPI Concept

Why Clock Required

Meaning of CPI = 1.



How to Set this CPI in Uniform Delay Pipeline

How to set this CPI in Non Uniform Delay pipeline.

Timing Diagram Concept

ALL GATE & NK PYQ's.

Pipelining Strategy

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

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

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

PIPELINE

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

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

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

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

$n t_n$

$\Rightarrow 4 \times 4 = 16$ cycle.

O/P end

S_4				I_1				I_2				I_3				I_4
S_3			I_1				I_2				I_3				I_4	
S_2		I_1				I_2				I_3				I_4		
S_1	I_1				I_2				I_3				I_4			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

I/P end

Cycle

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

Non-PIPELINE

O/P end

	1	2	3	4	5	6	7	8	9	10	11
S_4				I_1	I_2	I_3	I_4				
S_3			I_1	I_2	I_3	I_4					
S_2		I_1	I_2	I_3	I_4						
S_1	I_1	I_2	I_3	I_4							

I/P end

Cycle

PIPELINE

k = 4

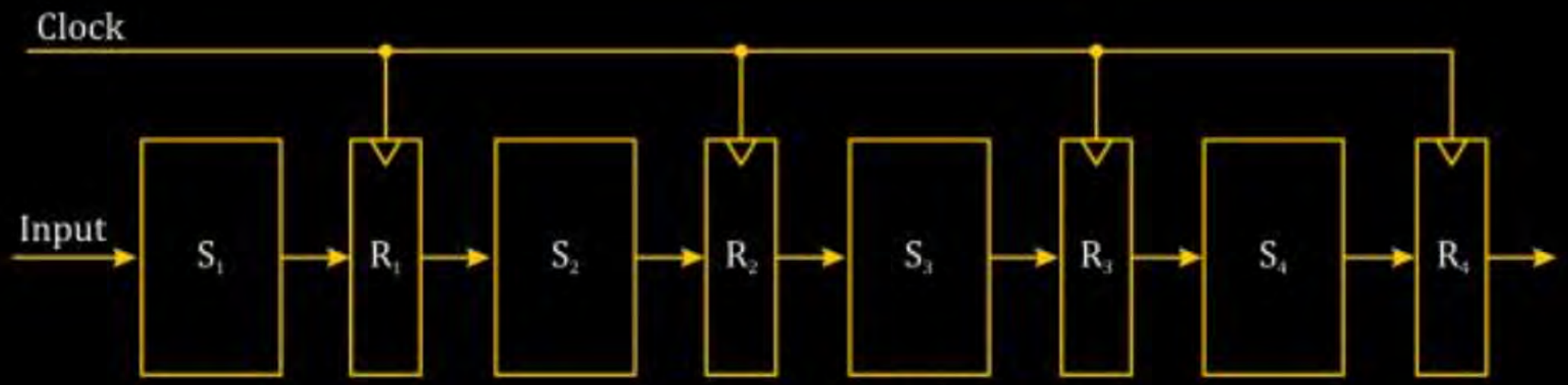
n = 4

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

$$= k * t_p + (n-1)\,t_p$$

PIPELINE

PIPELINE Design



Four-segment pipeline.

Q. COA 2017: CS

NIC PYQ SERIES

In a 10-bit computer instruction format, the size of address field is 3-bits. The computer uses expanding OP code technique and has 4 two-address instructions and 16 one-address instructions. The number of zero address instructions it can support is

A 256

B 356

C 640

D 756

Ans [C].

← 10 bit →		
OP	AF1	AF2
4bit	3bit	3bit

Primitive

← 10 bit →	
OP	AF1
7bit	3bit

Derived

← 10 bit →
OPCODE
10bit

Further Derived.

Primitive: Total Number of operation $= 2^4 = 16$ operation.

Given 2AI $= 4$

#Free opcode After Allocating 2AI $= 16 - 4 = 12$

Derived: Total # operation in 1AF $=$ Free opcode $\times 2$

Increment bit in opcode

$\Rightarrow 12 \times 2^{7-4} \Rightarrow 12 \times 2^3 \Rightarrow 12 \times 8 = 96$

Give LAF = 16

Free opcode After Allocating IAI = 96 − 16 = 80.

Total # operation in OAF = $80 \times 2^{10-7} \Rightarrow 80 \times 2^3$

$\Rightarrow 80 \times 8$

$= 640$ Ans

Q.

Design D_1	500ns	450ns	600ns	800ns
Design D_2	600ns	820ns	780ns	650ns.

Q What is the cycle time, latency of one Instruction & Throughput in the Pipeline Design D_1 & D_2 if in Both Design D_1 & D_2 having Buffer Register b/w each stages have a Delay of 20 nsec?

Soln

Design D_1	500ns	450ns	600ns	800ns
Design D_2	600ns	820ns	780ns	650ns

Bubble Delay = 20nsec.

Design D_1

$$t_{P_{D_1}} = \max\left(\text{Stage Delay} + \text{Buffer Delay}\right)$$

$$\Rightarrow \max(800 + 20)$$

cycle time $\boxed{t_{P_{D_1}} = 820\,\text{nsec}}$

Latency of One Instn | Latency of Instn

$$ET_{D_1} = 4 \times 820 = 3280\,\text{nsec}.$$

Design D_2

$$t_{P_{D_2}} = \max\left(\text{Stage Delay} + \text{Buffer Delay}\right)$$

cycle time $= 820 + 20$

$$\boxed{t_{P_{D_2}} = 840\,\text{nsec}}.$$

Latency of One Instn

$$ET_{D_2} = \underline{4 \times 840} = 3360\,\text{nsec}$$

Soln

Design D_1	500ns	450ns	600ns	800ns
Design D_2	600ns	820ns	780ns	650ns

Bubble Delay = 20nsec.

1 Instn

Design D_1

$ET_{D_1} = [k+(n-1)]t_p \Rightarrow [4+(1-1)] \times 820$

$ET_{D_1} = 4 \times 820 \Rightarrow \boxed{3280 \text{ nsec}}$

Throughput $= \frac{1}{t_p} = \frac{1}{820\text{ns}}$

Design D_2

$ET_{D_2} = [k+(n-1)]t_p \Rightarrow [4+(1-1)] \times 840$

$ET_{D_2} = 4 \times 840 \Rightarrow \boxed{ET_{D_2} = 3360 \text{ nsec}}$

Throughput $= \frac{1}{840}$ nsec.

Soln

$$\underline{D_1} = \left(\underline{820ns}, 3280ns, \frac{1}{820}\right)$$

$$D_2 = \left(840ns, 3360ns, \frac{1}{840}\right)$$

Pipeline D_1 Performance is better than Pipeline D_2.

PART I

- Pipeline Concept
- Performance Evaluation
- Efficiency & Throughput
- Uniform & Non Uniform Delay PIPELINE.

$\boxed{CPI = 1}$

PART II

CPI = 1 cycle

Hazards.

$CPI \neq 1$ $\Rightarrow$ 1 + Something

Extra cycle
$\Updownarrow$
(Stalls)
$\Updownarrow$
(Bubbles)

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

PIPE LINE - STAGES.

1. 2 stage PIPELINE:

FI	EX

2 stage Pipeline.

PIPE LINE - STAGES.

② 4 STAGE PIPE :

IF : Instn Fetch
ID : Instn Decode
EX : Execute
RS|WB : Write Back.

I_1:	IF	ID	EX	WB		
I_2:		IF	ID	EX	WB	
I_3:			IF	ID	EX	WB
I_4:						

PIPE LINE - STAGES.

3 6 STAGE PIPELINE = FI
DI
CO
FO
EI
WO.

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

Note

Stages may be varying.

Like sometimes 2 stage
4 stage
5 stage
6 stage
etc.

given in Question.

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

But we generally we use Classical Pipeline

Classical PIPELINE	OR	MIPS Pipeline	OR	Standard Pipeline	OR	RISC Pipeline.

RISC Pipeline: '5 stage'

RISC | MIPS Pipeline

5 Stage Pipeline

RISC PIPELINE

In The RISC Pipeline 5 Stages:

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

IF
ID
EX
MA
WB.

RISC Pipeline

1. IF (Instn Fetch).
2. ID (Decode & Register Read)
3. EX
4. MA
5. WB.

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 (Decode & Register Read).

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

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

Note

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).

(Register write)

5. Write Back {WB Stage} : In this stage Register write (Operand storing into reg. file) operation performed.

Register Write

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

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

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 : is a Situation that makes the Pipeline to Idle or Stalls.

1. Structural Hazards (Resource Conflict)
2. Data Hazards. (Data/operand)
3. Control Hazards. (Branch Instruction).

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". or Bubbles
- When stall in present in the pipeline then CPI ≠ 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

Hazards Meaning/working.

5 Stage Pipeline.

Normal Execution

I_1 :

I_2 :

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

Hazards Meaning/working.

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 $r_4 \leftarrow r_1 * r_5$

5 Stage Pipeline.

Instruction I_2 Data Dependant on I_1

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

r_1

Stalls

Bubble / Extra cycle

1 Structural Dependency / Hazards.

Structural Dependency is created in the Pipeline when Two or More Phase Require the Same Resource at the Same time & they are Not able to Run Simultaneously.

- Structural Dependency is created in the Pipeline Due to Resource Conflict.
- Resource May be Registers, functional Unit, ALU, Memory etc.

Instruction Fetch (IF) ⇒ MEM.

Instn Decode

Execute [ALU]

Memory Access ⇒ [MEM].

Structural Dependency

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1:	MEM	ID	EX	MEM	WB		
I_2:		MEM	ID	EX	MEM	WB	
I_3:			MEM	ID	EX	MEM	WB
I_4:				MEM			

'Resource Conflict'

CC4

In clock cycle [CC] 4

Both I_1 & I_4 Accessing the Same Resource 'Memory'.

This Situation is Called Resource Conflict so So. (I_1 & I_4) Not go for execution

So Here Both I_1 & I_4 Can not go for execution.

So keep I_4 into the waiting Untill the Resource become Available

This waiting Creates 'Stalls|Bubbles|Extra cycle' in the Pipeline.

Structural Dependency

	CC1	CC2	CC3	CC4	CC5	CC6	CC7
I_1:	MEM	ID	EX	MEM	WB		
I_2:		MEM	ID	EX	MEM	WB	
I_3:			MEM	ID	EX	MEM	WB
I_4:				MEM	MEM	MEM	MEM

1 2 3

3 Extra cycle

Stalls / Bubbles

Here I_1 is accessing the Memory to Store the Data.

&

I_4 is Accessing the Memory to Fetch the Instruction

- Memory
  - (I-Cache) (Instn Cache) Instruction | CODE Memory [IM | CM]
  - Data Memory [DM] (D-Cache) Data Cache

To minimize the Stalls Due to Structural Dependency One Hardware mk ism is Used called as Renaming.

Renaming state the Divide the memory into independent module to store the Instruction & Data Separately Called Instruction/CODE memory (CM/IM) & Data memory (DM) Respectively.

Structural Dependency

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	
I_1:	CM	ID	EX	DM	WB			
I_2:		CM	ID	EX	DM	WB		
I_3:			CM	ID	EX	DM	WB	
I_4:				CM	ID	EX	DM	WB

No Stalls.

THANK YOU!