Introduction to parallel processing and pipelining

8/3/2022

Parallel processing

- **Parallel processing** is a method of simultaneously breaking up and running program tasks on multiple microprocessors, thereby reducing processing time. Parallel processing may be accomplished via a computer with two or more processors or via a computer network.

- Parallel processing is also called **parallel computing**

- Any system that has more than one CPU can perform parallel processing, as well as multi-core processors which are commonly found on computers today.

- Multi-core processors are IC chips that contain two or more processors for better performance, reduced power consumption and more efficient processing of multiple tasks. These multi-core set-ups are similar to having multiple, separate processors installed in the same computer

- **How parallel processing works?**

- Divide a complex task into multiple parts with a software tool and assign each part to a processor, then each processor will solve its part, and the data is reassembled by a software tool to read the solution or execute the task.

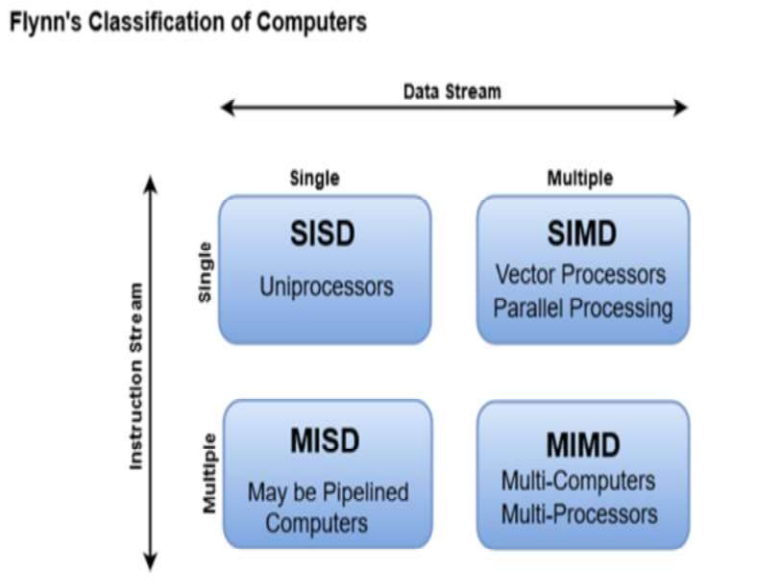
- Typically each processor will operate normally and will perform operations in parallel as instructed, pulling data from the computer's memory. Processors will also rely on software to communicate with each other so they can stay in sync concerning changes in data values. Assuming all the processors remain in sync with one another, at the end of a task, software will fit all the data pieces together.

Types of parallel processing

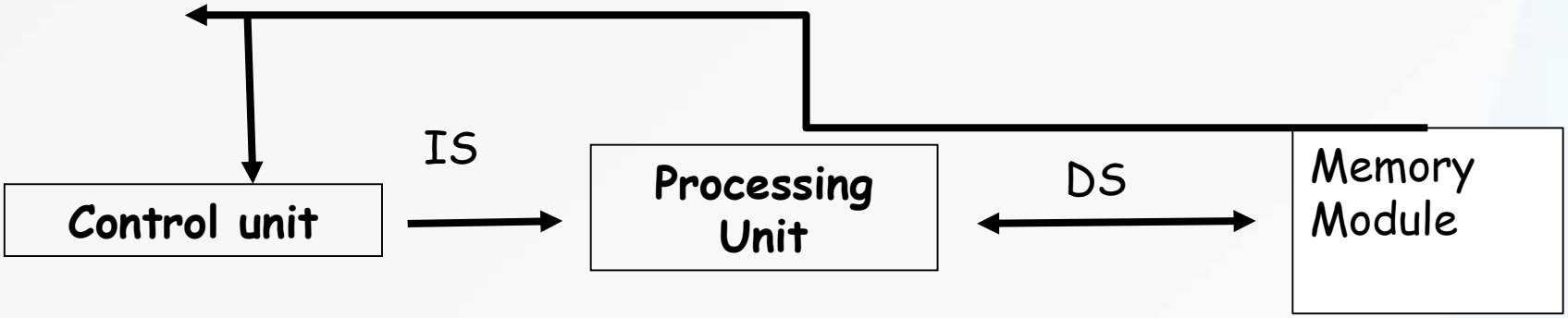
- Flynn's Classifications [1966]
  - Based on multiplicity of instruction streams & data stream in a computer.
- Feng's Classification [1972]
  - Based on serial & parallel processing.
- Handler's Classification [1977]
  - Determined by the degree of parallelism & pipeline in various subsystem level.

Flynn's Classification

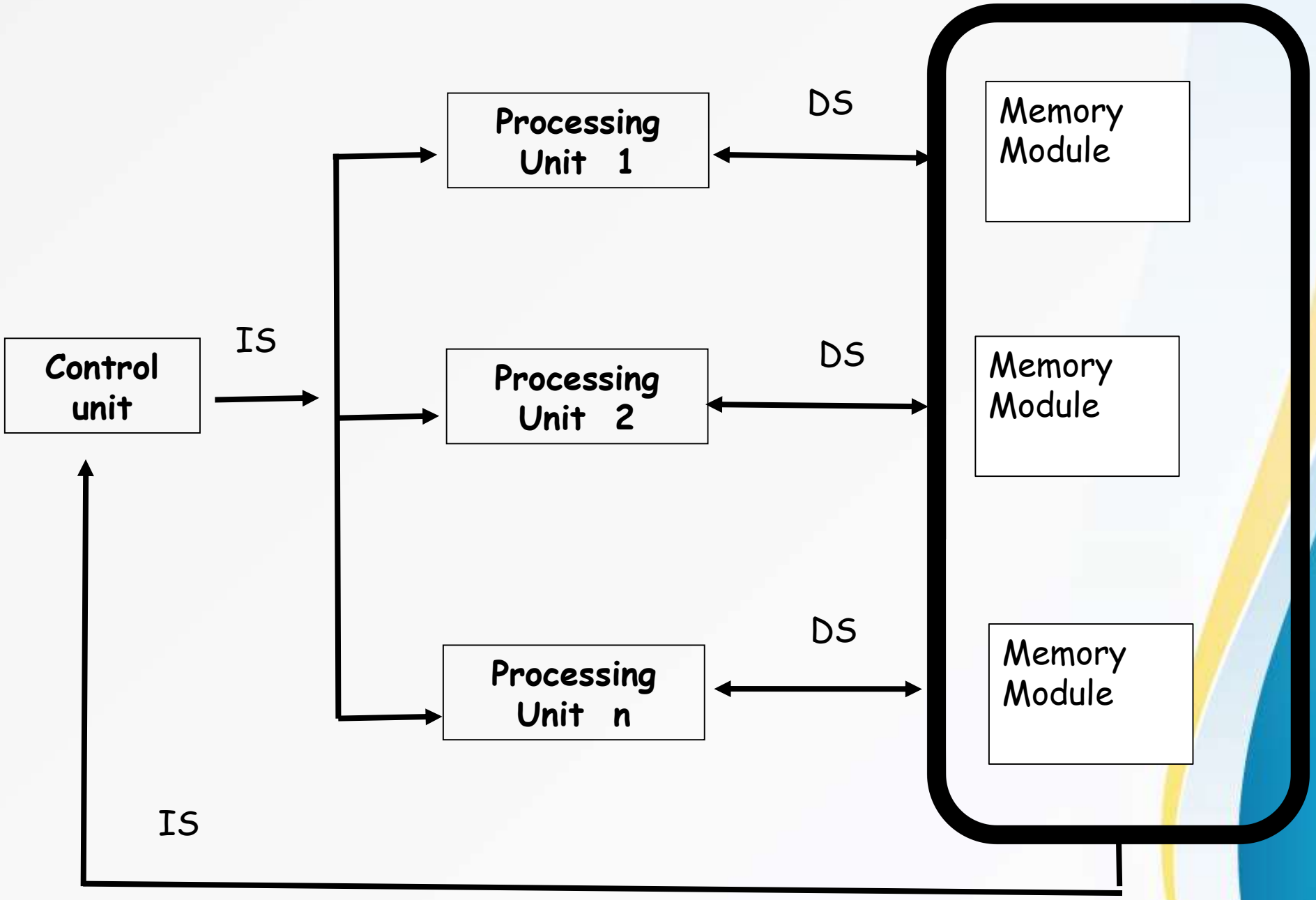
- The sequence of instructions read from memory constitutes an instruction stream.
- The operations performed on the data in the processor constitute a data stream.
- **SISD** (Single Instruction Single Data):
  - Uniprocessors.
- **MISD** (Multiple Instruction Single Data):
  - No practical examples exist
- **SIMD** (Single Instruction Multiple Data):
  - Specialized processors
- **MIMD** (Multiple Instruction Multiple Data):
  - General purpose, commercially important



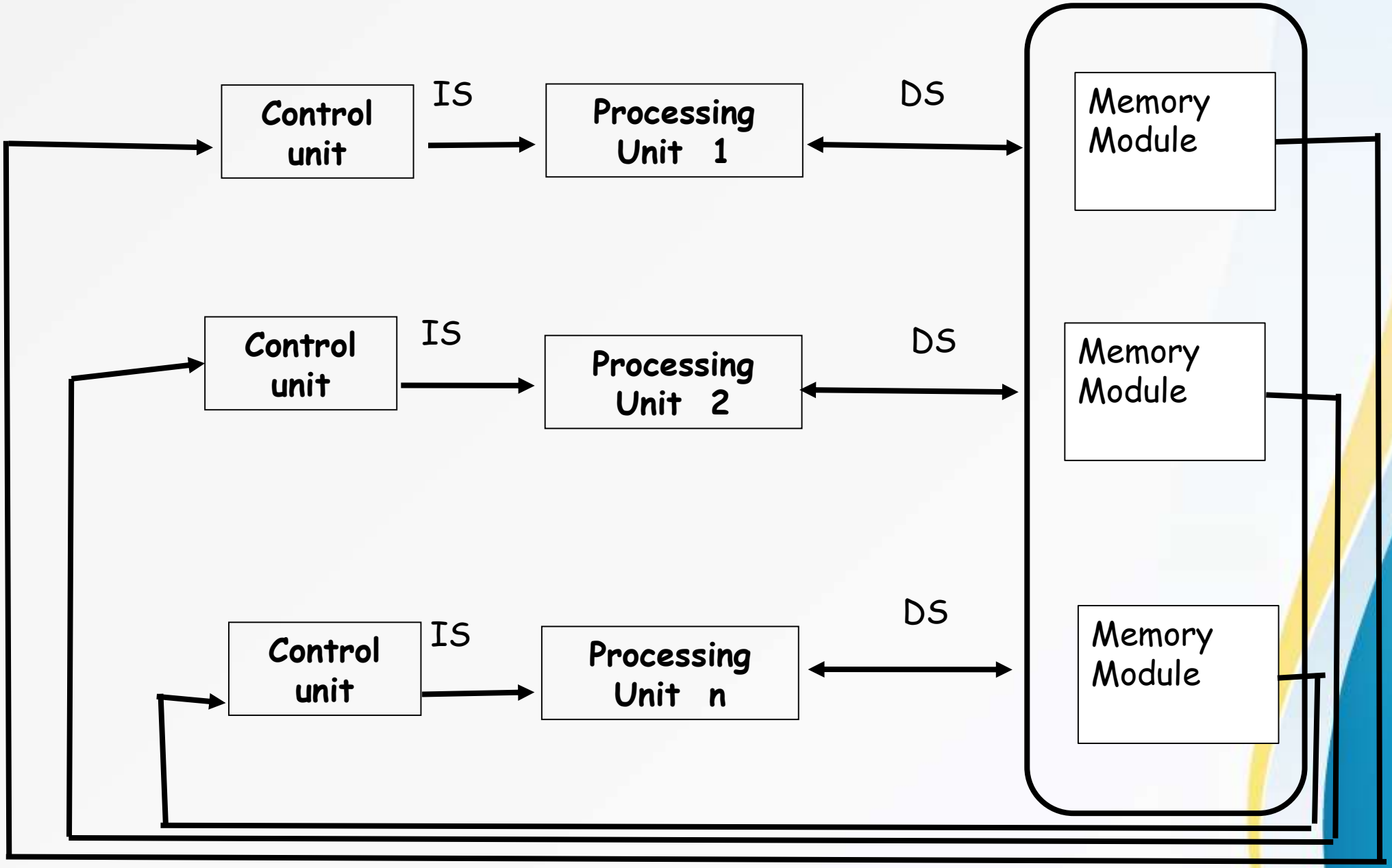
SISD



SIMD


Control unit
IS
Processing Unit 1
Processing Unit 2
Processing Unit n
DS
DS
DS
Memory Module
Memory Module
Memory Module
IS

MIMD


Control unit
IS
Processing Unit 1
DS
Memory Module
Control unit
IS
Processing Unit 2
DS
Memory Module
Control unit
IS
Processing Unit n
DS
Memory Module

Classification for MIMD Computers

- **Shared Memory:**
  - Processors communicate through a shared memory.
  - Typically processors connected to each other and to the shared memory through a bus.
- **Distributed Memory:**
  - Processors do not share any physical memory.
  - Processors connected to each other through a network.

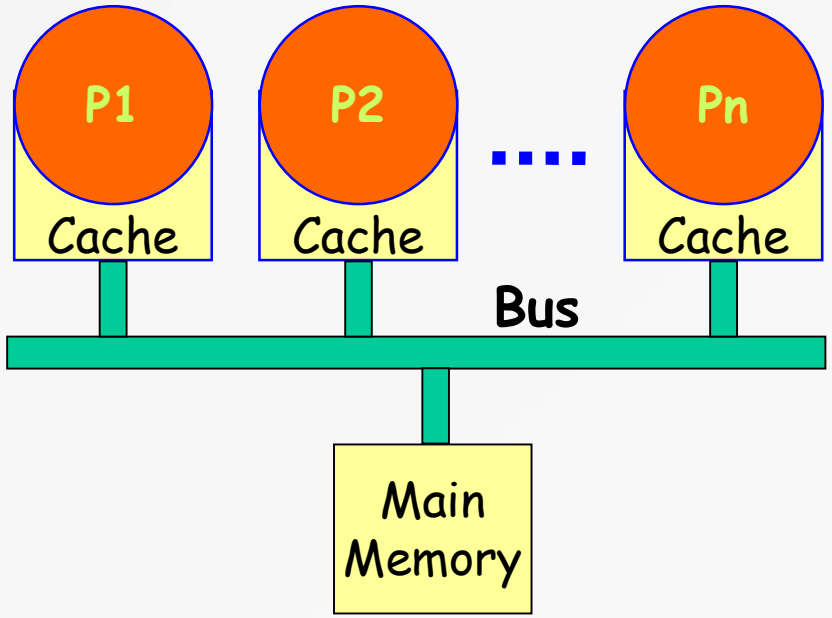
Shared Memory

- Shared memory located at a centralized location:
  - May consist of several interleaved modules --- same distance (access time) from any processor.
  - Also called Uniform Memory Access (UMA) model.

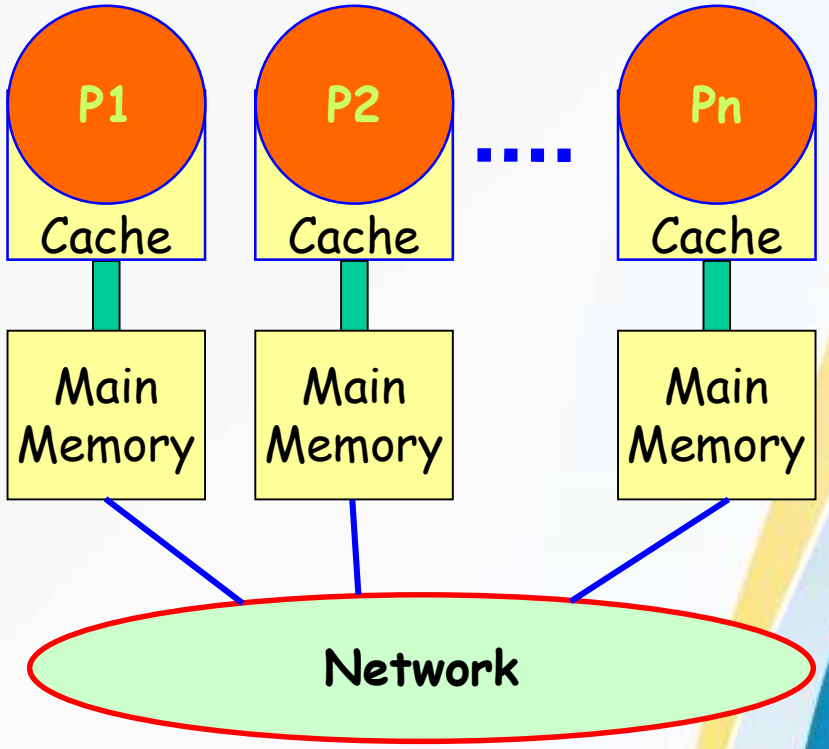
Distributed Memory

- Memory is distributed to each processor:
  - Improves scalability.
- Non-Uniform Memory Access (NUMA)
  - (a) Message passing architectures – No processor can directly access another processor's memory.
  - (b) Distributed Shared Memory (DSM)- Memory is distributed, but the address space is shared.

UMA vs. NUMA Computers



(a) UMA Model



(b) NUMA Model

Parallel Processing DEMANDS Concurrent Execution

- An efficient form of information processing which emphasizes the exploitation of concurrent events in computing process.
  - Parallel events may occur in multiple resources during the same interval of time.
  - Simultaneous events may occur at the same time.
  - Pipelined events may occur in overlapped time spans.

Feng's Classification

- Degree of parallelism is used to classify various computer architecture.
- Maximum number of binary bits that can be processed within a unit time by a computer is called as Maximum Parallelism Degree[p].

$$P_a = \frac{\sum_{i=1}^{T} P_i}{T}$$

- ✓ **P_i → be the no. of bits that can be processed within i^{th} processor cycle.**
- ✓ **T** → Processor clock cycle index i = 1,2,3,…,T
- ✓ **P_a** → Average parallelism degree.

Handler's Classification

- Wolfgang Handler has proposed a classification scheme for identifying the parallelism degree & pipelining degree built into the H/W structure of a computer system
- A computer system can be characterized by a six independent entities as:-

$$T = \{ K * K' , D * D' , W * W' \}$$

Where

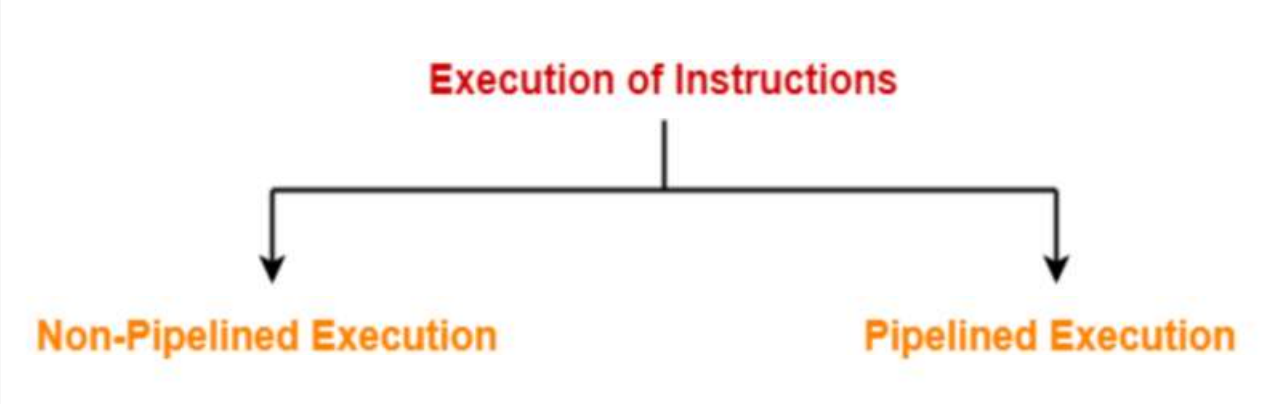
- ❖ K → No. of processor [PCU] within the computer.
- ❖K' → No. of processor [PCU] that can be pipelined.

- ❖D → No. of ALU or PE under the control of 1 PCU.
- ❖D' → No. of ALU or PE that can be pipelined.

- ❖W → Word length of an ALU of a PE.
- ❖W' → No. of pipeline stages in all ALU or in PE.

Pipelining in Computer Architecture

- ✓ A program consists of several number of instructions.
- ✓ These instructions may be executed in the following two ways-

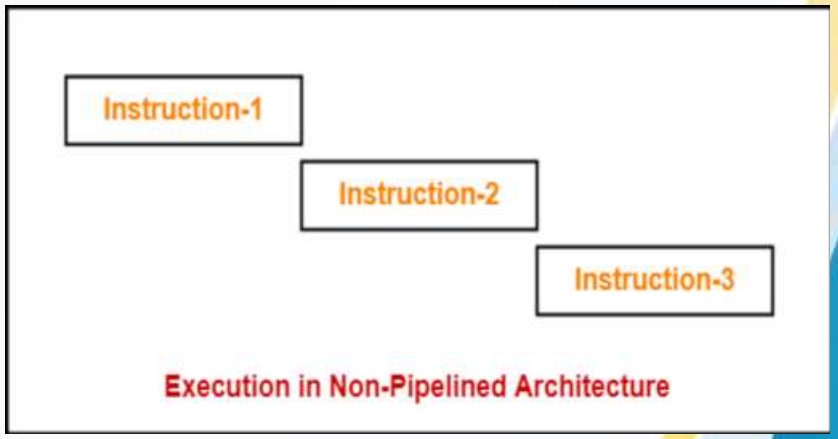


1Non-Pipelined Execution

All the instructions of a program are executed sequentially one after the other.

A new instruction executes only after the previous instruction has executed completely.

This style of executing the instructions is highly inefficient.



If time taken for executing one instruction = t, then-

Time taken for executing 'n' instructions = n x t

2. **Pipelined Execution-**

- In pipelined architecture, Multiple instructions are executed parallelly.This style of executing the instructions is highly efficient
- A pipelined processor does not wait until the previous instruction has executed completely.
- Rather, it fetches the next instruction and begins its execution.

In pipelined architecture,

- The hardware of the CPU is split up into several functional units.
- Each functional unit performs a dedicated task.
- The number of functional units may vary from processor to processor.

- These functional units are called as stages of the pipeline.
- Control unit manages all the stages using control signals.
- There is a register associated with each stage that holds the data.
- There is a global clock that synchronizes the working of all the stages.
- At the beginning of each clock cycle, each stage takes the input from its register.
- Each stage then processes the data and feed its output to the register of the next stage.

Four-Stage Pipeline-

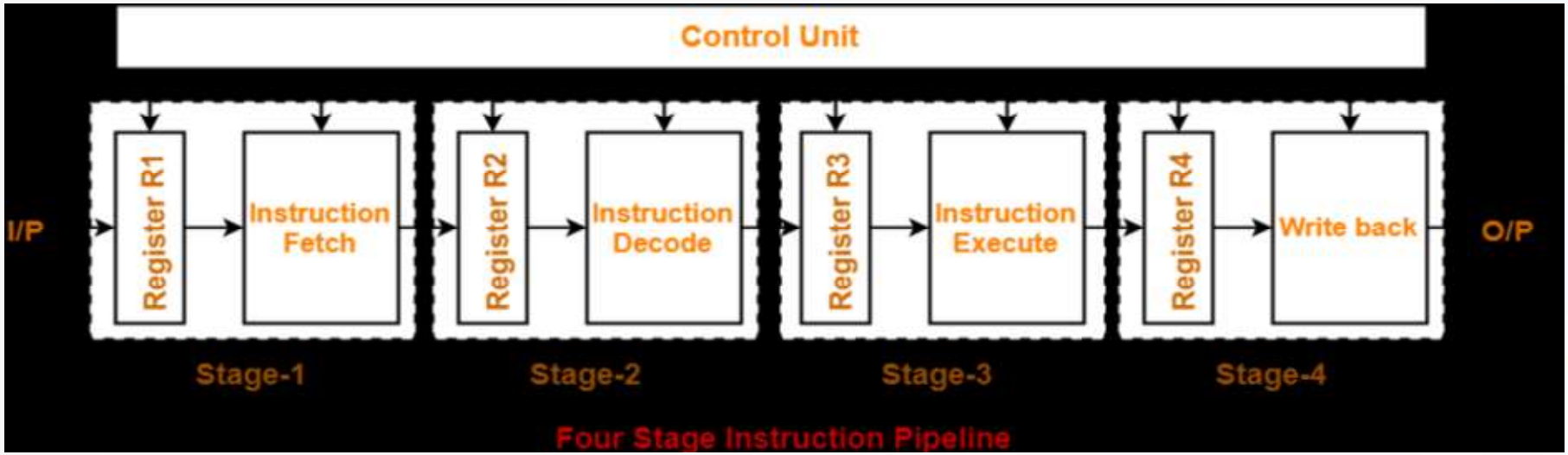
In four stage pipelined architecture, the execution of each instruction is completed in following 4 stages-

- Instruction fetch (IF)
- Instruction decode (ID)
- Instruction Execute (IE)
- Write back (WB)

To implement four stage pipeline,

✓ The hardware of the CPU is divided into four functional units.

✓ Each functional unit performs a dedicated task.



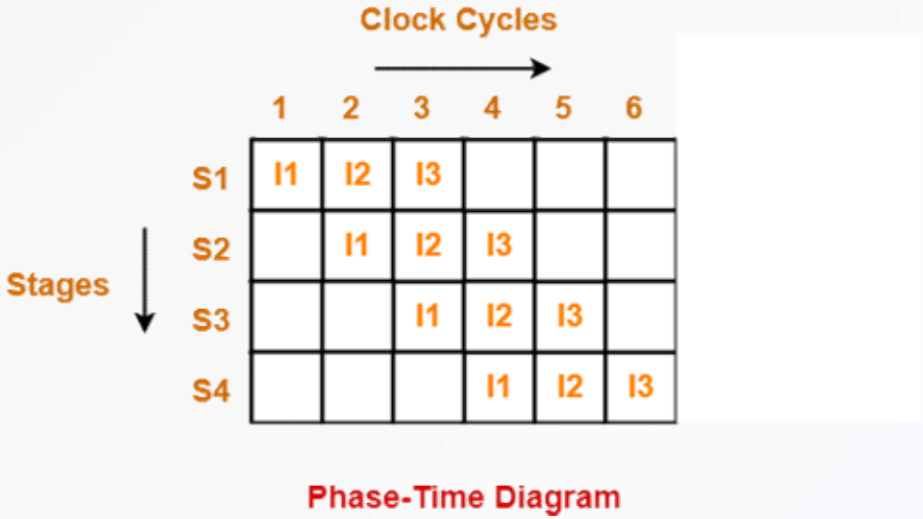
Execution-

In pipelined architecture,Instructions of the program execute parallelly.

When one instruction goes from nth stage to (n+1)th stage, another instruction goes from (n-1)th stage to nth stage.

Phase-Time Diagram-

- ✓ Phase-time diagram shows the execution of instructions in the pipelined architecture.
- ✓ The following diagram shows the execution of three instructions in four stage pipeline architecture.



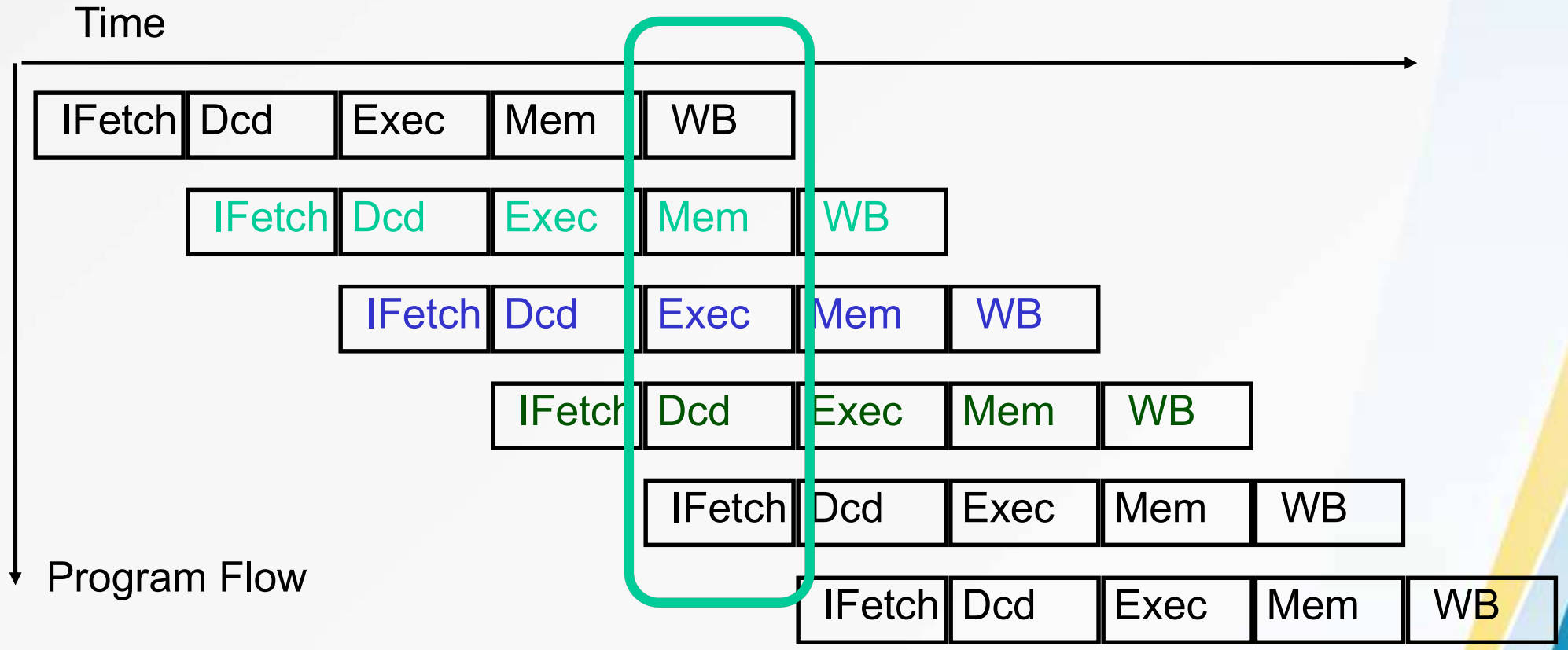
Phase-Time Diagram

Time taken to execute three instructions in four stage pipelined architecture = 6 clock cycles.

Pipelining

- Pipelining incorporates the concept of overlapped execution:
  - Used in many everyday applications without our notice.
  - Has proved to be a very popular and successful way to exploit ILP(instruction level parallelism):
  - Instruction pipes are being used in almost all modern processors.

Pipelined Execution



Advantages of Pipelining

- An n-stage pipeline:
  - Can improve performance upto n times.
- Not much investment in hardware:
  - No replication of hardware resources necessary.
  - The principle deployed is to keep the units as busy as possible.
- Transparent to the programmers:
  - Easy to use

Basic Pipelining Terminologies

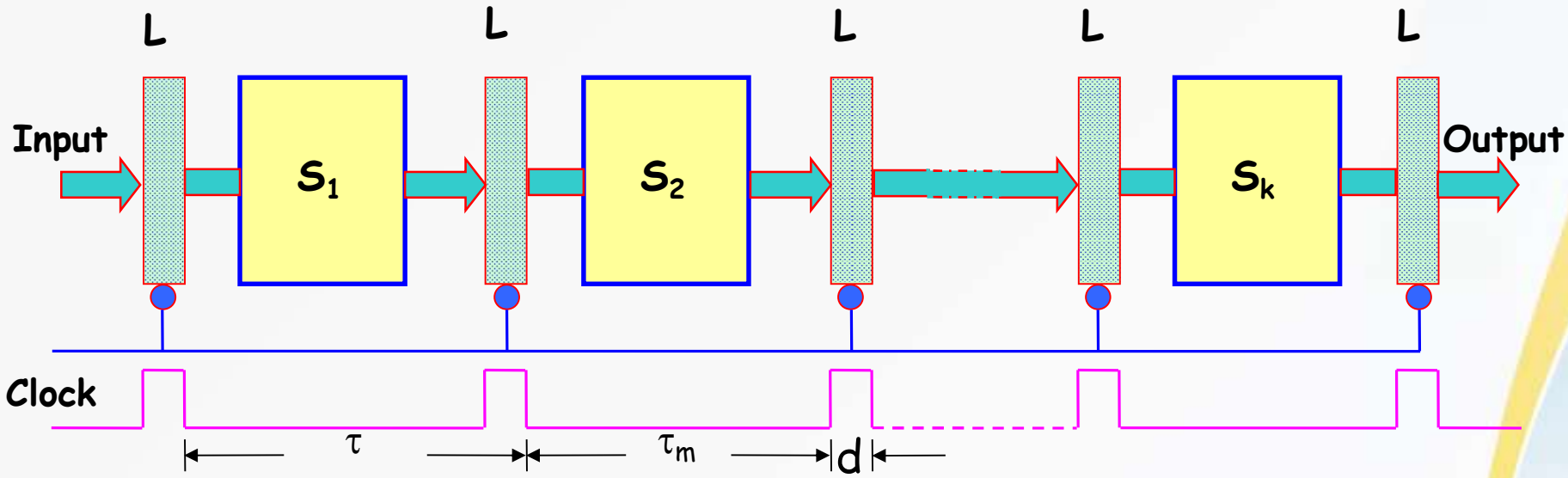
- Synchronous pipeline:
  - Pipeline cycle is constant (clock-driven).
- Asynchronous pipeline:
  - Time for moving from stage to stage varies
- Pipeline cycle (or Processor cycle):
  - The time required to move an instruction one step further in the pipeline.

Precedence relation

- A set of subtask { $T_1, T_2, \ldots\ldots, T_n$} for a given task T ,that some task T_j can not start until some earlier task T_i ,where (i<j)finishes.
- Pipeline consists of cascade of processing stages.
- Stages are combinational circuits over data stream flowing through pipe.
- Stages are separated by high speed interface latches (Holding intermediate results between stages.)
- Control must be under a common clock.

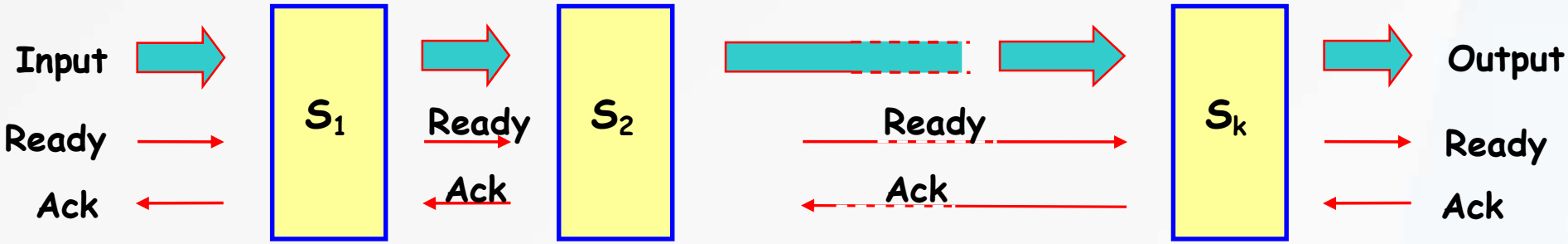
Synchronous Pipeline

- Transfers between stages are simultaneous.
- One task or operation enters the pipeline per cycle.

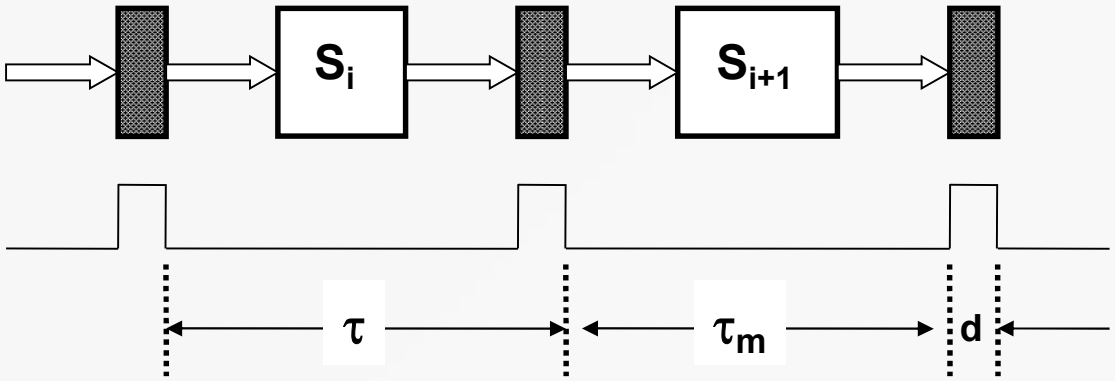


Asynchronous Pipeline

- Transfers performed when individual stages are ready.
- Handshaking protocol between processors.



- Different amounts of delay may be experienced at different stages.
- Can display variable throughput rate.



Pipeline cycle : τ

Latch delay : *d*

$$\tau = \max \{\tau_m\} + d$$

Pipeline frequency : *f*

$$f = 1 / \tau$$

Example on Clock period

- Suppose the time delays of the 4 stages are τ_1 = 60ns, τ_2 = 50ns, τ_3 = 90ns, τ_4 = 80ns & the interface latch has a delay of d = 10ns.
- Hence the cycle time of this pipeline can be granted to be like :-
- $\cdot$ = max { $\cdot_m$ } + d
- τ = 90 + 10 =100ns
- Clock frequency of the pipeline

  (f) = 1/100

  =10 Mhz
- If it is non-pipeline then = 60 + 50 + 90 + 80

  =280ns

Ideal Pipeline Speedup

- *k-stage* pipeline processes *n* tasks in k + (n-1) clock cycles:
  - *k* cycles for the first task and *n-1* cycles for the remaining *n-1* tasks.
- Total time to process *n* tasks in pipeline

$$T_k = [\, k + (n-1)] \, \tau$$

- For the non-pipelined processor

$$T_1 = n \, k \, \tau$$

Pipeline Speedup Expression

Speedup=

$$S_k = \frac{T_1}{T_k} = \frac{n\,k\,\tau}{[\,k + (n-1)]\,\tau} = \frac{n\,k}{k + (n-1)}$$

- Maximum speedup = $S_k \rightarrow K$,*for* $n >> K$ *(n=infinity)*

Efficiency of pipeline

- The percentage of busy time-space span over the total time span.
  - n:- no. of task or instruction
  - k:- no. of pipeline stages
  - τ:- clock period of pipeline
- Hence pipeline efficiency can be defined by:-

$$\eta = \frac{n}{k+(n-1)}$$

efficiency=speedup/speedupmax=speedup/k

Throughput of pipeline

Number of result task that can be completed by a pipeline per unit time.

Throughput = Number of instruction / Total time to complete the instruction
=n/{k+(n-1)}*clock period=efficiency/clock period

Throughput of pipeline

- Number of result task that can be completed by a pipeline per unit time.

$$W = \frac{n}{k^*\tau+(n-1)\tau} = \frac{n}{[k+(n-1)]\tau} = \frac{\eta}{\tau}$$

- Idle case w = $1/\tau$ = **f when** η **=1**.
- Maximum throughput = frequency of linear pipeline

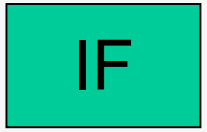
In instruction pipelining,Speed up, Efficiency and Throughput serve as performance measures of pipelined execution.

Pipeline Registers

- Pipeline registers are essential part of pipelines:
  - There are 4 groups of pipeline registers in 4 stage pipeline.
- Each group saves output from one stage and passes it as input to the next stage:
  - IF/ID
  - ID/EX
  - EX/MEM
  - MEM/WB
- This way, each time "something is computed"...
  - Effective address, Immediate value, Register content, etc.
  - It is saved safely in the context of the instruction that needs it.

Pipeline Registers

inst. 1



inst. 2

inst. 3

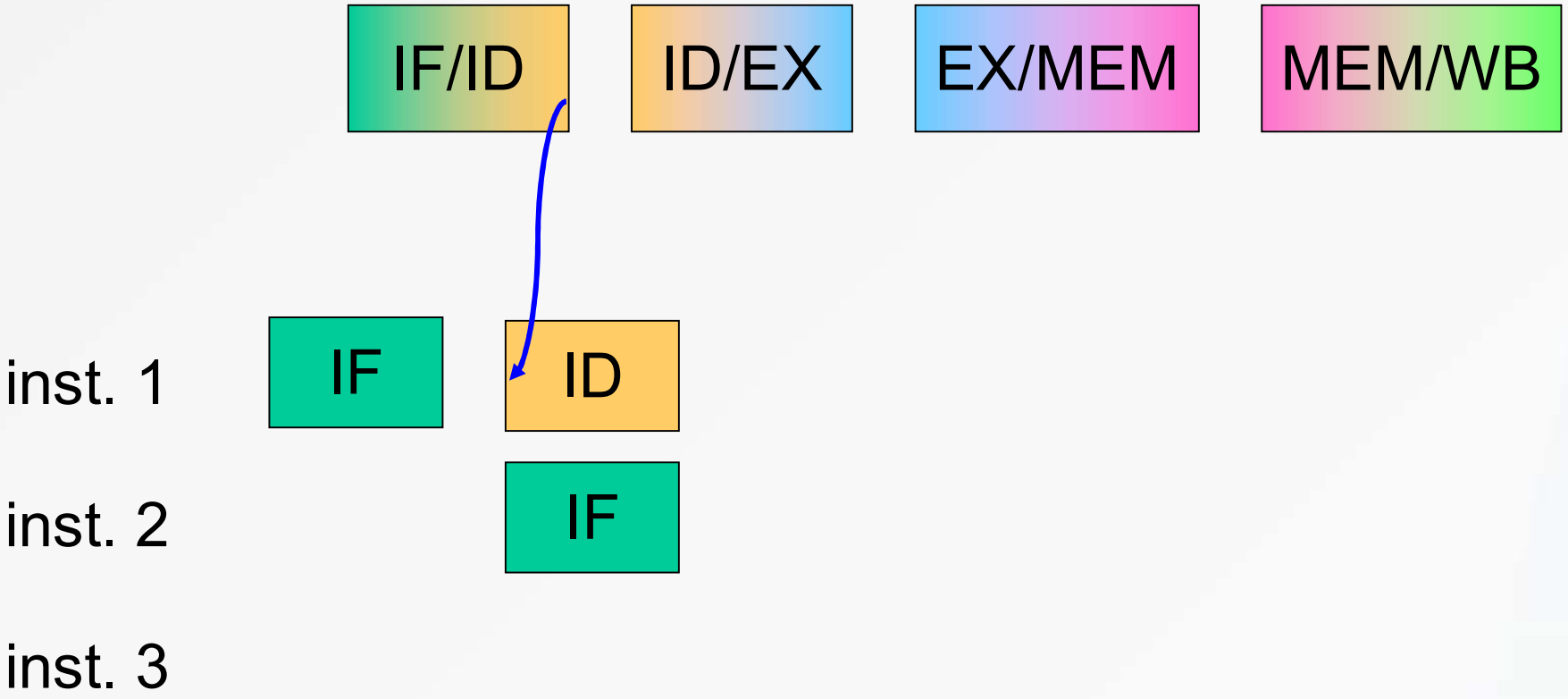
Pipeline Registers

inst. 1

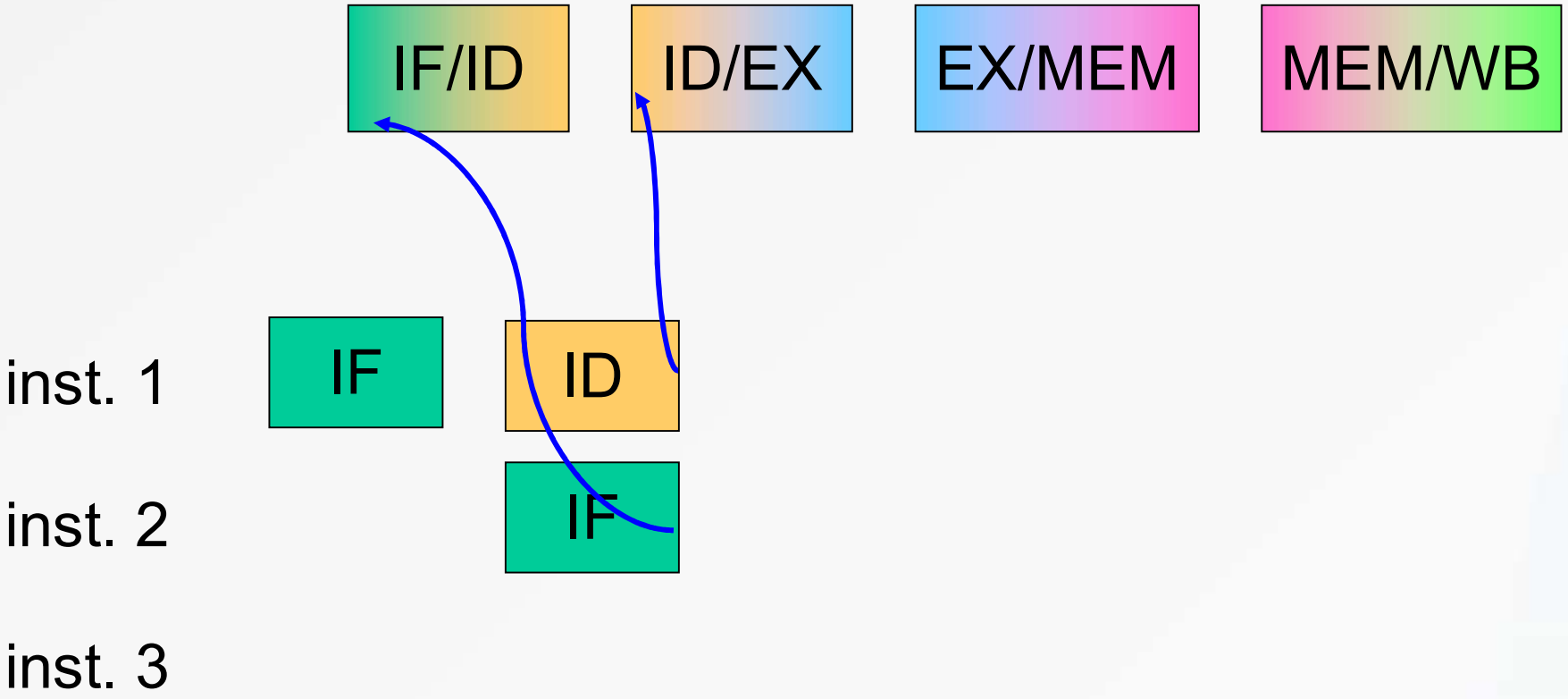
inst. 2

inst. 3

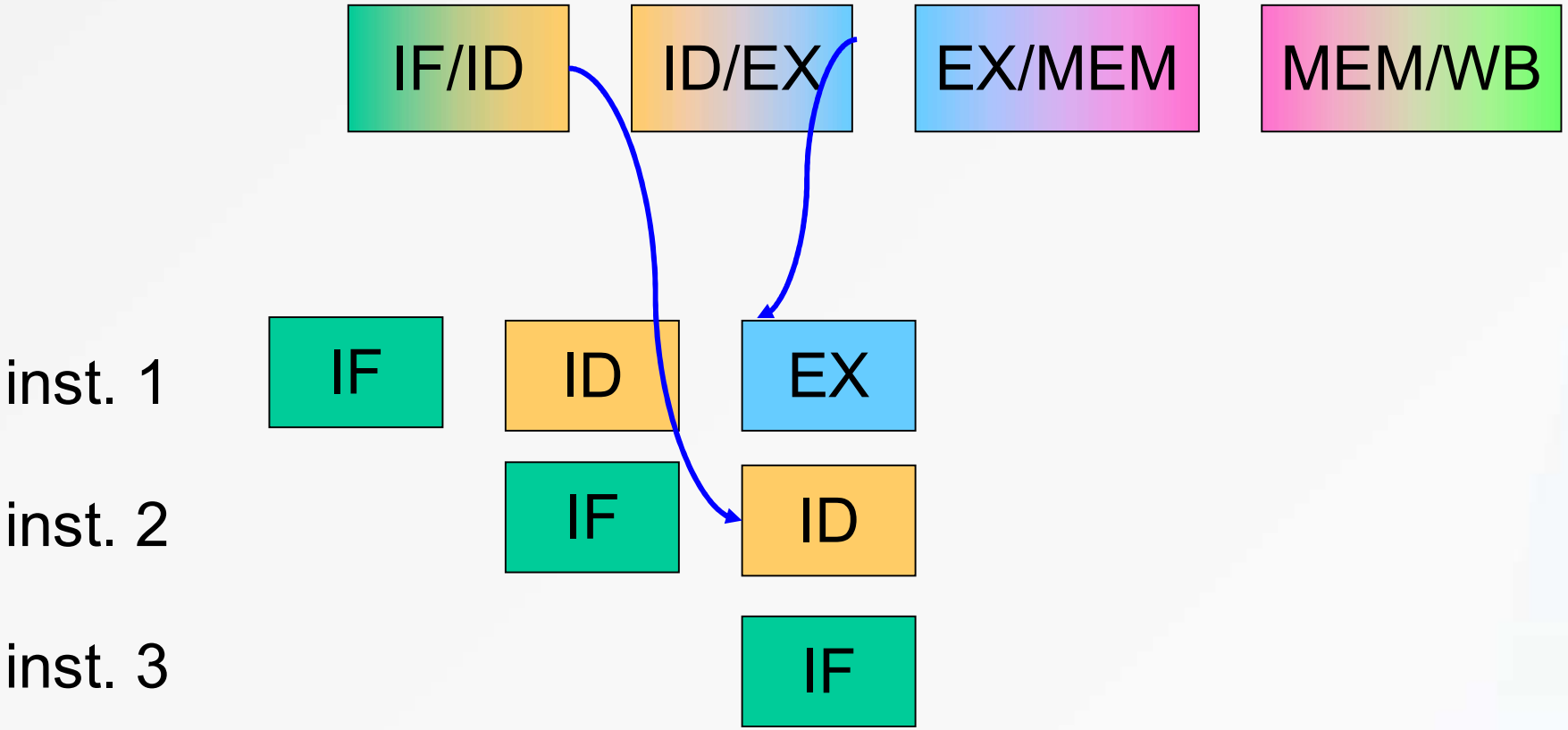
Pipeline Registers



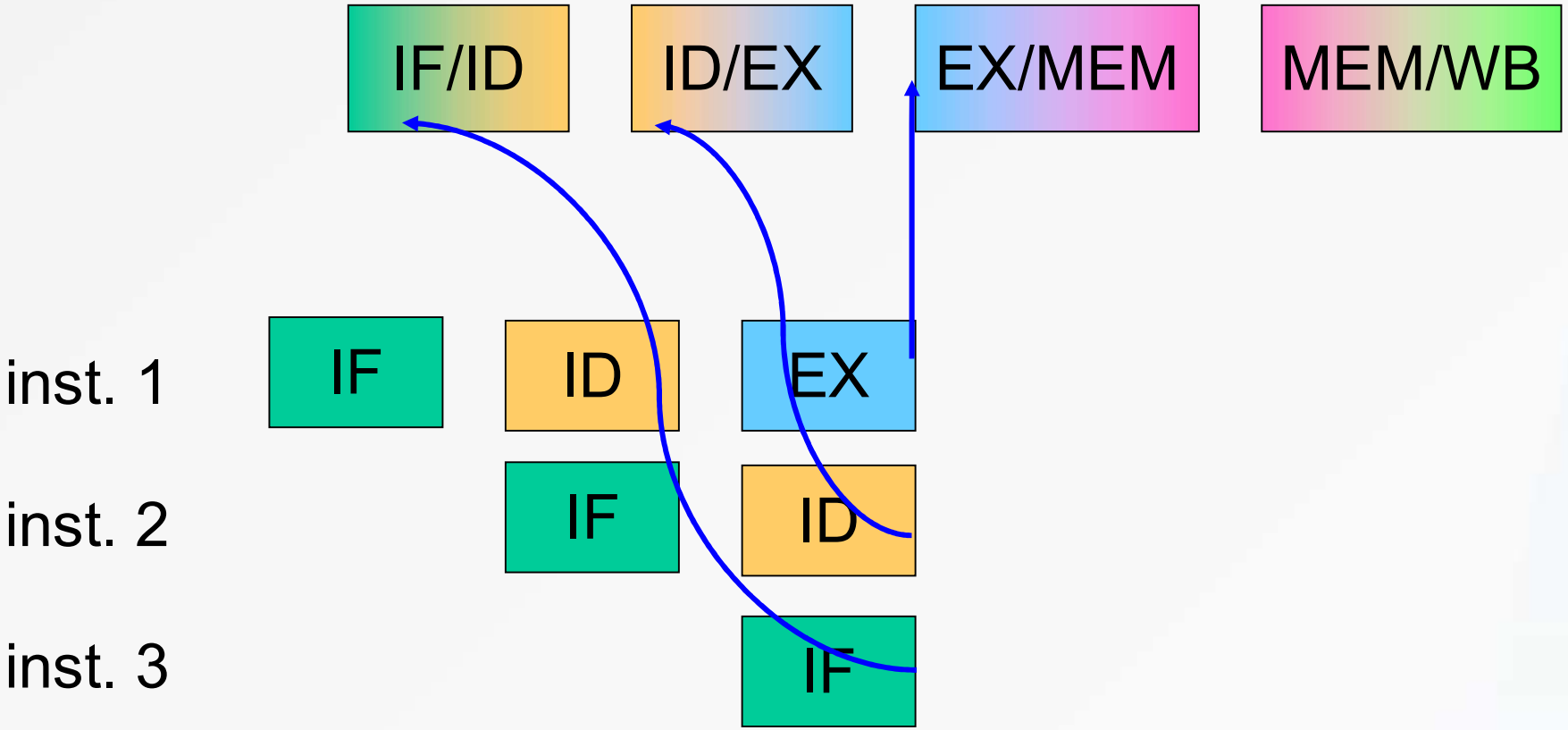
Pipeline Registers



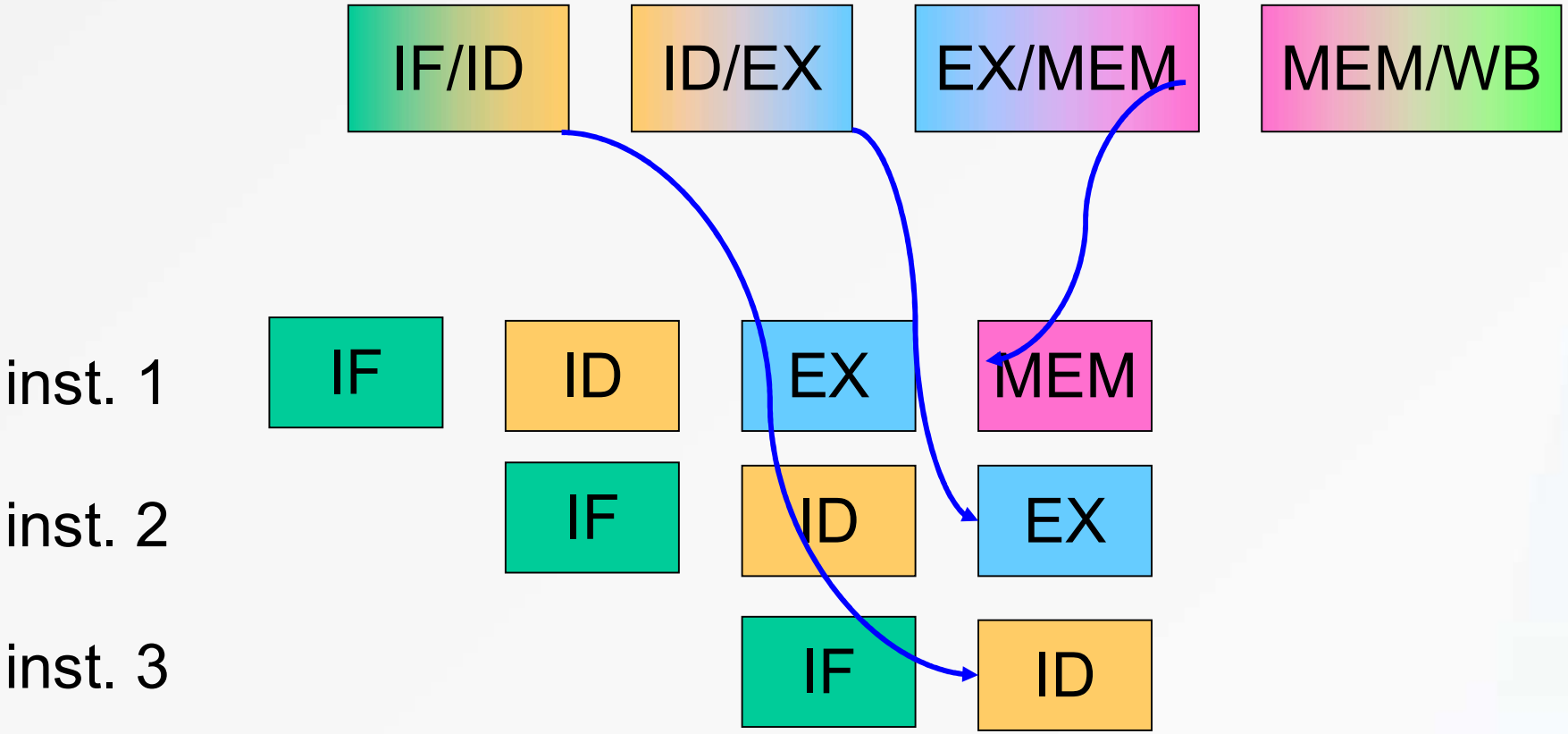
Pipeline Registers



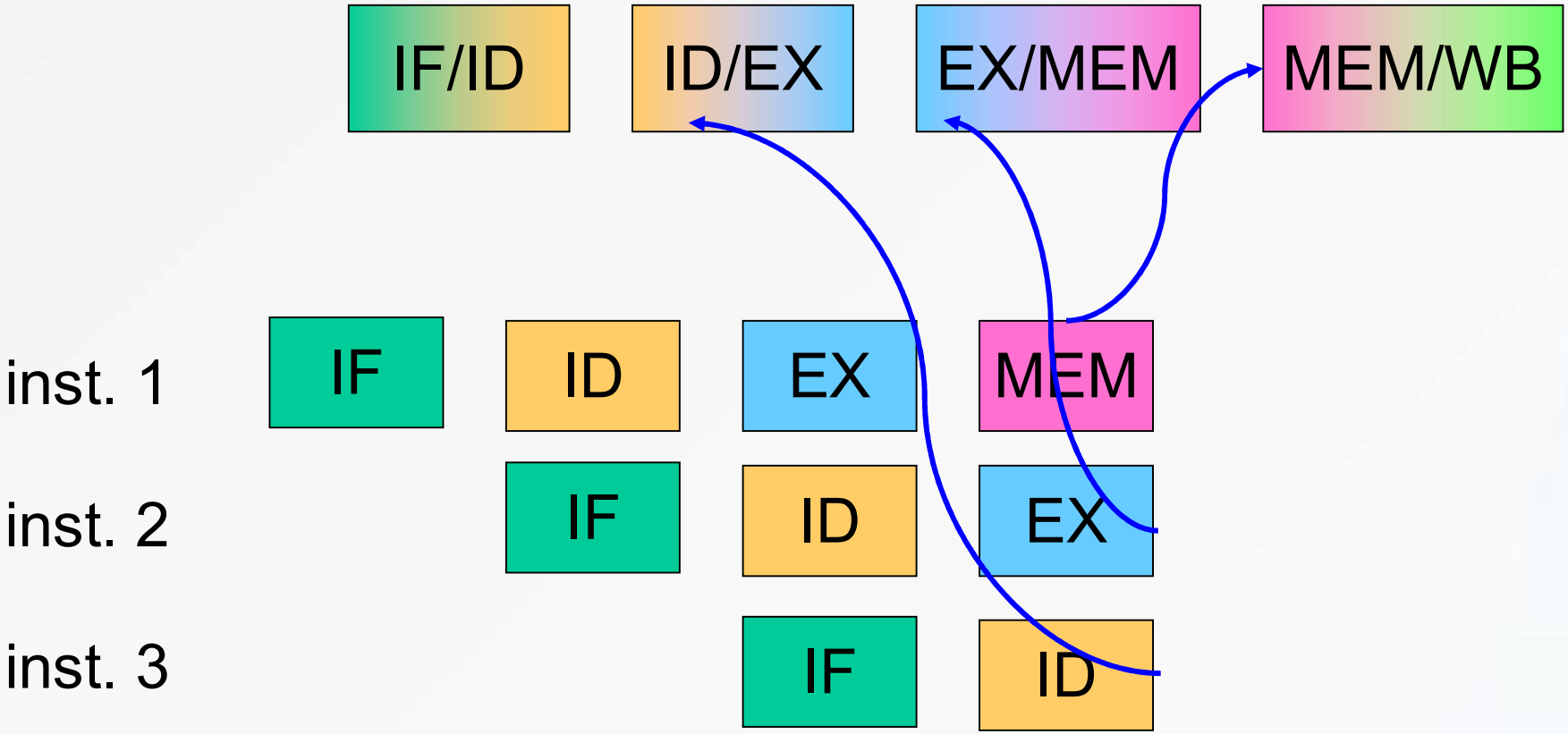
Pipeline Registers



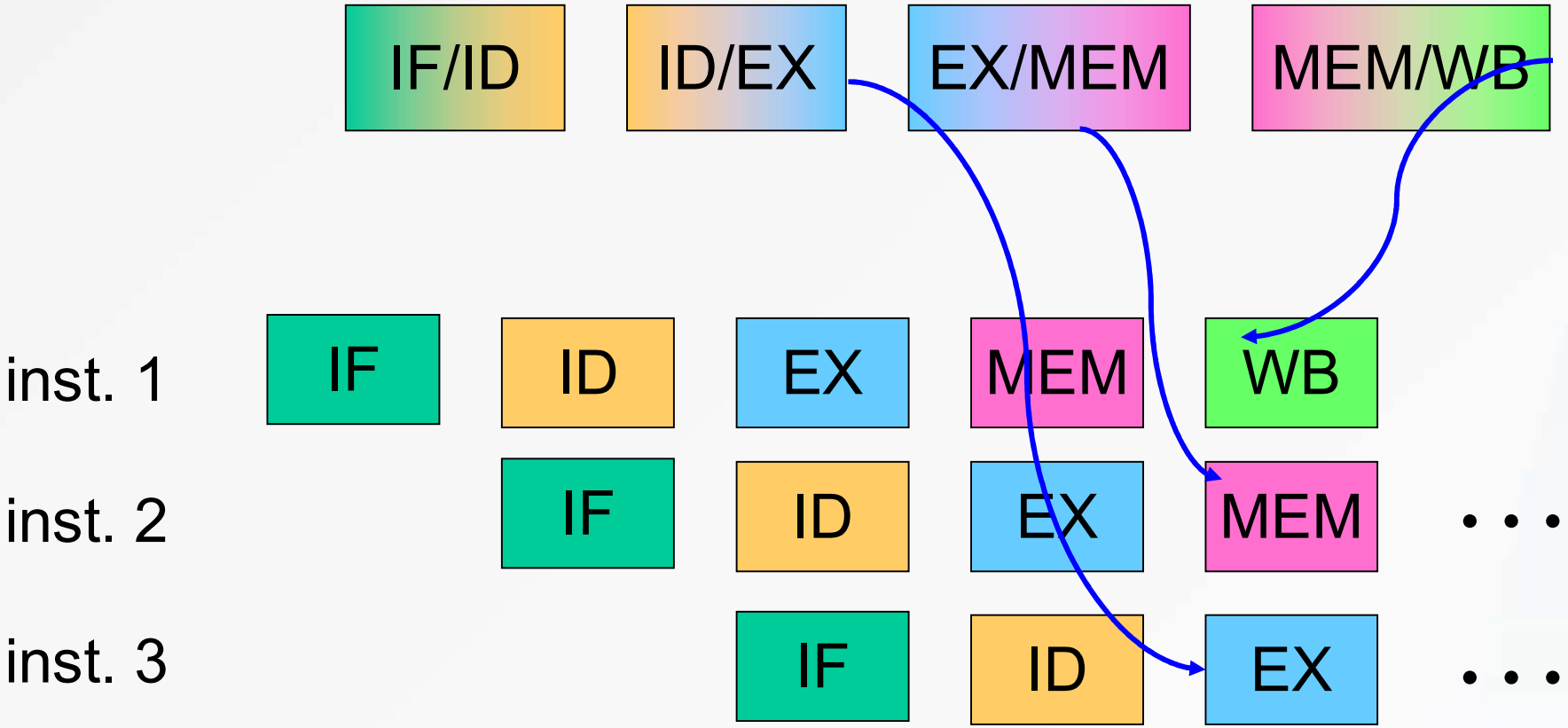
Pipeline Registers



Pipeline Registers



Pipeline Registers



- Typically, we will not think too much about pipeline registers and one just assumes that values are passed "magically" down stages of the pipeline.

example-1:

Consider a pipeline having 4 phases with duration 60, 50, 90 and 80 ns. Given latch delay is 10 ns. Calculate-

1. Pipeline cycle time
2. Non-pipeline execution time for an instruction
3. Pipeline time for 1000 tasks
4. Nonpipeline time for 1000 tasks
5. speedup
6. efficiency
6. Throughput

Solution:

Given-

Four stage pipeline is used

Delay of stages = 60, 50, 90 and 80 ns

Latch delay or delay due to each register = 10 ns

Part-01: Pipeline Cycle Time-

Cycle time= Maximum delay due to any stage + Delay due to its register

= Max { 60, 50, 90, 80 } + 10 ns

= 90 ns + 10 ns

= 100 ns

Part-02: Non-Pipeline Execution Time-

Non-pipeline execution time for one instruction= 60 ns + 50 ns + 90 ns + 80 ns

= 280 ns

Part-03: Pipeline Time For 1000 Tasks-

Pipeline time for 1000 tasks= Time taken for 1st task + Time taken for remaining 999 tasks

= 1 x 4 clock cycles + 999 x 1 clock cycle

= 4 x cycle time + 999 x cycle time

= 4 x 100 ns + 999 x 100 ns

= 400 ns + 99900 ns

= 100300 ns

Part-04: Sequential Time For 1000 Tasks-

Non-pipeline time for 1000 tasks= 1000 x Time taken for one task

= 1000 x 280 ns

= 280000 ns

Part-05: Speedup=280000/100300=2.79

part-06: efficiency=sppedup/k=2.79/4=0.69

Part-07: Throughput=1000/100300

example-02

A four stage pipeline has the stage delays as 150, 120, 160 and 140 ns respectively. Registers are used between the stages and have a delay of 5 ns each.

1. Pipeline cycle time
2. Non-pipeline execution time for an instruction
3. Pipeline time for 1000 tasks
4. Nonpipeline time for 1000 tasks
5. speedup
6. efficiency
6. Throughput

Solution-

Given-

- Four stage pipeline is used
- Delay of stages = 150, 120, 160 and 140 ns
- Delay due to each register = 5 ns
- 1000 data items or instructions are processed

Cycle Time-

Cycle time= Maximum delay due to any stage + Delay due to its register

= Max { 150, 120, 160, 140 } + 5 ns

= 160 ns + 5 ns

= 165 ns

Pipeline Time To Process 1000 Data Items-

Pipeline time to process 1000 data items= Time taken for 1st data item + Time taken for remaining 999 data items

= 1 x 4 clock cycles + 999 x 1 clock cycle

= 4 x cycle time + 999 x cycle time

= 4 x 165 ns + 999 x 165 ns

= 660 ns + 164835 ns

= 165495 ns

= 165.5 μs

Example-03

We have 2 designs D1 and D2 for a pipeline processor. D1 has 5 stage pipeline with execution time of 3 ns, 2 ns, 4 ns, 2 ns and 3 ns. While the design D2 has 8 pipeline stages each with 2 ns execution time. How much time can be saved using design D2 over design D1 for executing 100 instructions?
(hints:assume latch delay time 0)

214 ns
202 ns
86 ns
200 ns

Cycle Time in Design D1-

Cycle time= Maximum delay due to any stage + Delay due to its register
= Max { 3, 2, 4, 2, 3 } + 0
= 4 ns

Execution Time For 100 Instructions in Design D1-

Execution time for 100 instructions= Time taken for 1st instruction + Time taken for remaining 99 instructions
= 1 x 5 clock cycles + 99 x 1 clock cycle
= 5 x cycle time + 99 x cycle time
= 5 x 4 ns + 99 x 4 ns
= 20 ns + 396 ns
= 416 ns

Cycle Time in Design D2-

Cycle time= Delay due to a stage + Delay due to its register
= 2 ns + 0
= 2 ns

Execution Time For 100 Instructions in Design D2-

Execution time for 100 instructions= Time taken for 1st instruction + Time taken for remaining 99 instructions
= 1 x 8 clock cycles + 99 x 1 clock cycle
= 8 x cycle time + 99 x cycle time
= 8 x 2 ns + 99 x 2 ns
= 16 ns + 198 ns
= 214 ns

Time saved== 416 ns – 214 ns= 202 ns

Example-04

- Consider an unpipelined processor:
  - Takes 4 cycles for ALU and other operations
  - 5 cycles for memory operations.
  - Assume the relative frequencies:
    - ALU and other=60%,
    - memory operations=40%
  - Cycle time =1ns
- Compute speedup due to pipelining:
  - Ignore effects of branching.
  - Assume pipeline overhead = 0.2ns

Solution

- Average instruction execution time for large number of instructions:
  - unpipelined= 1ns * (60%*4+ 40%*5) =4.4ns
  - Pipelined=1.2ns
- Speedup=4.4/1.2=3.7 times

Example-05

- Consider an unpipelined processor:
  - Takes 4 cycles for ALU and
  - 3 cycles for branch operations
  - 5 cycles for memory operations.
  - Assume the relative frequencies of these opeartions are 40%, 20% and 40% respectively.
  - Cycle time =1ns
- Compute speedup due to pipelining:
  - Ignore effects of branching.
  - Assume pipeline overhead = 0.2ns

Example-06

Consider a non-pipelined processor with a clock rate of 2.5 gigahertz and average cycles per instruction of 4. The same processor is upgraded to a pipelined processor with five stages but due to the internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume there are no stalls in the pipeline. The speed up achieved in this pipelined processor is-

3.2
3.0
2.2
2.0

Cycle Time in Non-Pipelined Processor-

Frequency of the clock = 2.5 gigahertz

Cycle time= 1 / frequency= 1 / (2.5 gigahertz)= 0.4 ns

Non-Pipeline Execution Time-

Non-pipeline execution time to process 1 instruction
= Number of clock cycles taken to execute one instruction
= 4 clock cycles = 4 x 0.4 ns= 1.6 ns

Cycle Time in Pipelined Processor-

Frequency of the clock = 2 gigahertz
Cycle time= 1 / frequency= 1 / (2 gigahertz)= 0.5 ns

Pipeline Execution Time-

Since there are no stalls in the pipeline, so ideally one instruction is executed per clock cycle.
So,

Pipeline execution time = 1 clock cycle = 0.5 ns

Speed Up-

Speed up= Non-pipeline execution time / Pipeline execution time
= 1.6 ns / 0.5 ns = 3.2

Example-07

The stage delays in a 4 stage pipeline are 800, 500, 400 and 300 picoseconds. The first stage is replaced with a functionally equivalent design involving two stages with respective delays 600 and 350 picoseconds.

The throughput increase of the pipeline is _____%.

Execution Time in 4 Stage Pipeline-

Cycle time= Maximum delay due to any stage + Delay due to its register= Max { 800, 500, 400, 300 } + 0

= 800 picoseconds

Thus, Execution time in 4 stage pipeline = 1 clock cycle = 800 picoseconds.

Throughput in 4 Stage Pipeline-

Throughput= Number of instructions executed per unit time= 1 instruction / 800 picoseconds

Execution Time in 5 Stage Pipeline-

Cycle time = Maximum delay due to any stage + Delay due to its register

= Max { 600, 350, 500, 400, 300 } + 0 = 600 picoseconds

Thus, Execution time in 2 stage pipeline = 1 clock cycle = 600 picoseconds.

Throughput in 5 Stage Pipeline-

Throughput= Number of instructions executed per unit time= 1 instruction / 600 picoseconds

Throughput Increase-

Throughput increase= { (Final throughput – Initial throughput) / Initial throughput } x 100

= { (1 / 600 – 1 / 800) / (1 / 800) } x 100= { (800 / 600) – 1 } x 100= (1.33 – 1) x 100= 0.3333 x 100

= 33.33 %

Example-08

A non-pipelined single cycle processor operating at 100 MHz is converted into a asynchronous pipelined processor with five stages requiring 2.5 ns, 1.5 ns, 2 ns, 1.5 ns and 2.5 ns respectively. The delay of the latches is 0.5 ns.

The speed up of the pipeline processor for a large number of instructions is-

4.5
4.0
3.33
3.0

Cycle Time in Non-Pipelined Processor-

Frequency of the clock = 100 MHz
Cycle time= 1 / frequency= 1 / (100 MHz)= 0.01 μs=10ns

Non-Pipeline Execution Time-
Non-pipeline execution time to process 1 instruction= Number of clock cycles taken to execute one instruction= 1 clock cycle= 10 ns

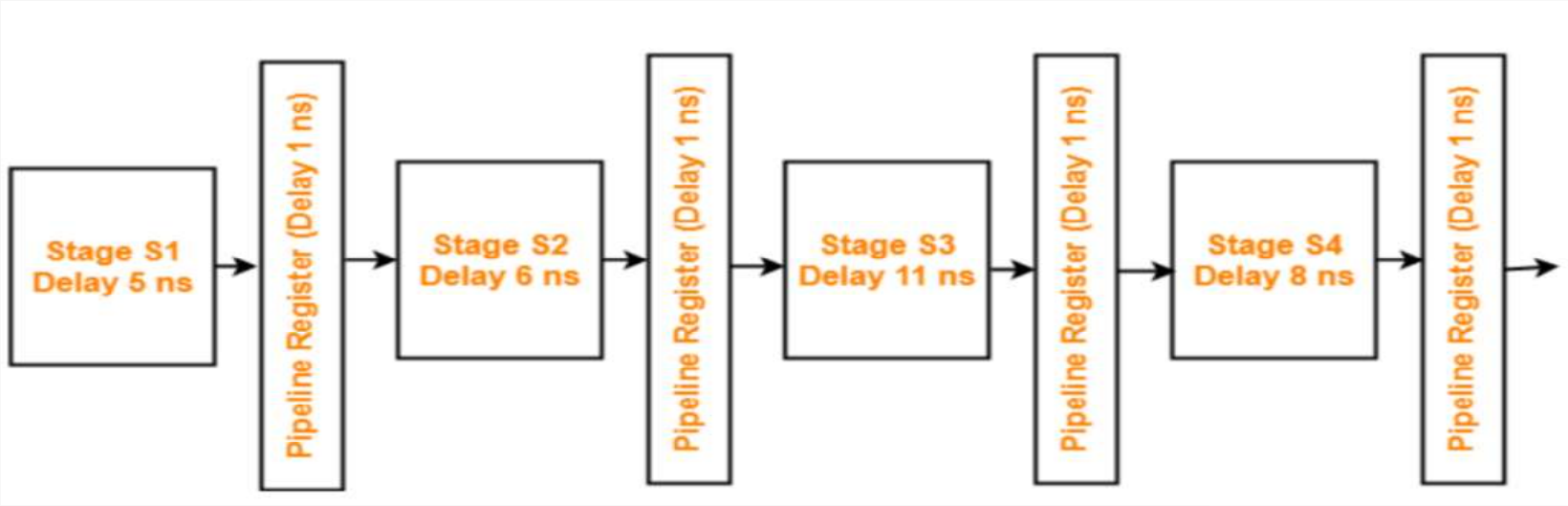
Cycle Time in Pipelined Processor-
Cycle time= Maximum delay due to any stage + Delay due to its register
= Max { 2.5, 1.5, 2, 1.5, 2.5 } + 0.5 ns= 2.5 ns + 0.5 ns= 3 ns

Pipeline Execution Time-
Pipeline execution time= 1 clock cycle= 3 ns

Speed Up-
Speed up= Non-pipeline execution time / Pipeline execution time
= 10 ns / 3 ns
= 3.33

Example-09

Consider an instruction pipeline with four stages (S1, S2, S3 and S4) each with combinational circuit only. The pipeline registers are required between each stage and at the end of the last stage. Delays for the stages and for the pipeline registers are as given in the figure-



What is the approximate speed up of the pipeline in steady state under ideal conditions when compared to the corresponding non-pipeline implementation?

4.0
2.5
1.1
3.0

Non-Pipeline Execution Time-

Non-pipeline execution time for 1 instruction= 5 ns + 6 ns + 11 ns + 8 ns= 30 ns

Cycle Time in Pipelined Processor-
Cycle time= Maximum delay due to any stage + Delay due to its register= Max { 5, 6, 11, 8 } + 1 ns
= 11 ns + 1 ns= 12 ns

Pipeline Execution Time-
Pipeline execution time= 1 clock cycle= 12 ns

Speed Up-

Speed up= Non-pipeline execution time / Pipeline execution time
= 30 ns / 12 ns
= 2.5

Why Pipelining RISC Processors is Easy

- Recall the main principles of RISC:
  - All operands are in registers
  - The only operations that affect memory are loads and stores.
  - Few instruction formats, fixed encoding (i.e., all instructions are the same size)
- Although pipelining could conceivably be implemented for any architecture,
  - It would be inefficient.
  - Pentium has characteristics of RISC and CISC --- CISC Instructions are internally converted to RISC-like instructions.