Parallel Processing

Nahin Ul Sadad
Lecturer
CSE, RUET

Parallel Processing

Parallel processing is a type of computation where many calculations or the execution of processes are carried out simultaneously.

Large problems can often be divided into smaller ones, which can then be solved at the same time.

There are several different forms of parallel processing: pipelining, instruction level parallelism, parallel processor architecture etc.

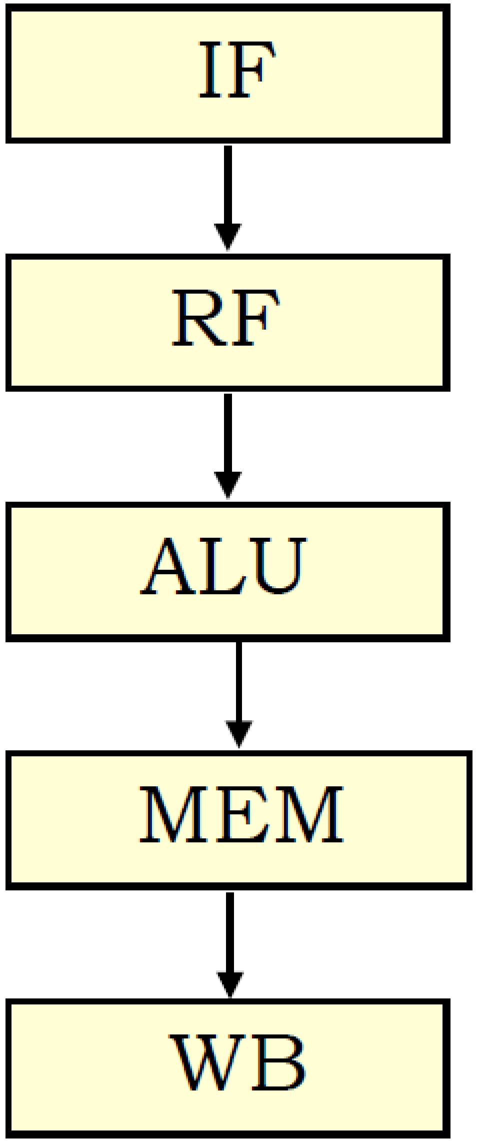
Pipelining

It is a process in which several storages of the CPU are used to execute more than one instruction concurrently.

We’ll study the classic 5-stage pipeline:

1. **Instruction Fetch (IF) stage:** Maintains PC, fetches instruction and passes it to RF.
2. **Register File (RF) stage:** Reads source operands from register file, passes them to ALU.
3. **ALU stage:** Performs indicated operation in ALU, passes result to MEM.
4. **Memory (MEM) stage:** If it’s a Load instruction, use ALU result as an address, pass mem data (or ALU result if not Load) to WB.
5. **Write-Back (WB) stage:** writes result back into register file.

Pipelining



Normal Execution of Instructions

Instructions	Clock Cycle																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I_1	IF	RF	ALU	MEM	WB															
I_2						IF	RF	ALU	MEM	WB										
I_3											IF	RF	ALU	MEM	WB					
I_4																IF	RF	ALU	MEM	WB

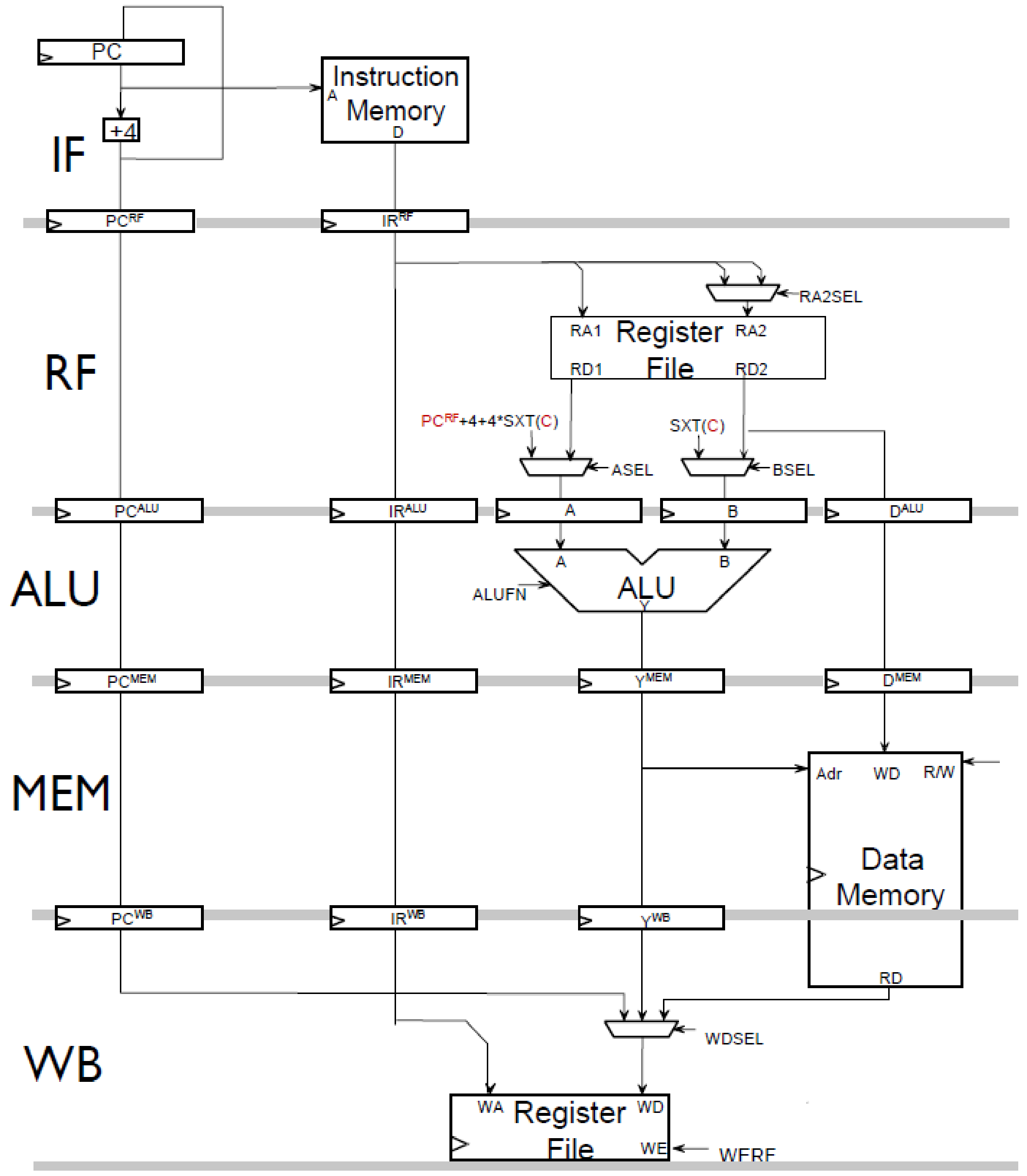
Normally for I_1 to I_4 instruction needs $4 \times 5 = 20$ clock cycle.

Pipelining

Instructions	Clock Cycle																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I_1	IF	RF	ALU	MEM	WB															
I_2		IF	RF	ALU	MEM	WB														
I_3			IF	RF	ALU	MEM	WB													
I_4				IF	RF	ALU	MEM	WB												

But with the use of pipeline, it is done with 8 clock cycle. Note that these instructions don’t have any dependency (Data/Control).

5-Stage Pipelined Datapath



1. Pipeline registers separate different stages:
   - IF – instruction fetch
   - RF – register file access
   - ALU – compute result
   - MEM – memory access
   - WB – write back to reg. file

2. Each stage services one instruction per cycle.

Pipeline Diagram

Pipeline diagram represents pipeline utilization over time.

Consider this instruction sequence:

```
LOAD R1, [4]
LOAD R2, [8]
SUB R6, R7
XOR R9, R10
MUL R12, R13
ADD R15, 1
```

Stages	**Cycles**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
IF	LD	LD	SUB	XOR	MUL	ADD				
RF		LD	LD	SUB	XOR	MUL	ADD			
ALU			LD	LD	SUB	XOR	MUL	ADD		
MEM				LD	LD	SUB	XOR	MUL	ADD	
WB					LD	LD	SUB	XOR	MUL	ADD

Pipeline Hazards

Pipelining tries to overlap the execution of multiple instructions, but an instruction may depend on something produced by an earlier instruction

1. A data value → Data hazard
2. The program counter → Control hazard (branches, jumps, exceptions)

Data Hazards

Data Hazards occur when an instruction depends on the result of previous instruction and that result of instruction has not yet been computed.

Consider this instruction sequence:

```
ADD R2, R1
SUB R2, R3
MUL R6, R7
XOR R9, R10
```

Stages	Cycles									
	1	2	3	4	5	6	7	8	9	10
IF	ADD	SUB	MUL	XOR						
RF		ADD	**SUB**	MUL	XOR					
ALU			ADD	SUB	MUL	XOR				
MEM				ADD	SUB	MUL	XOR			
WB					**ADD**	SUB	MUL	XOR		

SUB read R2 on cycle 3 but **ADD** does not update it until end of cycle 5.

Resolving Hazards

1. **Strategy 1:** Stall. Wait for the result to be available by freezing earlier pipeline stages.
2. **Strategy 2:** Bypass (aka Forward). Route data to the earlier pipeline stage as soon as it is calculated.
3. **Strategy 3:** Speculate
   - ➢ Guess a value and continue executing anyway
   - ➢ When actual value is available, two cases
     - ❑ Guessed correctly → do nothing
     - ❑ Guessed incorrectly →kill & restart with correct value

Resolving Data Hazards (Stall Strategy)

Wait for the result to be available by freezing earlier pipeline stages. This solution is simple but it wastes cycles.

```
ADD R2, R1
SUB R2, R3
MUL R6, R7
XOR R9, R10
```

Stages	Cycles										
	1	2	3	4	5	6	7	8	9	10	11
IF	ADD	SUB	MUL	MUL	MUL	MUL	XOR				
RF		ADD	SUB	SUB	SUB	SUB	MUL	XOR			
ALU			ADD	**NOP**	**NOP**	**NOP**	SUB	MUL	XOR		
MEM				ADD	**NOP**	**NOP**	**NOP**	SUB	MUL	XOR	
WB					ADD	**NOP**	**NOP**	**NOP**	SUB	MUL	XOR

Resolving Data Hazards (Bypass Strategy)

Route data to the earlier pipeline stage as soon as it is calculated. This solution is more expensive and requires more circuitry.

```
ADD R2, R1
SUB R2, R3
MUL R6, R7
XOR R9, R10
```

Stages	Cycles									
	1	2	3	4	5	6	7	8	9	10
IF	ADD	SUB	MUL	XOR						
RF		ADD	SUB	MUL	XOR					
ALU			ADD	SUB	MUL	XOR				
MEM				ADD	SUB	MUL	XOR			
WB					ADD	SUB	MUL	XOR		

Control Hazards

Control hazards occur when we don't know which instruction to execute next.

Consider this instruction sequence:

```
LOOP:
ADD R3, 2
MUL R6, R5
CMP R1, R2
JL LOOP
```

Stages	Cycles									
	1	2	3	4	5	6	7	8	9	10
IF	ADD	MUL	**CMP**	**JL**						
RF		ADD	MUL	**CMP**	**JL**					
ALU			ADD	MUL	**CMP**	**JL**				
MEM				ADD	MUL	**CMP**	**JL**			
WB					ADD	MUL	**CMP**	**JL**		

Calculation of jump address is done at **IF** stage for **JL** instruction and it depends on **CMP** instruction which will update FLAG register at the end of **WB** stage.

Control Hazards (Stall Strategy)

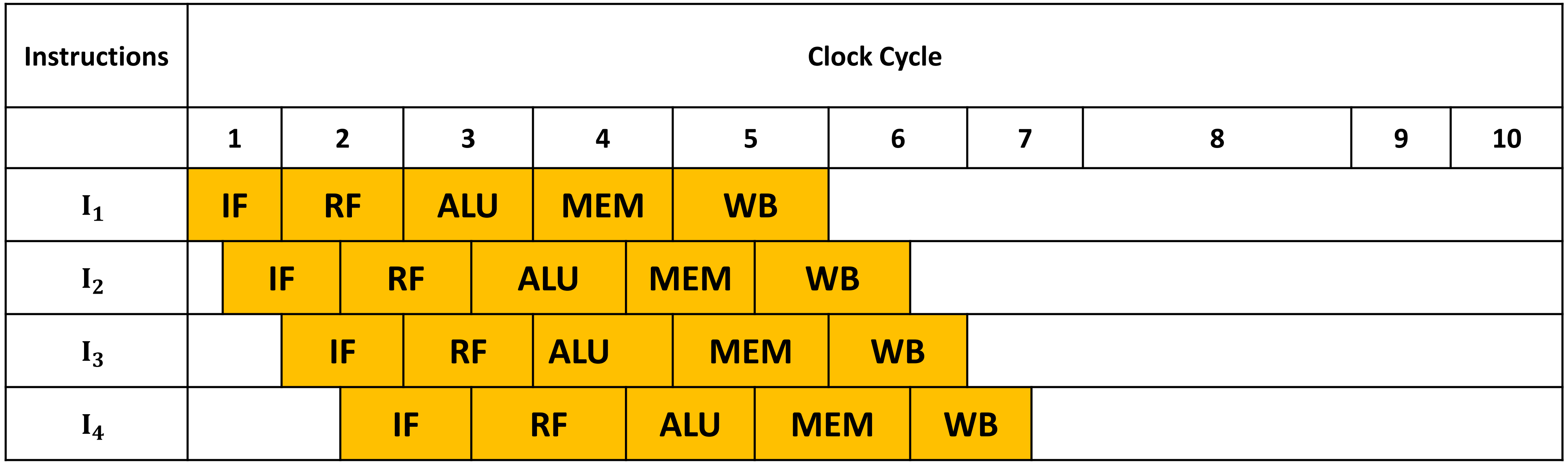
```
LOOP:
ADD R3, 2
MUL R6, R5
CMP R1, R2
JL LOOP
```

Stages	Cycles												
	1	2	3	4	5	6	7	8	9	10	11	12	13
IF	ADD	MUL	CMP	NOP	NOP	NOP	NOP	JL	ADD				
RF		ADD	MUL	CMP	NOP	NOP	NOP	NOP	JL	ADD			
ALU			ADD	MUL	CMP	NOP	NOP	NOP	NOP	JL	ADD		
MEM				ADD	MUL	CMP	NOP	NOP	NOP	NOP	JL	ADD	
WB					ADD	MUL	CMP	NOP	NOP	NOP	NOP	JL	ADD

JL instruction will be delayed until execution of **CMP** instruction which will update FLAG register at the end of **WB** stage. Then **JL** instruction will change PC address at **IF** stage.

Super Pipelining

Super Pipelining is an alternative approach to achieve better performance. Many pipeline stages perform task that requires less than half of a clock cycle, so a double interval clock speed allow the performance of two tasks in one clock cycle.



With the use of super pipelining, it is done with 6.5 clock cycle. Note that these instructions don’t have any dependency (Data/Control).

Scalar Pipelining

A Super Scalar process consists of multiple independent pipelines. Each pipeline consists of multiple stages, so that each one can handle multiple instructions at a time.

Instructions	Clock Cycle						
	1	2	3	4	5	6	7
I_1	IF	RF	ALU	MEM	WB		
I_2	IF	RF	ALU	MEM	WB		
I_3		IF	RF	ALU	MEM	WB	
I_4		IF	RF	ALU	MEM	WB	

With the use of scalar pipelining, it is done with 6 clock cycle. Note that these instructions don’t have any dependency (Data/Control).

Instruction Level Parallelism

The degree to which the instruction of a program can be executed parallels is called Instruction Level Parallelism (ILP).

For example, 4th degree ILP is shown below:

Instructions	Clock Cycle						
	1	2	3	4	5	6	7
I_1	IF	RF	ALU	MEM	WB		
I_2	IF	RF	ALU	MEM	WB		
I_3	IF	RF	ALU	MEM	WB		
I_4	IF	RF	ALU	MEM	WB		

With the use of scalar pipelining with 4th degree ILP, it is done with 5 clock cycle. Note that these instructions don't have any dependency (Data/Control).

Flynn’s Taxonomy

A taxonomy first introduced by Flynn in 1972 is still the most common way of categorizing systems with parallel processing capability. Flynn picked two characteristics that he considered essential:

1. The number of instruction streams and
2. The number of data streams.

Flynn proposed the following categories of computer systems:

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

Flynn's Taxonomy

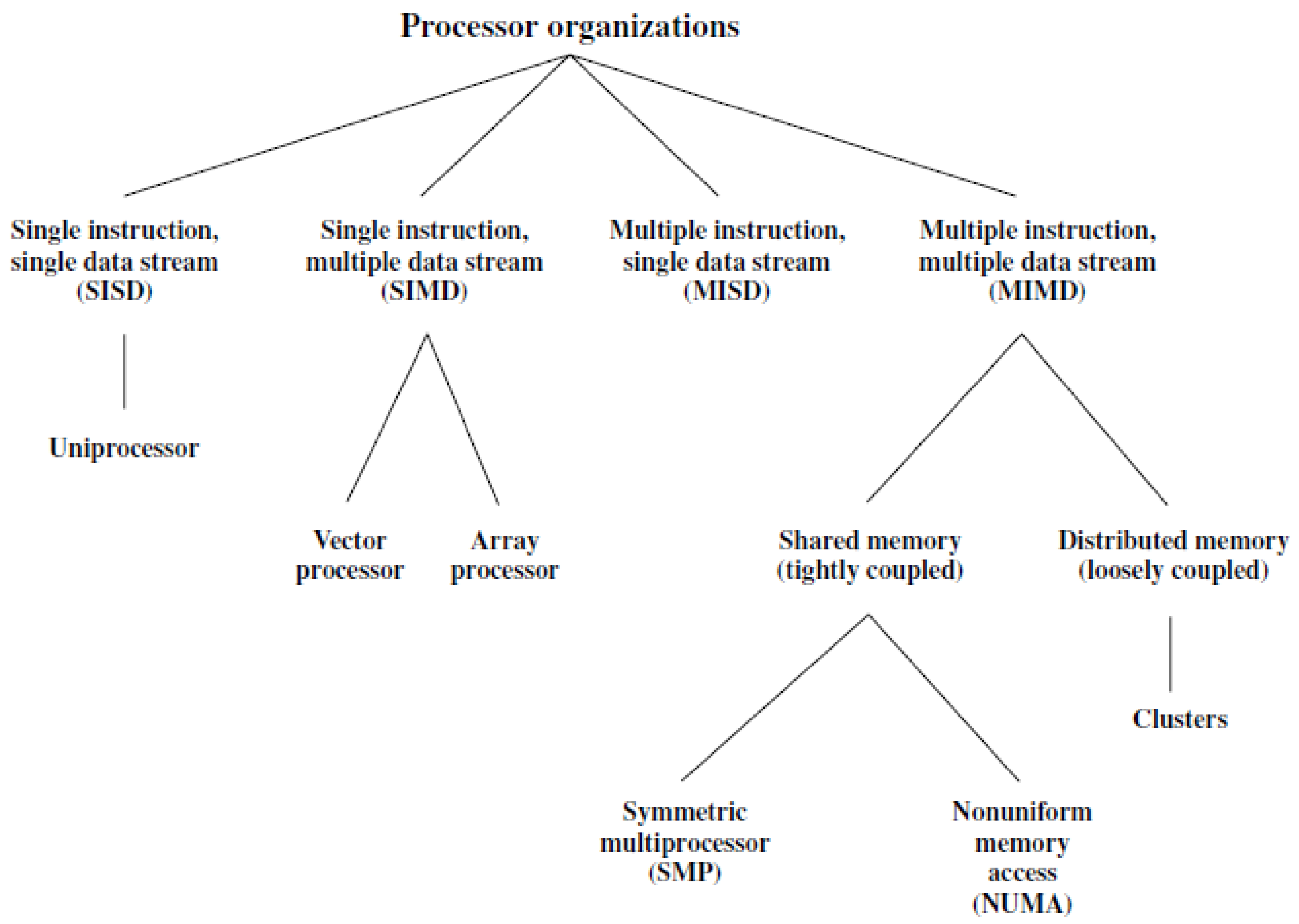

Processor organizations
Single instruction, single data stream (SISD)
Single instruction, multiple data stream (SIMD)
Multiple instruction, single data stream (MISD)
Multiple instruction, multiple data stream (MIMD)
Uniprocessor
Vector processor
Array processor
Shared memory (tightly coupled)
Distributed memory (loosely coupled)
Clusters
Symmetric multiprocessor (SMP)
Nonuniform memory access (NUMA)

Figure: Taxonomy of Parallel Processor Architectures.

Flynn's Taxonomy

Flynn proposed the following categories of computer systems:

1. **Single instruction, single data (SISD) stream:** A single processor executes a single instruction stream to operate on data stored in a single memory.

   For example, all traditional uniprocessor computers (having only one CPU) fall in this category from personal computers to large mainframes.

2. **Single instruction, multiple data (SIMD) stream:** A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that each instruction is executed on a different set of data by the different processors.

   For example, vector and array processors fall into this category where processors with one instruction unit that fetches an instruction and then commands many data units to carry it out in parallel, each with its own data.

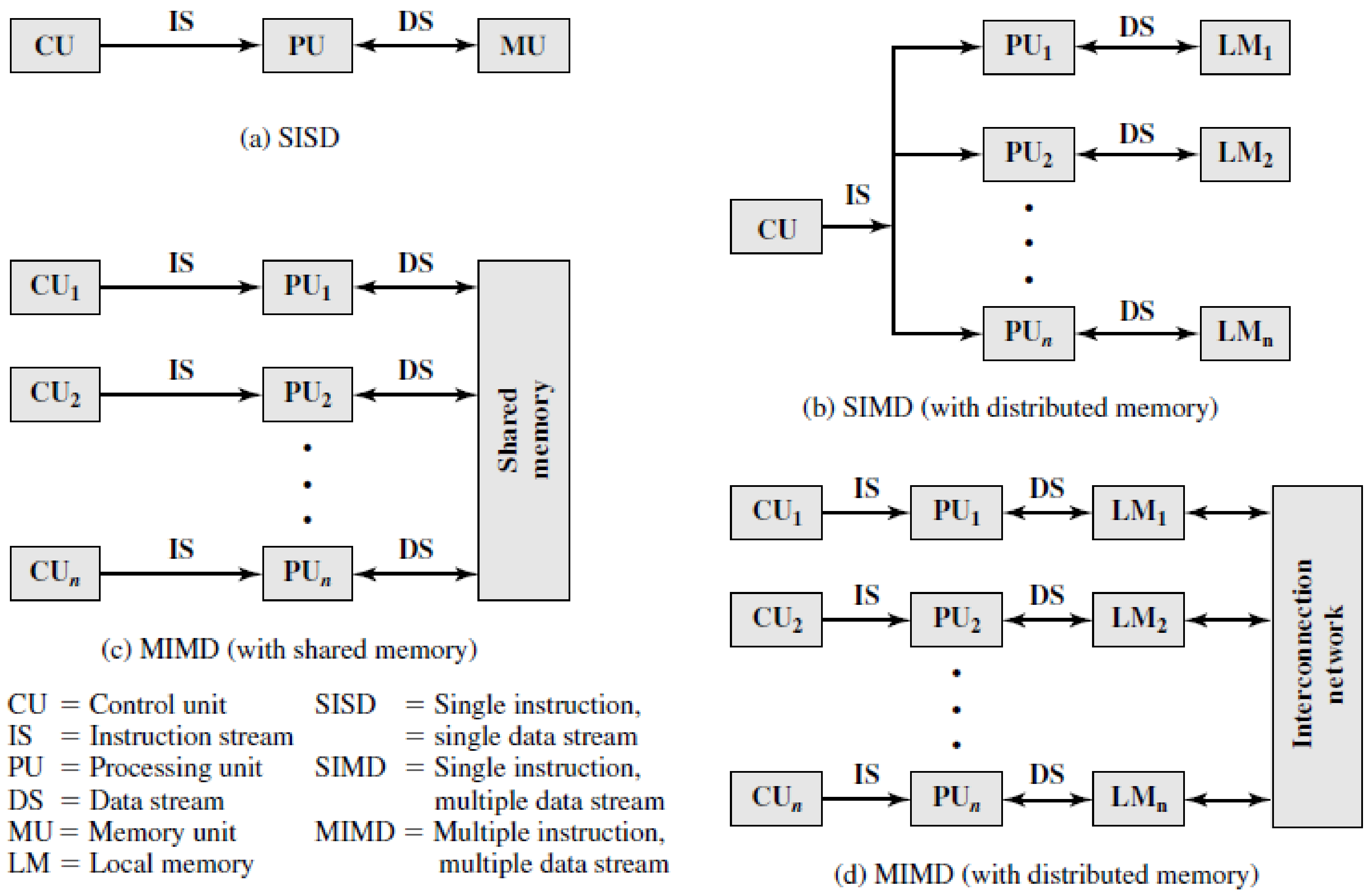
Flynn's Taxonomy

3. **Multiple instruction, single data (MISD) stream:** A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. This structure is not commercially implemented.

4. **Multiple instruction, multiple data (MIMD) stream:** A set of processors simultaneously execute different instruction sequences on different data sets.

   For example, SMPs, clusters and NUMA systems fit into this category. All distributed systems are also fall into this category.

Flynn’s Taxonomy



CU	= Control unit	SISD	= Single instruction,
IS	= Instruction stream		= single data stream
PU	= Processing unit	SIMD	= Single instruction,
DS	= Data stream		multiple data stream
MU	= Memory unit	MIMD	= Multiple instruction,
LM	= Local memory		multiple data stream

Figure 17.2 Alternative Computer Organizations

Thank you ☺