

# Pipelining: Introduction

# Outline

- Introduction
- What is pipelining?
- How is it implemented?
- Performance Parameters
  - Speedup, Throughput
- Optimal number of stages
- Examples of pipelining
  - Fixed Point Multiplier
  - Floating point adder





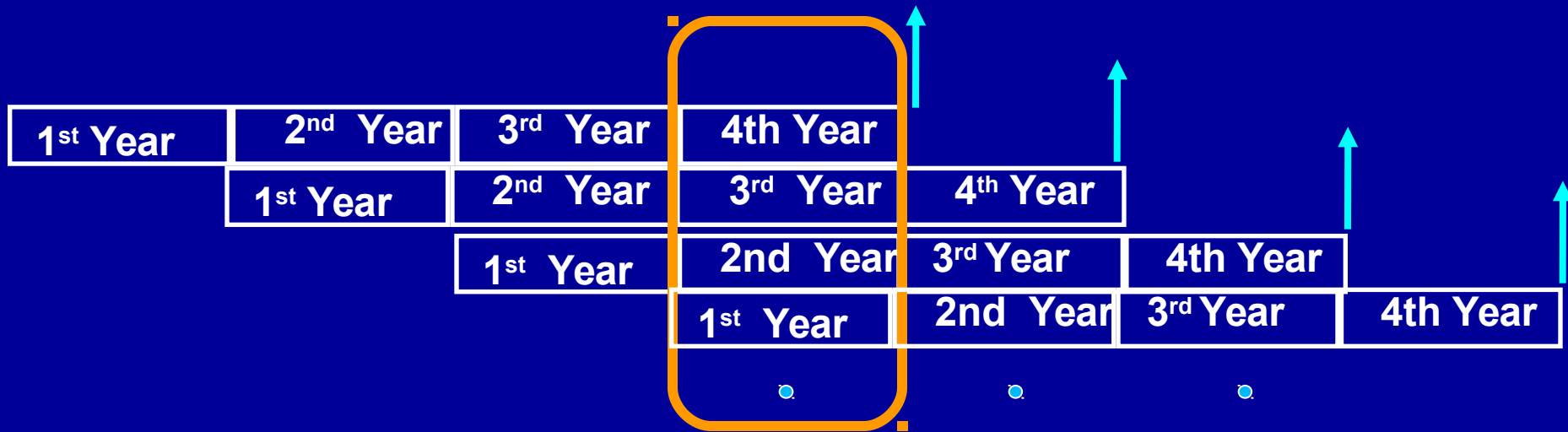
# Another Example

- Consider two alternate ways in which an engineering college can work:
  - **Approach-1.** Admit a batch of students and next batch admitted only after already admitted batch completes (i.e. admit once every 4 years).



# Another Example

- **Approach-2. Admit students every year**
  - In the second approach admit a new batch of students every year
  - Average number of students graduating per year increases four times



# Basic Concepts

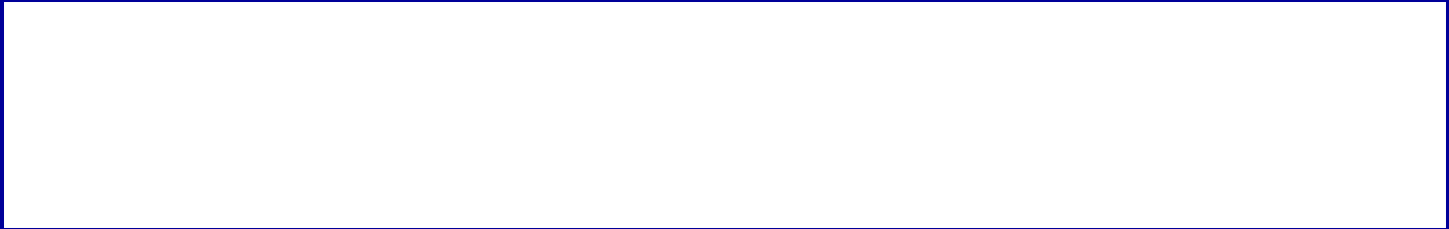
## ➤ What is pipelining?

- It is an implementation technique where multiple tasks are performed in an **overlapped** manner.

## ➤ When can it be implemented?

- It can be implemented when a task can be divided into two or subtasks, which can be performed **independently**.

**Example:** A task takes time  $t$

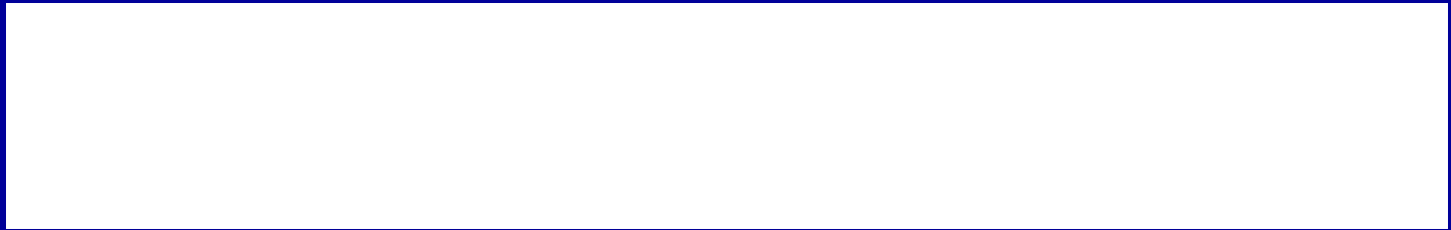


time





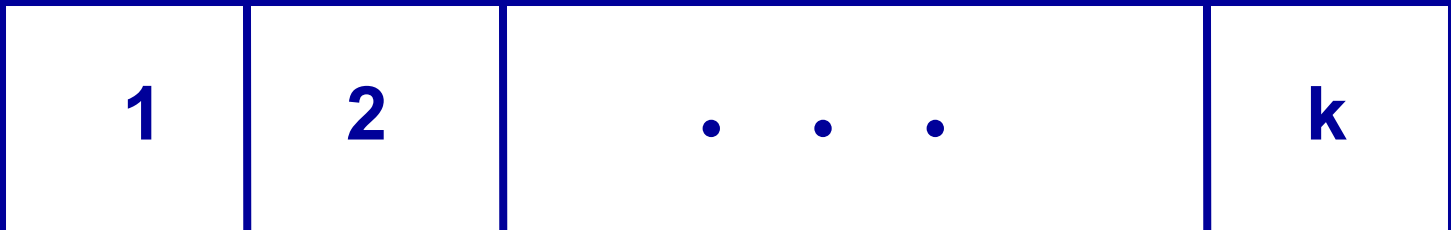
**Example:** A task takes time  $t$



time



The task is divided in  $k$  subtasks



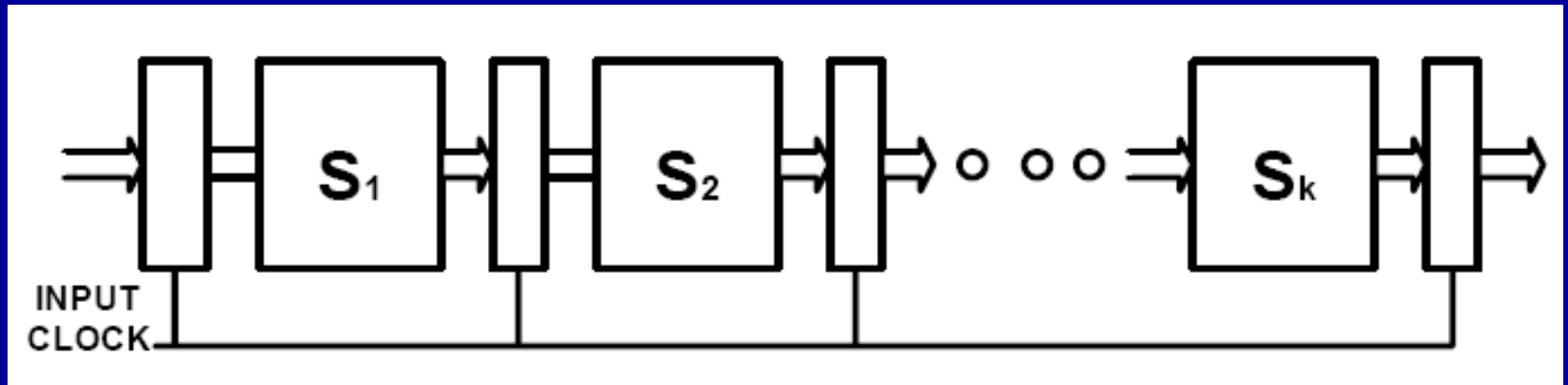
time



# Synchronous Pipeline

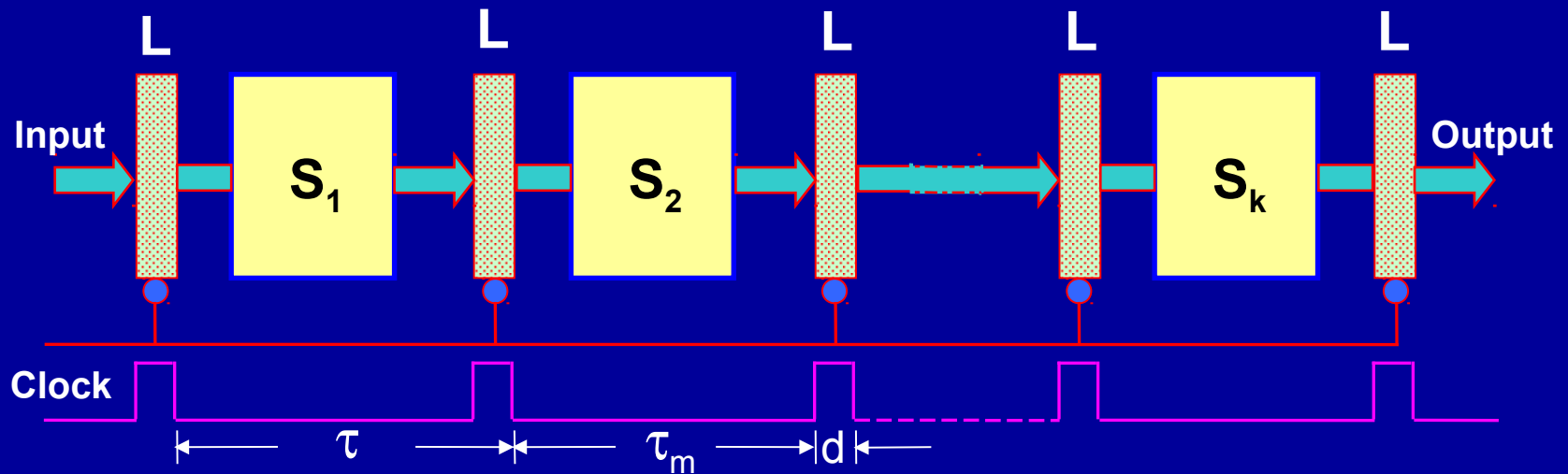
## ➤ How is it implemented?

- Different subtasks are performed by different hardware blocks known as **stages**
- The result produced by each stage is temporarily **buffered** in latches and then passed on to the next stage



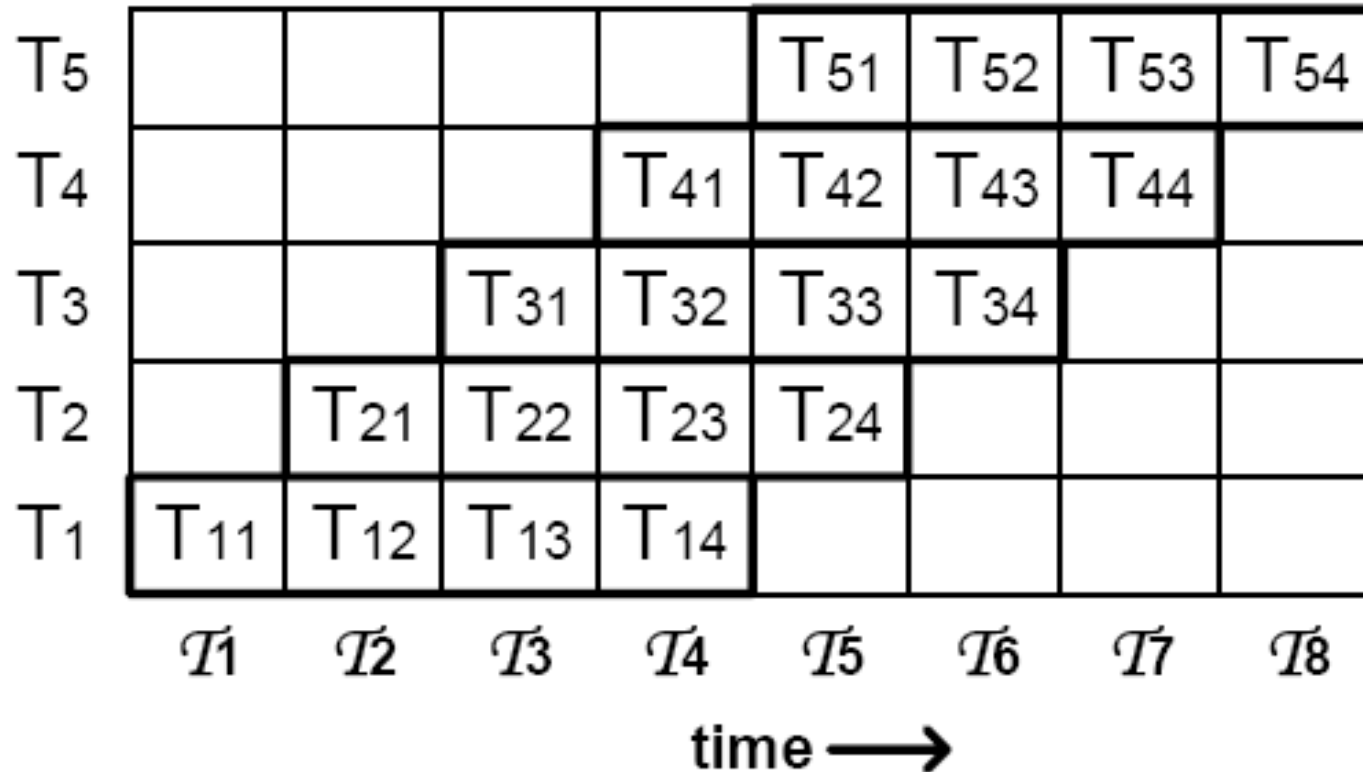
# Synchronous Pipeline

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



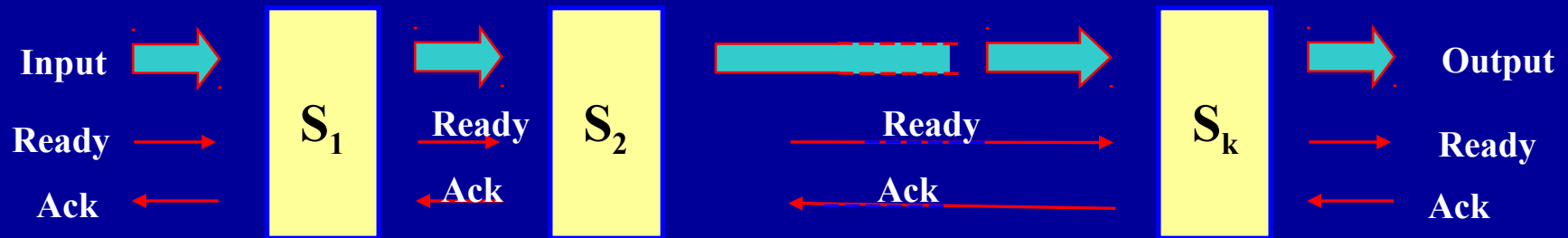
# Synchronous Pipeline

➤ How the tasks are executed?



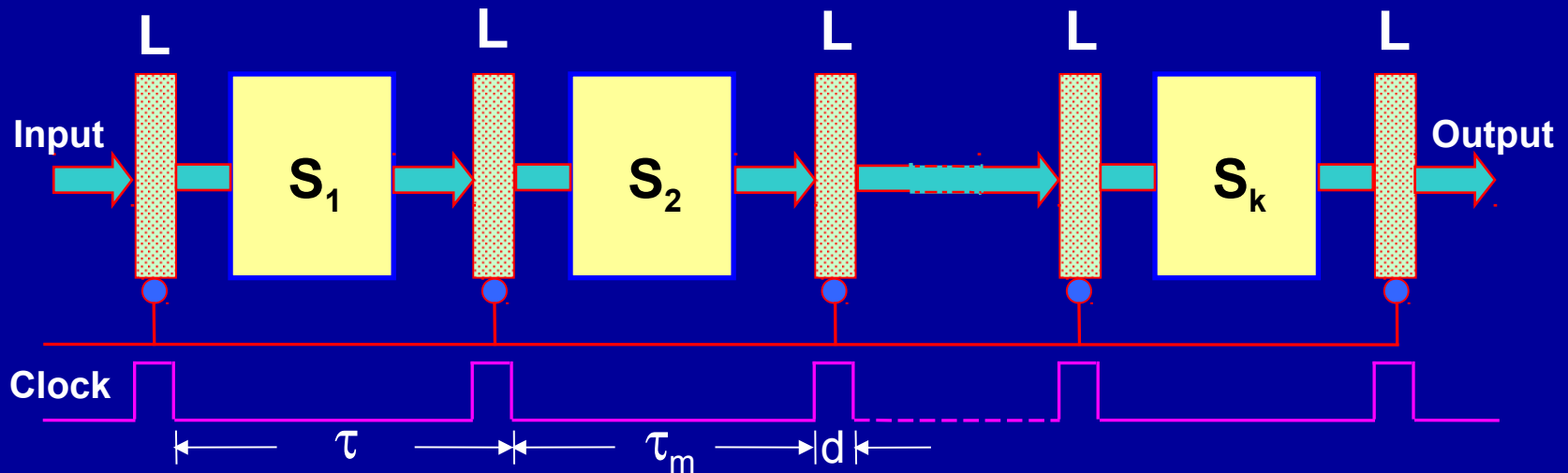
# Asynchronous Pipeline

- 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

# A Few Pipeline Concepts



Pipeline cycle :  $\tau$

Latch delay :  $d$   
 $\tau = \max \{ \tau_m \} + d$

Pipeline frequency :  $f$   
 $f = 1 / \tau$

# 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

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

- For the non-pipelined processor

$$T_1 = n k \tau$$

# Pipeline Speedup Expression

➤ Speedup =

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

- Observe that the memory bandwidth must increase by a factor of  $S_k$
- Otherwise, the processor would stall waiting for data to arrive from memory



# Pipeline Applications

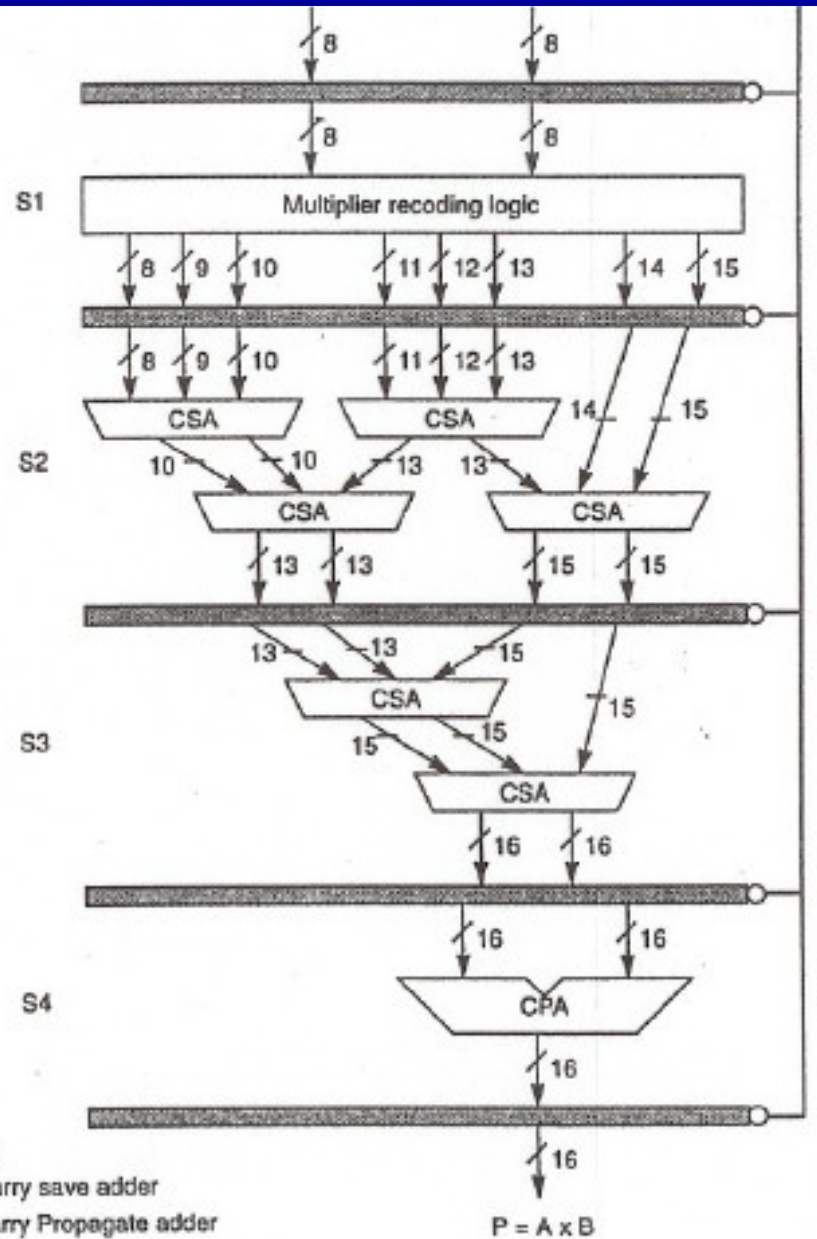
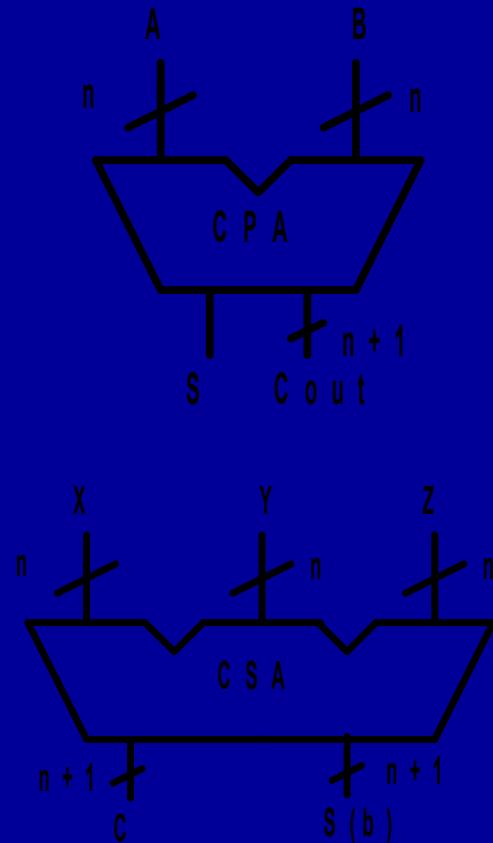
- Historically, there are two different types of pipelines:
  - Arithmetic pipelines
  - Instruction pipelines
- Arithmetic pipelines (e.g. FP multiplication) are not popular in general purpose computers:
  - Needs a continuous stream of arithmetic operations
  - e.g. Vector processors operating on an array
- On the other hand instruction pipelines being used in almost every modern processor
- Computers execute billions of instructions, so instruction **throughput** is what matters

# Pipelined Fixed Point Multiplier

# Pipelined Fixed Point Multiplier

									1	0	1	1	0	1	0	1	=	$A$
								×)	1	0	0	1	0	0	1	1	=	$B$
									1	0	1	1	0	1	0	1	=	$P_0$
							1		0	1	1	0	1	0	1	0	=	$P_1$
						0	0		0	0	0	0	0	0	0	0	=	$P_2$
					0	0	0		0	0	0	0	0	0	0	0	=	$P_3$
				1	0	1	1		0	1	0	1	0	0	0	0	=	$P_4$
			0	0	0	0	0		0	0	0	0	0	0	0	0	=	$P_5$
		0	0	0	0	0	0		0	0	0	0	0	0	0	0	=	$P_6$
	+	1	0	1	1	0	1		0	1	0	0	0	0	0	0	=	$P_7$
0	1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	=	$P$

# Pipelined Fixed Point Multiplier



# Pipelined Floating Point Adder

# Pipelined Floating Point Adder

Floating point number is represented by  $N = (-1)^s F \times 2^E$

Addition is performed in four steps

**Step 1:** Adjustment of the significand of the number with lesser exponent to match the larger exponent:

$8.96 \times 10^1 + 48.6 \times 10^{-1} \rightarrow 8.96 \times 10^1 + .486 \times 10^1$

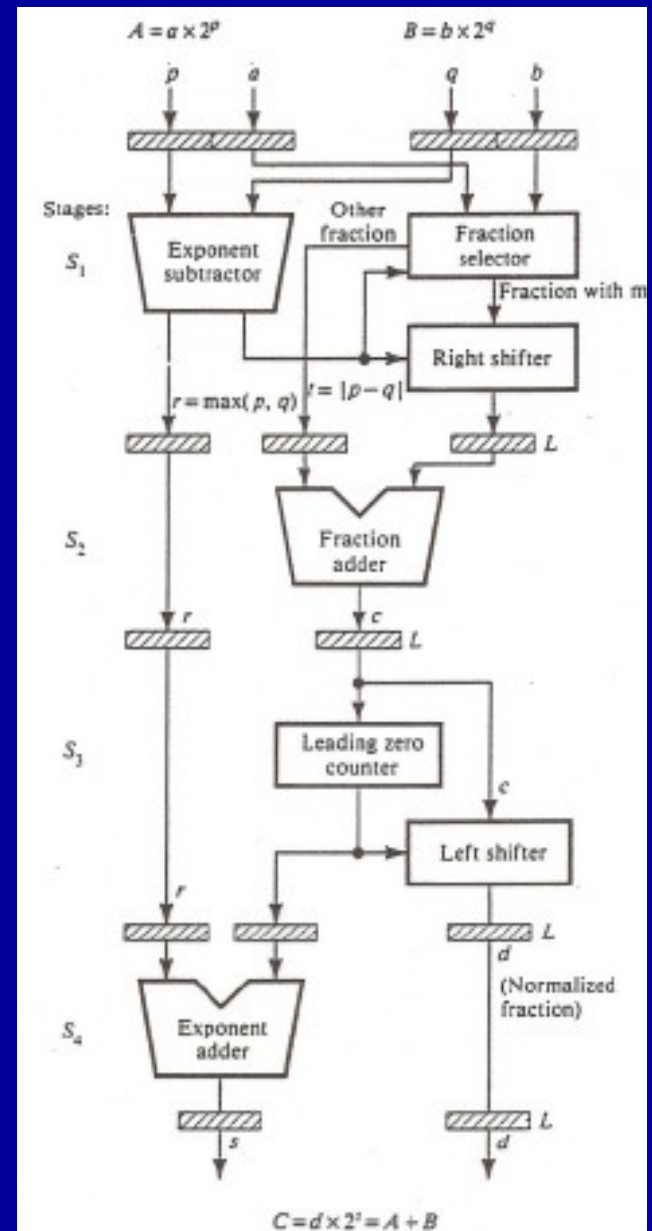
**Step 2:** Add the significands

$8.96 + 0.486 \rightarrow 9.246$

**Step 3:** Normalize the sum

$9.246 \times 10^1 \rightarrow .9246 \times 10^2$

**Step 4:** Round off the sum



# Conclusion

- Introduced the basic concepts of pipelining
- What is pipelining?
  - It is an implementation technique where multiple tasks are performed in an **overlapped** manner
- When can it be implemented?
  - It can be implemented when a task can be divided into two or subtasks, which can be performed **independently**
- Observed that the time required to perform an individual task does not **decrease**, but **throughput** increases
- Examples of pipelining considered
  - Fixed point multiplier
  - Floating point adder

# Pipeline Performance Parameters

- **Clock Period**
  - $\tau = \text{Max} \{ \text{time delay of a stage} \}_1^k + \text{other delays}$
- **Frequency**
  - Reciprocal of the clock period  $f = 1/\tau$
- **Speedup**
  - k stage pipeline, n tasks
  - $S_k = \frac{n \cdot k}{k + (n - 1)} \rightarrow k \text{ when } n \gg k.$
- **Efficiency**
  - Ratio of its actual speedup to the ideal speedup  $\eta = S_k/k$
- **Throughput**
  - Number of instructions that can be completed per cycle  $w = \eta/\tau$



Q&A:

Q 1: Explain the concept of instruction pipeline.

Q 2: How do you evaluate the performance enhancement of a pipeline processor with  $d$  number of phases with respect to a processor without pipeline?

Q 3: Why is a two stage instruction pipeline unlikely to cut the instruction cycle time in half, compared with the use of no pipeline?

Q 4: Explain the concept of delayed branching technique.

Q 5: What is a loop buffer? How loop buffer is used to handle the branching in pipeline processor?

Thanks!