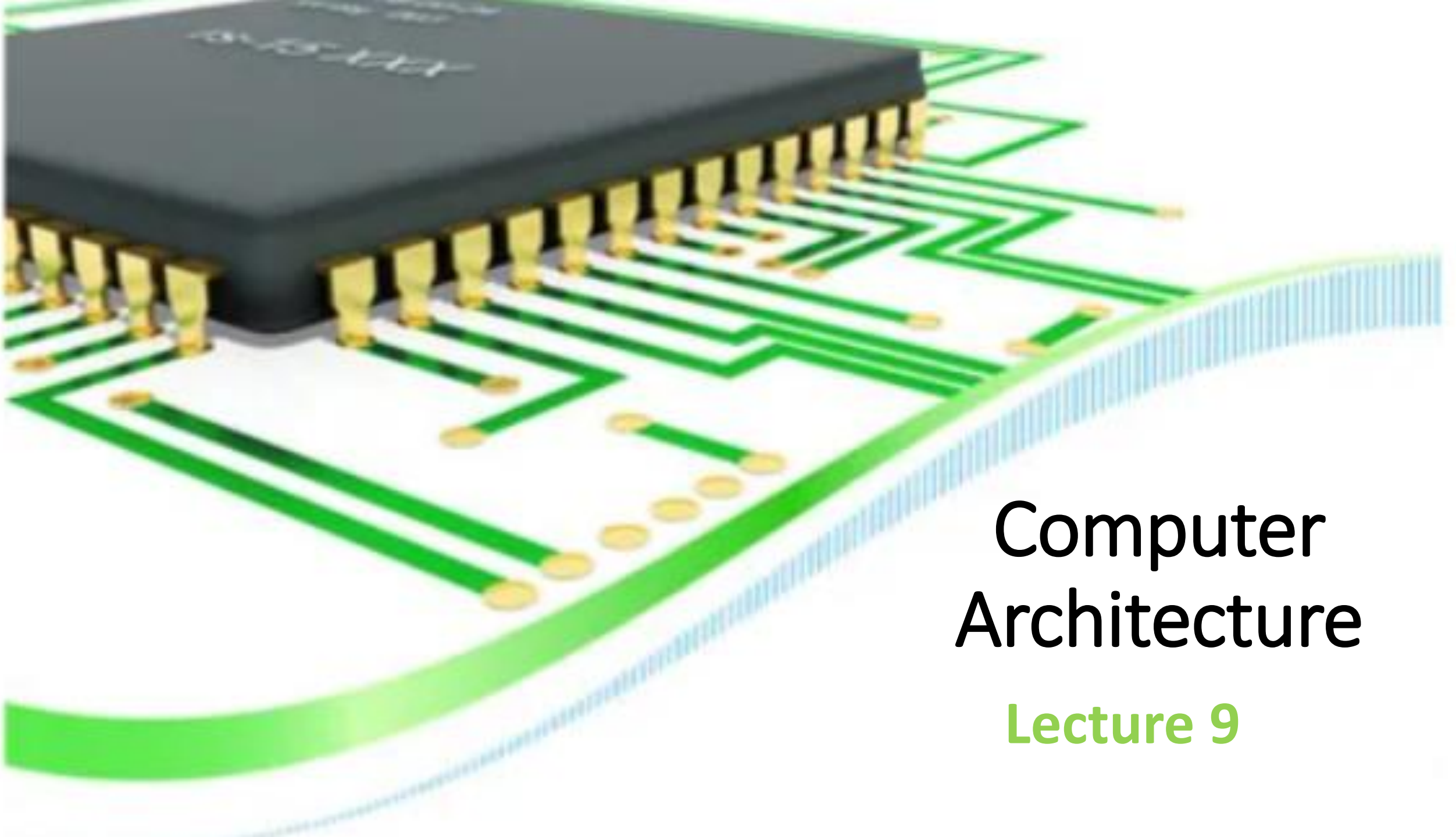


Computer Architecture

Lecture 9

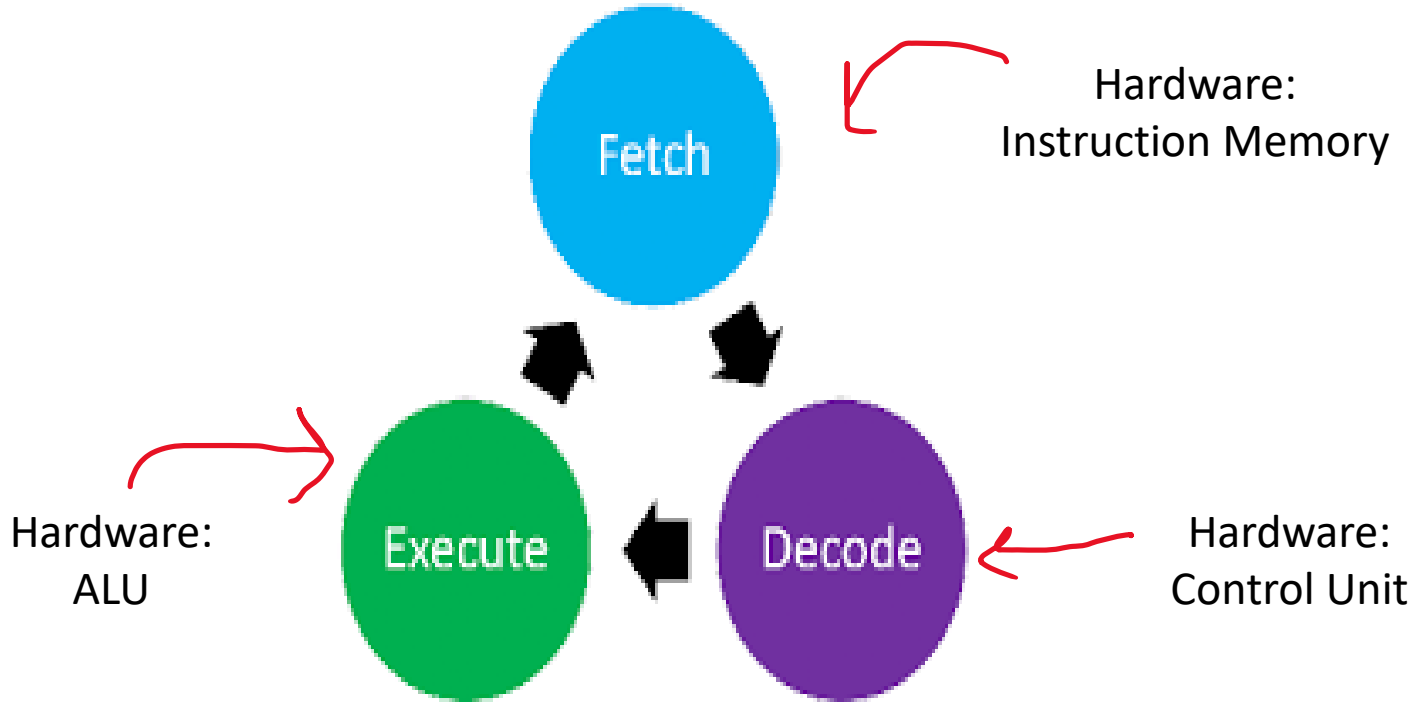
Agenda

- Instruction Cycle
- Pipelining
- Hazards

Instruction Cycle

- A program residing in the memory unit of a computer consists of a sequence of instructions.
- The processor executes these instructions by going through a cycle for each instruction.

Instruction Cycle

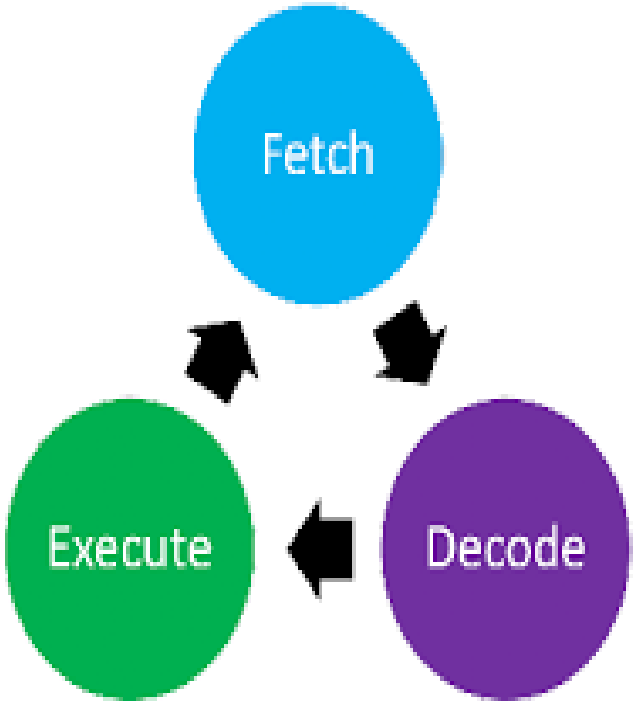


Instruction Cycle

- The instruction cycle represents the series of steps that a computer's central processing unit (CPU) goes through to execute a single machine instruction.

- Each instruction cycle consists of at least the following stages:
  1. Fetch instruction from memory.
  2. Decode the instruction.
  3. Execute the instruction.

- Instruction cycle is also known as the fetch-decode-execute cycle.



Fetch Stage

- The fetch is the initial stage of the instruction cycle. It involves retrieving the next instruction from memory.
- The CPU sends the contents of the PC to the MAR and sends a read command on the control bus
- In response to the read command (with an address equal to PC), the memory returns the data stored at the memory location indicated by the PC on the data bus.
- The CPU copies the data from the data bus into its MDR (MBR).
- Then, the CPU copies the data from the MDR to the instruction register for instruction decoding.
- The PC is incremented so that it points to the next instruction.

Fetch Stage

- Sequence of microoperations representing fetch stage:
  - t1: MAR ← PC
  - t2: MBR ← M[MAR], PC ← PC+1
  - t3: IR ← MBR

Assume that each instruction is one cell

PC 301

~~300~~

MBR

add R3, R1, R2

MAR

300

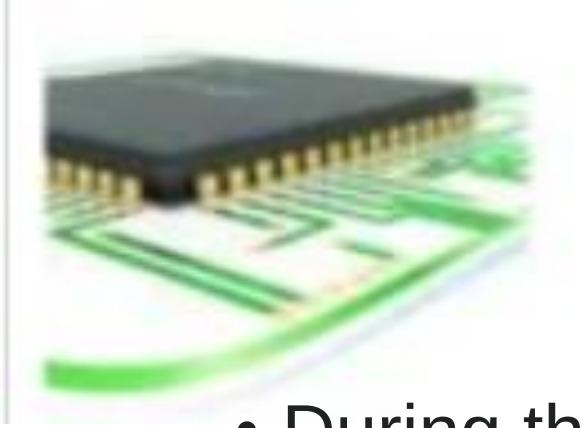
IR

add R3, R1, R2

Memory

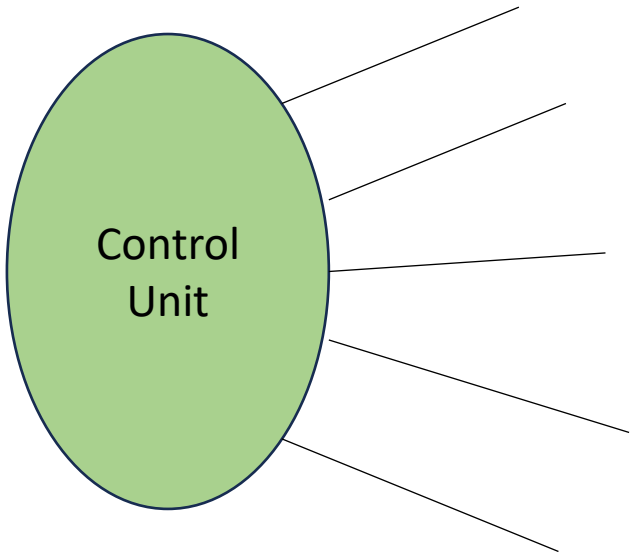
300	add R3, R1, R2
301	sub R6, R4, R5

Remember: Everything is in binary



Decode Stage

- During this stage, the instruction presented in the instruction register is interpreted by the control unit.



WAW, I am smart and
full of magic..
I hope you could know
me better

Execute Stage

- The CPU sends the decoded instruction as a set of control signals to the corresponding computer components.

- The instruction usually involves arithmetic or logic, the ALU is utilized.

Instruction cycle

In some architecture, the instruction cycle is 5 stages:

1. Fetch instruction from memory.
2. Decode the instruction.
3. Fetch the operand (Memory Access).
4. Execute the instruction.
5. Write back

In some architecture, the instruction cycle is 6 stages:

1. Fetch instruction from memory.
2. Decode the instruction.
3. Fetch the operand.
4. Execute the instruction.
5. Memory Access.
6. Write back

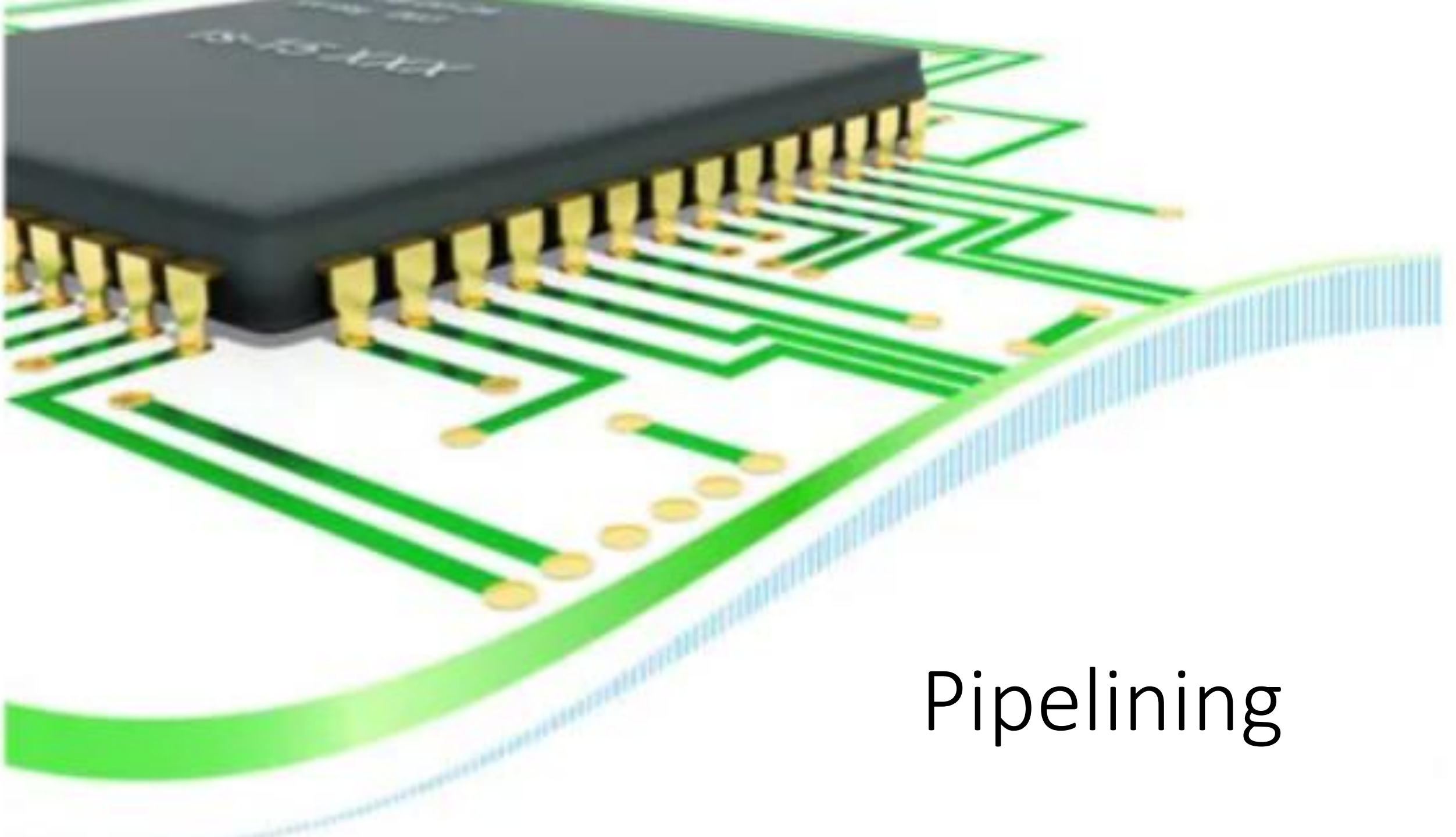
Instruction cycle

Fetch the operand /Memory Access:

Reading the operands from the data memory, or the registers.

Write back:

Writing the result back in a register or in data memory.



Pipelining

Sequential Processing mode

- Sequential (Serial) mode executes instructions in the sequence in which they are written in the program.
- Assume a 5-stage instruction cycle (IF = Instruction Fetch, ID = Instruction Decode, MEM = Memory access, EX = Execute, WB = Register write back).
- And We would like to execute a program with 3 instructions.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Instruction 1	IF1	ID1	MEM1	EX1	WB1										
Instruction 2						IF2	ID2	MEM2	EX2	WB2					
Instruction 3											IF3	ID3	MEM3	EX3	WB3

Pipelining

- Instruction pipelining is a technique for implementing instruction-level parallelism within a single processor.

- Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps performed by different processor units with different parts of instructions processed in parallel.

Pipelining Processing

- Assume a 5-stage instruction cycle (IF = Instruction Fetch, ID = Instruction Decode, MEM = Memory access, EX = Execute, WB = Register write back).
- And We would like to execute a program with 3 instructions on a machine that supports pipelining

	1	2	3	4	5	6	7
Instruction1	IF1	ID1	MEM1	EX1	WB1		
Instruction2		IF2	ID2	MEM2	EX2	WB2	
Instruction3			IF3	ID3	MEM3	EX3	WB3

Speed up

Example1:

Assume having a *non-pipelined* machine with 5 stages, where each stage needs 20ns to finish, what is the execution time for 3 instructions.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Instruction 1	IF1	ID1	MEM1	EX1	WB1										
Instruction 2						IF2	ID2	MEM2	EX2	WB2					
Instruction 3											IF3	ID3	MEM3	EX3	WB3

20 ns **20 ns** **20 ns** **20 ns** **20 ns**

Clock cycle time on a non-pipelined machine → t_n

Execution time$_{non\text{-}pipelined}$ = 3 x (20+20+20+20+20)
= 300 ns

Execution time$_{non\text{-}pipelined}$ = n x t_n

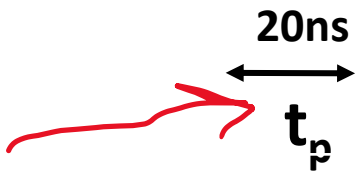
n is the number of instructions (tasks)

Speed up

Example1:

Assume *a pipelined* is introduced to the machine, where each stage needs 20ns to finish, what is the execution time for 3 instructions.

	1	2	3	4	5	6	7
Instruction1	IF1	ID1	MEM1	EX1	WB1		
Instruction2		IF2	ID2	MEM2	EX2	WB2	
Instruction3			IF3	ID3	MEM3	EX3	WB3



Clock cycle time on a pipelined machine

Execution time$_{pipelined}$ = (5+3-1) x 20 = 140 ns

Execution time$_{pipelined}$ = (k+n-1) x t_p

k is the number of stages

Speed up

Example1:

Calculate the speed up gained when the pipeline is introduced.

$$\text{Speed up} = \frac{\text{Execution time}_{\text{non-pipelined}}}{\text{Execution time}_{\text{pipelined}}}$$

$$= \frac{n \times t_n}{(k+n-1) \times t_p} = \frac{300}{140} = 2.14$$

Speed up

$$\text{Speed up} = \frac{\text{Execution time}_{\text{non-pipelined}}}{\text{Execution time}_{\text{pipelined}}}$$

$$= \frac{n \times t_n}{(k+n-1) \times t_p}$$

Where

- t_n is Clock cycle time on a non-pipelined machine.
- t_p is Clock cycle time on a pipelined machine
- k is the number of stages
- n is the number of instructions or tasks.

But really not all stages take the same time.

Speed up

Example2:

Assume having a *non-pipelined* machine, where each stage needs as shown is the table, what is the execution time for 3 instructions?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Instruction 1	IF1	ID1	MEM1	EX1	WB1										
Instruction 2						IF2	ID2	MEM2	EX2	WB2					
Instruction 3											IF3	ID3	MEM3	EX3	WB3

20 ns **40 ns** **30 ns** **20 ns** **10 ns**

t_n

Execution time$_{non\text{-}pipelined}$ = 3 x (20+40+30+20+10)
= 360 ns

Speed up

Example2:

Assume *a pipelined* is introduced to the machine, where each stage needs is given as follows, what is the execution time for 3 instructions?

	1	2	3	4	5	6	7
Instruction1	IF1	ID1	MEM1	EX1	WB1		
Instruction2		IF2	ID2	MEM2	EX2	WB2	
Instruction3			IF3	ID3	MEM3	EX3	WB3

Where
IF → 20 ns
ID → 40 ns
MEM → 30 ns
EX → 20 ns
WB → 10 ns

t_p

Execution time$_{pipelined}$ = (5+3-1) x 40 = 280 ns

What is the t_p in this case?

Take the max time as tp

Speed up

Example2:

Calculate the speed up gained when the pipeline is introduced.

$$\text{Speed up} = \frac{\text{Execution time}_{\text{non-pipelined}}}{\text{Execution time}_{\text{pipelined}}}$$

$$= \frac{n \times t_n}{(k+n-1) \times t_p} = \frac{360}{280} = 1.28$$

Latch delay

- Latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals.

- They are used in digital systems as temporary storage elements to store binary information.

Speed up

Example2:

Assume *a pipelined* is introduced to the machine with latch 2 ns, where each stage needs is given as follows, what is the execution time for 3 instructions?

	1	2	3	4	5	6	7
Instruction1	IF1	ID1	MEM1	EX1	WB1		
Instruction2		IF2	ID2	MEM2	EX2	WB2	
Instruction3			IF3	ID3	MEM3	EX3	WB3

Where
IF → 20 ns
ID → 40 ns
MEM → 30 ns
EX → 20 ns
WB → 10 ns

t_p

Execution time$_{pipelined}$ = (5+3-1) x 42 = 294 ns

What is the t_p in this case?

Take the max time + latch as tp

Speed up

Example2:

Calculate the speed up gained when the pipeline with latch is introduced.

$$\text{Speed up} = \frac{\text{Execution time}_{\text{non-pipelined}}}{\text{Execution time}_{\text{pipelined}}}$$

$$= \frac{n \times t_n}{(k+n-1) \times t_p} = \frac{360}{294} = 1.22$$

Check your understanding

Consider a machine where each instruction passes by 4 stages, the 4 stages need 20, 30, 10, and 15 ns respectively. If 50 tasks to be executed and a pipeline with latch 3 ns is applied to the machine. Calculate the Speed up gained.

Solution:

Execution time$_{non\text{-}pipelined}$ = n x t_n = 50x(20+30+10+15) = 3750 ns

Execution time$_{pipelined}$ = (k + n-1) x t_p = (4+49) x33 = 1749 ns

Speedup = 3750/1749 = 2.14

Maximum Speed up

Note:

When we deal with millions of instructions, we could calculate the maximum speed up directly as follows:

$$\textbf{Speed up} = \frac{n \times t_n}{(k+n-1) \times t_p}$$

$$\textbf{Maximum Speed up} = \frac{t_n}{t_p}$$

Problem 1

- Assume that the individual stages of on a machine takes the following execution time:

IF	ID	EX	MEM	WB
300ps	400ps	500ps	200ps	100ps

- Assume that when pipelining, each pipeline stage costs 10ps extra between pipeline stages.

a) What is the clock cycle time in a pipelined and non-pipelined processor?

b) If you could split one of the pipeline stages into 2 equal halves, which one would you choose? What is the new cycle time?

Solution:

a) Clock cycle time in a non-pipelined $t_n = 300 + 400 + 500 + 200 + 100 = 1500$ ps

Clock cycle time in a pipelined $t_p = 500 + 10 = 510$ ps

a) Splitting the longest stage is the only way to reduce the cycle time. After splitting it, the new cycle time is based on the new longest stage.

Old longest stage is EX.

New Clock cycle time = 410ps

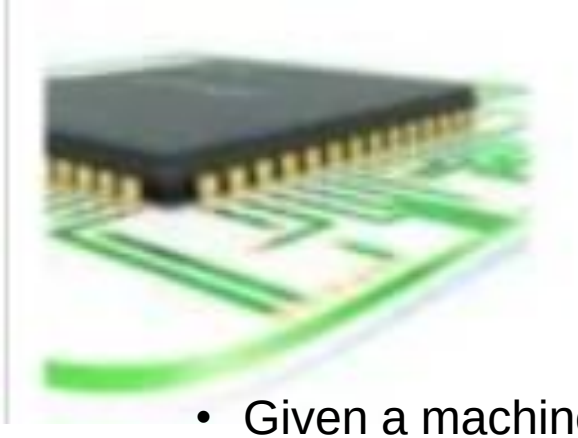
Problem 2

- Draw a space-time diagram for a five-segment pipeline showing the time it takes to process eight tasks. Determine the number of clock cycles that it takes to process 100 tasks in a five-segment pipeline.

Solution:

Segment/Cycles	1	2	3	4	5	6	7	8	9	10	11	12	
S1	T1	T2	T3	T4	T5	T6	T7	T8					
S2		T1	T2	T3	T4	T5	T6	T7	T8				
S3			T1	T2	T3	T4	T5	T6	T7	T8			
S4				T1	T2	T3	T4	T5	T6	T7	T8		
S5					T1	T2	T3	T4	T5	T6	T7	T8	

Number of clock cycles = $k + n-1 = 5 + 100 – 1 = 104$ clock cycles



Problem 3

- Given a machine with a 4-stage pipeline (Fetch, Decode, Execute and Write-Back), assuming that 5 instructions independent of each other need to be executed, fill the table with the detailed stages that will occur in each clock cycle and write down how many clock cycles needed to finish the whole execution of the 5 instructions. You could add rows as many as you need

Time	Details
1	
2	
3	

Problem 3

- Given a machine with a 4-stage pipeline (Fetch, Decode, Execute and Write-Back), assuming that 5 instructions independent of each other need to be executed, fill the table with the detailed stages that will occur in each clock cycle and write down how many clock cycles needed to finish the whole execution of the 5 instructions. You could add rows as many as you need

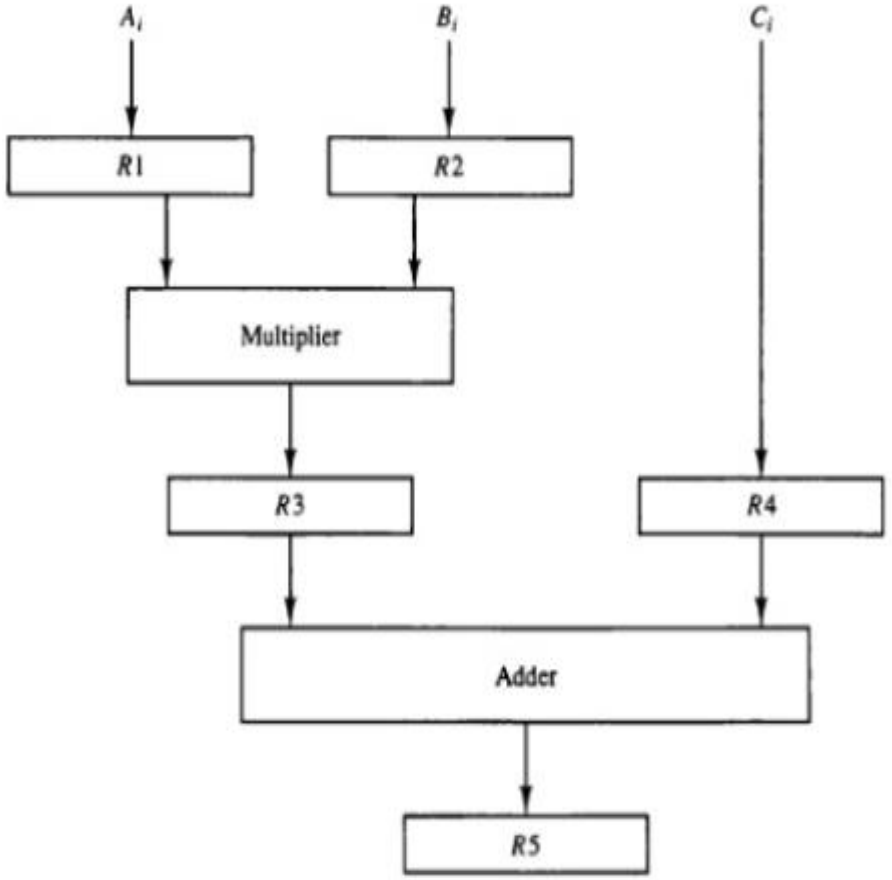
Solution:

Time	Details
1	IF1
2	IF2, ID1
3	IF3, ID2, EX1
4	IF4, ID3, EX2, WB1
5	IF5, ID4, EX3, WB2
6	ID5, EX4, WB3
7	Ex5, WB4
8	WB5

Problem 4

The pipeline of the below figure has the following propagation times: 20 ns for the operands to be read from memory into registers R1 and R2, 35 ns for the signal to propagate through the multiplier, 5 ns for the transfer into R3, and 30 ns to add the two numbers into R5.

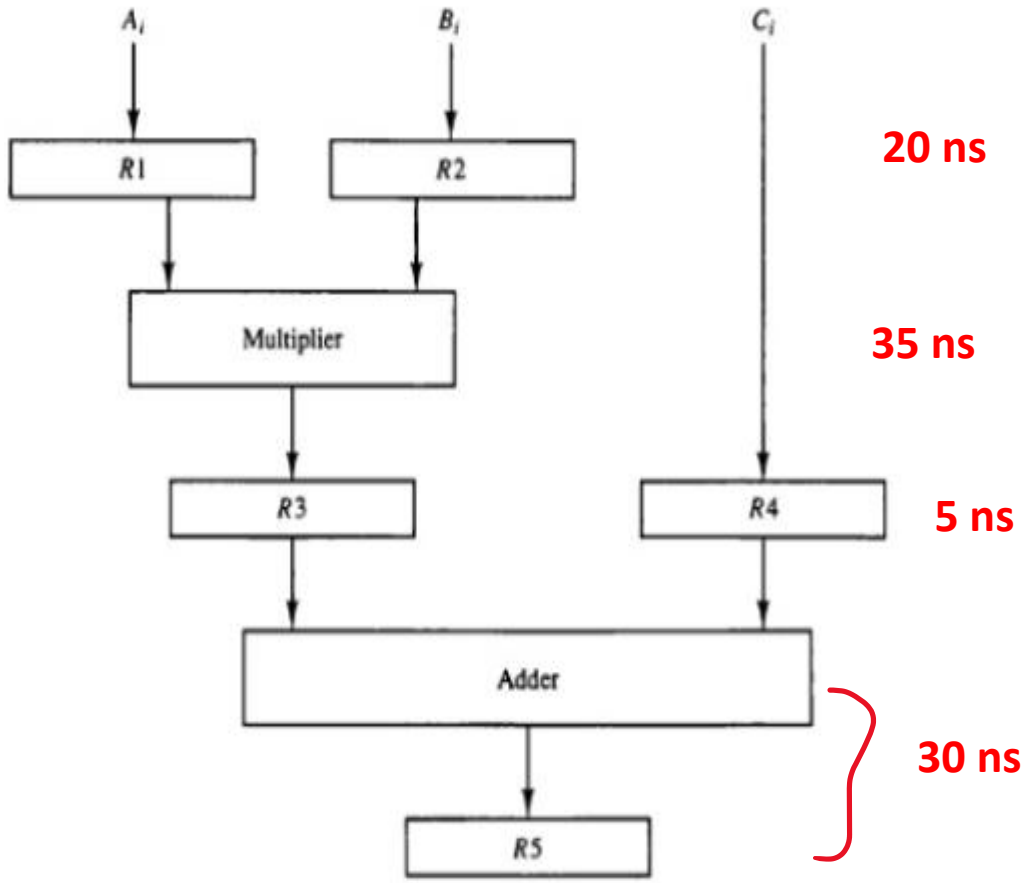
a) What is the minimum clock cycle time that can be used?
b) A non-pipeline system can perform the same operation by removing R3 and R4. How long will it take to multiply and add the operands without using the pipeline?
c) What is the maximum speed up that can be achieved?



Problem 4

The pipeline of the below figure has the following propagation times: 20 ns for the operands to be read from memory into registers R1 and R2, 35 ns for the signal to propagate through the multiplier, 5 ns for the transfer into R3 and R4, and 30 ns to add the two numbers into R5.

a) What is the minimum clock cycle time that can be used?



Solution:

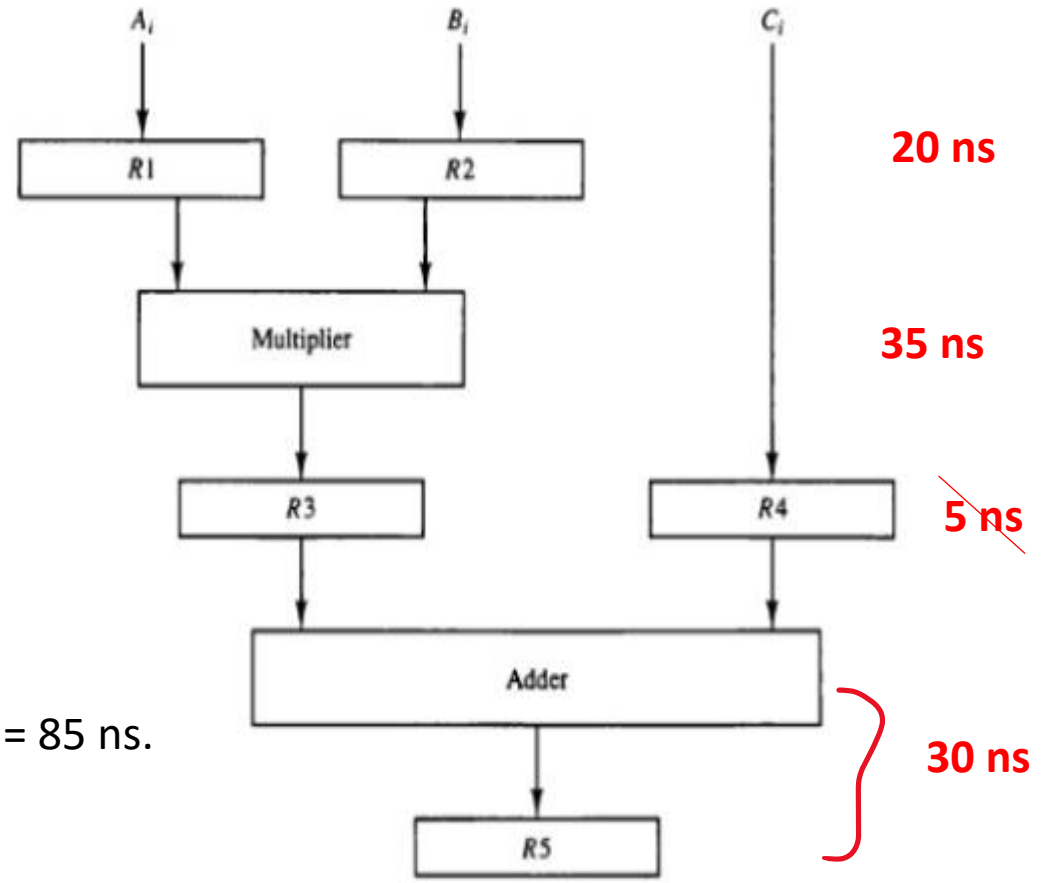
Minimum clock cycle time = t_p = 35 ns.

Problem 4

The pipeline of the below figure has the following propagation times: 20 ns for the operands to be read from memory into registers R1 and R2, 35 ns for the signal to propagate through the multiplier, 5 ns for the transfer into R3 and R4, and 30 ns to add the two numbers into R5.

b) A non-pipeline system can perform the same operation by removing R3 and R4. How long will it take to multiply and add the operands without using the pipeline?

c) What is the maximum speed up that can be achieved?



Solution:

b) Time on non-pipelined machine if no R3 or R4 = 20 + 35 + 30 = 85 ns.

c) Maximum Speedup = t_n / t_p = 85/ 35 = 2.42

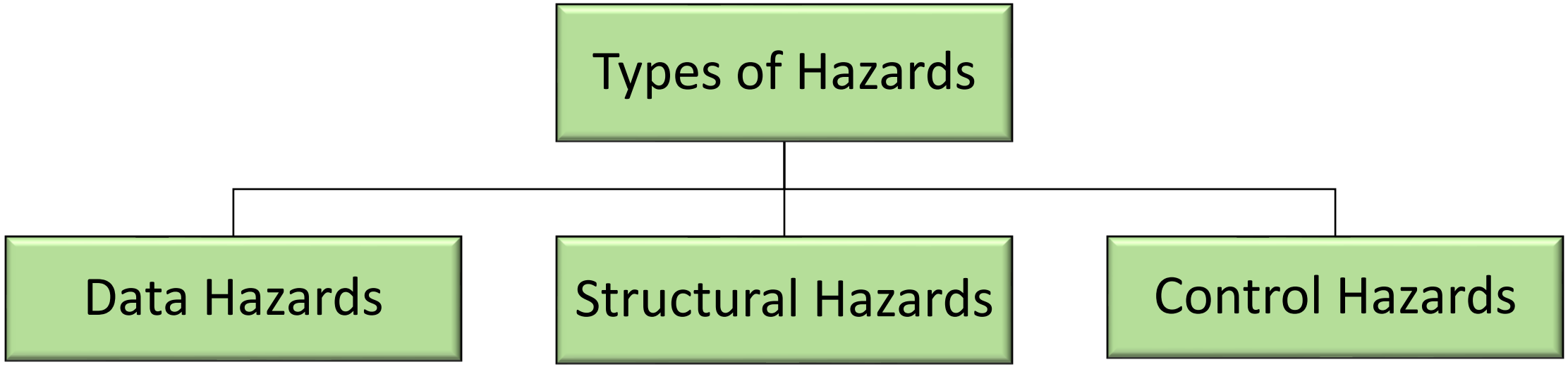
BUT

There are conditions that can occur in a ***pipelined*** machine that prevent the execution of a subsequent instruction in a particular cycle

Hazards

Pipelining Hazards

- Pipeline Hazards are situations that prevent the next instruction from executing in its designated clock cycle.
- Hazards reduce the performance from the ideal speedup gained by pipelining.



Pipelining Hazards

- ***Data Hazards:***

They arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in the pipeline.

- ***Structural Hazards:***

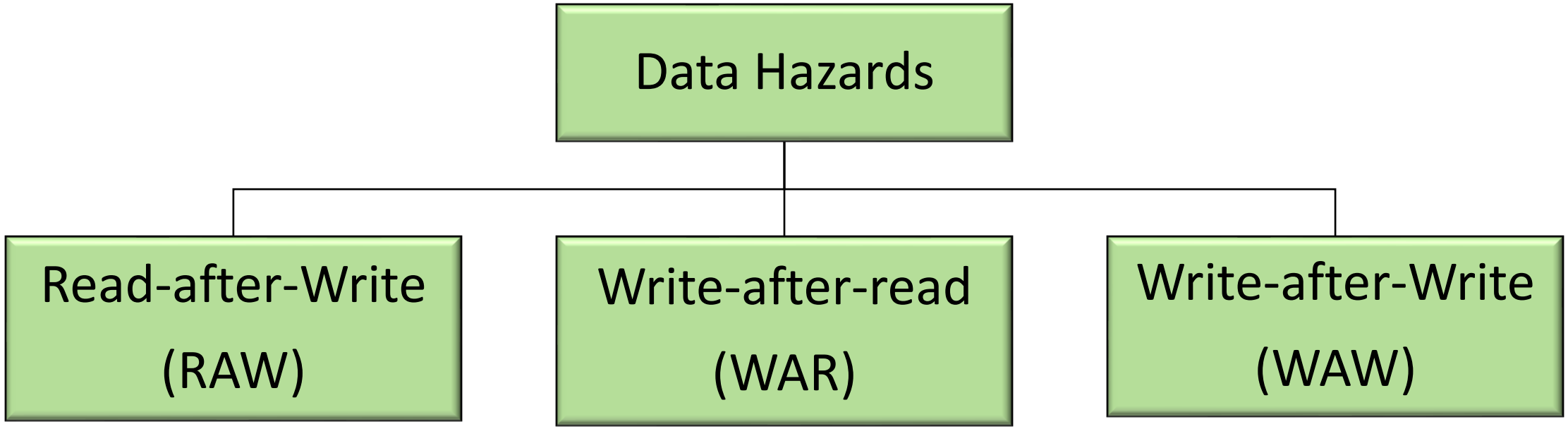
They arise from resource conflicts when the hardware can’t support all possible combinations of overlapping instructions.

- ***Control Hazards:***

They arise from pipelining of branches and other instructions that change the Program Counter (PC).

Data Hazards

- Data hazards include write-after-read (WAR), read-after-write (RAW), and write-after-write (WAW)



Data Hazards: RAW

Instruction 1: add R3, R1, R2

Instruction 2: add R5, R3, R4

	1	2	3	4	5	6
Instruction1	IF1	ID1	MEM1	EX1	WB1	
Instruction2		IF2	ID2	MEM2	EX2	WB2

Instruction 2 tries to read a source before instruction 1 writes to it.

A read-after-write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved.

Data Hazards: WAR

Instruction 1: add R4, R1, R5

Instruction 2: add R5, R2, R3

Instruction 2 tries to write a destination before it is read by instruction 1.

A write-after-read (WAR) data hazard represents a problem with concurrent execution.

In some situations, instruction 2 may finish before instruction 1 (i.e., with concurrent execution), it must be ensured that the result of register R5 is not stored before instruction1 has had a chance to fetch the operands.

Data Hazards: WAW

Instruction 1: add R4, R1, R2

Instruction 2: add R4, R5, R6

Instruction 2 tries to write an operand before it is written by instruction 1.
A write-after-write (WAW) data hazard may occur in a concurrent execution environment.

The write-back (WB) of instruction 2 must be delayed until instruction 1 finishes executing.

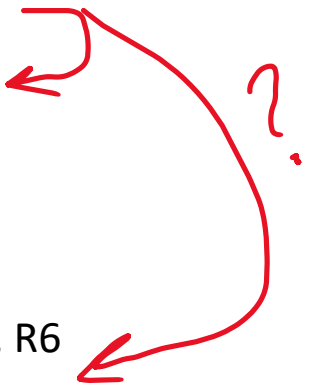
Structural Hazards

- A structural hazard occurs when two (or more) instructions that are already in pipeline need the same resource.
- The result is that instruction must be executed in series rather than parallel for a portion of pipeline.
- Example: A situation in which multiple instructions are ready to enter the execute instruction phase and there is a single ALU (Arithmetic Logic Unit).
- One solution to such resource hazard is to increase available resources, such as having multiple ports into main memory and multiple ALU (Arithmetic Logic Unit) units.

Control Hazards

- Control hazard occurs when the pipeline makes wrong decisions on branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded.

```
beq R1, R2, label
add R5, R2, R3
|
|
|
label: sub R3, R4, R6
```



How Hazards are Resolved?

- In the design of pipelined computer processors, a **pipeline stall** is a delay in the execution of an instruction in order to resolve a hazard.
- During this, the values of PC are preserved until the instruction causing the conflict has passed through the execution stage. Such an event is often called a **bubble.**
- In some architectures, the bubble is implemented by feeding NOP ("no operation") instructions in between the instruction causing the hazards.

How Hazards are Resolved?

Instruction 1: add R3, R1, R2

NOP

Instruction 2: add R5, R3, R4

But, a stall causes the pipeline performance to degrade.

Instruction 1: add R3, R1, R2

sub R7, R6, R8

Instruction 2: add R5, R3, R4

Programmers could perform instruction reordering, but it is not a solution that we could depend on.

Operand Forwarding (Bypassing)

- The processor is updated with special hardware- Bypass unit - which checks if the result is in the buffer then select to use that result instead of reading from register file.
- This is based on the assumption that instruction 1 EX stage produces a result at the first half of clock cycle and instruction 2 Operand Fetch or MEM at the second half.
- It is used with RAW data hazards.

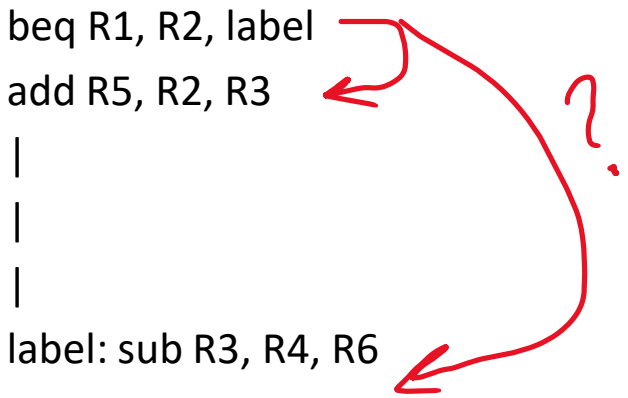
Instruction 1: add R3, R1, R2

Instruction 2: add R5, R3, R4

	1	2	3	4	5	6
Instruction1	IF1	ID1	MEM1	EX1	WB1	
Instruction2		IF2	ID2	MEM2	EX2	WB2

Branch prediction

- It is used with control hazards.
- The idea depends on using history of branch to predict the current branch (taken/not taken). The history is stored in "branch-target buffer“.



	1	2	3	4	5	6
Branch instruction	IF1	ID1	MEM1	EX1	WB1	
Next predicted instruction		IF2	ID2	MEM2	EX2	WB2

Thank You