Pipelining + Parallelism

Topics to Cover

- ➢‘Organizational techniques’ to Improve Processor Speed
- Pipelining
- Superscalar
- Super-pipeline

- **Parallelism**
- Instruction-level parallelism (ILP)
- ➢Pipelining
- ➢Superscalar
- Machine-level parallelism
- ➢Multicore systems
- ➢Cluster Computers
- ❖Flynn’s Taxonomy of Computers

Pipelining

- It divides the instruction processing into a series of stages.
- Each stage performs a specific task, and
- Multiple instruction can be in various stages of execution simultaneously.

1. Multi-Stage Pipeline

Instruction Format

Opcode	Address

- **Instruction pipelining** is an organizational approach, to improve the processor performance.
- As in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as output at the other end.
- Each step in the **instruction cycle** (fetch -> decode -> execute) takes at least one tick of the system clock, called a clock cycle.
- But this does not mean that the processor must wait until all steps are completed before beginning to process the next instruction.
- The processor can execute the steps in parallel, a technique known as **pipelining**. (e.g. overlapping of instruction processing steps)

Six-Stages of an Instruction

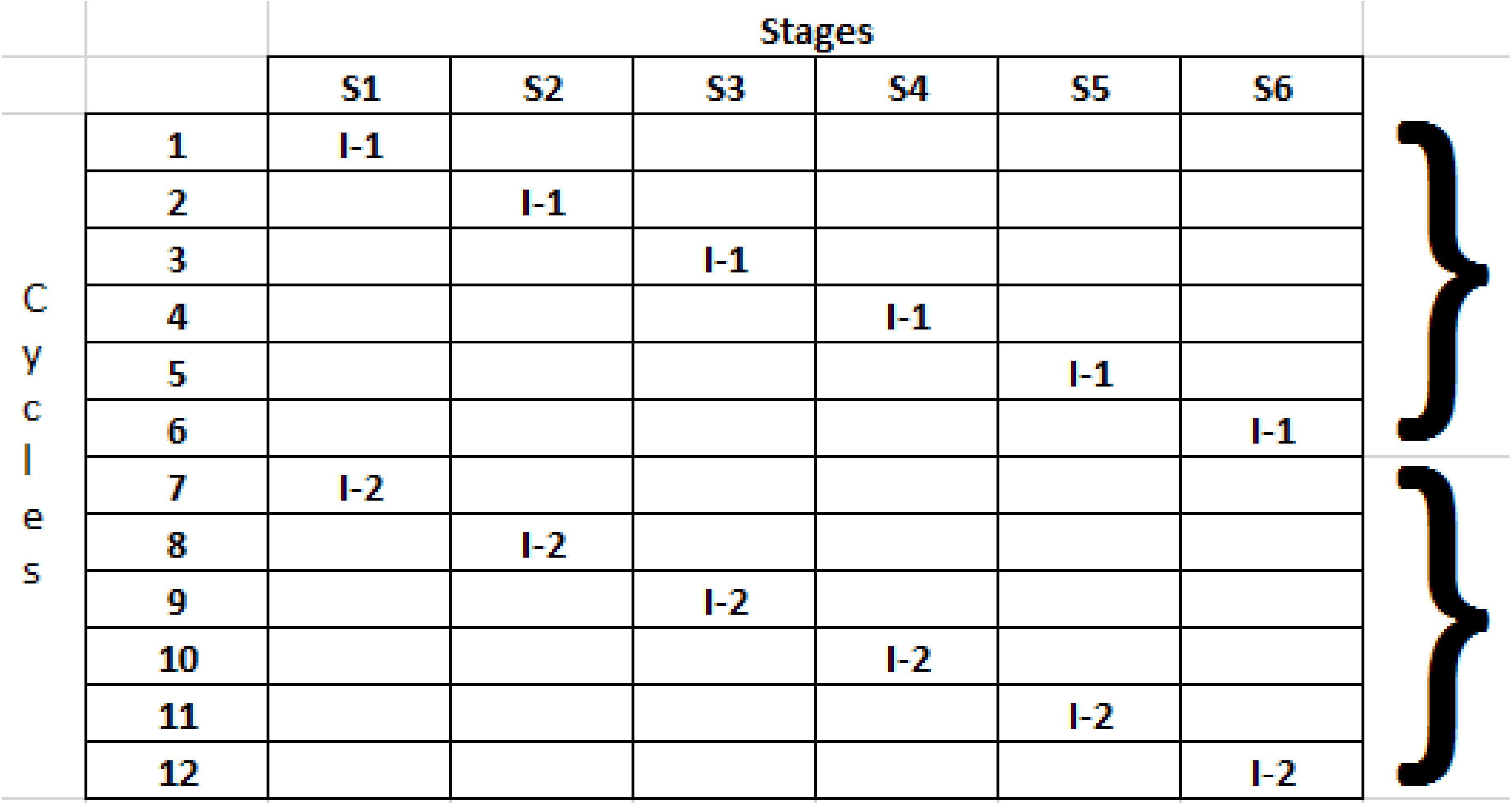
- The six-stages of an instruction are listed below:

1. **Fetch instruction (FI):** Read the next expected instruction into a buffer. As told by PC.
2. **Decode instruction (DI):** Determine the opcode and the operand.
3. **Calculate operands (CO):** Calculate the effective address of each source operand. This may involve address calculation.
4. **Fetch operand (FO):** Fetch each operand from memory to register.
5. **Execute instruction (EI):** Perform the indicated operation/result.
6. **Write operand (WO):** Store the result in memory.

Non-Pipelined Instruction Execution (Fig. Next)

- Let’s assume that each execution stage in the processor requires a single clock cycle.
- Figure uses a grid to represent a six-stage *non-pipelined* processor.
- When instruction I-1 has finished stage S6, instruction I-2 begins.
- Twelve clock cycles are required to execute the two instructions.
- In other words, for **k** execution stages, **n** instructions require **(n*k)** cycles to process.
- Of course, it represents a major waste of CPU resources because each stage is used only one-sixth of the time.

6-Stage Non-Pipelined Instruction Execution



		Stages					
		S1	S2	S3	S4	S5	S6
Cycles	1	I-1					
	2		I-1				
	3			I-1			
	4				I-1		
	5					I-1	
	6						I-1
	7	I-2					
	8		I-2				
	9			I-2			
	10				I-2		
	11					I-2	
	12						I-2

1st instruction

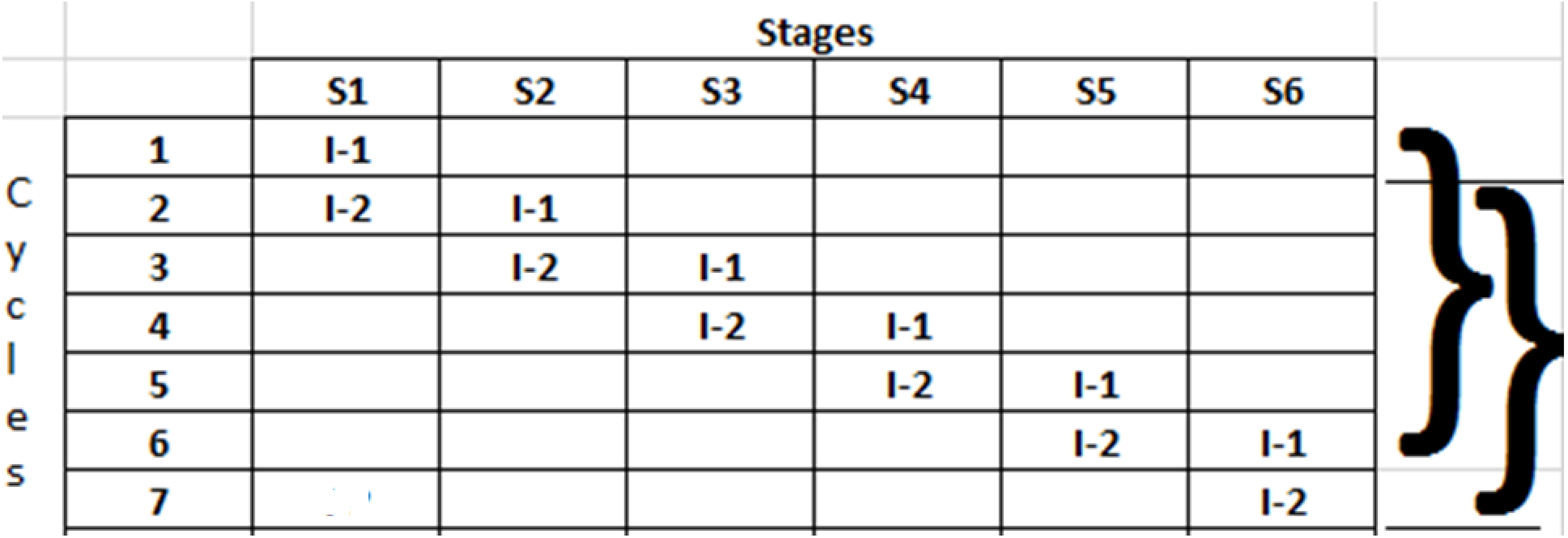
2nd instruction

(n*k) Cycles => 12 cycles

Pipelined Execution (Fig. Next Slide)

- If, on the other hand, a processor supports pipelining, a new instruction can enter stage S1 during the second clock cycle.
- Meanwhile, the first instruction has entered stage S2.
- This enables the overlapped execution of the two instructions.
- In Figure, two instructions I-1 and I-2, are shown progressing through the pipeline.
- I-2 enters stage S1 as soon as I-1 has moved to stage S2.
- As a result, only seven clock cycles are required to execute I-1 & I-2.
- When the pipelining is full, all six stages are in use all the time.
- In general, for **k** execution stages, **n** instructions require **k+(n-1)** cycles.

6-Stage Pipelined Instruction Execution



	Stages					
Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3		I-2	I-1			
4			I-2	I-1		
5				I-2	I-1	
6					I-2	I-1
7						I-2

New Instruction executed Per cycle.

k + (n-1) cycles => 7 cycles

Q. In a six-stage pipelined processor, how many instructions can be executed in 12 clock cycles? Ans: 7.

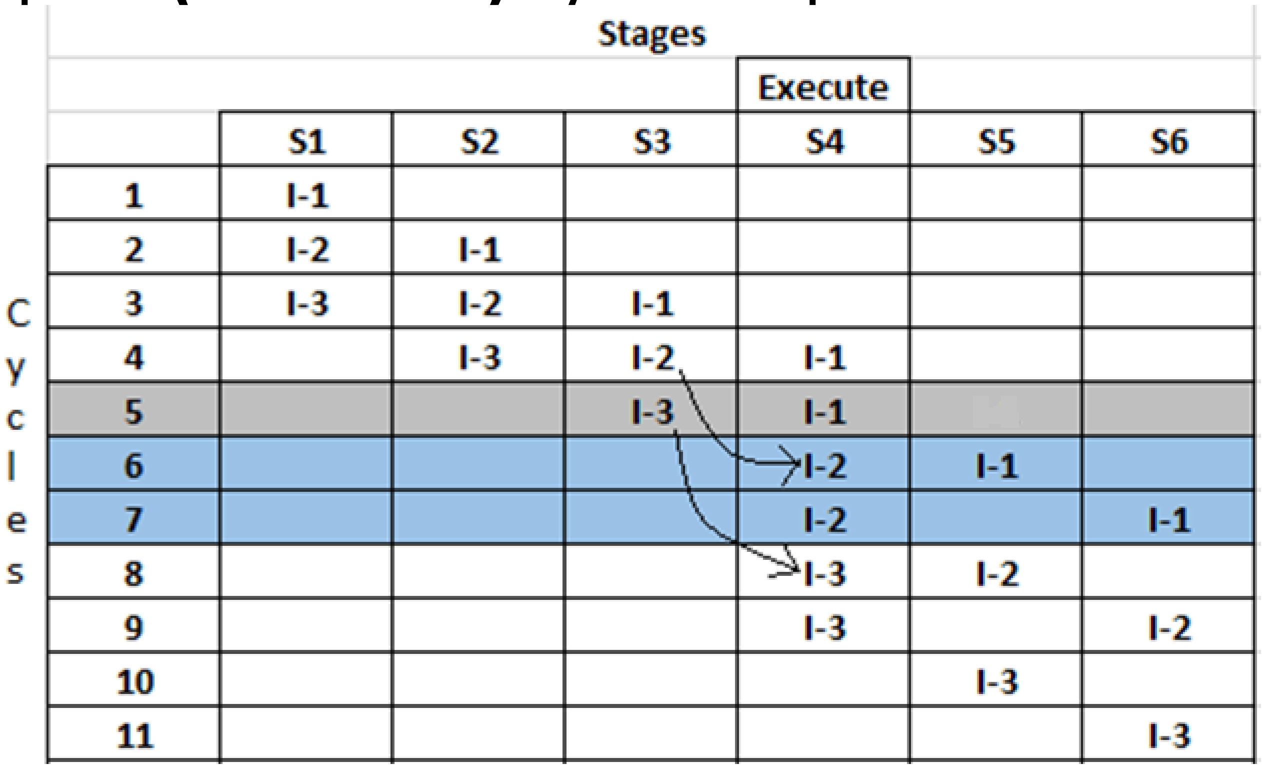
2. Superscalar Architecture (Fig. Next Slide)

- A **superscalar** processor has two or more execution pipelines, making it possible for two instructions to be in the execution stage at the same time. For n-pipelines, n-instructions can execute during the same clock cycle.
- In the previous pipeline example, we assumed that the ‘instruction execution’ stage (S4) required a single clock cycle.
- That was an overly simplistic approach.
- What would happen if stage S4 required two clock cycles?
- Then a bottleneck would occur, as shown in Figure next slide.
- Instruction I-2 cannot enter stage S4 until I-1 has completed the stage, so I-2 has to wait one more cycle before entering stage S4.

Q. Define a way of increasing the efficiency of the pipeline. Ans: Superscalar approach.

Without Super-Scalar Pipelining

- As more instructions enter the pipeline, wasted cycles occur (shaded in grey & blue).
- In general, for **k** stages (where one stage requires 2 execute cycles), **n** instruction require **(k + 2n – 1)** cycles to process.



Cycles	Stages					
				Execute		
	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3	I-3	I-2	I-1			
4		I-3	I-2	I-1		
5			I-3	I-1		
6				I-2	I-1	
7				I-2		I-1
8				I-3	I-2	
9				I-3		I-2
10					I-3	
11						I-3

One cycle Wait for I-2

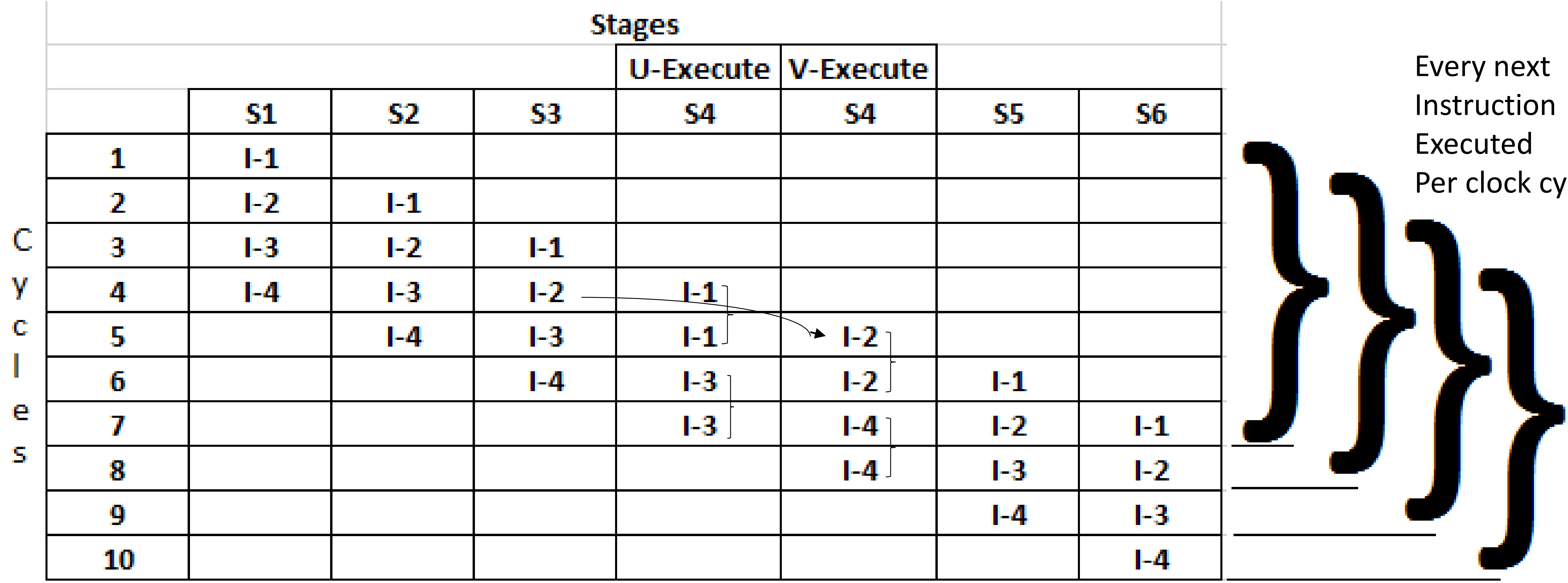
Two cycles Wait for I-3

k+(2n-1) Cycles
=> 11 cycles

With Super-Scalar Pipelining (Fig. Next Slide)

- When a superscalar processor design is used, multiple instructions can be in the execution stage at the same time.
- For n-pipelines, n-instructions can execute during the same clock cycle.
- Let us introduce a second pipeline (superscalar) into our 6-staged pipeline and assume that execution stage S4 requires two clock cycles.
- In Figure, odd-numbered instructions enter the *u-pipeline* and even-numbered instructions enter the *v-pipeline*.
- This removes the wasted cycles, and it is now possible to process **n** instructions in **(k + n)** cycles.

Two Pipelined Stages (Superscalar)



Cycles

Stages							
				U-Execute	V-Execute		
	S1	S2	S3	S4	S4	S5	S6
1	I-1						
2	I-2	I-1					
3	I-3	I-2	I-1				
4	I-4	I-3	I-2	I-1			
5		I-4	I-3	I-1	I-2		
6			I-4	I-3	I-2	I-1	
7				I-3	I-4	I-2	I-1
8					I-4	I-3	I-2
9						I-4	I-3
10							I-4

Every next
Instruction
Executed
Per clock cycle

(k + n) Cycles = > 10 cycles

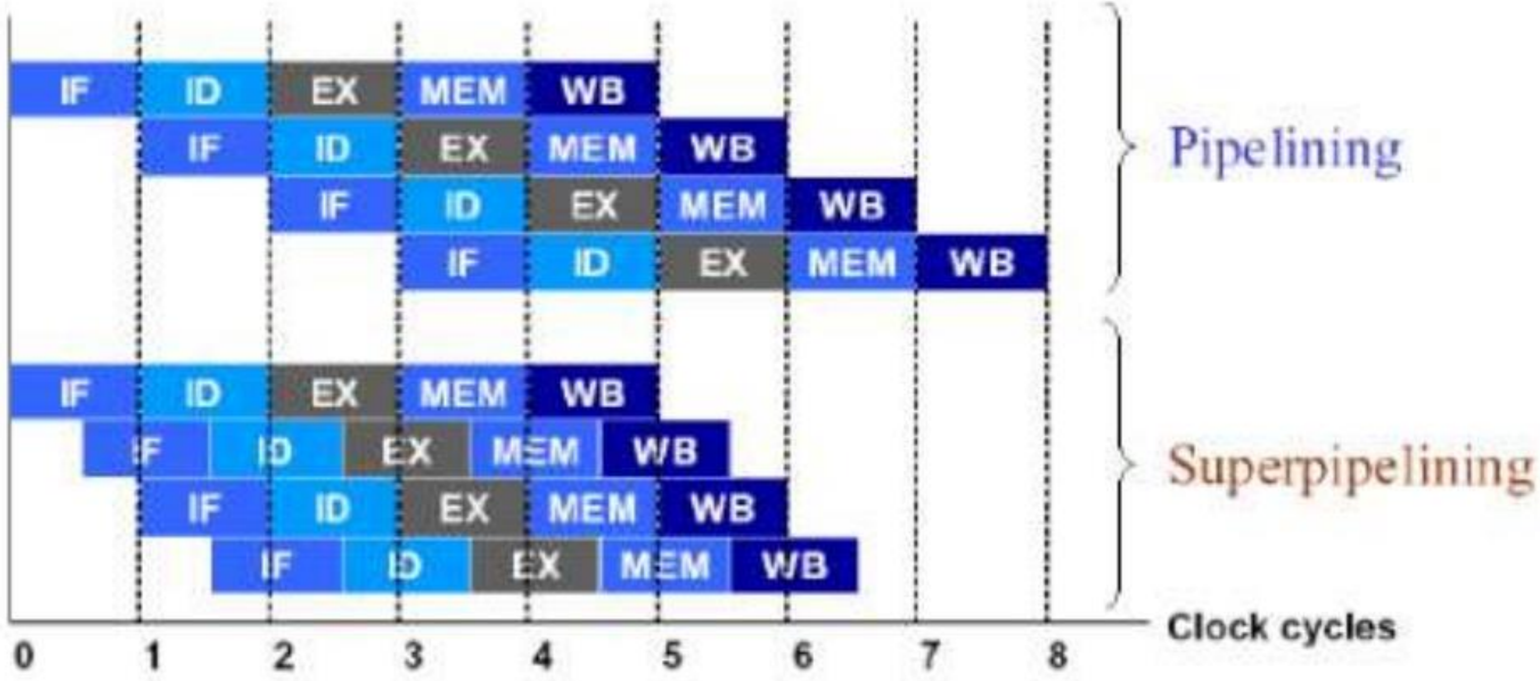
3. Super-Pipeline

- In a Super-pipeline, many pipeline stages need less than half a clock cycle.

- Super-pipeline is the breaking of stages of a given pipeline into smaller stages. (thus making the pipeline deeper) in an attempt to shorten the clock period and thus enhancing the instruction throughput by keeping more and more instructions in flight at a time.

Super-pipeline Performance

- The performance is shown below in the figure:

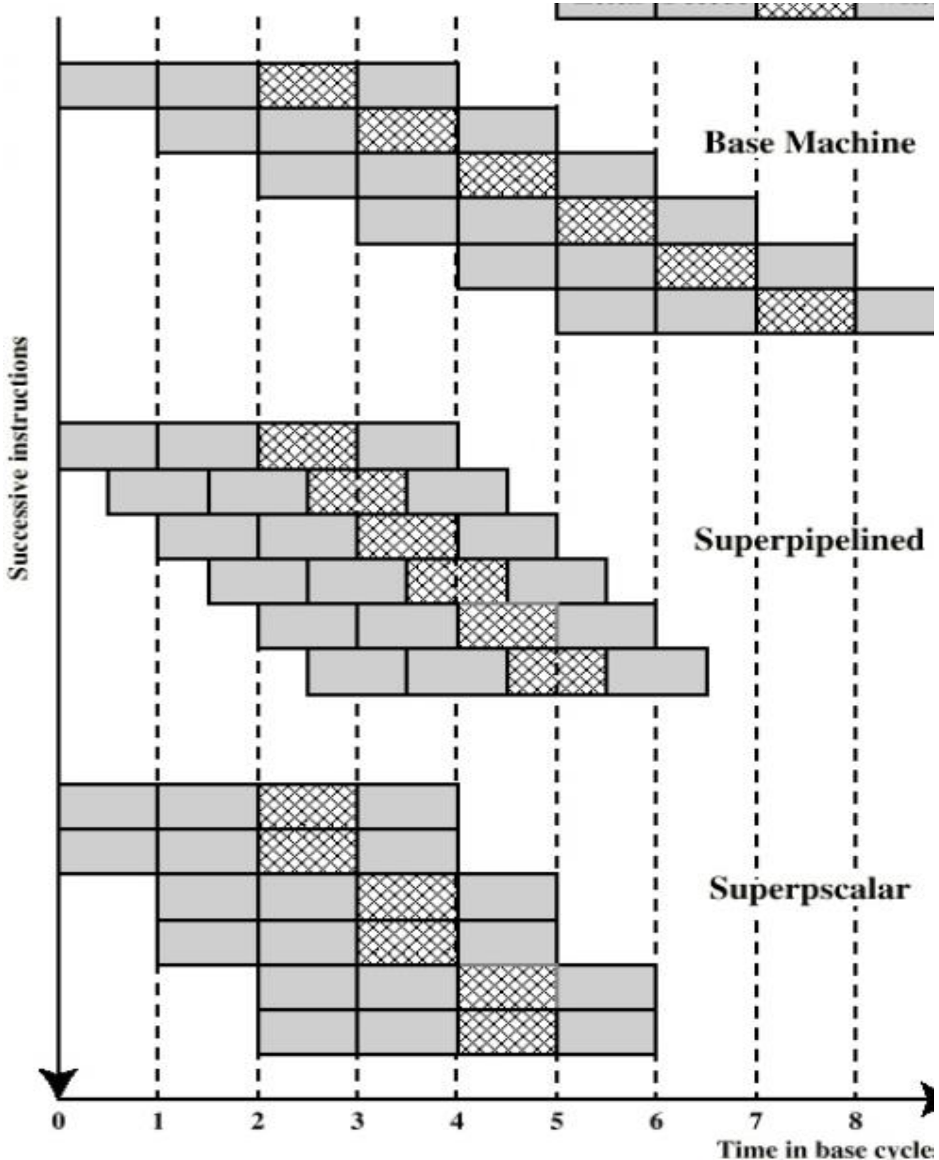


‘Super-Scalar’ VS ‘Super-Pipeline’

- Simple pipeline system performs only one pipeline stage per clock cycle.

- Super-pipeline system is capable of performing two pipeline stages per clock cycle.

- Super-scalar performs only one pipeline stage per clock stage in each parallel pipeline.



Pipeline Hazards/Problems

- Limits to Pipelining, Hazards prevent next instruction from executing during its designated clock cycles.

1. Structural hazards: Hardware cannot support some combination of instructions (single processor will limit to machine level parallelism).
2. Control hazards: Pipelining of branches causes later instruction fetches to wait for the result of the branch. (limited ILP)
3. Data hazards: Instruction depends on result of prior instruction still in the pipeline (data dependency).

➢ These might result in pipeline ‘stalls’ or ‘bubbles’ in the pipeline.

Preparatory Questions (Pipelining)

Q1. What is the ‘instruction pipelining’? How can we pipeline instructions?

Q2. In a six-stage pipelined processor, how many instructions can be executed in 18 clock cycles?

Q3. What is a ‘superscalar’ pipeline? How does it improve processor performance?

Q4. What is a ‘superpipeline’? How does it differ from a normal pipeline?

Q5. What are the ‘hazards’ to pipelining?

Parallelism

- Executing two or more operations at the same time is known as **Parallelism.** It is used in ‘high-performance computing’.
- In ‘Parallel processing’ the computer does the simultaneous data processing tasks concurrently (at one time).
- **Goal of Parallelism**
- Parallelism is done to increase the ‘computational speed’ of a computer system.
- It increases the computer’s processing capability and increases its throughput, ‘the amount of processing during a given interval of time’.

Types of ‘Parallelism’

- Parallelism can be of two types:

1) Instruction Level Parallelism (ILP) (Parallelism in Software)

i. Pipelining
ii. Superscalar

For uni-processor

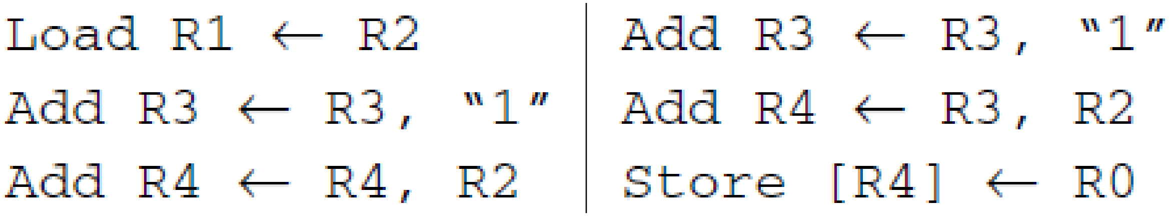
2) Machine Parallelism (Parallelism in Hardware)

i. Multi-core processors
ii. Multi-computers (Clusters)

For multi-processors

1) Instruction Level Parallelism (ILP)

- **Instruction-level parallelism** exists when instructions in a sequence are independent and thus can be executed in parallel by overlapping.
- As an example of the concept of ILP, consider the following two codes:



```
Load R1 ← R2          | Add R3 ← R3, “1”
Add R3 ← R3, “1”      | Add R4 ← R3, R2
Add R4 ← R4, R2       | Store [R4] ← R0
```

- The three instructions on the left are independent, and in theory all three can be executed in parallel.
- In contrast the three instruction on the right can not be executed in parallel because the second instruction uses the result of the first, and the third instruction uses the result of the second.

Instruction Level Parallelism (ILP)

- **Instruction-level parallelism (ILP):** is a measure of how many of the operations in a computer program can be performed simultaneously.
- Micro-architectural techniques that are used to exploit ILP include:

i. **Instruction pipelining:** where the execution of multiple instructions can be partially overlapped. (You have already studied this)

- In **Pipelining**, while an instruction is being executed in the ALU, the next instruction can be read from memory. (overlap fetch & execute)

ii. **Superscalar:** execution in which multiple execution units are used to execute multiple instructions in parallel.

ii. Superscalar Approach

- A parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- In a **Superscalar** computer, the system has redundant functional units.
- For example, the system may have two or more ALUs and be able to execute two or more instructions at the same time.
- ‘Parallel processing’ is established by distributing the data among the multiple functional units.
- As the amount of hardware increases with parallel processing, and with it, the cost of system increases.

Superscalar Processor with Multiple Functional Units

(Figure Next Slide)

- For example, the arithmetic, logic, and shift operations can be separated into three units.

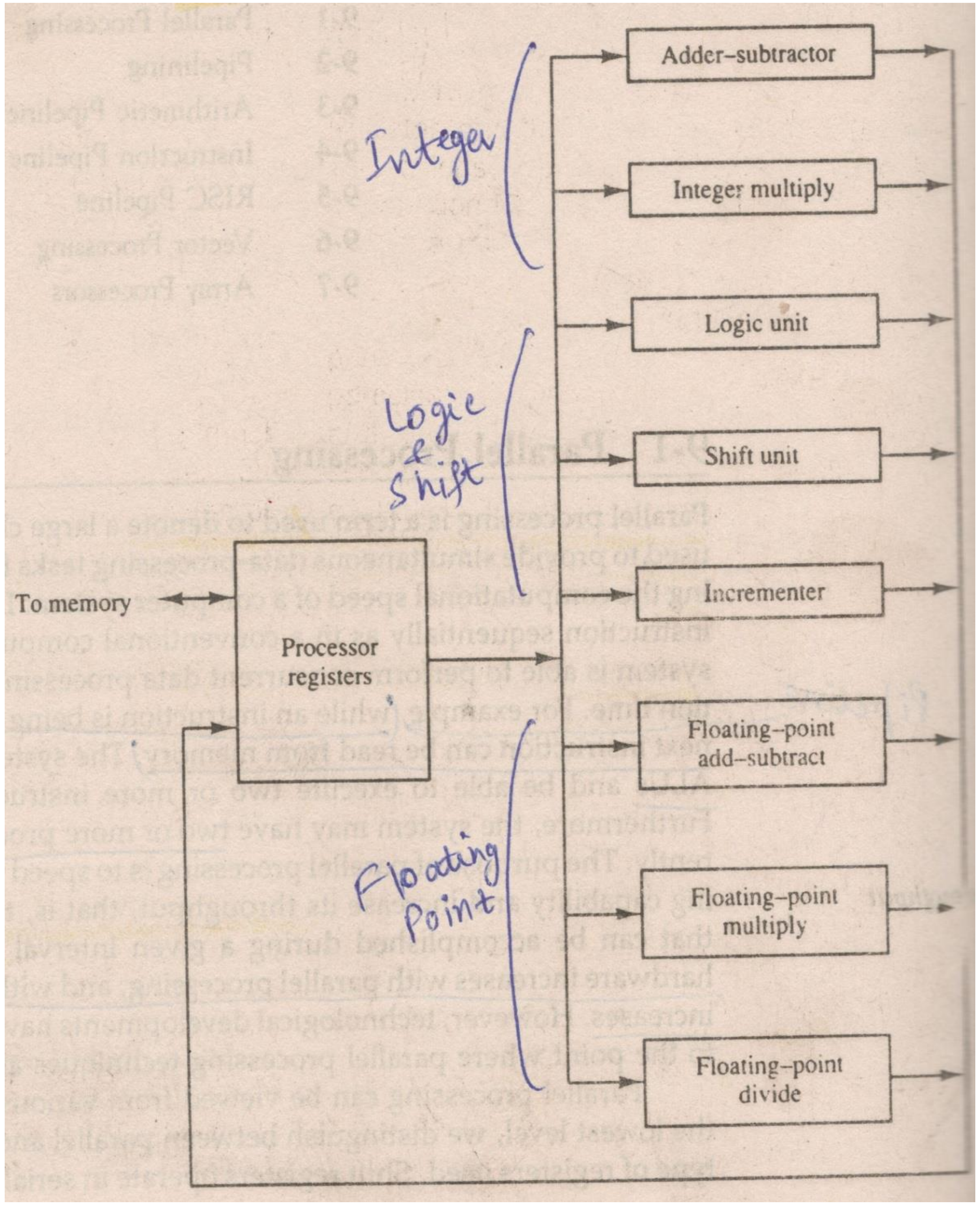
- All units are independence of each other, so one number can be shifted while another number is being incremented.

- The operands are diverted to each unit under the supervision of a complex ‘Control Unit’, which coordinates all the activities among the various components.

Multiple Functional Units

- Figure shows one possible way of separating the execution unit into eight functional units operating in parallel.

- The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands.



2. Machine Parallelism

- **Machine parallelism** is a measure of the ability of the processor to take advantage of instruction-level parallelism (ILP).

i. **Multi-core processors:** the system may have two or more processors operating concurrently. (You have already studied this)

- Such multi-core system will support ‘multi-threaded’ programs and ‘multi-tasking’.

ii. **Multi-computers (Clusters):** consist of multiple independent computers organized in a cooperative fashion. (e.g. networks)

- Clusters are ‘interconnected computers’ that can support workloads that are beyond the capacity of a single multiprocessor computer.

Final Note

- Both instruction-level and machine **parallelism** are important factors in enhancing performance.
- A program may not have enough instruction-level parallelism to take full advantage of machine parallelism. (e.g. not enough independent instructions, may not support multiple-threads).
- The use of a fixed-length instruction set architecture, as in a **RISC**, enhances instruction-level parallelism. (e.g. pipelining)
- On the other hand, limited machine parallelism will limit performance no matter what the nature of the program.

Types of Parallel Processor Systems

- The normal operation of a computer is to fetch instructions from memory and to execute them in the processor. (instruction cycle)
- The sequence of instructions read from memory constitutes an **instruction stream.**
- The operations performed on the data in the processor constitutes a **data stream.**
- Parallel processing may occur in the ‘instruction stream’, in the ‘data stream’ or both.
- ‘Flynn’ classified systems on the basis of these two streams in system.

Assignment 2

Flynn’s Taxonomy of Parallel Processor Systems

- On the basis of ‘instruction’ & ‘data streams’, Flynn classifies the organization of a computer system by the number of instruction and data items that are manipulated simultaneously.
- **Flynn’s classification:** divides computers into four major groups as:

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

1. Single Instruction, Single Data (SISD)

- In **SISD,** A single processor executes instructions sequentially from a single instruction stream, each instruction processes one data item stored in a single memory.

- Parallel processing in this case may be achieved by means of ‘multiple functional units’ or by ‘pipeline’ processing.

- Uniprocessor systems fall into this category.

2. Single Instruction, Multiple Data (SIMD)

- In **SIMD**, same instruction is processed in all processor (cores) with different data.
- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- All processors receive the same instruction from the control unit but operate on different items of data. (e.g GPU Graphics Processing Unit)
- The shared memory unit must contain different modules so that it can communicate with all processors simultaneously.
- Vector processors & array processors fall into this category.

3. Multiple Instructions, Single Data (MISD)

- In **MISD**, different instructions (processors) operate on the same data.

- This structure is only of theoretical interest and not commercially implemented. (since multiple processors on same data give same results)

- <u>Fault-tolerant systems</u> fall into this category. Such systems, must be able to continue working to a level of satisfaction in the presence of faults.

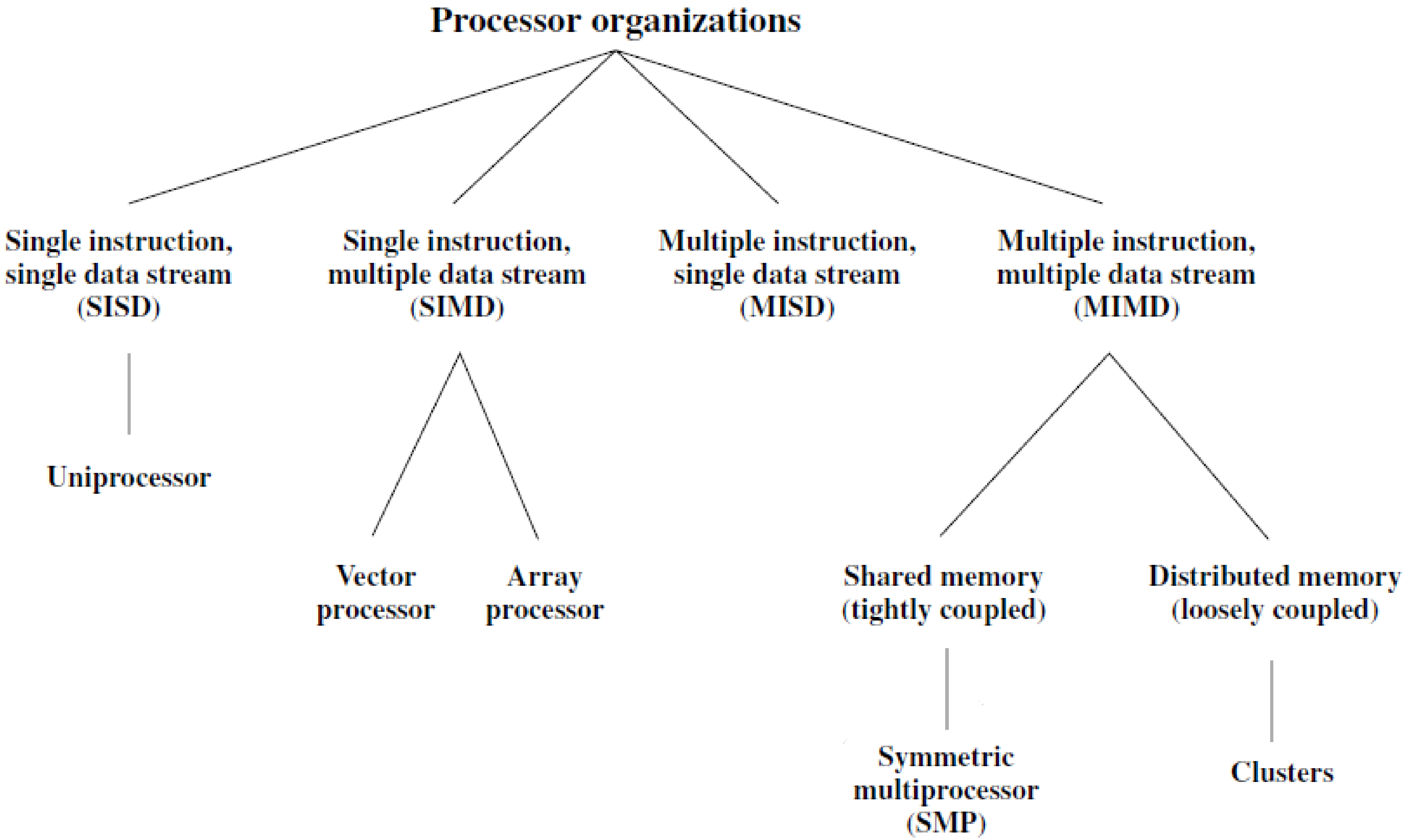
4. Multiple Instruction, Multiple Data (MIMD)

- In **MIMD**, a multiprocessor system is capable of processing (data streams) of several programs (instruction streams) at the same time.

- Each processor uses its own data and executes its own program.

- Multiprocessor systems and multi-computers (clusters) fall into this category.

Figure. A taxonomy of Parallel Processor Architecture



Processor organizations

Single instruction, single data stream (SISD)

Uniprocessor

Single instruction, multiple data stream (SIMD)

Vector processor

Array processor

Multiple instruction, single data stream (MISD)

Multiple instruction, multiple data stream (MIMD)

Shared memory (tightly coupled)

Symmetric multiprocessor (SMP)

Distributed memory (loosely coupled)

Clusters

Final Note

- **Flynn's classification** depends on the distinction between the performance of the 'control unit' and the 'data-processing unit'.

- It emphasizes the 'behavioural characteristics' of the computer system rather than its 'operational and structural interconnections'.

Preparatory Questions (Parallelism)

1. **What is ‘parallelism’? Describe its goal.**
2. **What are the two types of parallelism? Describe.**
3. **Describe ‘instruction-level parallelism’ and its types.**
4. **Describe ‘machine-level parallelism’ and its types.**
5. **Describe ‘Flynn’s taxonomy of parallel processor systems’ and its types.**