CSC 808: ADVANCED COMPUTER ARCHITECTURE

Prepared by: Assoc. Prof. O. N. Akande

MARCH 3, 2025

Computer Science Department, Nile University, Abuja

Table of Contents

Lesson 1: Introduction to Advanced Computer Architecture

Overview of Computer Architecture

Computer architecture refers to the design and organization of a computer system, focusing on how various components interact to execute instructions efficiently.

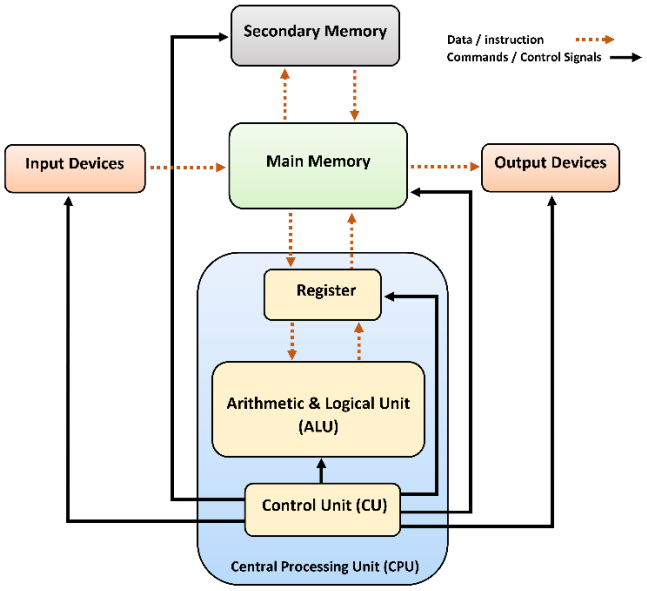


Figure 1: Block Diagram of a Basic Computer with Uniprocessor CPU.

Therefore, the study of computer architecture encompasses the study of the design and development of the hardware, Instruction Set Architecture (ISA), and system-level integration (these are called the components of computer architecture) to optimize performance, power consumption, and cost.

Components of Computer Architecture

(a). **Instruction Set Architecture (ISA):** Defines the set of machine instructions a processor can execute, including data types, registers, addressing modes, and execution control.

(b). **Microarchitecture:** The internal design of a processor, including pipeline structure, execution units, and cache organization, which determines how instructions are processed.

(c). **Memory Hierarchy:** captures how storage components (registers, cache, RAM, and secondary storage) are organized to balance speed and capacity efficiently.

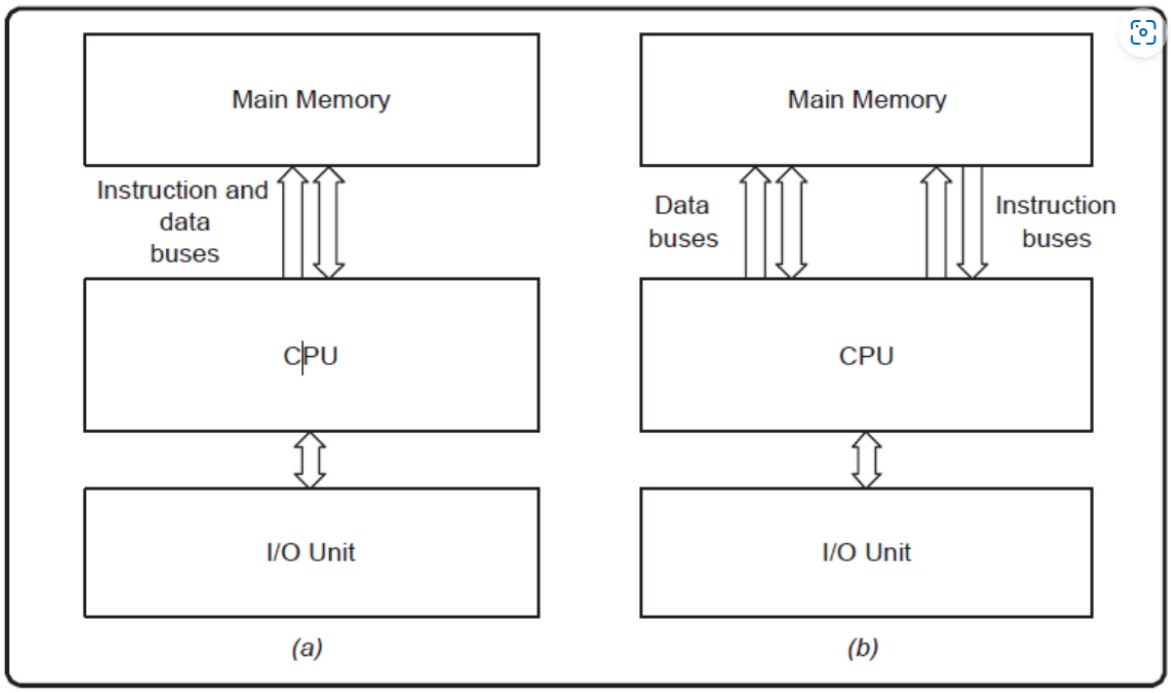
(d). **Input/Output (I/O) System:** Manages communication between the processor, memory, and peripherals, ensuring seamless data transfer.

(e). **Control Unit & Data Path:** Directs the operation of the processor, handling instruction decoding, execution, and data movement.

Evolution of Computer Architecture

(a). **Von Neumann Architecture (1940s):** Uses a **single memory** for both data and instructions. Instructions and data share the same communication bus.

(b). **Harvard Architecture (1940s):** Uses **separate memory** for data and instructions, allowing simultaneous access to both. This architecture is often used in real-time processing or low-power applications



Examples of Computer Architecture: Von Neumann Architecture (a) and Harvard Architecture (b)

Table 1: Comparison of **von Neumann** and **Harvard** Computer Architectures

Feature	Von Neumann Architecture	Harvard Architecture

Memory Structure	Shared memory for instructions and data	Separate memory for instructions and data
Data Transfer	Uses a single bus for both data and instructions	Uses separate buses for data and instructions
Speed	Slower due to potential bottlenecks	Faster due to parallel memory access
Complexity	Simpler and cost-effective	More complex and costly
Used In	General-purpose computers (PCs, servers)	Embedded systems, DSPs, microcontrollers
Modification of Instructions	Can modify instructions at runtime (self-modifying code)	Generally, does not allow self-modifying code
Power Consumption	Higher due to frequent memory access contention	Lower as separate buses optimize energy efficiency
Example Architectures	Intel x86, ARM Cortex-A series	AVR, DSP processors, ARM Cortex-M series
Application Areas	widely used in general-purpose computing due to its simplicity and ease of implementation.	Preferred in embedded systems, DSPs, and real-time applications due to its speed and efficiency.

Key Differences

i. **Memory Bottleneck**: Von Neumann architecture suffers from the **von Neumann bottleneck** because the processor cannot fetch an instruction and read/write data at the same time. Harvard architecture avoids this by using separate buses.

ii. **Flexibility**: Von Neumann allows programs to modify their own instructions dynamically, whereas Harvard architectures often have read-only instruction memory.

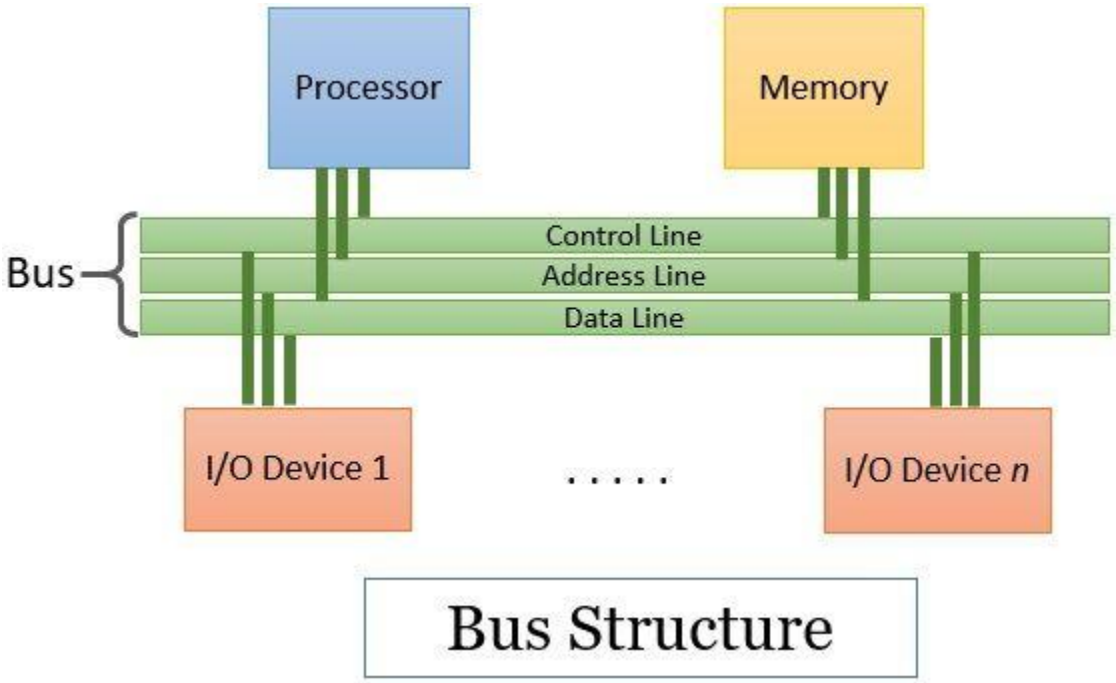
iii. **Performance**: Harvard architecture is generally **faster** because it allows simultaneous instruction and data fetching, making it ideal for real-time applications.

(c). **Complex Instruction Set Computing (CISC) Architectures:** Employs complex instructions, reducing the number of required instructions but increasing processing time per instruction.

(d). **Reduced Instruction Set Computing (RISC) Architectures:** Utilizes simpler instructions with a higher execution rate, optimizing pipeline efficiency.

(e). **Parallel and Distributed Architectures:** Enable simultaneous processing via multi-core processors, GPUs, and cloud computing models.

Bus Structure in Computer Organization



Bus Structure

In computer organization, a system bus is a set of electronic cables that connect key computer components, such as the CPU, memory, and input/output (I/O) devices. It serves as a communication pathway for transferring data and signals between these parts. The efficiency of the system bus plays a crucial role in overall system performance. A faster, more reliable bus allows for quicker communication between components, enhancing the overall speed and responsiveness of the computer.

(a). Data Bus

The data bus is responsible for carrying the actual data between the CPU, memory, and peripherals. The data bus is bidirectional, meaning it can carry data in both directions from the CPU to memory or an I/O device, and vice versa. This allows for the transfer of both read and write operations.

(b). Address Bus

The address bus is responsible for carrying the addresses of memory locations or I/O devices where data is to be read. It helps the CPU identify the location of the data in memory. The address bus is unidirectional, meaning data only flows in one direction from the CPU to memory or an I/O device.

(c). Control Bus

The control bus carries control signals that synchronize the actions of all components in the system. It acts as a coordination mechanism for different devices, ensuring that data is transferred at the right time and that the CPU and other components are working in harmony without conflicts. Typical control lines include.

i. **Memory Write:** Causes data on the bus to be written into the addressed location.
ii. **Memory Read:** Causes data from the addressed location to be placed on the bus.
iii. **I/O Write:** Causes data on the bus to be output to the addressed I/O port
iv. **I/O Read:** Causes data from the addressed I/O port to be placed on the bus.
v. **Transfer ACK:** Indicates that data have been accepted from or placed on the bus.
vi. **Bus Request:** Indicates that a module needs to gain control of the bus.
vii. **Bus Grant:** Indicates that a requesting module has been granted control of the bus.
viii. **Interrupt Request:** Indicates that an interrupt is pending.
ix. **Interrupt ACK:** Acknowledge that the pending interrupt has been recognized.
x. **Clock:** Used to synchronize operations.
xi. **Reset:** Initializes all modules

Future Trends of Bus Structure in Computer Graphics

As technology advances, bus structures are evolving to meet the increasing demands of modern computing. Some key trends in bus architecture include:

(a). Increased Bandwidth

The need for faster data transfer speeds is pushing the development of higher-bandwidth buses, such as PCIe Gen 4 and Gen 5, and future iterations of USB, which will support faster data rates for applications like virtual reality, 4K/8K video, and large-scale data processing.

(b). Integration of Buses

More integration of buses into single chips or circuits is occurring. For example, the advent of system-on-chip (SoC) designs has led to the integration of memory controllers, peripheral interfaces, and communication buses all within a single chip. This reduces complexity, power consumption, and size.

(c). Wireless Communication

Wireless buses are becoming increasingly popular for data transfer between devices. Technologies like Wi-Fi, Bluetooth, and 5G are being integrated into bus architectures for short- and long-range communication, reducing the need for physical cabling and enabling mobile computing.

(d). Low-Power Bus Systems

With the growing focus on energy efficiency, there is an increasing demand for low-power buses. Advanced power management features are being incorporated into bus systems to reduce power consumption, especially in mobile devices and IoT (Internet of Things) devices.

(e). Optical Buses

Optical buses, which use light instead of electrical signals, are emerging as a way to achieve extremely high-speed data transfer. These buses can reduce signal interference and improve data transfer rates significantly over long distances, especially in high-performance computing environments.

Bus Transmission Modes

Buses can be classified into **synchronous** and **asynchronous** types based on how they manage timing and data transfer.

(a). Synchronous Bus

A synchronous bus operates using a shared clock signal that synchronizes all data transfers. Every device connected to the bus follows the same timing sequence dictated by the clock. Therefore, all data transfers occur at **fixed time intervals** based on the clock signal. A **clock cycle** determines when data is placed on the bus and when it is read. It is used in **high-speed systems** where precise timing is required. It consumes more power due to continuous clock operation.

(b). Asynchronous Bus

An asynchronous bus does not use a shared clock signal. Instead, data transfers are controlled by handshaking signals between sender and receiver. It uses request (REQ) and acknowledge (ACK) signals to control data flow. Therefore, each device can transfer data at its own pace, making it flexible. Common in systems where devices operate at different speeds. It is more flexible as it allows devices with different speeds to communicate. It also consumes less power since the bus is only active during data transfers. Used in Universal Serial Bus (USB), Inter-Integrated Circuit (I^2C) and Serial Communications (RS-232).

Table 2: Synchronous vs Asynchronous Bus

Feature	Synchronous Bus	Asynchronous Bus
Clock Signal	Uses a common clock	No common clock; uses handshaking
Speed	Faster due to synchronized operation	Slower due to extra control signals
Flexibility	Less flexible; all devices must operate at the same clock speed	More flexible; different speed devices can communicate
Complexity	Easier to implement	More complex due to control logic
Power Usage	Higher (clock always running)	Lower (bus active only during transfer)
Application Areas	are best for **high-speed, predictable** data transfers where all components work at the same clock speed (e.g., RAM, CPU buses).	are better for **flexible, slower** data transfers where devices operate at different speeds (e.g., USB, serial communication).

Performance Metrics for Computer Architectures

Performance metrics are used to evaluate and compare the efficiency of a computer system. The key metrics include clock speed, CPI, MIPS, FLOPS, Throughput, and Latency.

(a). **Clock Speed:** Measured in GHz, indicates the number of cycles a processor can execute per second.

(b). **Cycles Per Instruction (CPI):** Defines the average number of cycles required to execute an instruction. It is calculated thus:

$$\text{CPI} = \frac{Total\ Cycles}{Total\ Instructions\ Executed}$$

Lower CPI values indicate better performance, as fewer cycles per instruction mean faster execution.

(c). **Million Instructions Per Second (MPIS):** Measures how many millions of instructions a processor executes per second. It is calculated thus:

$$\text{MIPS} = \frac{Instruction\ Count}{Execution\ TIme\ X\ 10^6}$$

Higher MIPS values generally indicate better performance, but it depends on the instruction set and workload.

Example:

A processor has a clock speed of 3 GHz and executes a program with 600 million instructions. The total number of clock cycles required to execute the program is 1.8 billion cycles.

(a) Calculate the CPI (Cycles Per Instruction).

(b) Calculate the MIPS (Million Instructions Per Second).

Solution:

(a) Calculating Cycles Per Instruction

The formula for **CPI** is:

$$\text{CPI} = \frac{Total\ Cycles}{Total\ Instructions\ Executed}$$

Substituting the given values:

$$\text{CPI} = \frac{1.8\times 10^9}{600 \times 10^6} = 3$$

So, the **CPI is 3**, meaning each instruction takes an average of 3 clock cycles to execute.

(b) Calculating Million Instructions Per Second (MIPS)

The formula for **MIPS** is:

$$\text{MIPS} = \frac{Instruction\ Count}{Execution\ TIme\ X\ 10^6}$$

Substituting the given values:

$$\text{MIPS} = \frac{3 \times 10^9}{3 \times 10^6}\ 1000$$

So, the processor runs at **1000 MIPS**, meaning it executes **1000 million (or 1 billion) instructions per second**.

(d). **Floating-Point Operations Per Second (FLOPS):** Measures the number of floating-point arithmetic operations a system can perform per second. Higher FLOPS values indicate better performance for floating-point calculations. It is calculated thus:

$$\text{FLOPS} = \frac{Number\ of\ Floating{-}Point\ Operations}{Execution\ TIme\ (seconds)}$$

If the processor's clock speed and floating-point operations per cycle are known:

FLOPS = Clock Speed (Hz) × FLOP per cycle × Number of Cores

Where: Clock Speed (Hz) is the processor's frequency (e.g. 3 GHz = 3×10^9 Hz), FLOP per cycle is the number of floating-point operations a single core executes per clock cycle and Number of Cores is the total cores in a multi-core processor.

Example:

A **4-core** processor runs at **2.5 GHz**, and each core executes **8 floating-point operations per cycle**. Its FLOPS is:

$$\textbf{FLOPS} = 2.5 \times 10^9 \times 8 \times 4 = 80 \times 10^9 = 80\ \text{G FLOPS}$$

This means the processor can perform 80 billion floating-point operations per second (80 GFLOPS). FLOPS is a key metric in scientific computing, AI, and high-performance computing (HPC).

(e). **Throughput:** Measures the number of instructions executed per unit time.

(f). **Latency:** The time taken to complete a single task or instruction from start to finish. It is measured in nanoseconds (ns) or milliseconds (ms). Lower latency is desirable, as it means quicker response times in applications like real-time systems and user interactions.

Modern Trends in Computer Architecture

(a). **Multi-Core Processors:** specifically designed to enhance parallel processing by integrating multiple processing units on a single chip.

(b). **GPUs and AI Accelerators:** specifically designed to optimize computing for graphics rendering and deep learning applications.

(c). **Quantum Computing:** Uses quantum bits (qubits) to perform complex computations beyond classical capabilities.

(d). **Edge and Cloud Computing:** specifically designed to shift processing power to distributed systems for efficiency and scalability.

TASK

The performance of two different computers A and B (having similar architecture) are being compared by a consultant as part of the evaluation process, computer A operates at 100MHz clock and gives 100 MIPS whereas Computer B operates at 120MHz clock and give 80 MIPS. Due to various reasons Computer B was chosen by the consultant. He also came out with few suggestions for improving the performance of Computer B in future design modifications. Some of his suggestions are given below.

(a). Replace the existing main memory with the faster memory

(b). Introduce a small cache memory

(c). Increase the clock frequency to 200 MHz.

Suppose u are asked to select one of these suggestions, keeping the cost as the main factor. Which one will u select (a, b, c).

Lesson 2: Computer Logic Designs

The study of Digital Logic is fundamental to Computer Architecture as it provides the building blocks for designing and understanding modern computing systems. It enables engineers to grasp how logic gates, Boolean algebra, combinational and sequential circuits form the foundation of processors, memory units, and control logic in computers. Digital logic is essential for understanding how CPUs execute instructions, perform arithmetic operations, manage data flow, and communicate with peripherals. In addition, Digital Logic Design is used to develop hardware, such as circuit boards and microchip processors. This hardware processes user input, system protocol and other data in computers, navigational systems, cell phones, street lights, digital lifts or other high-tech systems.

What is Digital Logic?

Digital logic is the foundation of computer architecture and is responsible for the design and operation of all digital circuits that make up a computer system. It involves using binary numbers (0s and 1s) to perform computations, control operations, and store information.

Role of Digital Logic in Computer Architecture

Digital logic is crucial in:

(a). **Processor Design** – Used to build Arithmetic Logic Units, control units, and registers.

(b). **Memory Systems** – Governs how data is stored and retrieved in RAM, cache, and storage.

(c). **Data Communication** – Defines protocols for data transfer within buses and networks.

(d). **Logic Gates & Circuits** – Used in designing circuits for computations and control mechanisms.

Key Components of Digital Logic

(a). **Logic Gates** – Basic building blocks of circuits (AND, OR, NOT, XOR, NAND, NOR).

(b). **Boolean Algebra** – The mathematical framework for logic operations.

(c). **Combinational Logic** – Circuits like multiplexers, decoders, and adders that compute without memory.

(d). **Sequential Logic** – Uses memory elements (flip-flops, registers) to store state information.

(e). **Finite State Machines (FSMs)** – Control logic for processors and embedded systems.

Overview Logic Gates

Logic gates are the fundamental building blocks of digital circuits. They perform basic logical functions that are essential for digital computing and electronic systems. A logic gate takes one or more binary inputs (0s and 1s) and produces a single binary output based on a specific logical operation.

Types of Logic Gates

There are seven basic types of logic gates:

(a). **AND Gate** – Outputs 1 only if all inputs are 1; otherwise, it outputs 0.

Input A	Input B	Output (A AND B)
0	0	0
0	1	0
1	0	0
1	1	1

(b). **OR Gate** – Outputs 1 if at least one input is 1; otherwise, it outputs 0.

Input A	Input B	Output (A OR B)
0	0	0
0	1	1
1	0	1
1	1	1

(c). **NOT Gate (Inverter)** – Outputs the opposite of the input (0 becomes 1, and 1 becomes 0).

Input	Output (NOT Input)
0	1
1	0

(d). **NAND Gate** – The inverse of the AND gate; outputs 0 only when all inputs are 1.

(e). **NOR Gate** – The inverse of the OR gate; outputs 1 only when all inputs are 0.

(f). **XOR Gate (Exclusive OR)** – Outputs 1 when the inputs are different; outputs 0 when they are the same.

Input A	Input B	Output (A XOR B)
0	0	0
0	1	1
1	0	1
1	1	0

(g). **XNOR Gate (Exclusive NOR)** – The inverse of the XOR gate; outputs 1 when the inputs are the same and 0 when they are different.

Input A	Input B	Output (A XNOR B)
0	0	1
0	1	0
1	0	0
1	1	1

Importance of Logic Gates

Logic gates are used in various applications, including:

(a). Computer processors and memory

(b). Digital signal processing

(c). Control systems in embedded devices

(d). Circuit design and automation

Introduction to Boolean Algebra

Boolean Algebra is a mathematical framework used for analyzing and designing digital circuits in computer architecture. It was developed by George Boole in the mid-19th century and provides a way to represent logical operations using binary values (0 and 1). Boolean algebra forms the foundation of logic gates, combinational circuits, and sequential circuits, which are essential for the functioning of computer processors and memory systems.

Key Concepts in Boolean Algebra

(a). **Boolean Variables** – Represented as **A, B, C, etc.**, where each variable can take a value of **0 (False)** or **1 (True)**.

(b). **Basic Logic Operations:**

i. **AND (· or ∧):** Outputs **1** only if both inputs are 1. ($A \cdot B = AB$)

ii. **OR (+ or ∨):** Outputs **1** if at least one input is 1. ($A + B$)

iii. **NOT (¬ or $\bar{}$):** Inverts the input ($\bar{A}$ means NOT A).

(c). **Laws of Boolean Algebra:**

i. **Identity Law:** $A + 0 = A$, $A \cdot 1 = A$

ii. **Null Law:** $A + 1 = 1$, $A \cdot 0 = 0$

iii. **Idempotent Law:** $A + A = A$, $A \cdot A = A$

iv. **Complement Law:** $A + \bar{A} = 1$, $A \cdot \bar{A} = 0$

v. **De Morgan's Theorems:** De Morgan's Theorem states that "an entire function could be complemented by replacing AND operators with OR operator, and vice versa; as well as complementing every variable and literal". The theorem is useful when simplifying expressions in which the product or sum of variables is inverted/complemented.

E.g. $\overline{(X+Y)} = \bar{X} \cdot \bar{Y}$

$\overline{(X.Y)} = \bar{X} + \bar{Y}$

Attempt this: $\overline{(M + \bar{N})\,(\bar{M} + N)}$

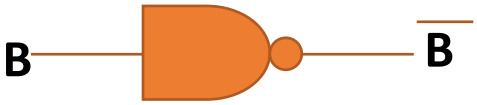
Implementing Logic Functions

This illustrates how to convert/derive one logic gate function from another, especially with NAND and NOR gates because they operate at a higher speed than AND & OR gate

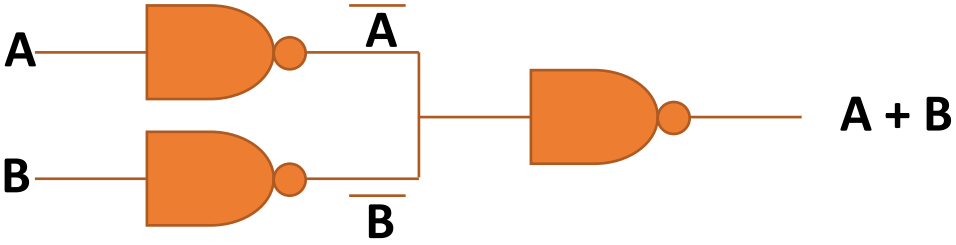
Implementing Logic Functions using NAND Gate

(a). Two NAND gates can be combined together to produce a AND gate

(b). A NOT gate can be derived from a NAND gate



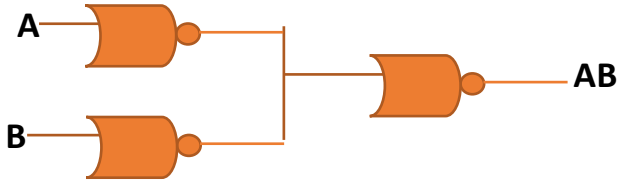
(c). OR gate can be derived from a combination of three NAND gates



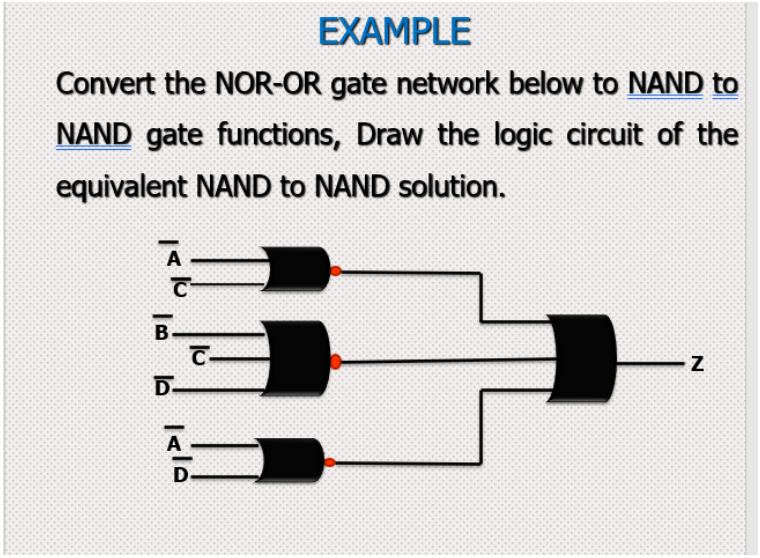
Implementing Logic Functions using NOR Gate

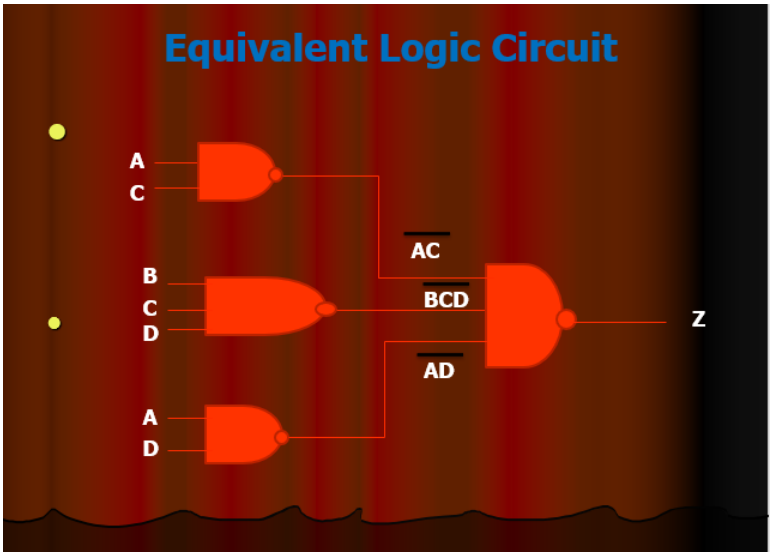
1. Two NOR gates can be combined together to form a OR gate

2. Three NOR gates can be combined together to form a AND gate



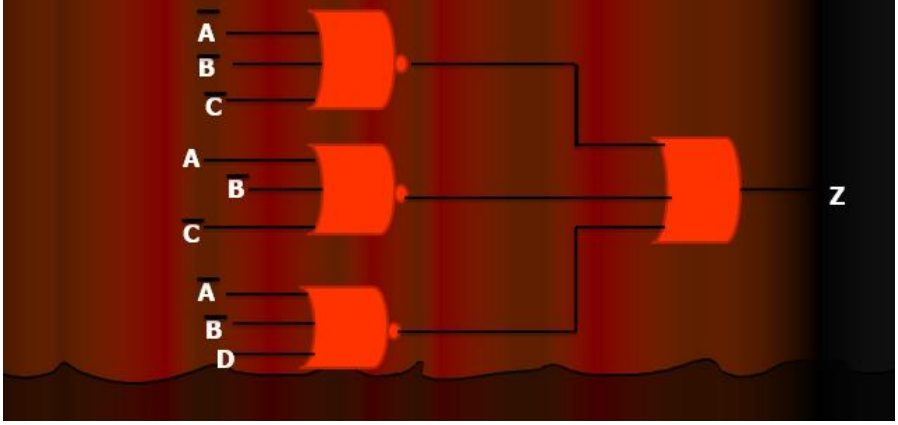
3. A NOT gate can be derived from a NOR gate





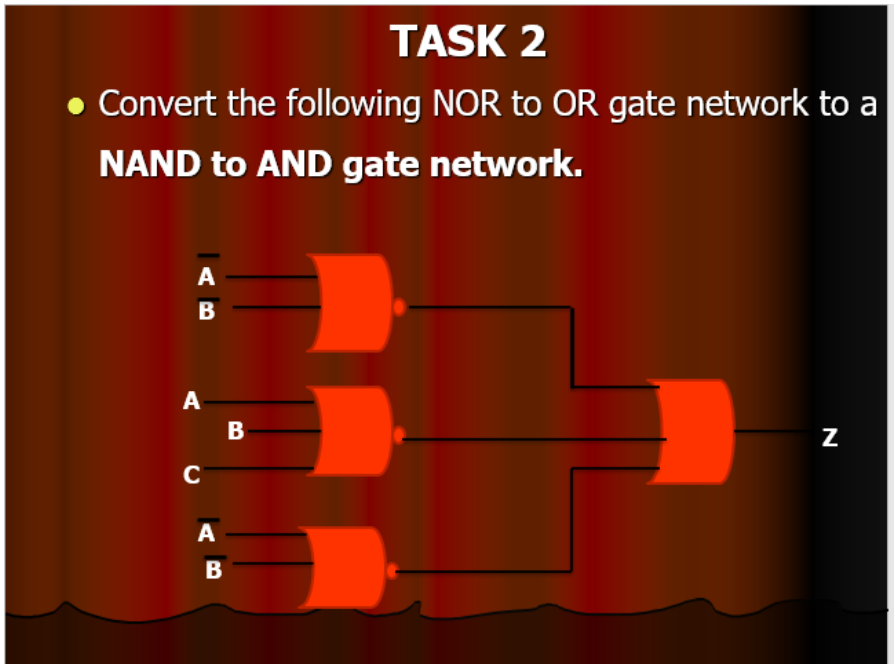
TASK 1

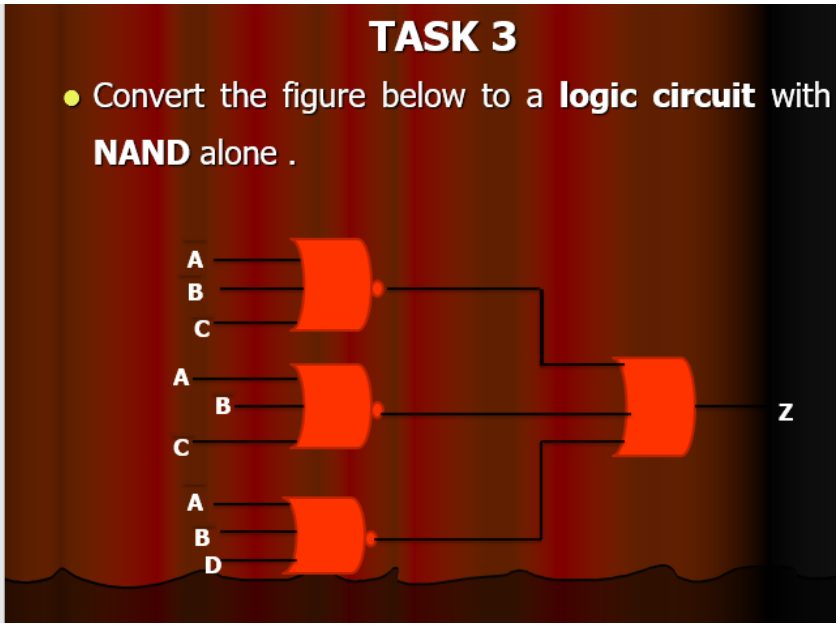
- Convert the figure below to a **logic circuit** with **NAND gates** alone .



TASK 2

- Convert the following NOR to OR gate network to a **NAND to AND gate network.**




TASK 3
Convert the figure below to a logic circuit with NAND alone .
A
B
C
A
B
C
A
B
D
Z

Lesson 3: Combinational Logic Circuit Design

A combinational logic circuit is a type of digital circuit where the output depends *only* on the present inputs and not on any past inputs or states. Unlike sequential circuits, combinational circuits do not have memory or feedback loops. They perform logical operations using basic logic gates and provide an immediate output for any given set of input values.

Basic Characteristics of Combinational Logic Circuits

(a). **No Memory:** The circuit does not store previous inputs; it only responds to current inputs.

(b). **Logical Operations:** It processes inputs based on Boolean algebra and logic gate combinations.

(c). **Immediate Response:** The output is generated as soon as inputs are applied, without any delay due to internal states.

(d). **Deterministic Behavior:** The same set of inputs always results in the same output.

Basic Building Blocks of Combinational Logic Circuits

Combinational logic circuits are built using basic logic gates, such as:

- **AND Gate** – Outputs 1 only when all inputs are 1.
- **OR Gate** – Outputs 1 when at least one input is 1.
- **NOT Gate (Inverter)** – Produces the complement of the input.
- **NAND, NOR, XOR, and XNOR Gates** – Used in various circuit designs to achieve specific logic operations.

Types of Combinational Logic Circuits

(a). **Arithmetic Circuits**

i. **Half Adder:** Adds two single-bit binary numbers.

ii. **Full Adder:** Adds three bits (including a carry-in).

iii. **Half Subtractor & Full Subtractor:** Perform binary subtraction.

iv. **Binary Multipliers:** Multiply binary numbers.

(b). **Data Processing Circuits**

i. **Multiplexer (MUX):** Selects one output from multiple inputs.
ii. **Demultiplexer (DEMUX):** Directs a single input to one of multiple outputs.
iii. **Encoder:** Converts multiple inputs into a smaller binary code.
iv. **Decoder:** Converts binary input into multiple outputs (e.g., BCD-to-7-segment display).

(c). **Comparison and Code Conversion Circuits**

i. **Magnitude Comparator:** Compares two binary numbers and determines equality or magnitude relations.
ii. **Parity Generator & Checker:** Used in error detection in data transmission.

Designing a Combinational Logic Circuit

The process of designing a combinational circuit involves the following steps:

(a). **Problem Definition:** Identify the required logic function and inputs/outputs.

(b). **Truth Table Construction:** List all possible input combinations and their corresponding outputs.

(c). **Boolean Expression Derivation:** Use Boolean algebra to express the desired output.

(d). **Simplification:** Simplify the Boolean expression using techniques like Karnaugh Maps (K-Maps) or algebraic manipulation.

(e). **Logic Gate Implementation:** Use logic gates to implement the simplified Boolean function.

(f). **Verification:** Test the circuit using simulation tools or hardware implementation.

Sum of Products (SOP) and Product of Sums (POS) in Combinational Logic Circuit Design

SOP and POS are two standard ways to express Boolean functions in combinational logic design. They help in simplifying and implementing logic circuits using logic gates.

Sum of Products (SOP)

The Sum of Products (SOP) form is a Boolean expression where multiple AND terms (products) are OR-ed together.

Principle of Operation:

(a). Each AND term represents a minterm (a combination of variables where the function is 1).

(b). The output is 1 for certain input combinations, represented as ANDed terms.

(c). The final expression is a sum (OR) of these products (ANDs).

Example:

Given a truth table with three variables (A, B, C), the SOP expression is derived from the rows where the output is **1**.

Truth Table Example

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

For this function:

- Rows where output = **1** → **($\bar{A}\bar{B}C$), ($\bar{A}BC$), ($A\bar{B}\bar{C}$), (ABC)**
- SOP expression:
  $F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + ABC$

Implementation:

(a). Uses AND gates for each product term.

(b). Uses an OR gate to sum the products.

(c). Preferred for AND-OR gate implementation.

Product of Sums (POS)

The Product of Sums (POS) form is a Boolean expression where multiple OR terms (sums) are AND-ed together.

Principle:

(a). Each OR term represents a maxterm (a combination of variables where the function is 0).

(b). The output is 0 for certain input combinations, represented as ORed terms.

(c). The final expression is a product (AND) of these sums (ORs).

Example:

Using the same truth table, POS is derived from rows where the output is **0**.

Truth Table Example

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

For this function:

(a). Rows where output = **0** → **$(\bar{A} + \bar{B} + \bar{C})$, $(\bar{A} + B + \bar{C})$, $(A + \bar{B} + \bar{C})$, $(A + \bar{B} + C)$**

(b). POS expression:

$F(A, B, C) = (\bar{A} + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})(A + \bar{B} + \bar{C})(A + \bar{B} + C)$

Combinational Network Designs

Example

1. A logic circuit has three inputs X, Y, Z and an output W. The output W gives a **HIGH** signal whenever the combinations of the inputs are all zeros or even. Whenever the combinations of the inputs are odd, a **LOW** output will be produced.

   (a). Draw a truth table to represent this scenario

   (b). Simplify the expression using either POS or SOP rules

   (c). Draw the logic circuit of the simplified expression.

2. An analog-digital converter was designed to monitor the dc voltage of a 12v storage battery on an orbiting spaceship. The converter's INPUT is a four bit binary number A B C D corresponding to the battery voltage in steps of 1v. The converter's binary outputs are fed to a logic circuit that is to produce a HIGH output whenever the binary output is greater than $0110_2 = 6_{10}$ that is when the battery voltage is greater than 6v

   (a). Draw a truth table to represent this scenario

   (b). From the truth table, obtain a simplified SOP expression using Boolean algebra techniques.

   (c). Draw the logic circuit of the simplified SOP expression.

   (d). Design a logic circuit to implement the expression using NAND gates only.

   Solution:

   Truth Table

	A	B	C	D	OUTPUT(Z)
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0

4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

TASK

A logic circuit has **four inputs D, C, B, A** that is encoded in 8421 natural binary form. The inputs in the range 0000=0 to 1011=11 represents the months of the year from JANUARY (0) to DECEMBER (11). Inputs in the range 1100 to 1111 (i.e. 12-15) cannot occur. The output of the circuit is a logical one (1) if the month represented by the input has 31 days, otherwise the input is FALSE.

(a). Draw a truth table to represent this scenario

(b). From the truth table, obtain a simplified SOP expression using Boolean algebra techniques.

(c). Draw the logic circuit of the simplified SOP expression.

(d). Design a logic circuit to implement the expression using NAND gates only.

Simplifying Boolean Functions using Karnaugh-Map

Karnaugh-Map (K-Map) is a graphical method for representing and simplifying Boolean expression. It is a two-dimensional form of simplifying Boolean expression into their simplest forms. 2^n input truth table will have 2^n input K- Map.

SLIDE

Lesson 4: Sequential Logic Circuits Designs

Sequential circuits are digital circuits that store and use the previous state information to determine their next state. Unlike combinational circuits, which only depend on the current input values to produce outputs, sequential circuits depend on both the current inputs and the previous state stored in memory elements. They are commonly used in digital systems to implement state machines, timers, counters, and memory elements and are essential components in digital systems design. Sequential circuits are commonly used in digital systems to implement state machines, timers, counters, and memory elements. The memory elements in sequential circuits can be implemented using flip-flops, which are circuits that store binary values and maintain their state even when the inputs change.

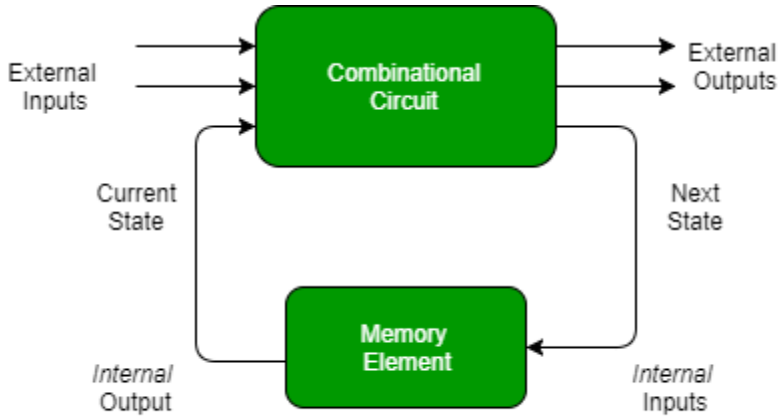


Figure: Sequential Circuit

As shown in the figure, there are two types of input to the combinational logic:

(a). External inputs which are not controlled by the circuit.

(b). Internal inputs, which are a function of a previous output state.

Types of Sequential Circuits

There are two types of sequential circuits

Asynchronous Sequential Circuit

These circuits do not use a clock signal but uses the pulses of the inputs. These circuits are faster than synchronous sequential circuits because there is clock pulse and change their state immediately when there is a change in the input signal. We use asynchronous sequential circuits when speed of operation is important and independent of internal clock pulse.

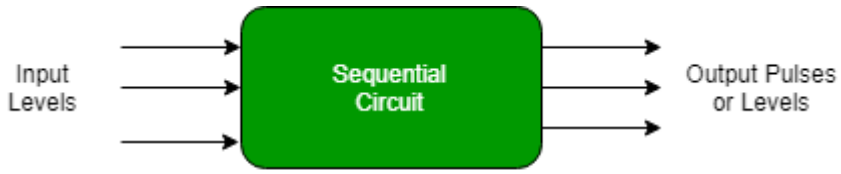


Figure: Asynchronous Sequential Circuit

Synchronous Sequential Circuit

These circuits use clock signal and level inputs (or pulsed) (with restrictions on pulse width and circuit propagation). The output pulse is the same duration as the clock pulse for the clocked sequential circuits. Since they wait for the next clock pulse to arrive to perform the next operation, so these circuits are bit slower compared to asynchronous. Level output changes state at the start of an input pulse and remains in that until the next input or clock pulse. If the output(s) depend only on the state of the FSM, it is called a **Moore machine.** And if the output(s) depend on both the state and the current input(s), it is called a **Mealy machine.**

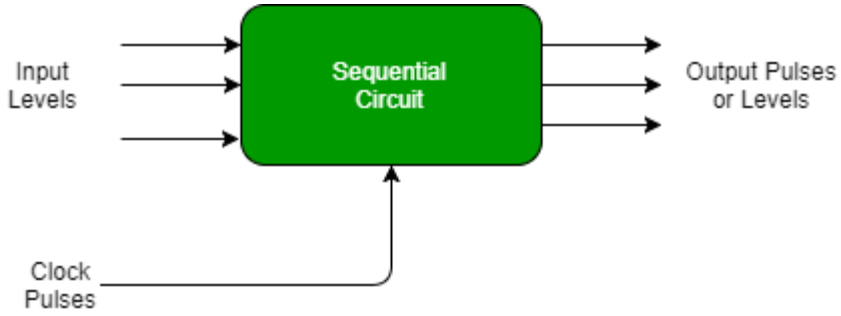


Figure: Synchronous Sequential Circuit

Synchronous sequential circuits are used in synchronous counters, flip flops, and in the design of MOORE-MEALY state management machines.

Clock Pulses

A **clock signal** is used to synchronize operations. It is typically a square wave that oscillates between high (1) and low (0) voltage levels, providing timing for sequential logic components such as flip-flops and registers. The amount of time spent at each level may be unequal. Although not a requirement, the timing pattern is usually uniform. However, most circuits are designed such that a *transition* of the clock signal triggers the circuit elements to start their respective operations.

As shown in Figure 3, there are three common ways a clock signal can be used for triggering:

1. Level-Triggered Clocking

In **level-triggered** circuits, the output responds whenever the clock signal is at a particular **level** (either HIGH or LOW). For example, in **latches**, the circuit is active as long as the clock is HIGH (or sometimes LOW), meaning the output can change continuously while the clock is at that level.

2. Edge-Triggered Clocking

Unlike level-triggered circuits, **edge-triggered** circuits only respond at the moment the clock transitions between states. There are two types:

a. Positive-Edge Triggered

A circuit is **activated** only at the moment the clock transitions from **LOW to HIGH (0 → 1)** as shown in Figure 3(b). This is commonly represented with an upward arrow (↑) in timing diagrams. Most D flip-flops and edge-triggered registers use positive-edge triggering to avoid constant changes while the clock is high.

b. Negative-Edge Triggered

Here, a circuit is **activated** only at the moment the clock transitions from **HIGH to LOW (1 → 0)** as shown in Figure 3(b). This is represented with a downward arrow (↓) in timing diagrams. Some flip-flops and memory elements use negative-edge triggering to operate at the opposite phase of a system.

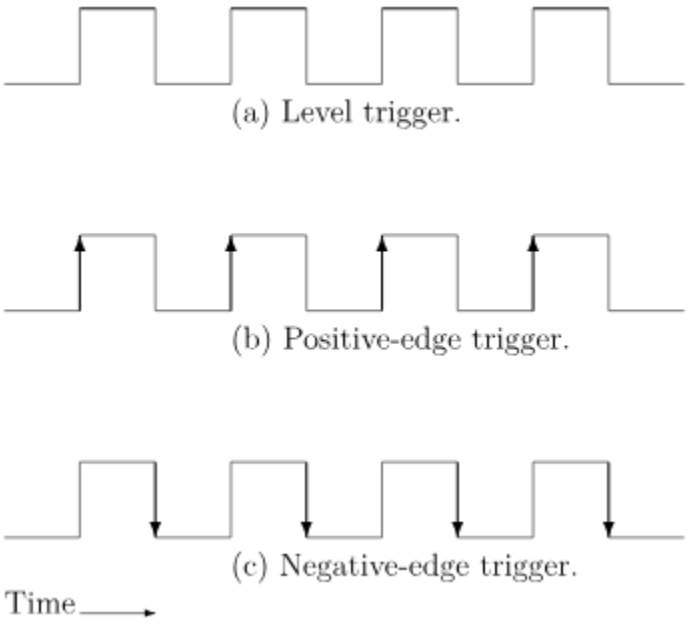


Figure 3**.** Clock signals. (a) For level-triggered circuits. (b) For positive-edge triggering. (c) For negative-edge triggering.

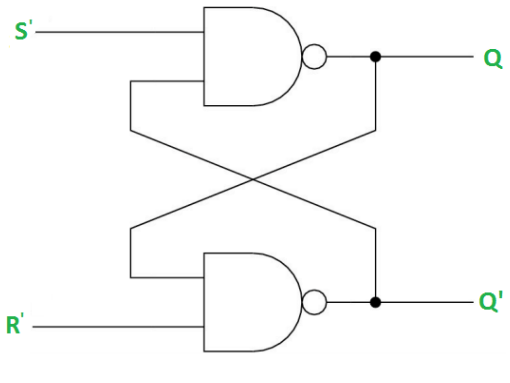
Latches

Latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals. They are level-triggered device used in digital systems as temporary storage elements to store binary information. Latches can be implemented using various digital logic gates, such as AND, OR, NOT, NAND, and NOR gates. Latches are widely used in digital systems for various applications, including data storage, control circuits, and flip-flop circuits. They are often used in combination with other digital circuits to implement sequential circuits, such as state machines and memory elements.

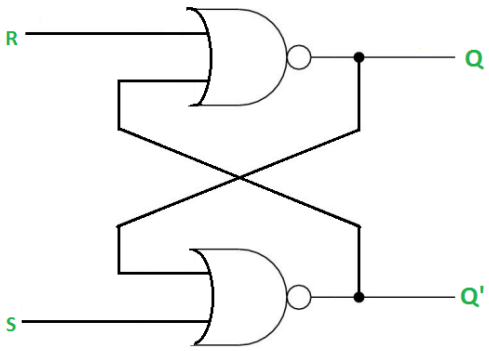
Types of Latches

(a). **SR (Set-Reset) Latch**

These are the simplest form of latches and are implemented using two inputs: S (Set) and R (Reset). The S input sets the output to 1, while the R input resets the output to 0. When both S and R inputs are at 1, the latch is said to be in an "undefined" state. They are also known as preset and clear states. The SR latch forms the basic building blocks of all other types of flip-flops.



a) SR latch using NAND gate



b) SR latch using NOR gate

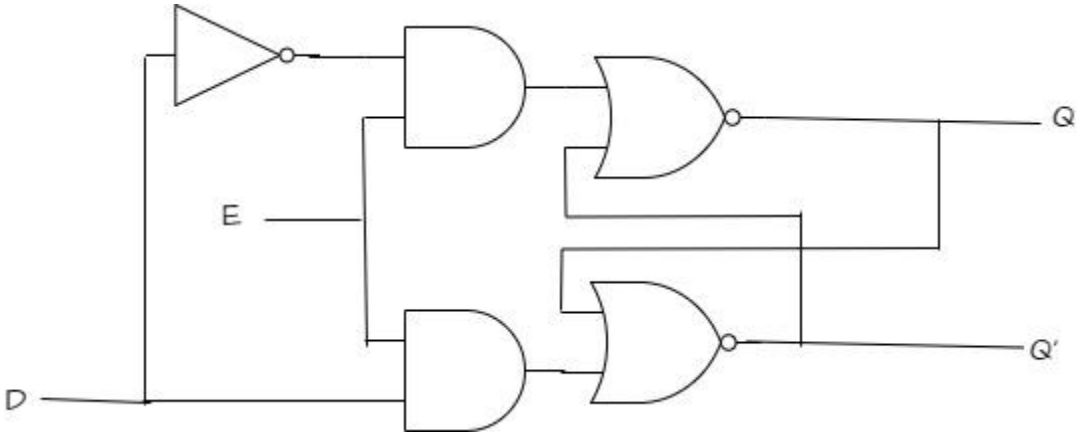
Truth Table of SR Latch

S	R	Q	Q '	Description
0	0	Latch	Latch	Latch/Hold (No Change)
0	1	0	1	Reset (Q = 0)
1	0	1	0	Set (Q = 1)

1	1	0	0	(undefined output)

(b). **D (Data) Latch**

D latches are also known as transparent latches and are implemented using two inputs: D (Data) and a clock signal. The output of the latch follows the input at the D terminal as long as the clock signal is high. When the clock signal goes low, the output of the latch is stored and held until the next rising edge of the clock. The D (Data) Latch is designed to eliminate the invalid state in an SR latch by ensuring that S and R are never both 1 at the same time.



Logic Diagram of D Latch

Truth Table of D Latch

E (Enable)	D (Data)	Q (Next State)	Q ' (Complement)	Description
0	X (Don't Care)	Q (Previous State)	Q' (Previous State)	Latch/Hold (No Change)
1	0	0	1	Reset (Q = 0)
1	1	1	0	Set (Q = 1)

When **E = 0**, the latch **holds** its previous state (Q remains unchanged).

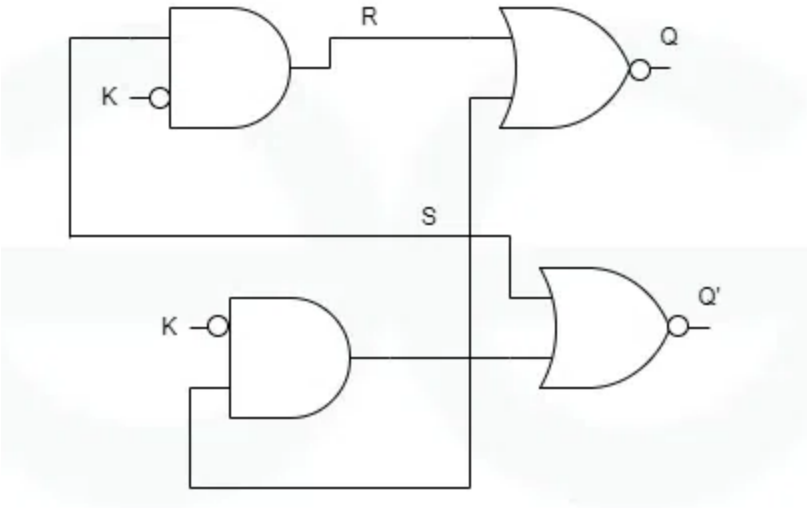
When **E = 1**, the latch **follows the input D**:

i. If D = 0, Q is reset to 0.

ii. If D = 1, Q is set to 1.

(c). **JK Latch**

JK latch has two inputs J and K. The output gets toggled when the J and K inputs are high. JK latch is just like SR latch, but it eliminates the undefined state of SR latch.



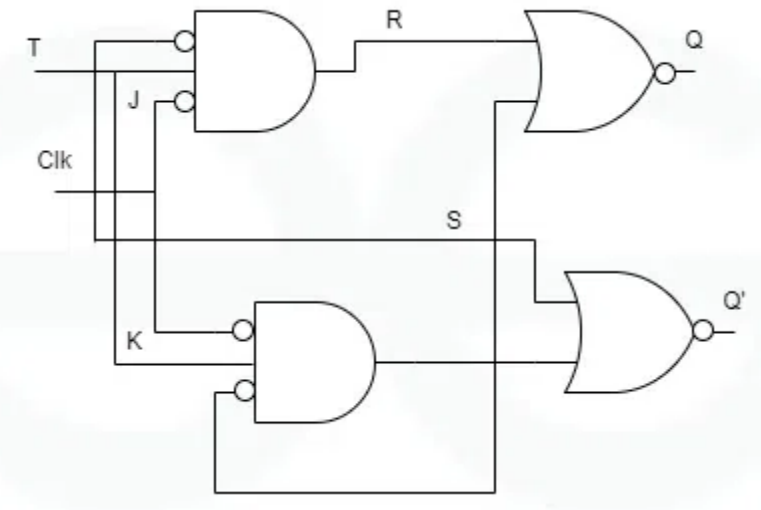
Logic Diagram of JK Latch

Truth Table of JK Latch

J (Set)	K (Reset)	Q (Next State)	Comment
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	Q’	Toggle

(d). **T (Toggle) Latch**

The **T (Toggle) Latch** is a modified version of the **JK Latch**, designed to operate with a single input (**T**) and a control signal (**Enable, E**). It is mainly used in counters and frequency dividers.



Truth Table of T Latch

E (Enable)	T (Toggle)	Q (Next State)	Q ‘ (Complement)	Description
0	X (Don’t Care)	Q (Previous State)	Q‘ (Previous State)	Latch/Hold (No Change)
1	0	Q (Previous State)	Q' (Previous State)	Latch/Hold (No Change)
1	1	Q' (Toggles State)	Q' (Toggles State)	Toggle Q (Flip State)

- When E = 0, the latch holds its previous state (no change in output).
- When E = 1:

  i. If T = 0, the latch maintains its current state.

  ii. If T = 1, the latch toggles (flips Q to its opposite state).

Flip-Flops

A flip-flop is a bistable digital circuit that stores one bit of data and is edge-triggered, meaning it changes state only on the rising or falling edge of a clock signal. Unlike latches, which are level-triggered, flip-flops respond only at specific moments, reducing race conditions and making them more stable for sequential logic applications.

So, flip-flop is similar to Latches, however latches use a level-based clock signal while Flip-Flops are triggered by a clock signal edge.

J	K	Q	Q_{next}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

JK Flip Flop

T	Q	Q_{next}
0	0	0
0	1	1
1	0	1
1	1	0

T Flip Flop

Designing Sequential Circuits

We will now consider a more general set of steps for designing sequential circuits. Design in any field is usually an iterative process, as you have no doubt learned from your programming experience. You start with a design, analyze it, and then refine the design to make it faster, less expensive, etc. After gaining some experience, the design process usually requires fewer iterations.

The following steps form a good method for a first working design:

(a). From the word description of the problem, create a state table and/or state diagram showing what the circuit must do. These form the basic technical specifications for the circuit you will be designing.

(b). Choose a binary code for the states, and create a binary-coded version of the state table and/or state diagram. For N states, the code will need log2N bits. Any code will work, but some codes may lead to simpler combinational logic in the circuit.

(c). Choose a particular type of Latches/flip-flop. This choice is often dictated by the components you have on hand.

(d). Add columns to the state table that show the input required to each flip-flop in order to effect each transition that is required.

(e). Simplify the input(s) to each flip-flop. Karnaugh maps or algebraic methods are good tools for the simplification process.

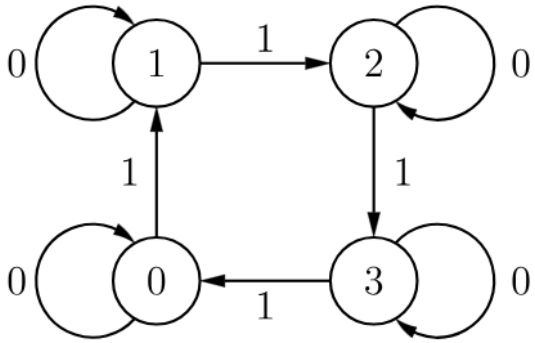
(f). Draw the circuit.

Examples

Design a counter that has an Enable input. When Enable=1 it increments through the sequence 0,1,2,3,0,1,… with each clock tick. Enable=0 causes the counter to remain in its current state.

Step 1: First, we create a state table and state diagram:

	$Enable = 0$	$Enable = 1$
Current	Next	Next
n	n	n
0	0	1
1	1	2
2	2	3
3	3	0



At each clock tick the counter increments by one if Enable=1. If Enable=0 it remains in the current state. We have only shown the inputs because the output is equal to the state.

Step 2: A reasonable choice is to use the binary numbering system for each state. With four states we need two bits. We will let $n=n_1n_0$, giving the state table:

		$Enable = 0$		$Enable = 1$	
Current		Next		Next	
n_1	n_0	n_1	n_0	n_1	n_0
0	0	0	0	0	1
0	1	0	1	1	0
1	0	1	0	1	1
1	1	1	1	0	0

Step 3: Since JK flip-flops are very general we will use those.

Step 4: We need two flip-flops, one for each bit. So, we add columns to the state table showing the input required to each JK flip-flop to cause the correct state transition. Referring to Figure 7.3.16, we see that JK=00 keeps the current state, JK=01 resets it (to 0), JK=10 sets it (to 1), and JK=11 complements the state. We use X when the input can be either 0 or 1.

		$Enable = 0$						$Enable = 1$					
Current		Next						Next					
n_1	n_0	n_1	n_0	J_1	K_1	J_0	K_0	n_1	n_0	J_1	K_1	J_0	K_0
0	0	0	0	0	X	0	X	0	1	0	X	1	X
0	1	0	1	0	X	X	0	1	0	1	X	X	1
1	0	1	0	X	0	0	X	1	1	X	0	1	X
1	1	1	1	X	0	X	0	0	0	X	1	X	1

Notice the “don't care” entries in the state table. Since the JK flip-flop is so versatile, including the “don't cares” helps find simpler circuit realizations.

Step 4: Simplifying the circuits using a Karnaugh maps with E as Enable pin

$J_0(E, n_1, n_0)$

E \ n_1n_0	00	01	11	10
0		X	X	
1	1	X	X	1

$K_0(E, n_1, n_0)$

E \ n_1n_0	00	01	11	10
0	X			X
1	X	1	1	X

$J_1(E, n_1, n_0)$

E \ n_1n_0	00	01	11	10
0			X	X
1		1	X	X

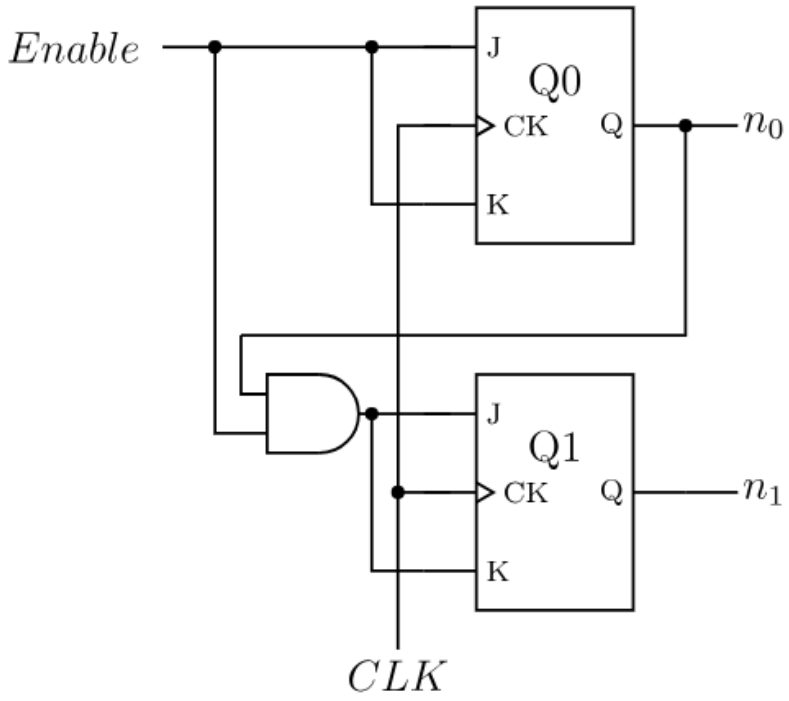
$K_1(E, n_1, n_0)$

E \ n_1n_0	00	01	11	10
0	X	X		
1	X	X	1	

This yields the funtions:

$$J_0(E, n_1, n_0) = E$$
$$K_0(E, n_1, n_0) = E$$
$$J_1(E, n_1, n_0) = E \cdot n_0$$
$$K_1(E, n_1, n_0) = E \cdot n_0$$

The circuit to implement this counter is:



TASK

(a). Design a counter that has an Enable input. When Enable = 1, it decrements through the sequence 3, 2, 1, 0, 3, 2,… with each clock tick. Enable = 0 causes the counter to remain in its current state.

(b). Design a counter that has an Enable input. When Enable = 1, it counts through the sequence 0, 2, 1, 3, 0, 2,… with each clock tick. Enable = 0 causes the counter to remain in its current state.

Designing Sequential Circuits

Sequential Logic Circuits and the SR Flip-flop

Lesson 5: Programming in Assembly Language Using MC68000

Lesson 6: Advances In Bus Architecture

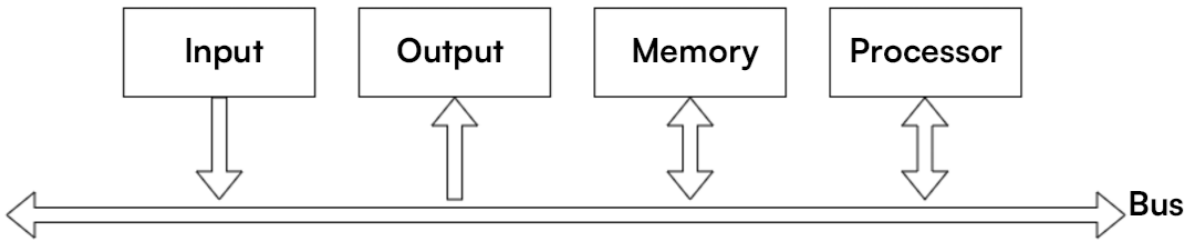
Bus architecture plays a crucial role in computer systems by facilitating communication between different hardware components such as the CPU, memory, and peripheral devices. Over the years, significant advancements have been made to enhance speed, efficiency, scalability, and power consumption. Modern bus architectures aim to address bottlenecks associated with traditional parallel bus designs by incorporating new techniques like serial communication, point-to-point links, and high-bandwidth interfaces.

Traditional Bus Architectures

This includes:

Single Bus System

A Single Bus Structure uses one shared communication line (bus) to transfer data and control signals between various components of the computer system. All devices connected to the bus use the same set of lines for transferring information to and from the processor, memory, and other peripherals.



Single Bus Structure

In the above structure, each component is connected to the bus, and only one device can use the bus at any given time.

The disadvantages of single bus structure are:

i. As all components share the same bus, data transfer speed is slow, causing a potential delay in communication.

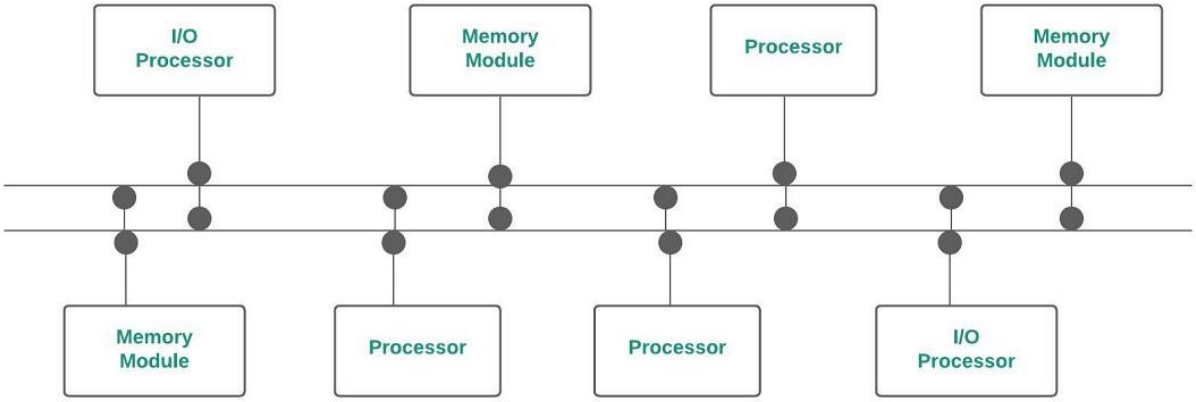
ii. The bus's bandwidth is fixed, and as more devices are added to the system, performance may degrade.

iii. In large systems with many devices, the single bus might not be able to handle the increasing demand for data transmission.

iv. If the bus malfunctions or is damaged, the entire system could fail.

Multi-Bus System

The multiple bi-directional buses means that in the system there are multiple buses that are bi-directional and are used to separate memory access and I/O operations, thereby, improving overall performance. It permits simultaneous transfers as many as buses are available. But here also the complexity of the system is increased.



Disadvantages of Multiple Bi-Directional Buses

(a). **Increased Complexity:** Managing multiple buses that can handle traffic in both directions is technically complex. It also requires advanced control logic to manage bus access, prevent conflicts, and coordinate data flow.

(b). **Bus Contention:** Since buses are bi-directional, multiple devices may try to use the bus at the same time, causing contention. Complex arbitration mechanisms are needed to decide who gets to use the bus at any given moment, which can introduce delays.

(c). **Signal Interference and Timing Issues:** Signals traveling in opposite directions can cause electrical interference. Maintaining signal integrity and timing synchronization becomes difficult, especially as clock speeds increase.

(d). **Cost:** Implementing multiple bi-directional buses requires more wiring/traces on the motherboard or chip, more sophisticated controllers and Higher manufacturing costs.

(e). Power Consumption: More buses connote more active components. Managing signals in both directions increase power usage, which can be critical in battery-powered systems like laptops and mobile devices.

(f). **Scalability Challenges:** Adding more devices or components becomes harder without overloading the buses. The system may need even more buses or more complex arbitration as it scales, which adds to the design challenges.

(g). **Debugging and Maintenance:** Troubleshooting data errors or performance issues is harder with multiple bi-directional paths. Fault isolation (finding out which device or bus is causing a problem) is more complicated.

Disadvantage	Explanation
Complexity	More difficult to design and manage.
Contention	Devices competing for bus access.
Interference	Risk of signal clashes and noise.
Cost	Increased hardware and design expense.
Power Usage	Higher energy consumption.
Scalability	Harder to expand the system.
Maintenance	More challenging to debug.

Synchronous vs. Asynchronous Buses

Synchronous bus architecture uses a clock signal to synchronize data transfer between devices in a computer system. The clock signal is generated by a master device and is used to control all slave devices connected to the bus.

Asynchronous bus architecture is a way for devices to communicate without a clock signal. Instead, devices signal each other to indicate when they are ready to transfer data.

Refer to Lesson one for a recap about bus structure

Recent Advancements in Bus Architecture

High-Speed Serial Buses

Traditional parallel buses suffer from signal degradation and crosstalk at high speeds. Newer architectures use serial transmission to reduce these issues while increasing bandwidth.

High-speed serial buses send encoded data that contains both data and clocking information in a single differential signal, so engineers can avoid the speed limitations in parallel buses. Today, high-speed serial links with data lanes running at 10 Gbps are common. Additionally, multiple

lanes of serial links can be coherently bonded together to form communication links with higher data throughputs.

By serializing the data and sending it at faster speeds, the number of pin counts in integrated circuits (ICs) can be reduced, which helps decrease device size. Furthermore, because the serial lanes can operate at a much faster clock speed, you can achieve better data throughput than that possible with parallel buses. Examples: PCI Express (PCIe), USB, SATA, and Thunderbolt.

Point-to-Point Communication

Unlike shared bus systems, point-to-point architectures provide dedicated links between components, minimizing contention and improving throughput. Example: PCIe, where each device has a dedicated lane for communication with the CPU. Imagine a traditional bus system as one big classroom where everyone must take turns to speak. Now, imagine point-to-point as private video calls between two people—no waiting your turn, no interruptions, and much faster communication.

Traditional vs Point-to-Point

Traditional Bus	Point-to-Point Bus
Shared by all devices	Dedicated link between two devices
Slower speeds	Ultra-high speeds
Contention problems	No contention
Hard to scale	Easy to scale with more links
Older tech (parallel)	Modern (serial, high-speed)

Cache-Coherent Interconnects

In modern multi-core and multi-processor systems, each core often has its own private cache. However, when multiple cores access or modify shared data, inconsistencies can occur if changes in one cache are not visible to others. This is known as the cache coherence problem. However, a Cache-coherent interconnects are advanced bus architectures that ensure all processors and cores see the most recent, consistent view of shared data across their caches. They use built-in protocols to automatically manage data consistency.

By maintaining a unified, coherent memory space, it improves performance, power efficiency, and reduces software complexity, allowing efficient data sharing without relying on software management of data movement between components. As systems have evolved from single-core

to multi-core and many-core processors, the need to maintain coherent data across multiple caches became critical. Traditional buses couldn't handle this complexity efficiently.

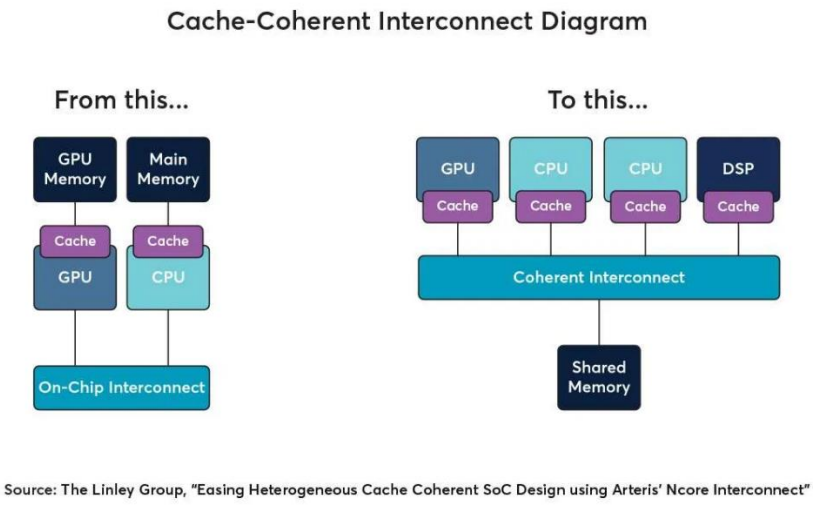
Modern interconnect technologies were introduced to solve this, such as:

(a). Intel QuickPath Interconnect (QPI)

(b). AMD Infinity Fabric

(c). ARM AMBA Coherent Interconnect (CCI)

(d). Cache Coherent Interconnect for Accelerators (CCIX)



Source: The Linley Group, "Easing Heterogeneous Cache Coherent SoC Design using Arteris' Ncore Interconnect"

Optical Bus Technology

Based on optical communication network technologies, the industrial optical bus combines the multi-level optical splitting and multi-terminal access capabilities of the optical distribution network to provide competitive performance indicators for the field bus, further improving the bandwidth and reducing the latency. Optical bus technology uses light to transmit data and control systems, and is used in a variety of industries and applications. Some application areas are:

(a). **Industrial Optical Bus Control System (OCS)**: Uses optical bus technology to reduce the use of copper cables and eliminate traditional intermediate links. OCS is a next-generation industrial process control system that supports real-time transmission

(b). **Supercomputers:** Optical interconnects can support gigahertz transfer rates and are free from capacitive bus loading, crosstalk, and electromagnetic interference.

(c). **Home networks:** Optical communications technology can be used to transmit information over long distances.

Example of Optical Bus Technologies:

(a). **Silicon Photonics** (Intel, IBM) for on-chip optical communication.

(b). **Optical PCI Express (PCIe)** in experimental stages.

(c). **Optical backplanes** in high-speed switches and servers.

Fibre Optic Cable Vs Optical Bus Technology

Fibre Optic Cable is a medium used to transmit data as light signals over long distances with high speed and low loss.

Optical Bus Technology, on the other hand, refers to a bus system or architecture that uses optical (light-based) communication between system components (like CPUs, memory, and I/O devices) within a computer or network.

Fibre Optic Cable vs Optical Bus Technology

Fibre Optic Cable	Optical Bus Technology
Physical cable carrying light signals, often over long distances (networking, internet backbones, etc.).	A system design where data paths inside a computer or system use optical communication instead of electrical signals.
Common in telecom and internet infrastructure.	Emerging inside high-performance computing and data centers.
Example: Undersea internet cables.	Example: Silicon Photonics interconnects inside servers.

Advanced Memory Buses

Advanced Memory Buses are high-speed communication pathways designed to efficiently transfer data between the CPU and memory (RAM), supporting the increasing demand for higher bandwidth, lower latency, and better scalability in modern systems.

Examples of Advanced Memory Bus Technologies

Technology	Description
DDR (Double Data Rate)	Successive generations like DDR4, DDR5 increase speed and reduce power consumption.

HBM (High Bandwidth Memory)	Stacked memory close to the processor, using wide buses for ultra-fast data transfer (common in GPUs).
GDDR (Graphics DDR)	Specialized for graphics cards and high-performance tasks.
Infinity Fabric (AMD)	Connects CPU cores and memory with high-speed links.
QuickPath Interconnect (QPI - Intel)	High-speed point-to-point link between CPU and memory.

Energy-Efficient Bus Designs

As modern computing devices (from smartphones to servers) demand higher performance while minimizing power consumption, energy-efficient bus designs have become a key advancement in bus architecture. These low-power buses are essential for mobile and embedded systems. They are bus systems specifically optimized to:

(a). Reduce power usage during data transmission.

(b). Minimize heat generation.

(c). Extend battery life (in mobile devices).

(d). Lower operating costs (in data centers).

Key Techniques Used

Technique	Description
Dynamic Voltage and Frequency Scaling (DVFS)	Adjusts voltage and frequency based on workload.
Low-Power Signaling	Uses differential signaling and lower voltages to reduce energy use.
Sleep and Idle Modes	Buses enter low-power states when not active.
Data Compression	Reduces the amount of data transferred to save energy.
Clock Gating	Turns off parts of the bus clock when not in use.

Examples of Energy-Efficient Bus Technologies:

(a). Mobile Industry Processor Interface (MIPI): Used in smartphones and tablets, optimized for low power.

(b). Low-Power DDR (LPDDR): Memory bus for mobile and embedded systems.

(c). PCI Express (PCIe) Power Management: Advanced states like L1, L2 for reduced power during inactivity.

(d). USB Power Delivery (USB-PD): Manages power use dynamically based on device needs.

Network-on-Chip (NoC)

As chips now have dozens to hundreds of cores, traditional bus systems can't handle the traffic efficiently. Network-on-Chip (NoC) is an advanced bus architecture designed to handle communication between multiple cores, memory, and I/O devices inside a single chip (like a multi-core processor or system-on-chip). Instead of using traditional shared buses or point-to-point connections, NoC uses network-style communication, similar to how data moves across the internet, but on a much smaller scale within the chip. NoC improves the traditional bus systems by:

(a). Managing communication like a network.

(b). Allowing multiple data packets to travel simultaneously.

(c). Ensuring low latency and high bandwidth.

Benefits of NoC

Feature	Advantage
Scalability	Supports many-core processors.
Parallelism	Multiple data paths active at once.
Power Efficiency	Optimized data routing saves energy.
Reliability	Reduces bottlenecks and improves fault tolerance.

Examples of NoC Use:

(a). Multi-core CPUs and GPUs.

(b). System-on-Chip (SoC) designs (like smartphones).

(c). AI accelerators and high-performance computing.

Lesson 7: Advances in Processor Architecture

Processor architecture has undergone significant advancements over the years to improve computational speed, power efficiency, and scalability. The evolution has been driven by increasing demands for high-performance computing, artificial intelligence, mobile devices, and cloud computing. Here are some key advancements in modern processor architectures and their impact on computing.

Multicore and Manycore Architectures

(a). **Multicore Processors**: Multiple processor cores are integrated onto a single chip to enhance parallel processing. Each core can execute instructions independently, leading to improved performance and energy efficiency.
(b). **Manycore Processors**: Extend multicore concepts with a significantly larger number of cores (e.g., 32+ cores), often used in High-Performance Computing (HPC) and AI workloads.

Examples: Intel Core i9, AMD Ryzen 9, NVIDIA Grace CPU Superchip.

Superscalar and Out-of-Order Execution

A Superscalar processor is a type of CPU design that can execute multiple instructions simultaneously during a single clock cycle by using multiple execution units (like arithmetic, logic, and load/store units). This increases processing speed by allowing more work to be done in parallel. On the other hand, Out-of-Order Execution (OoOE) is a technique where the CPU does not strictly follow the original program instruction order; instead, it dynamically rearranges the execution of instructions to avoid delays caused by slow operations (like waiting for data from memory). By executing independent instructions while others are waiting, Out-of-Order Execution improves overall CPU efficiency and performance. Together, Superscalar and Out-of-Order Execution are key features in modern high-performance processors that maximize instruction throughput and minimize idle time. Examples of processors using this technology are: ARM Cortex-A76, Intel Alder Lake, AMD Zen Architecture.

Vector Processing and Single Instruction, Multiple Data (SIMD)

Vector Processing is a computer architecture technique designed to handle multiple data elements at once using a single instruction. Instead of processing one piece of data at a time, vector processors work on entire arrays (vectors) of data simultaneously, making them very efficient for tasks like scientific computing, graphics, and large-scale data processing where the same operation repeats over many data points.

A key concept behind vector processing is SIMD (Single Instruction, Multiple Data). In SIMD architecture, one instruction is applied to multiple data elements in parallel. This is especially

useful in workloads such as image processing, audio processing, and machine learning, where the same calculation needs to be performed across a large dataset.

Modern CPUs and GPUs include SIMD extensions (like Intel's SSE, AVX, or ARM's NEON) that speed up these parallel tasks by handling vectorized operations efficiently within general-purpose processors.

Heterogeneous Computing and Specialized Accelerators

Heterogeneous Computing is an advanced computing approach that uses different types of processors within the same system to handle different kinds of tasks more efficiently. Instead of relying only on the CPU (Central Processing Unit), a heterogeneous system combines other specialized processors like GPUs (Graphics Processing Units), FPGAs (Field-Programmable Gate Arrays), TPUs (Tensor Processing Units), and AI accelerators. Each processor is optimized for specific workloads, which improves performance and reduces power consumption. **Example:** When streaming a video, the CPU may handle the operating system, the GPU renders the graphics, and a video accelerator decodes the video stream—all working together seamlessly.

Specialized Accelerators

These are hardware components designed to perform specific tasks much faster than a general-purpose CPU. Examples include:

(a). **GPUs** for graphics and parallel data processing.
(b). **TPUs** for machine learning and AI tasks.
(c). **FPGAs** for customizable high-speed processing.
(d). **Video and encryption accelerators** for media and security tasks.

Energy-Efficient and Low-Power Architectures

Energy-Efficient and Low-Power Architectures are modern computer designs focused on reducing power consumption while maintaining good performance, especially important in mobile devices, embedded systems, IoT devices, and data centers. As technology advances, processors are required to perform more tasks without draining batteries or producing excessive heat. These architectures are designed to balance speed, power usage, and thermal output, making systems more sustainable and cost-effective. Examples include ARM Cortex-A78 + Cortex-A55, Intel Hybrid Architecture.

Key Techniques Used:

(a). Dynamic Voltage and Frequency Scaling (DVFS): Automatically adjusts processor speed and power based on workload.
(b). Power Gating: Shuts down inactive parts of a processor to save power.

(c). Clock Gating: Disables the clock signal to unused circuits, reducing unnecessary switching.
(d). Big.LITTLE Architecture: Combines high-power cores for heavy tasks and low-power cores for light tasks within the same processor (common in smartphones).
(e). Use of smaller transistors (nanometer technology) that consume less energy.
(f). Efficient sleep and idle modes to minimize power during inactivity.

Quantum and Neuromorphic Computing

Quantum Computing is a revolutionary computing model that uses the principles of quantum mechanics to perform computations. Unlike traditional computers that use bits (which are either 0 or 1), quantum computers use qubits that can exist as 0, 1, or both at the same time (a property called superposition). Quantum computers can process massive amounts of possibilities at once, making them powerful for solving complex problems such as cryptography, drug discovery, optimization, and big data analysis.

However, quantum computing is still in its early stages and requires extremely cold environments and specialized hardware.

Neuromorphic Computing is inspired by how the human brain works. It uses special processors designed to mimic neurons and synapses, allowing the system to process data through spikes and events, just like the brain. This approach is extremely efficient for tasks involving pattern recognition, learning, vision, and speech. Neuromorphic chips consume very low power and are highly parallel, making them ideal for artificial intelligence and machine learning applications. Companies like Intel (with its Loihi chip) and IBM (with TrueNorth) are leading in developing neuromorphic processors.

These technologies represent the future of computing, pushing beyond traditional architectures to tackle problems in new, faster, and smarter ways.

Security Enhancements in Modern Processors

Modern processors are designed with built-in security features to protect systems from advanced cyber threats like malware, data breaches, and side-channel attacks. As cyberattacks become more sophisticated, hardware-level security has become essential to protect sensitive data and system integrity.

Key Security Features in Modern Processors

Security Feature	Description
Trusted Execution Environments (TEE)	Creates a secure, isolated area in the processor to run sensitive code and protect data (e.g., Intel SGX, ARM TrustZone).
Hardware Encryption	Built-in support for fast, efficient encryption and decryption to secure data (e.g., AES-NI instructions).
Secure Boot	Ensures that only verified, trusted software loads during the system startup process to prevent malware injection.
Memory Protection	Techniques like Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR) prevent attackers from exploiting memory.
Control-Flow Enforcement Technology (CET)	Blocks common attack techniques like Return-Oriented Programming (ROP).
Side-Channel Attack Mitigations	Protections against attacks like Spectre and Meltdown that exploit CPU data leakage.
Physical Attack Resistance	Prevents tampering with embedded systems, such as smart cards and IoT devices.

Conclusion

Advancements in processor architecture continue to drive innovation across various computing domains. The integration of AI accelerators, improved parallelism, energy-efficient designs, and security features ensures that modern processors can handle increasingly complex workloads with greater efficiency. The future of processor design will likely focus on neuromorphic computing, quantum processing, and even tighter integration of heterogeneous computing units to maximize performance and power efficiency.

Lesson 8: Advances In Memory Architecture

Memory is a vital component of any computer system. As computing demands increase (big data, AI, gaming, IoT), traditional memory technologies face limitations in speed, capacity, power consumption, and cost. To address these challenges, memory architectures have evolved significantly.

Limitations of Traditional Memory Architectures

Before understanding the advances, recall the conventional memory hierarchy:

Registers → Cache → Main Memory (RAM) → Secondary Storage (HDD/SSD)

Common challenges include:

(a). **Latency**: Delay in accessing data.

(b). **Bandwidth**: Limited speed in transferring data.

(c). **Power consumption**: High in DRAM and HDD.

(d). **Scalability**: Difficulty in increasing size without performance drops.

(e). **Cost**: High-performance memory is expensive.

Key Advances in Memory Architecture

Cache Innovations

Cache Innovations are some of the most important advances in memory architecture, designed to make data access faster, smarter, and more efficient. Since processors operate much faster than main memory (RAM), caches store frequently accessed data closer to the CPU to reduce waiting time. Modern systems have introduced several innovative cache techniques to improve performance, reduce power consumption, and handle complex workloads

Key Cache Innovations

Innovation	Description
Multi-Level Caches (L1, L2, L3)	Modern processors use multiple layers of cache. L1 is the fastest and closest to the CPU, while L2 and L3 are larger and slower but still faster than RAM. This hierarchy helps balance speed and storage capacity.
Non-Uniform Cache Architecture (NUCA)	In large multi-core processors, caches are spread across the chip, and access time depends on the physical distance between the core and the cache block, improving scalability.
Victim Caches	A small cache that stores recently evicted blocks from the main cache to reduce conflict misses and improve hit rates.
Inclusive, Exclusive, and Non-Inclusive Caches	Different cache strategies are used to manage how data is shared across cache levels to minimize duplication and maximize speed.
Adaptive Cache Replacement Policies	Modern caches use smart algorithms to decide which data to keep or remove based on usage patterns, improving cache efficiency.
Cache Prefetching	Predicts which data the CPU will need next and loads it into the cache ahead of time, reducing delays.
Shared Caches for Multi-core Processors	Shared L3 caches allow multiple processor cores to efficiently exchange data without going back to main memory.
Write-back vs. Write-through Improvements	Enhancements in how and when data is written back to memory help optimize performance and power usage.

Non-Volatile Memory (NVM)

Non-Volatile Memory (NVM) refers to memory technologies that retain data even when power is turned off. Unlike traditional volatile memory like DRAM (which loses data when the system shuts down), NVM provides permanent storage with faster speeds closer to RAM. NVM represents a major advancement in memory architecture, offering a blend of speed, durability, and data retention.

Popular NVM Technologies

Technology	Feature	Example
Flash	Common in SSDs	USB drives, smartphones
Magnetoresistive RAM (MRAM)	Fast, durable, low power	Embedded systems
Resistive RAM (ReRAM)	High density	AI accelerators
Phase Change Memory (PCM)	Fast read/write	Data centers

Why is NVM Important

Feature	Benefit
Persistence	Data is saved even without power.
Speed	Faster than traditional storage (like HDDs and SSDs), and some NVM types approach DRAM speeds.
Low Power Consumption	Uses less energy since there's no need to constantly refresh data.
Durability	Withstands more write cycles and has longer life compared to flash storage.
Instant-On Systems	Enables devices to resume work immediately after power loss or shutdown.

NVM's Role in Modern Architectures

(a). Acts as a bridge between fast volatile memory (like DRAM) and slower storage (like hard drives).

(b). Supports faster boot times and quicker application loading.

(c). Helps in building persistent memory systems, where even running programs can continue after power failure.

(d). Reduces system complexity by combining memory and storage into one layer in some designs.

Non-Volatile Memory (NVM) is a game-changing advancement that provides high-speed, power-efficient, and persistent storage within modern computer architectures, helping improve system performance, reliability, and user experience.

3D Memory Technologies

3D Memory Technologies represent a major breakthrough in memory architecture by stacking multiple layers of memory cells vertically instead of spreading them out horizontally on a flat surface (2D). This vertical stacking greatly increases storage capacity, improves speed, and reduces power consumption, all while using less physical space on a chip. This innovation helps meet the growing demand for higher performance, smaller devices, and energy efficiency in modern computing systems.

Examples of 3D Memory Technologies

Memory Type	Description
3D NAND Flash	Widely used in SSDs, smartphones, and memory cards. Memory cells are stacked in dozens or hundreds of layers to store more data in the same footprint.
High Bandwidth Memory (HBM)	Stacks memory dies close to the processor using **through-silicon vias (TSVs)** for ultra-fast data transfer, common in graphics cards and AI systems.
3D DRAM	Future DRAM designs using vertical stacking for higher capacities and lower power use.
Hybrid Memory Cube (HMC)	A high-speed memory that stacks multiple DRAM layers with fast interconnects for advanced computing systems.

Why is 3D Memory Important?

☑ Feature	☑ Benefit
Higher Density	More storage in a smaller area.
Faster Data Transfer	Shorter distances between layers improve speed.
Lower Power Usage	More efficient design reduces energy consumption.
Better Performance for AI, Gaming, and Servers	Supports demanding workloads with high data throughput.
Scalability	Easier to add more layers than making chips wider.

Uses of 3D Memory Technologies

(a). Solid-State Drives (SSDs) with 3D NAND for faster storage.

(b). Graphics Processing Units (GPUs) using HBM for high-speed gaming and AI.

(c). Supercomputers and data centers using advanced 3D memory for big data tasks.

(d). Mobile devices needing small, powerful, and energy-efficient memory.

3D Memory Technologies stack memory vertically to achieve higher capacity, better speed, and greater energy efficiency in modern computing. This design is essential for handling the increasing demands of today's high-performance systems like AI, cloud computing, and mobile devices.

Hybrid Memory Cube (HMC)

The Hybrid Memory Cube (HMC) is an advanced memory technology that completely rethinks traditional DRAM design by stacking multiple layers of high-speed memory vertically on top of a logic layer. These layers are connected using Through-Silicon Vias (TSVs), which are tiny vertical electrical connections passing through the layers to allow extremely fast communication.

HMC provides massive bandwidth, lower power consumption, and better scalability than traditional memory, making it ideal for high-performance computing tasks like supercomputers, AI systems, data centers, and networking equipment.

Key Features of HMC

☑ Feature	☑ Description
3D Stacking	Multiple DRAM layers are stacked vertically on a logic layer to save space and improve performance.
Through-Silicon Vias (TSVs)	Vertical connections that allow fast data transfer between layers.
Logic Layer	The base layer manages memory control, error correction, and data routing, offloading tasks from the CPU.
High Bandwidth	Offers extremely fast data transfer rates, much higher than traditional DDR memory.
Energy Efficiency	Consumes less power per bit transferred, making it ideal for large-scale systems.

Scalability	Easy to expand memory capacity by adding more cubes without major redesigns.

Why is HMC Important?

(a). Provides 15x bandwidth of traditional DDR memory.

(b). Reduces power consumption by up to 70% compared to conventional DRAM.

(c). Saves space with compact vertical stacking.

(d). Improves data integrity with built-in error correction.

(e). Perfect for workloads requiring massive parallel processing, like AI, big data, and scientific simulations.

Where is HMC Used?

(a). Supercomputers

(b). Artificial Intelligence (AI) accelerators

(c). Cloud servers and data centers

(d). High-end networking devices

(e). Graphics and gaming systems

The Hybrid Memory Cube (HMC) is a next-generation memory architecture that delivers exceptional speed, energy efficiency, and scalability by using a smart 3D stacking design with fast interconnects. It is a key innovation powering the future of high-performance computing.

Persistent Memory (PMEM)

Persistent Memory (PMEM), also known as Storage-Class Memory (SCM), is an innovative memory technology that combines the speed of traditional RAM with the non-volatility of storage. Unlike regular DRAM, which loses all data when power is off, PMEM retains data even without power, making it ideal for systems that need high-speed access to large amounts of data with the ability to recover quickly after power loss. PMEM bridges the gap between fast, volatile memory (like DRAM) and slower, persistent storage (like SSDs and HDDs), helping improve performance, reliability, and data durability in modern systems.

Key Features of PMEM

☑ Feature	☑ Description
Non-Volatile	Retains data after power loss, like storage devices.
High-Speed Access	Accesses data almost as fast as DRAM.
Large Capacity	Offers higher storage capacity at a lower cost compared to DRAM.
Byte-Addressable	Allows applications to read/write data at the byte level, not just in blocks like traditional storage.
Direct Access (DAX)	Enables software to directly access PMEM without going through traditional storage I/O operations.

Why is PMEM Important?

(a). Faster recovery after crashes or power failures since data remains available.

(b). Improves performance in data-heavy applications like databases and analytics.

(c). Reduces the need to frequently move data between RAM and storage.

(d). Lowers costs by allowing large memory systems without relying solely on expensive DRAM.

(e). Supports real-time analytics, cloud computing, and high-availability systems.

Where is PMEM Used?

(a). Enterprise databases (e.g., SQL, Oracle).

(b). Big data platforms (e.g., Hadoop, Spark).

(c). Cloud services.

(d). High-performance computing.

(e). In-memory computing applications.

Memory Disaggregation

Memory Disaggregation is a modern memory architecture concept where memory is separated (disaggregated) from individual servers and pooled into a shared resource that multiple servers can access over a high-speed network. Instead of each computer having its own dedicated RAM, many systems share a large, centralized memory pool. This approach allows data centers and cloud systems to use memory more efficiently, reduce waste, and dynamically allocate memory to the servers or applications that need it most.

Key Features of Memory Disaggregation

☑ Feature	☑ Description
Centralized Memory Pool	Memory is separated from CPUs and shared across multiple servers.
Dynamic Allocation	Systems request memory from the pool based on workload needs.
High-Speed Interconnects	Uses fast network technologies (like **CXL**, **RDMA**, or **InfiniBand**) to connect servers to the memory pool.
Scalability	Easily add more memory without upgrading individual servers.
Resource Optimization	Reduces idle memory and ensures better utilization.

Why is Memory Disaggregation Important?

(a). Eliminates memory waste caused by unused RAM in underutilized servers.

(b). Enables flexible and scalable computing, as memory can grow separately from processing power.

(c). Reduces data movement across servers, which saves bandwidth and energy.

(d). Improves performance for applications needing large amounts of memory, like big data analytics, machine learning, and cloud services.

(e). Cuts costs by allowing hardware upgrades independently (no need to buy new servers just for more memory).

Where is Memory Disaggregation Used?

(a). Large cloud data centers (like AWS, Google Cloud, Microsoft Azure).

(b). High-Performance Computing (HPC) systems.

(c). AI/ML workloads that need access to massive shared memory.

(d). Enterprise data centers looking to improve efficiency.

Processing-In-Memory (PIM)

Processing-In-Memory (PIM) is a groundbreaking memory architecture innovation that integrates processing capabilities directly within the memory chips. Instead of constantly transferring data back and forth between the CPU and memory for processing, PIM allows certain computations to happen inside the memory itself, reducing the need for data movement.

This drastically improves performance, reduces power consumption, and is especially useful for tasks that handle large amounts of data, such as artificial intelligence (AI), machine learning (ML), big data analytics, and scientific simulations.

Key Features of PIM

☑ Feature	☑ Description
Integrated Processing	Adds lightweight processors or logic circuits inside the memory.
Reduced Data Movement	Processes data where it is stored, minimizing the need to send data to the CPU.
High Parallelism	Performs many operations in parallel across memory units.
Energy Efficiency	Uses less power by cutting down on data transfers.
Speed Improvement	Reduces latency for large data operations.

Why is PIM Important?

(a). Overcomes the "memory wall" problem (the speed gap between CPU and memory).

(b). Saves energy by reducing expensive data transfers.

(c). Accelerates data-intensive workloads, like AI, graphics processing, databases, and video processing.

(d). Improves system bandwidth and throughput.

(e). Supports real-time processing for emerging technologies such as IoT and edge computing.

Examples of PIM Technologies

(a). **Samsung** and **SK Hynix** are developing PIM-enabled DRAM.

(b). **HBM-PIM (High Bandwidth Memory with PIM)** – combining high-speed memory with in-memory processing.

(c). **UPMEM** – offers commercial PIM chips for accelerating big data workloads.

Where is PIM Used?

(a). Artificial Intelligence (AI) and Machine Learning (ML).

(b). Big Data Analytics.

(c). Real-time video/image processing.

(d). Scientific computing and simulations.

(e). Cloud and edge computing.

Impacts of Advanced Memory Architectures

Area	Impact
Performance	Faster data processing and reduced latency.
Power Efficiency	Lower energy consumption.
Capacity	Support for large datasets (AI, Big Data).
Scalability	Ability to expand without significant redesign.

Cost	More economical solutions for high-speed storage.

Conclusion

Memory architecture is evolving rapidly to support modern computing demands. From cache improvements to non-volatile and 3D technologies, the goal remains clear: faster, bigger, cheaper, and more efficient memory systems.

Reference Materials

(a). Robert G. Plantz (2021), **Introduction to Computer Organization:** ARM Assembly Language Using the Raspberry Pi

(b). "Computer Architecture: A Quantitative Approach" by John L. Hennessy & David A. Patterson.