William Stallings Computer Organization and Architecture 8th Edition

Chapter 17

Parallel Processing

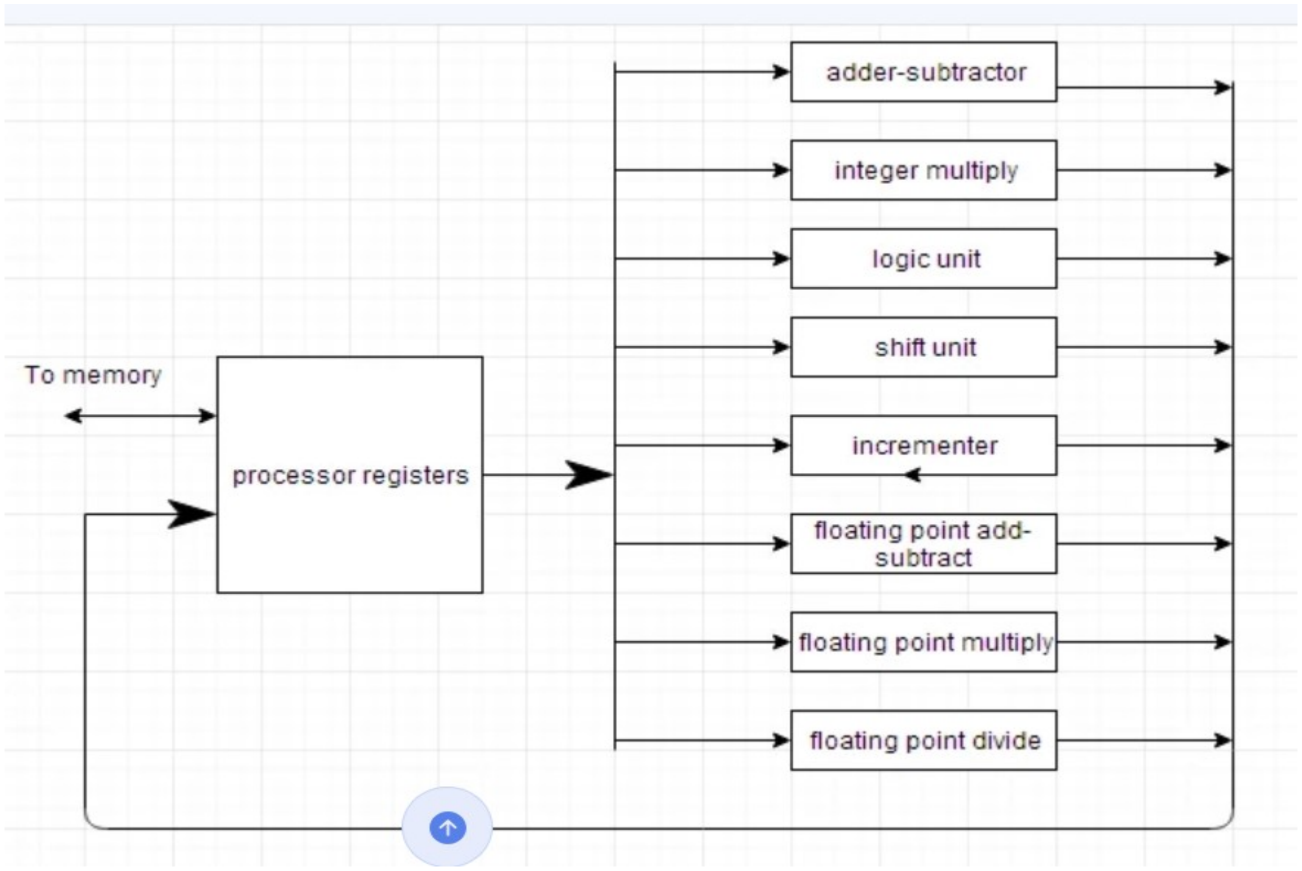
6.0	**Multiprocessor Configurations**		06	CO4
	6.1	Flynn’s classification, Parallel processing systems and concepts		
	6.2	Introduction to pipeline processing and pipeline hazards		
	6.3	Design issues of pipeline architecture, Instruction pipelining: Six Stage instruction pipeline.		
	6.4	8086 Instruction (Arithmetic Instructions, Logical Instructions, Data transfer instructions)		
	Self-Learning; Pin Diagram of 8086, Minimum Mode and Maximum mode with timing diagram			
		Total	**45**	

Multiprocessor Configurations

1. Flynn's classification,
2. Parallel processing concepts,
3. Introduction to pipeline processing and pipeline hazards,
4. design issues of pipeline architecture,
5. Instruction pipelining

Parallel Processing Concepts

- Executing multiple tasks concurrently
- To fulfil increasing demands for higher performance
- Need to process data concurrently to achieve better throughput
- capable of concurrent data processing to achieve faster execution times.
- Parallelism
  - Multiple functional units-several ALU’s
  - Multiple processors-several processors operating concurrently



All these units will work concurrently and produce the required output.

Flynn’s Classification-Types of Parallel Processor Systems

- Based on
  - Internal organization of processors
  - Interconnection Structures

Instruction Stream

Instruction from main memory to processor

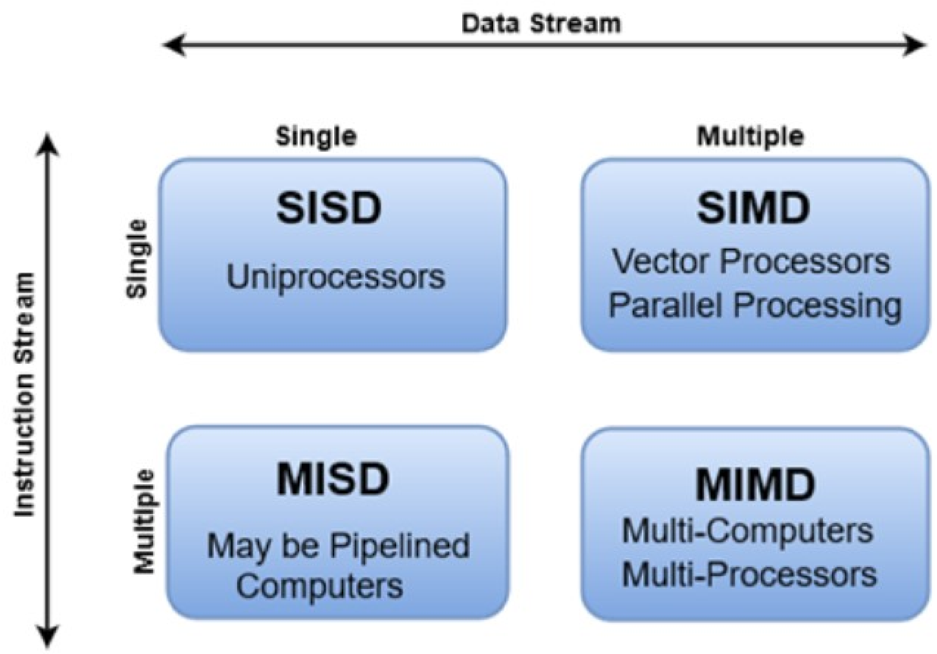
Data Stream

Operands flowing to and from the processor

FLYNN’s CLASSIFICATION Multiple Processor Organization

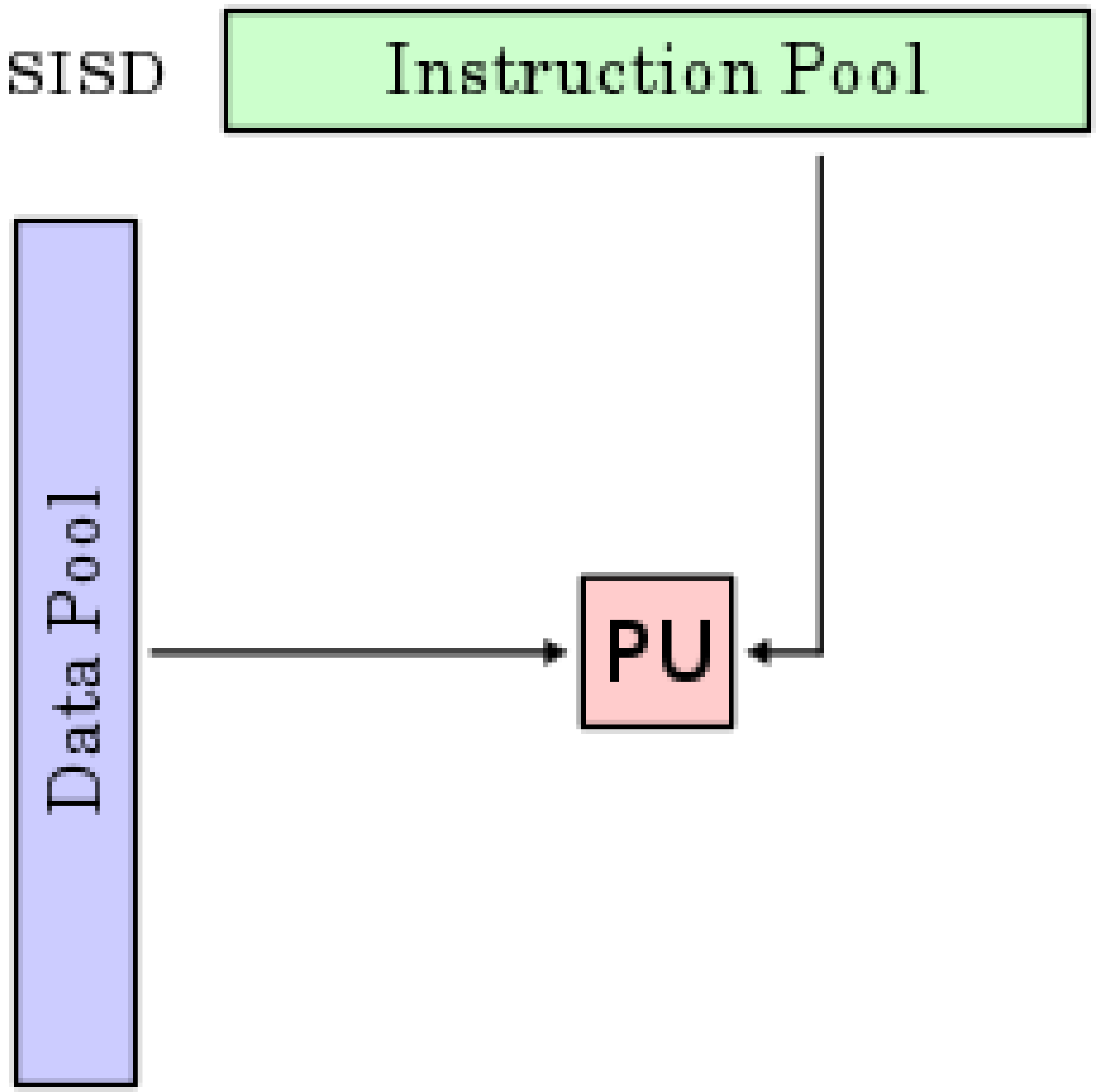
- Single instruction, single data stream - **SISD**
- Single instruction, multiple data stream - **SIMD**
- Multiple instruction, single data stream - **MISD**
- Multiple instruction, multiple data stream- **MIMD**

Flynn's Classification of Computers



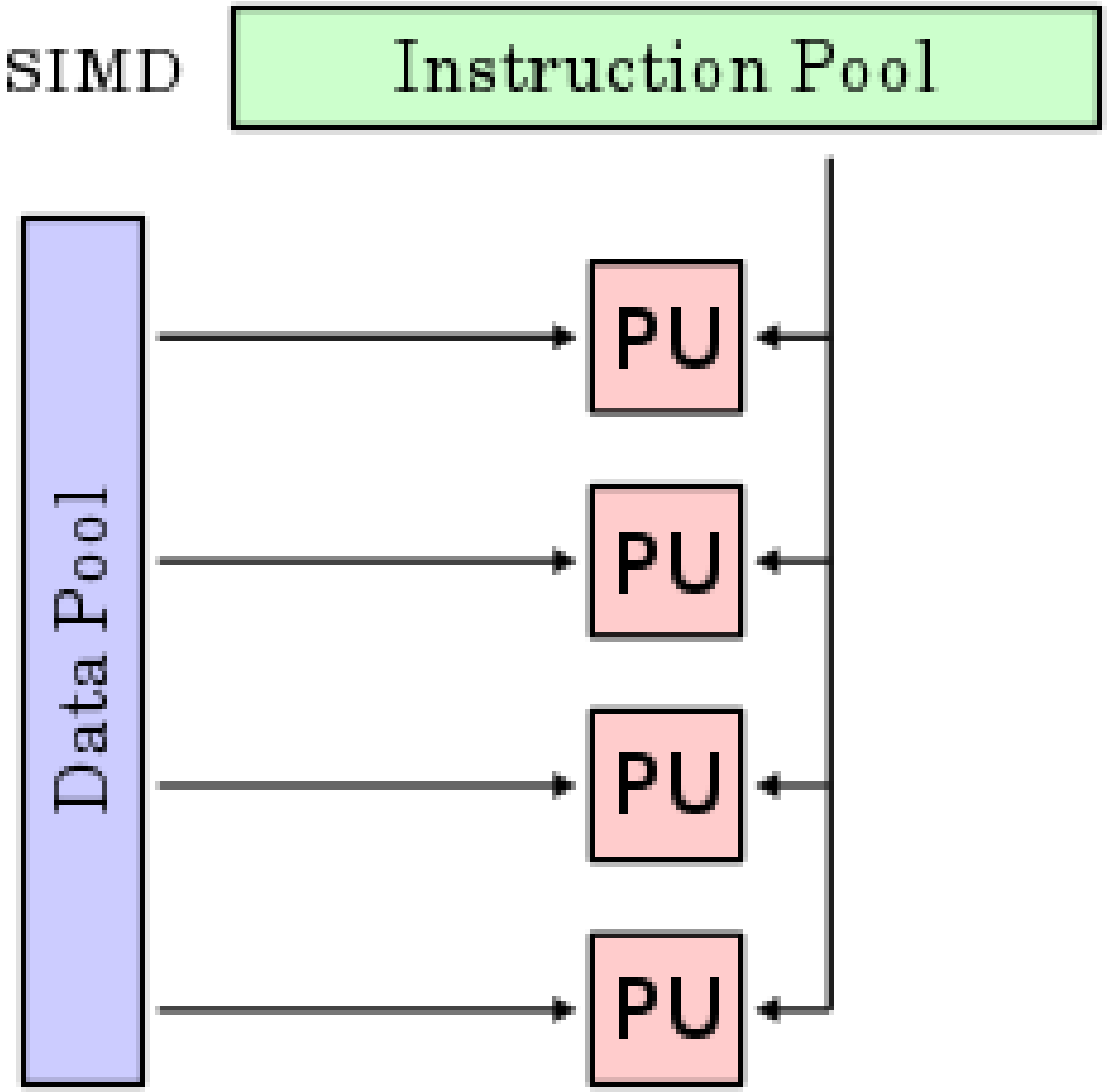
Single Instruction, Single Data Stream - SISD

- A single processor executes a single instruction stream to operate on data stored in a single memory
- Uni-processor
- *Each "PU" (processing unit) does not necessarily correspond to a processor, just some functional unit that can perform processing.*
- *The PU's are indicated as such to show relationship between instructions, data, and the processing of the data.*


SISD
Instruction Pool
Data Pool
PU

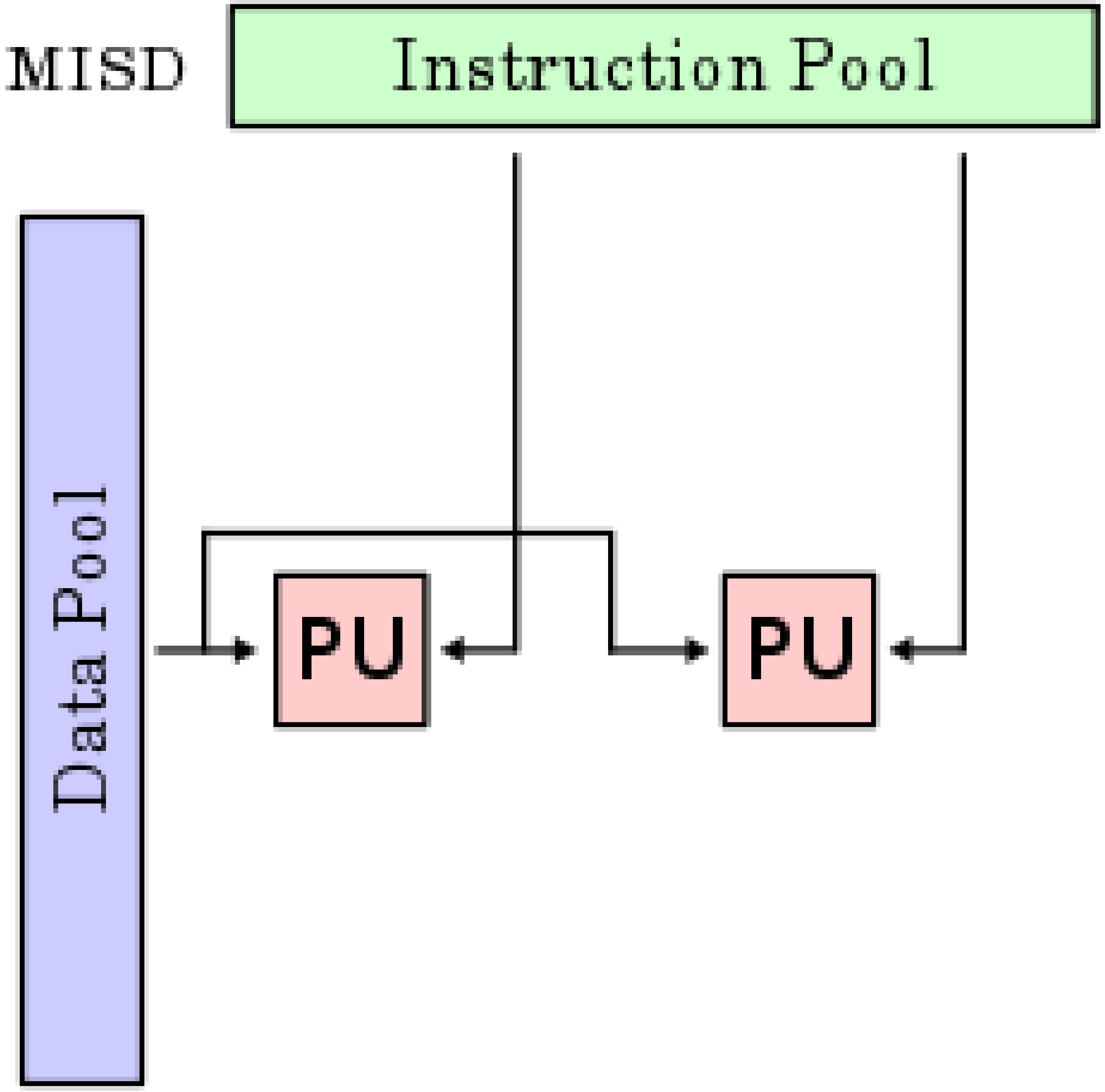
Single Instruction, Multiple Data Stream - SIMD

- A single machine instruction controls the simultaneous execution of a number of processing elements
- Each processing element has an associated data memory, so that each instruction is executed on a different set of data by the different processors.
- Vector and array processors


SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU

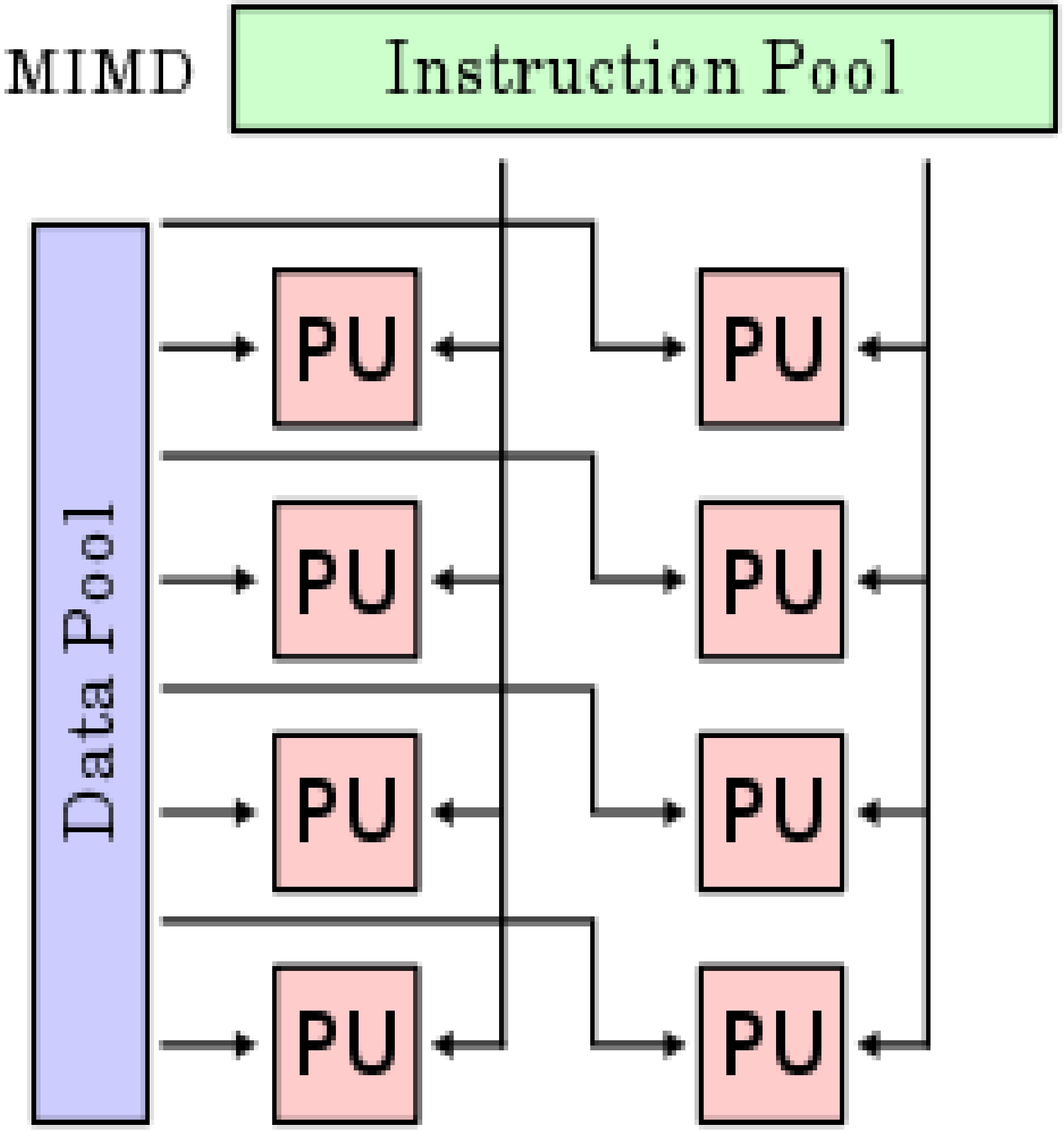
Multiple Instruction, Single Data Stream - MISD

- A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence.
- Each processing unit operates on the data independently via separate instruction stream.
- This structure is not commercially implemented.


MISD
Instruction Pool
Data Pool
PU
PU

Multiple Instruction, Multiple Data Stream- MIMD

- Set of processors
- Simultaneously execute different instruction sequences on different sets of data
- SMPs, NUMA systems
  - SMP-Symmetric multiprocessing--multiple identical processors are interconnected to a single shared main memory,
  - Non-Uniform Memory Access-configuring a cluster of microprocessors in a multiprocessing system so they can share memory locally.


MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU

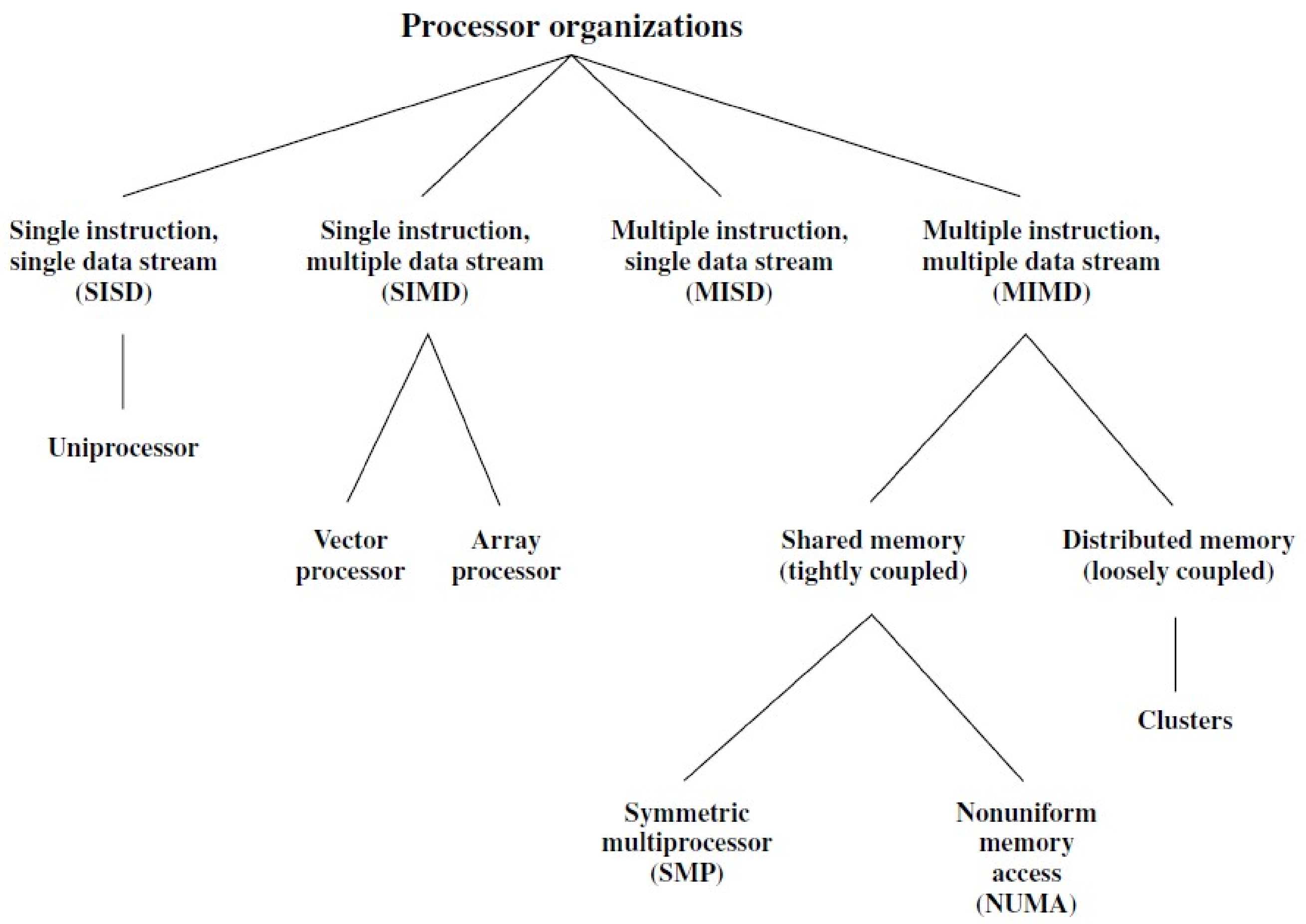
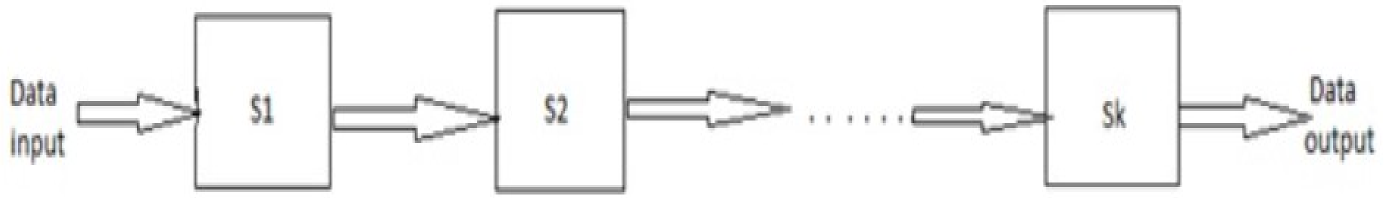

Processor organizations
Single instruction, single data stream (SISD)
Single instruction, multiple data stream (SIMD)
Multiple instruction, single data stream (MISD)
Multiple instruction, multiple data stream (MIMD)
Uniprocessor
Vector processor
Array processor
Shared memory (tightly coupled)
Distributed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

Figure 17.1 A Taxonomy of Parallel Processor Architectures

Introduction to pipeline processing and pipeline hazards

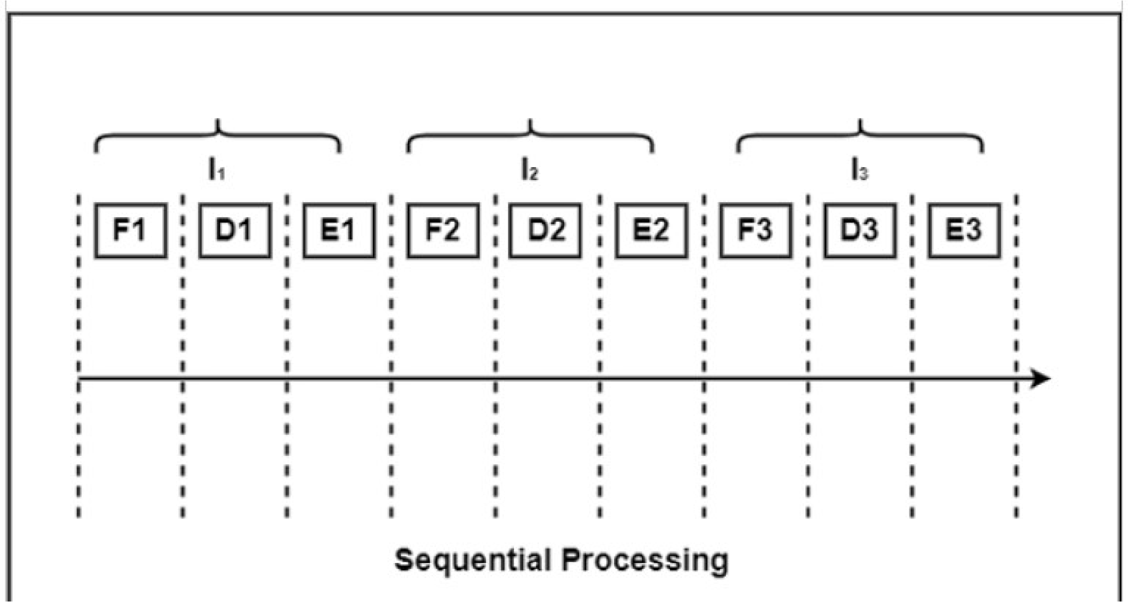
- Pipelining- Temporal overlapping of processing
- Subdivide input task(process) into a sequence of subtasks
- Each executed by specialized hardware that operates concurrently with other stages of pipeline
- Improvement of system throughput
- In Pipelining, specialized hardware stages are also known as **hardware segments**,.

- linearly connected to perform a fixed function over a stream of data flowing from one end to the other
- A processor supporting such a hardware architecture computation is known as **pipeline processor.**

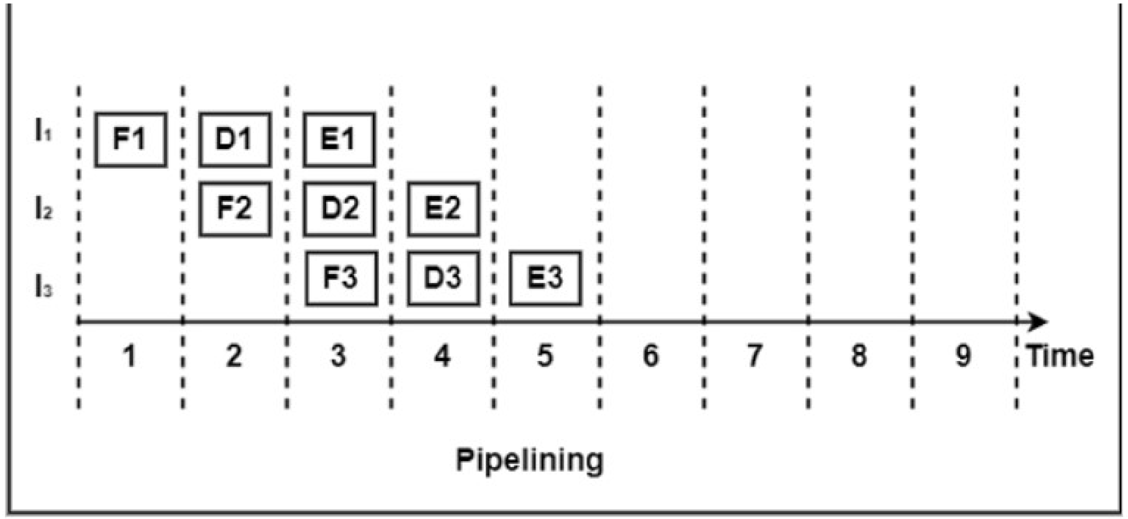


Basic Structure of PIPELINE PROCESSOR

Pipeline processors are constructed with k processing stages. Data inputs such as operands are fed into the pipeline at the first stage S1. The processed results are passed from stage Si to stage Si+1, for all i = 1, 2, 3… k-1



Sequential Processing



Pipelining

- **Pipelining** allows the next **instructions** to be fetched while the processor is performing arithmetic operations
- Holds them in a buffer close to the processor until each **instruction** operation can be performed.
- The staging of **instruction** fetching is continuous.
- each task is subdivided into **multiple successive subtasks**.
- A pipeline phase is defined for each subtask to execute its operations.
- The process continues until the processor has executed all the instructions and all subtasks are completed.
- https://www.techtarget.com/whatis/definition/pipelining#:~:text=With%20pipelining%2C%20the%20next%20instructions,for%20each%20instruction%20is%20performed.
- Merits & demerits

- There are two types of pipelining:

Arithmetic pipelining
Instruction pipelining

- Arithmetic pipelining-The integer arithmetic and floating point arithmetic) are performed by two separate units to introduce parallelism. These units perform scalar operations involving one pair of operands at a time.

- Instruction pipelining- subcycles of instruction are performed simultaneously to reduce overall processing time-instruction pipelining.

- As the instruction process is divided into four stages so it is also known as Four Stage Instruction Pipeline.

- Fetch: read the instruction from the memory.
- Decode: decode the opcode and fetch source operand if necessary.
- Execute: Perform the operation specified by the instruction.
- Store: store the result in the destination.

- four distinct hardware units are needed- capable of performing their tasks simultaneously and without interfering with one another. Information from one stage is passed to the next stage with the help of the buffers.

INSTRUCTION PIPELINING

- New inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.
- The first stage fetches an instruction and buffers it.
- When the second stage is free, the first stage passes it the buffered instruction.
- While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction- *Instruction prefetch* or *fetch overlap*

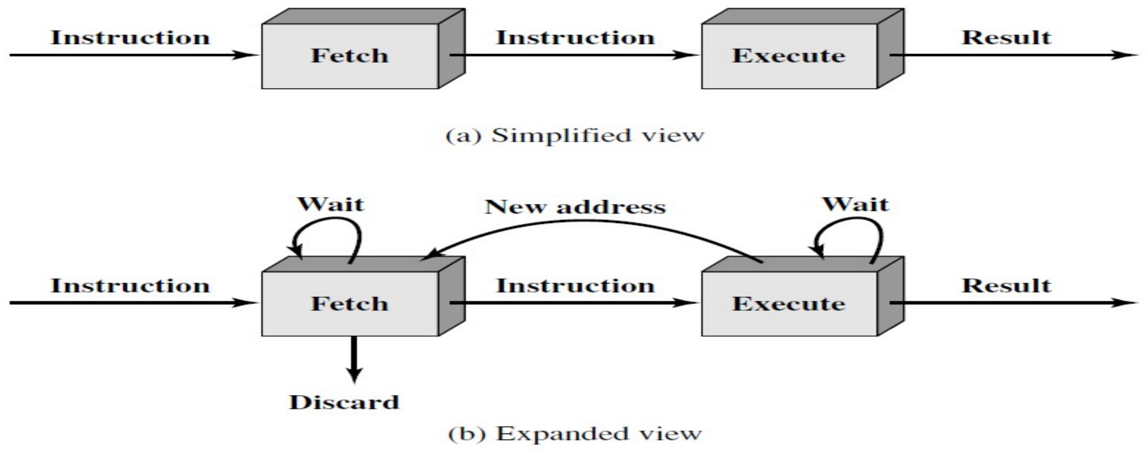


Figure 12.9 Two-Stage Instruction Pipeline

- This requires more registers to store data between stages
- If the fetch and execute stages were of equal duration, the instruction cycle time would be halved.

- Doubling of execution rate is due to

  - Execution will involve reading and storing operands and the performance of some operation (waiting for fetch **stage-more time for execution)**

  - A conditional branch instruction makes the address of the next instruction to be fetched unknown

  - To gain further speedup, the pipeline must have more stages

SIX STAGE OF INSTRUCTION PIPELINING

- **Fetch Instruction(FI)**

  Read the next expected instruction into a buffer
- **Decode Instruction(DI)**

  Determine the opcode and the operand specifiers.
- **Calculate Operands(CO)**

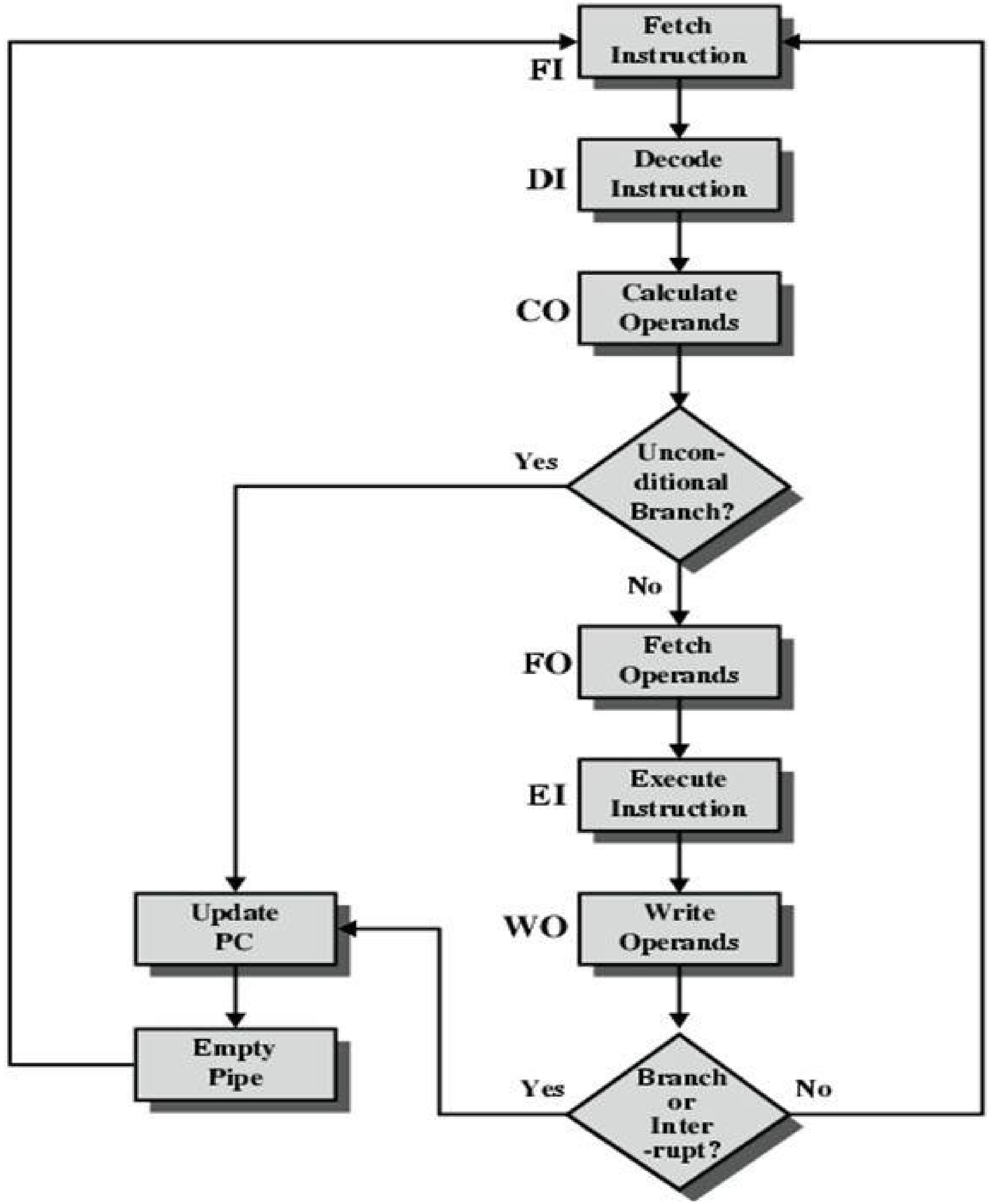
  Calculate the effective address of each source operanc
- **Fetch Operands(FO)**

  Fetch each operand from memory. Operands in regist need not be fetched.
- **Execute Instruction(EI)**

  Perform the indicated operation and store the result
- **Write Operand(WO)**

  Store the result in memory.

Six Stage Instruction Pipeline



Timing diagram for 6 stage instruction pipeline

Time $\longrightarrow$

	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

Figure 12.10 Timing Diagram for Instruction Pipeline Operation

- six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.
- the timing is set up assuming that each instruction requires all six stages (a load instruction does not need the WO stage)
- Also, the diagram assumes that all of the stages can be performed in parallel
- Assumes no memory conflicts-memory access can occur simultaneously
- Fig 12.11-Conditional branch in Instruction pipelining

Pipeline hazards

- Any reason that causes the pipeline to stall is called hazard or conflict
- A **pipeline hazard** occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution-***pipeline bubble*.**

- Resource usage

(inter instruction dependencies)

- Job scheduling problems (where different tasks get executed at pre-determined time or when the right event happens.)

3 types of Hazards

- RESOURCE CONFLICTS
  - insufficient resources
- DATA or DATA DEPENDENCY CONFLICTS
  - instruction in pipeline depend on result of previous instruction which still in pipeline yet to be completed
- BRANCH DIFFICULTIES-Contents of PC get changed

RESOURCE HAZARD

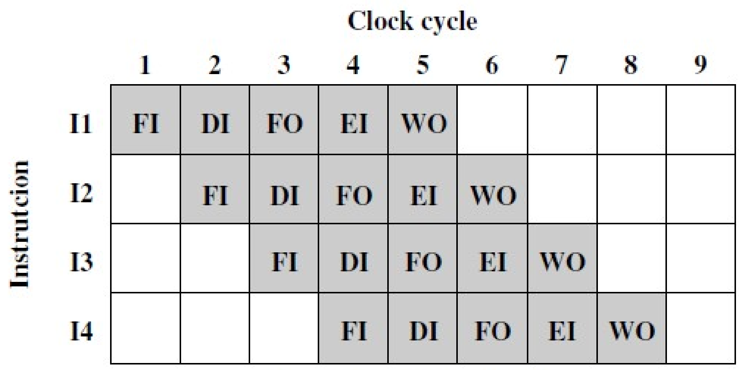
- when two (or more) instructions that are already in the pipeline need the same resource.
- executed in serial rather than parallel

- Stalling pipeline causes Structural Hazard

- Commonly need Memory Access

  – Use separate cache for **instruction** and **data**

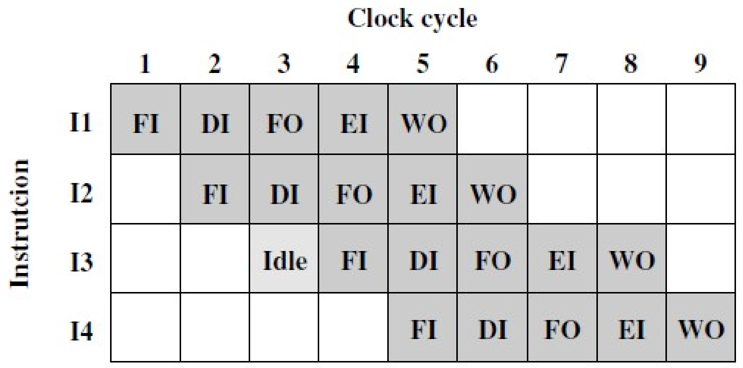
Resource conflict



Clock cycle

Instrucion	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			FI	DI	FO	EI	WO		
I4				FI	DI	FO	EI	WO	

(a) Five-stage pipeline, ideal case



Clock cycle

Instrucion	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			Idle	FI	DI	FO	EI	WO	
I4					FI	DI	FO	EI	WO

(b) I1 source operand in memory

Figure 12.15 Example of Resource Hazard

- Fig a-each stage takes one clock cycle.
- Fig b- main memory has a single port and that all instruction fetches and data reads and writes must be performed one at a time

- the fetch instruction stage of the pipeline must idle for one cycle before beginning the instruction fetch for instruction I3

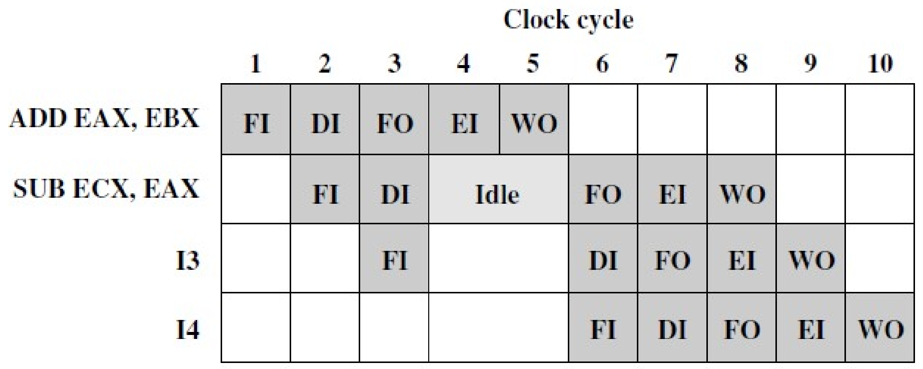
Resource conflict

- multiple instructions are ready to enter the execute instruction phase and there is a single ALU.
- Solution- increase available resources, such as having multiple ports into main memory and multiple ALU units.

DATA HAZARD

- when there is a conflict in the access of an operand location.
- the program produces an incorrect result because of the use of pipelining-

ADD EAX, EBX /* EAX = EAX + EBX
SUB ECX, EAX /* ECX = ECX - EAX



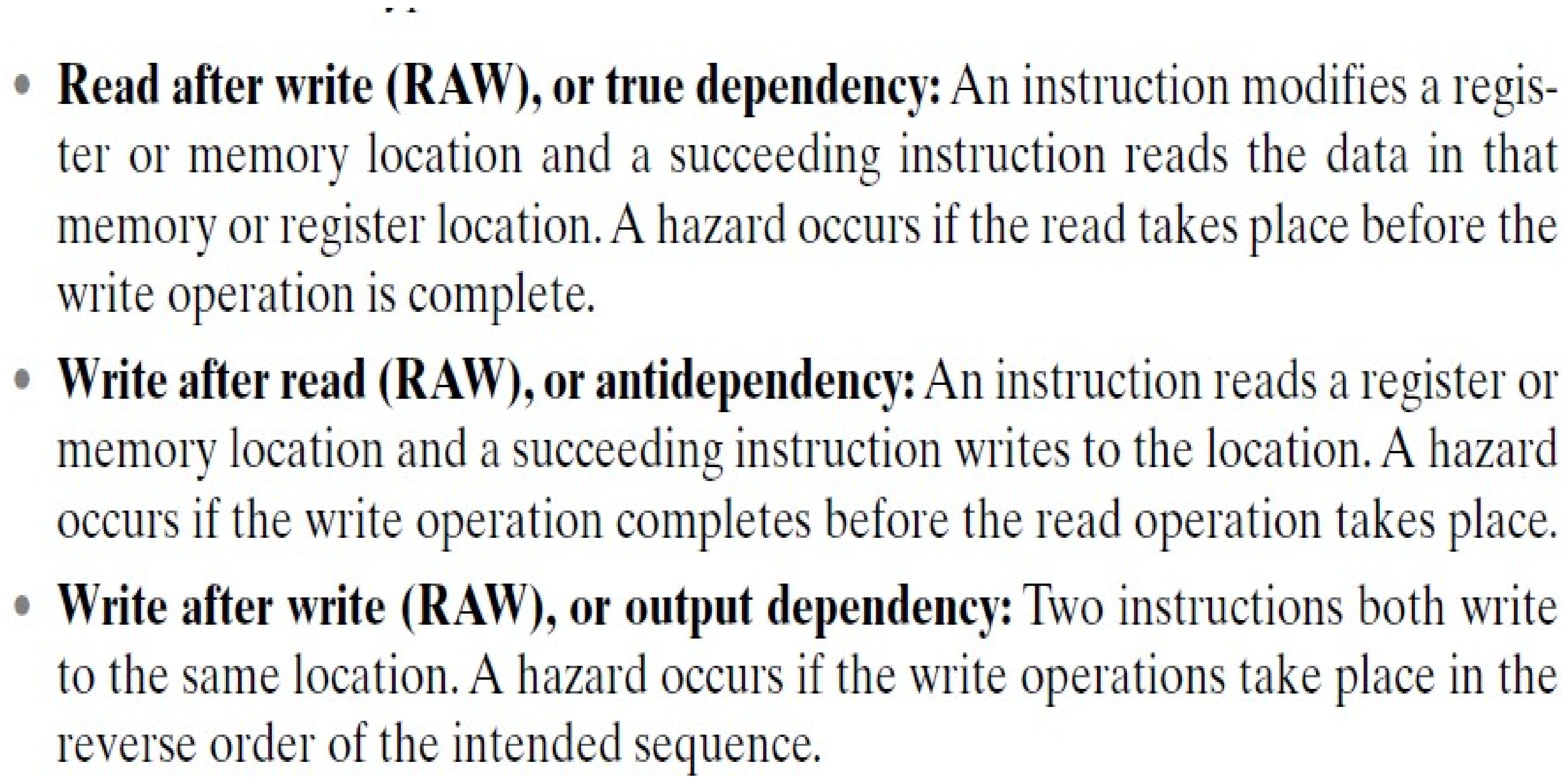
Clock cycle

	1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX	FI	DI	FO	EI	WO					
SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
I3			FI			DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

Figure 12.16 Example of Data Hazard

- The ADD instruction does not update register EAX until the end of stage 5, which occurs at clock cycle 5.
- But the SUB instruction needs that value at the beginning of its stage 2, which occurs at clock cycle 4.
- To maintain correct operation, the pipeline must stall for two clocks cycles.

DATA HAZARD



- **Read after write (RAW), or true dependency:** An instruction modifies a register or memory location and a succeeding instruction reads the data in that memory or register location. A hazard occurs if the read takes place before the write operation is complete.
- **Write after read (RAW), or antidependency:** An instruction reads a register or memory location and a succeeding instruction writes to the location. A hazard occurs if the write operation completes before the read operation takes place.
- **Write after write (RAW), or output dependency:** Two instructions both write to the same location. A hazard occurs if the write operations take place in the reverse order of the intended sequence.

  – RAR?

- Rearrange the pipeline- pipeline scheduling

BRANCH HAZARDS

- Flush Pipeline
- Delayed branching
- Conditional branching

Design issues of pipeline architecture

- **Instruction Pipeline design**
  - Instruction Issuing
    - In order issuing, out of order issuing or re order issuing
- **Arithmetic Pipeline design**
  - Fixed point , floating point , integer arithmetic

Principles of designing pipelined processors

- **Proper data buffering (Preloading data into a reserved area of memory)** to avoid congestion and smooth pipeline operations
- Instruction **dependence** relationship
- **Logic hazards** should be detected and resolved
- Avoid Collisions and structural hazards by proper **sequencing**
- **Reconfiguration** of the pipeline should be possible