Module 1:

Fundamentals of Computer Organization

Dr. Vignesh Ramamoorthy. H
Associate Professor
AIIT – AUB

AMITY UNIVERSITY BENGALURU

Module 1: Fundamentals of Computer Organization

- Introduction to Computer Systems
- Difference between organization and architecture
- Functional components: ALU, registers, control unit, memory, I/O
- Overview of instruction execution (Fetch, Decode, Execute)
- Concept of bus systems and data flow

AMITY UNIVERSITY BENGALURU

Module 1: Fundamentals of Computer Organization

- **Introduction to Computer Systems**
- Difference between organization and architecture
- Functional components: ALU, registers, control unit, memory, I/O
- Overview of instruction execution (Fetch, Decode, Execute)
- Concept of bus systems and data flow

Introduction to Computer Systems

- A computer system is an integrated set of hardware and software designed to perform computation, manage data, and control tasks automatically.
- It includes **hardware** (the physical components) and **software** (the instructions or programs), working in harmony to solve problems and perform various operations.

Introduction to Computer Systems

Definition and Purpose of a Computer System

- A computer system is a programmable electronic device that accepts input, processes data, stores information, and produces output.
- To automate tasks such as calculations, data processing, information storage, communication, and control

Introduction to Computer Systems

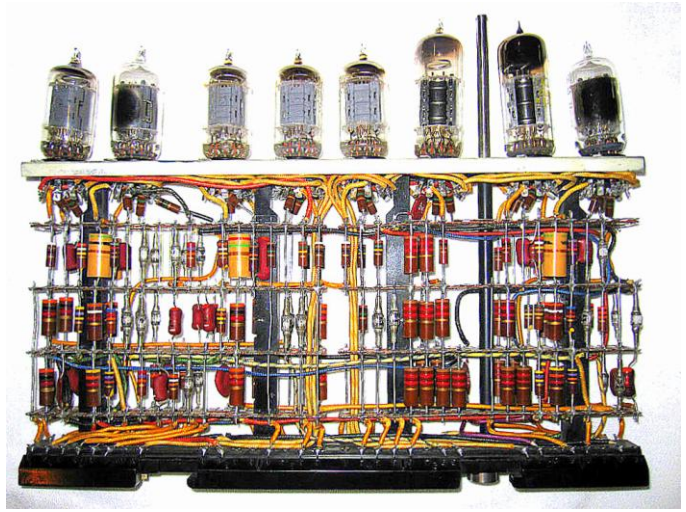
Evolution of Computer Systems

a) Generations of Computing:

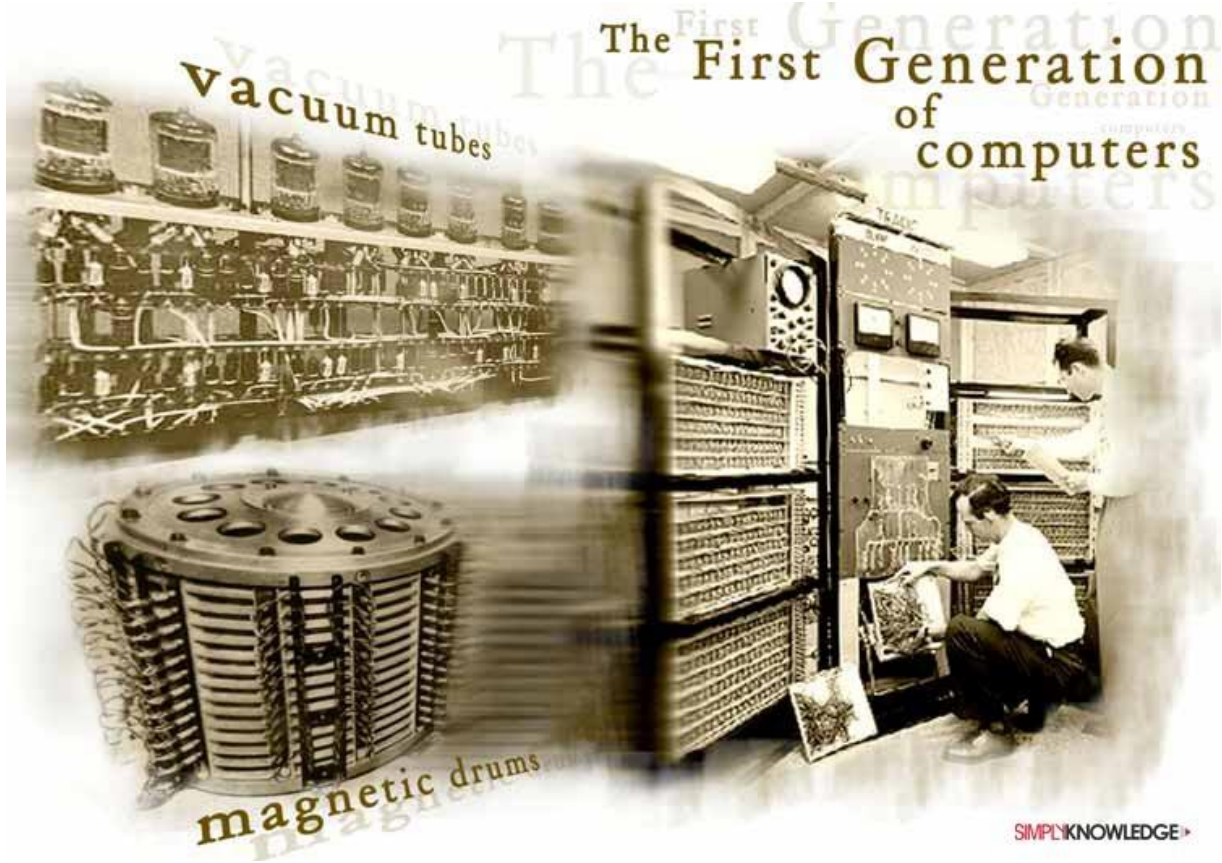
1. **First Generation (1940s–1950s)** – Used vacuum tubes; slow and large (e.g., ENIAC - Electronic Numerical Integrator and Computer).
2. **Second Generation (1950s–1960s)** – Used transistors; more reliable and compact.
3. **Third Generation (1960s–1970s)** – Used integrated circuits; faster and more efficient.
4. **Fourth Generation (1970s–present)** – Used microprocessors; led to PCs and laptops.
5. **Fifth Generation (Present and Beyond)** – Focuses on AI and parallel processing.

Introduction to Computer Systems

Evolution of Computer Systems

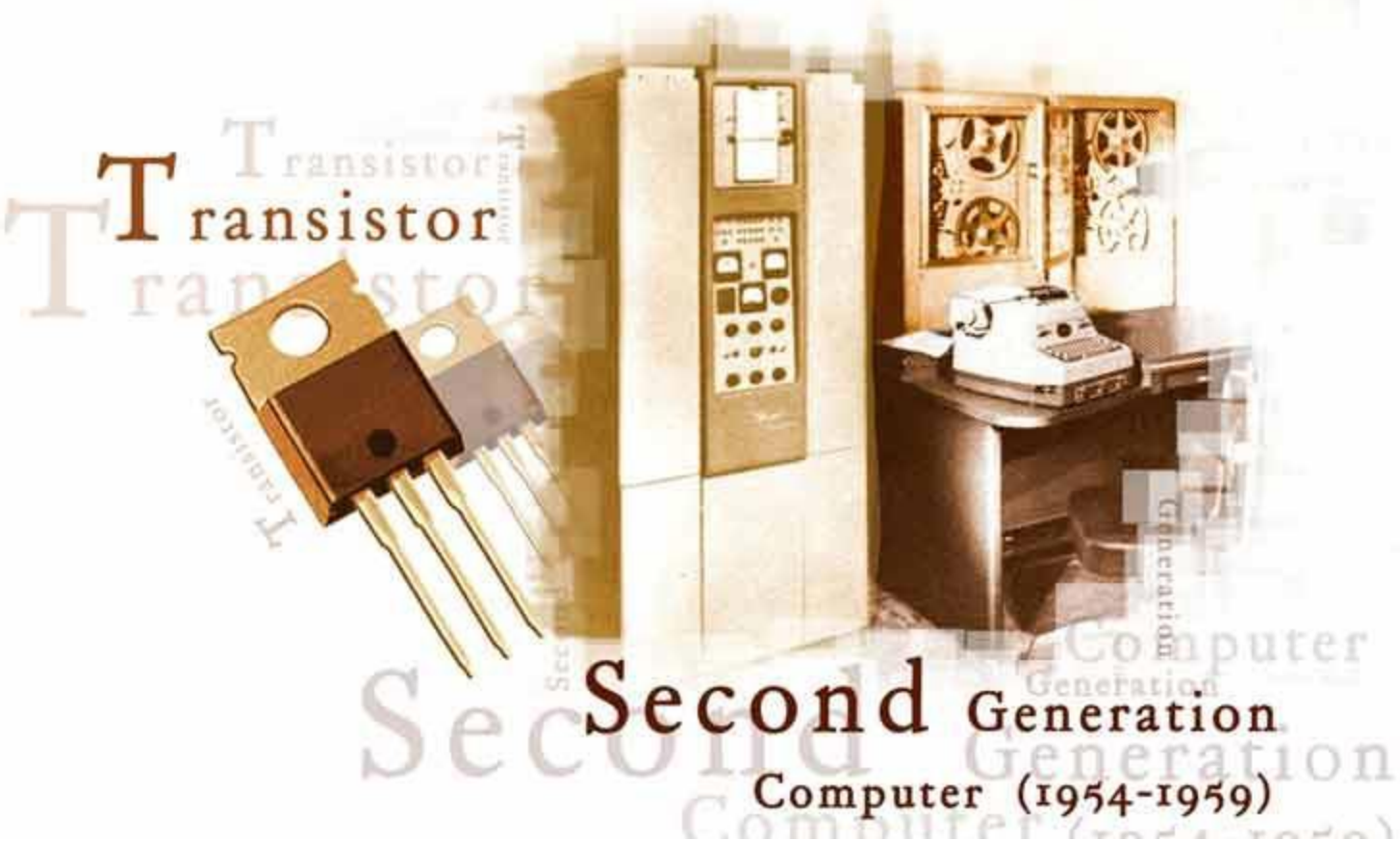


Vacuum Tubes



Introduction to Computer Systems

Evolution of Computer Systems



Introduction to Computer Systems

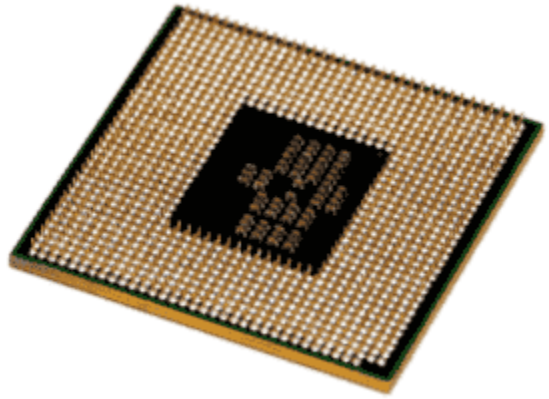
Evolution of Computer Systems

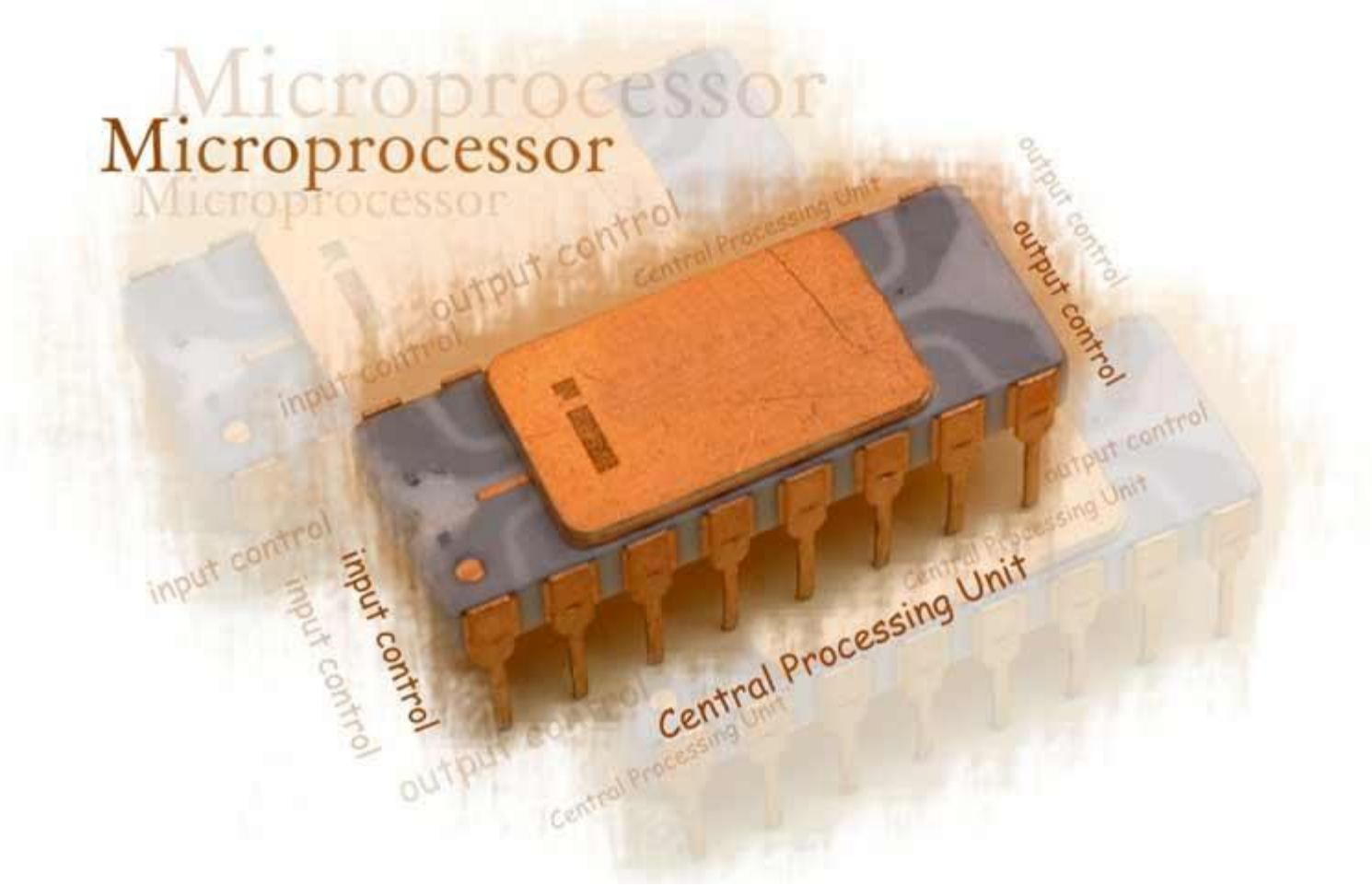
Third generation Computer

Integrated Circuit

Introduction to Computer Systems

Evolution of Computer Systems





Introduction to Computer Systems

Evolution of Computer Systems

Fifth Generation Computer

- Technology :AI (Artificial Intelligence), Nano technology

Introduction to Computer Systems

Evolution of Computer Systems

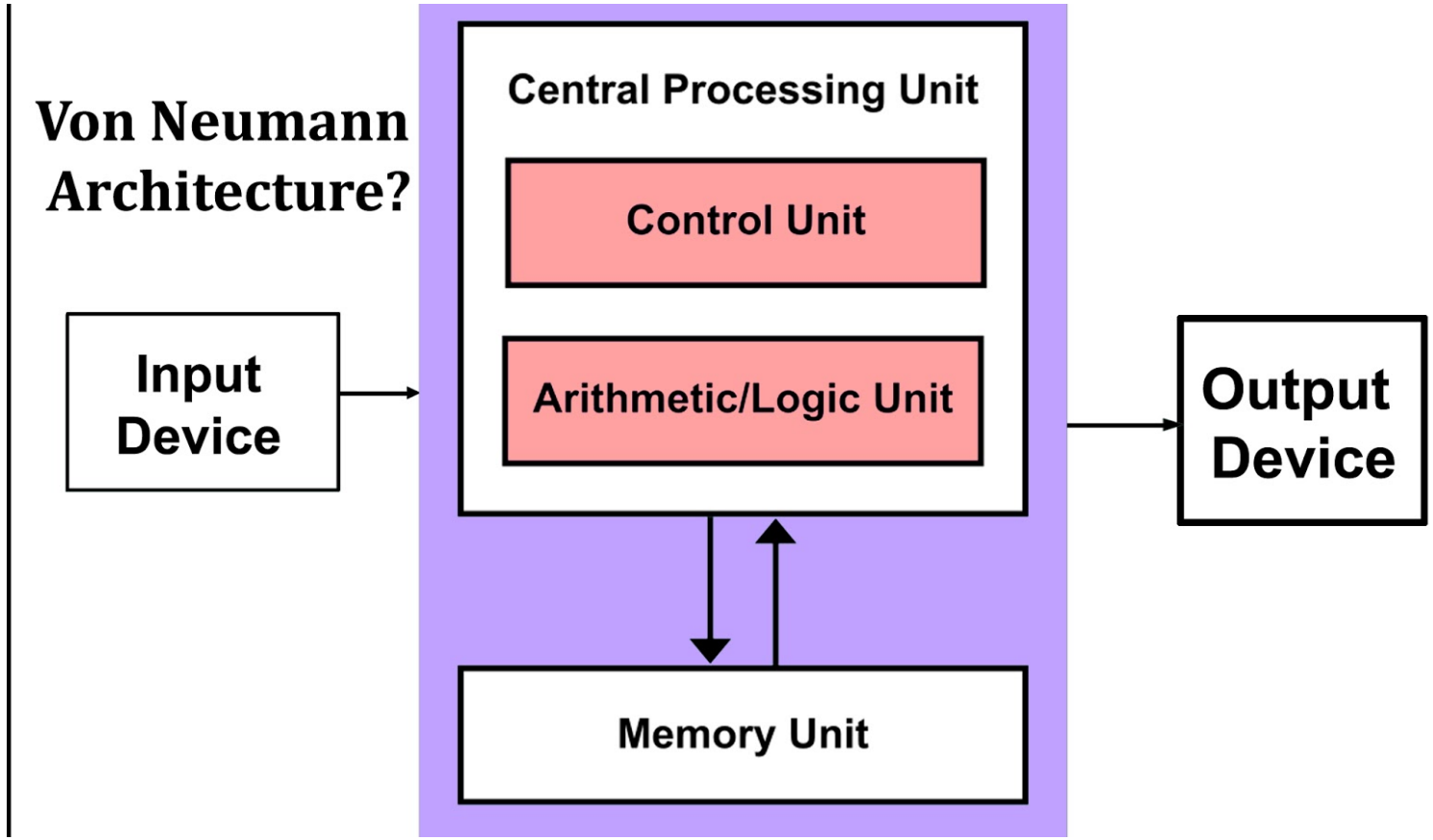
b) Von Neumann Model:

- Proposed by John von Neumann in 1945.
- Describes a system architecture where the CPU, memory, and input/output are integrated.
- Uses the **stored-program concept**, meaning both data and programs are stored in the same memory.

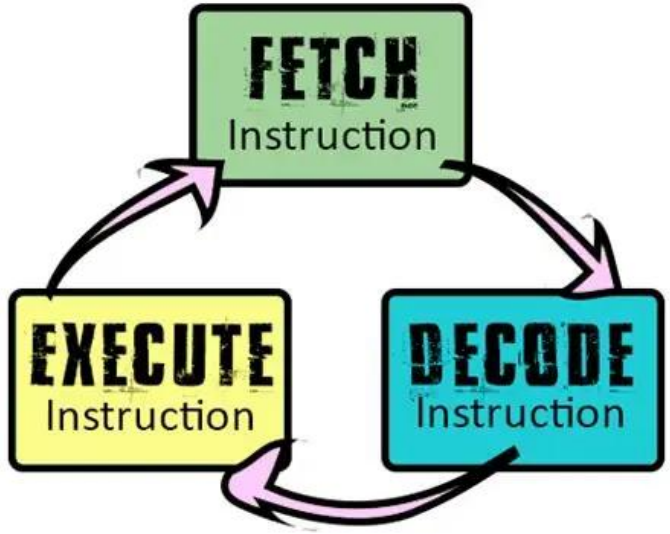
Introduction to Computer Systems

Evolution of Computer Systems

b) Von Neumann Model:



Introduction to Computer Systems



Basic Operational Concepts

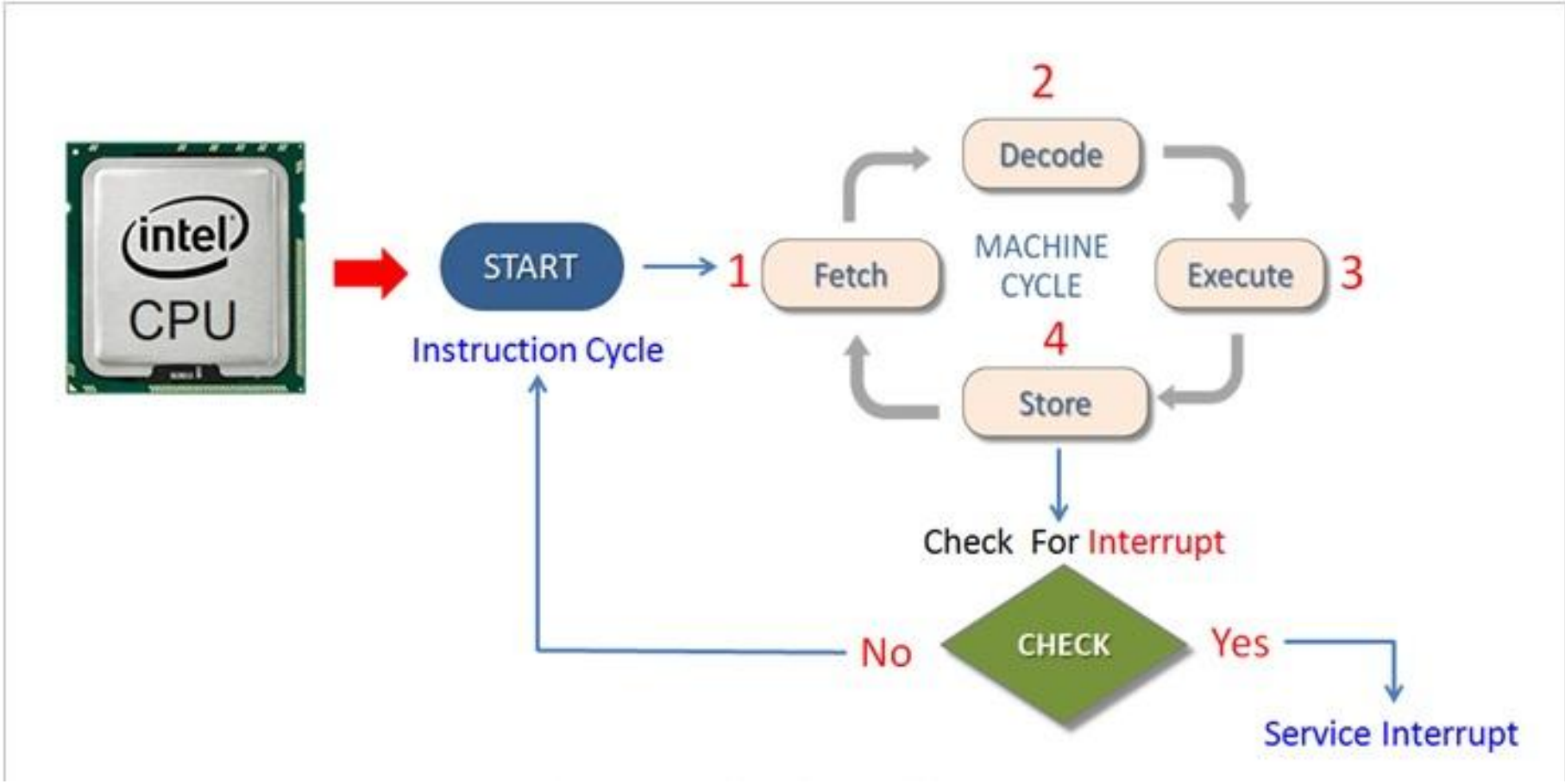
a) Instruction Cycle (Fetch-Decode-Execute):

- **Fetch** – CPU retrieves an instruction from memory.
- **Decode** – Instruction is interpreted/decoded.
- **Execute** – CPU performs the operation (e.g., addition, data move).

Introduction to Computer Systems

Basic Operational Concepts

a) Instruction Cycle (Fetch-Decode-Execute)

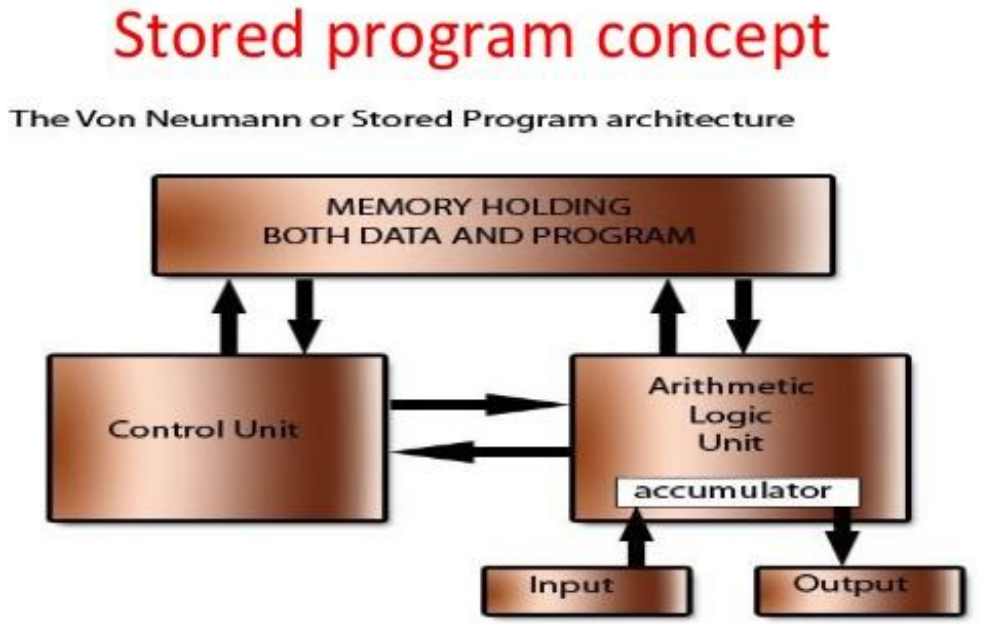


Introduction to Computer Systems

Basic Operational Concepts

b) Stored-Program Concept:

- Programs are stored in memory alongside data.
- CPU reads and executes instructions sequentially from memory.
- This allows for flexibility and automation.



→An accumulator is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU (central processing unit).

Introduction to Computer Systems

Software Interface

a) System Software:

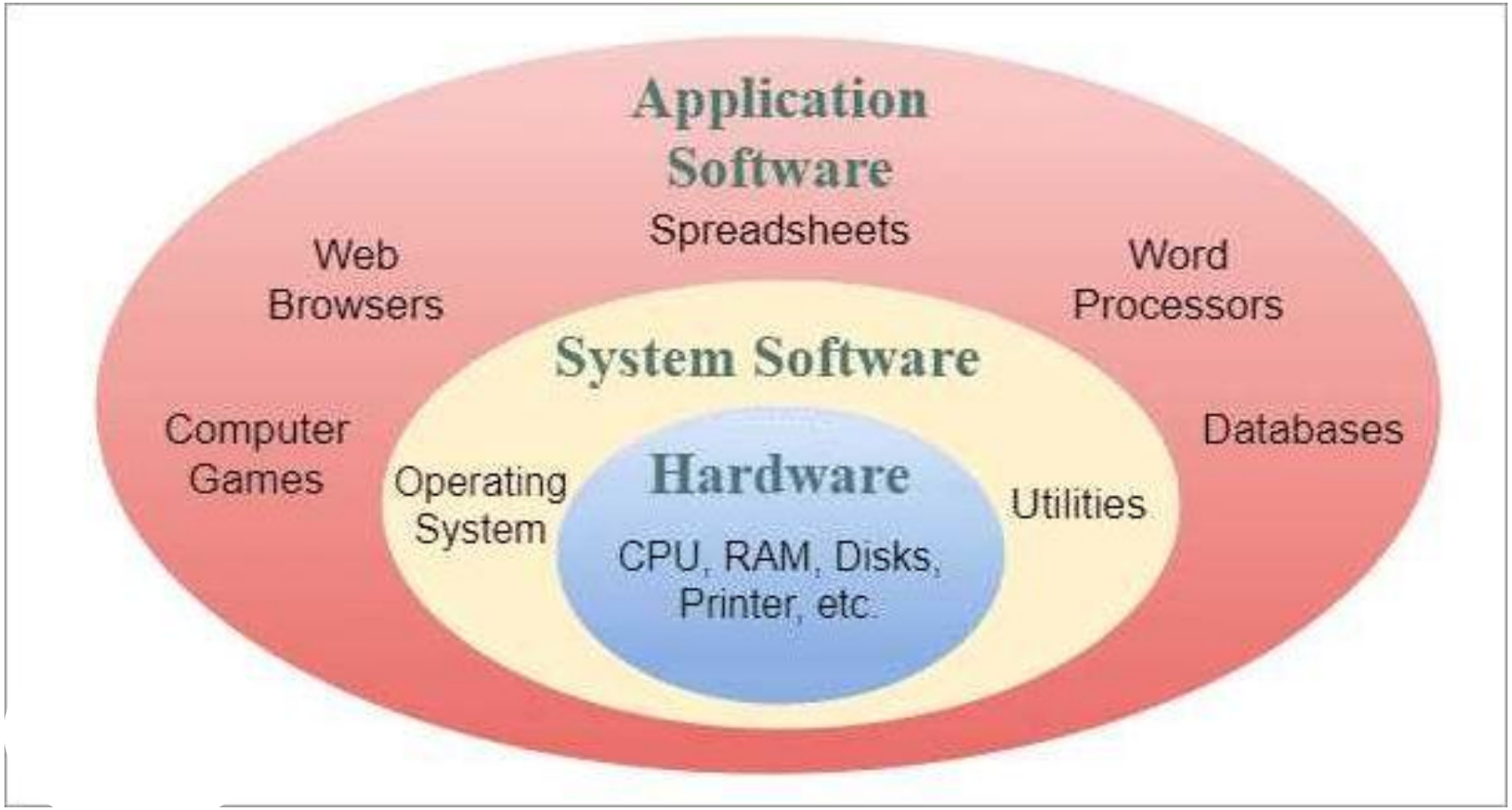
- Controls and manages hardware resources.
- Examples: Operating systems (Windows, Linux), compilers, assemblers, and device drivers.

b) Application Software:

- Performs user-specific tasks.
- Examples: MS Word, web browsers, media players, and games.

Introduction to Computer Systems

Software and Hardware Interface



Introduction to Computer Systems

Hardware Interface

- **Hardware** refers to the **physical components** of a computer system that can be touched and seen.
- It provides the foundation upon which software operates. Without hardware, software has no platform to run on.

Introduction to Computer Systems

Types of Hardware Interface

- **Input Devices** – Allow users to interact with the computer (e.g., keyboard, mouse, scanner).
- **Output Devices** – Display or produce results (e.g., monitor, printer, speakers).
- **Processing Unit** – The **CPU (Central Processing Unit)** executes instructions and performs calculations.
- **Storage Devices** – Store data permanently or temporarily (e.g., hard drives, SSDs, RAM).
- **Motherboard and Components** – Connects all parts of the system and enables communication between them.

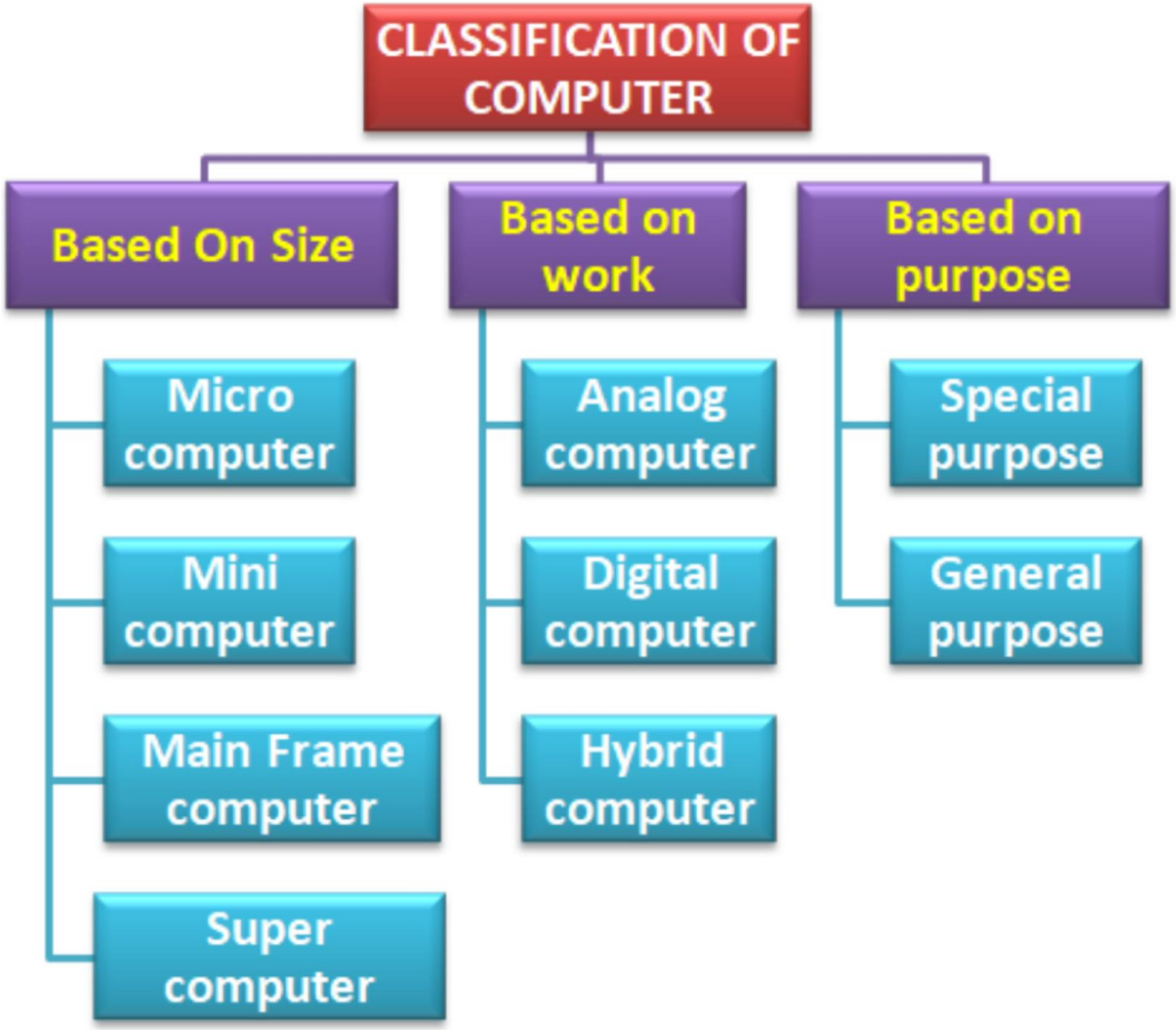
Introduction to Computer Systems

Hardware Interface

- Hardware is controlled and accessed through **system software** (like OS drivers), which acts as a bridge between **raw physical components** and **user-level applications**.
- For example, pressing a key on the keyboard (hardware) triggers an electrical signal interpreted by the **keyboard driver** (system software), which in turn is used by a **word processor** (application software).

Introduction to Computer Systems

Classification of Computers



General-purpose vs. Special-purpose Computers

- General-purpose computers: Designed to perform a wide variety of tasks (e.g., desktops, laptops). They can run many types of programs (e.g., MS Word, browsers, games).
- Special-purpose computers: Designed for a specific task (e.g., embedded systems in washing machines, ATMs, calculators). They are optimized for efficiency and speed in a fixed function.

Digital vs. Analog Computers

- Digital computers: Process data in binary form (0s and 1s). Most modern computers are digital. They are accurate, fast, and programmable.
- Analog computers: Work with continuous data (e.g., temperature, speed). They simulate physical systems but are less common now.

Introduction to Computer Systems

Classification of Computers

Introduction to Computer Systems

Analog vs Digital computers

- An **analog computer** is a type of computing device that uses **continuous physical phenomena** such as electrical, mechanical, or hydraulic quantities to model and solve problems.
- Unlike **digital computers**, which operate using **discrete** values (binary numbers), analog computers represent data with **continuous** variables, allowing them to process analog data directly.
- Historically, **analog computers** were widely used for *simulations and calculations* in various fields, but they have largely been replaced by faster and more powerful digital computers since the 1970s.

Introduction to Computer Systems

Classification of Computers

- **Microcomputers**: Single-user systems (e.g., PCs, laptops). Suitable for general applications.
- **Minicomputers**: Mid-range systems used by small organizations for shared computing.
- **Mainframes**: Powerful multi-user systems used in large organizations for bulk data processing.
- **Supercomputers**: Extremely fast systems used for scientific simulations, weather forecasting, and complex computations

AMITY UNIVERSITY BENGALURU

Module 1: Fundamentals of Computer Organization

- Introduction to Computer Systems
- **Difference between Organization and Architecture**
- Functional components: ALU, registers, control unit, memory, I/O
- Overview of instruction execution (Fetch, Decode, Execute)
- Concept of bus systems and data flow

Difference between Computer Organization and Architecture

Aspect	*Computer Architecture*	*Computer Organization*
Definition	Describes *what* a computer does (attributes visible to the programmer)	Describes *how* a computer is implemented (hardware realization)
Focus	Logic, design, instruction set, addressing methods	Circuit-level implementation, control signals, data paths
Examples	Instruction Set Architecture (ISA), data types, memory addressing	Control units, ALUs, data bus, registers
Design Role	Conceptual structure and functional behavior	Physical realization and optimization

Difference between Computer Organization and Architecture

Aspect	*Computer Architecture*	*Computer Organization*
Relevance to Users	User-level interaction (compilers, assembly programmers, application developers)	Relevant to system designers, hardware engineers, and embedded developers
Abstraction Level	High-level abstraction; concerned with what the system should do	Low-level abstraction; concerned with how the system actually performs
Performance Impact	Affects system efficiency and instruction-level performance	Affects speed, cost, and power consumption of implementation

Difference between Computer Organization and Architecture

Aspect	*Computer Architecture*	*Computer Organization*
Design Tools	Uses simulation tools, performance models, instruction set design	Uses digital logic tools like VHDL, Verilog, circuit simulators
Implementation	Can be the same across systems with different organizations (e.g., same ISA in different CPUs)	Can vary significantly even with the same architecture (e.g., RISC vs. CISC styles)
Examples of Concern	Number of registers, instruction formats, opcodes	Cache design, memory hierarchy, pipelining, branch prediction

AMITY UNIVERSITY BENGALURU

Module 1: Fundamentals of Computer Organization

- Introduction to Computer Systems
- Difference between Organization and Architecture
- **Functional components: ALU, registers, control unit, memory, I/O**
- Overview of instruction execution (Fetch, Decode, Execute)
- Concept of bus systems and data flow

Functional Components:

ALU, Registers, Control Unit, Memory, I/O

- The **Control Unit** fetches and decodes instructions.
- **Registers** temporarily store data.
- The **ALU** executes operations on this data.
- Results are stored back in registers or **memory**.
- **I/O systems** allow data to be transferred in and out of the system.

Functional Components: ALU

Arithmetic Logic Unit (ALU)

The **Arithmetic Logic Unit (ALU)** is a core component of the **Central Processing Unit (CPU)**. It is a **combinational digital circuit** designed to perform two main types of operations:

- **Arithmetic operations**: such as addition, subtraction, multiplication (in some systems), and division.
- **Logic operations**: such as AND, OR, NOT, XOR, and comparisons (e.g., equal to, less than).

Functional Components: ALU

Key Functions and Roles of the ALU:

1. Execution of Arithmetic and Logic Operations:

- Performs basic mathematical functions needed for program execution.
- Executes logical decisions used in conditional branching and comparison-based instructions.
- Common operations:
    a) Addition/Subtraction (used in address calculation, loop counters)
    b) AND/OR/XOR/NOT (used in masking, decision making, encryption)
    c) Shift/rotate operations (used in multiplication, division, bit manipulation)

Functional Components: ALU

Key Functions and Roles of the ALU:

2. Interaction with Registers:

- The ALU receives its **operands** from **CPU registers** (typically from a register file like accumulator, general-purpose registers).
- The **results** of operations are sent back to registers or written to memory via buses.

Functional Components: ALU

Key Functions and Roles of the ALU:

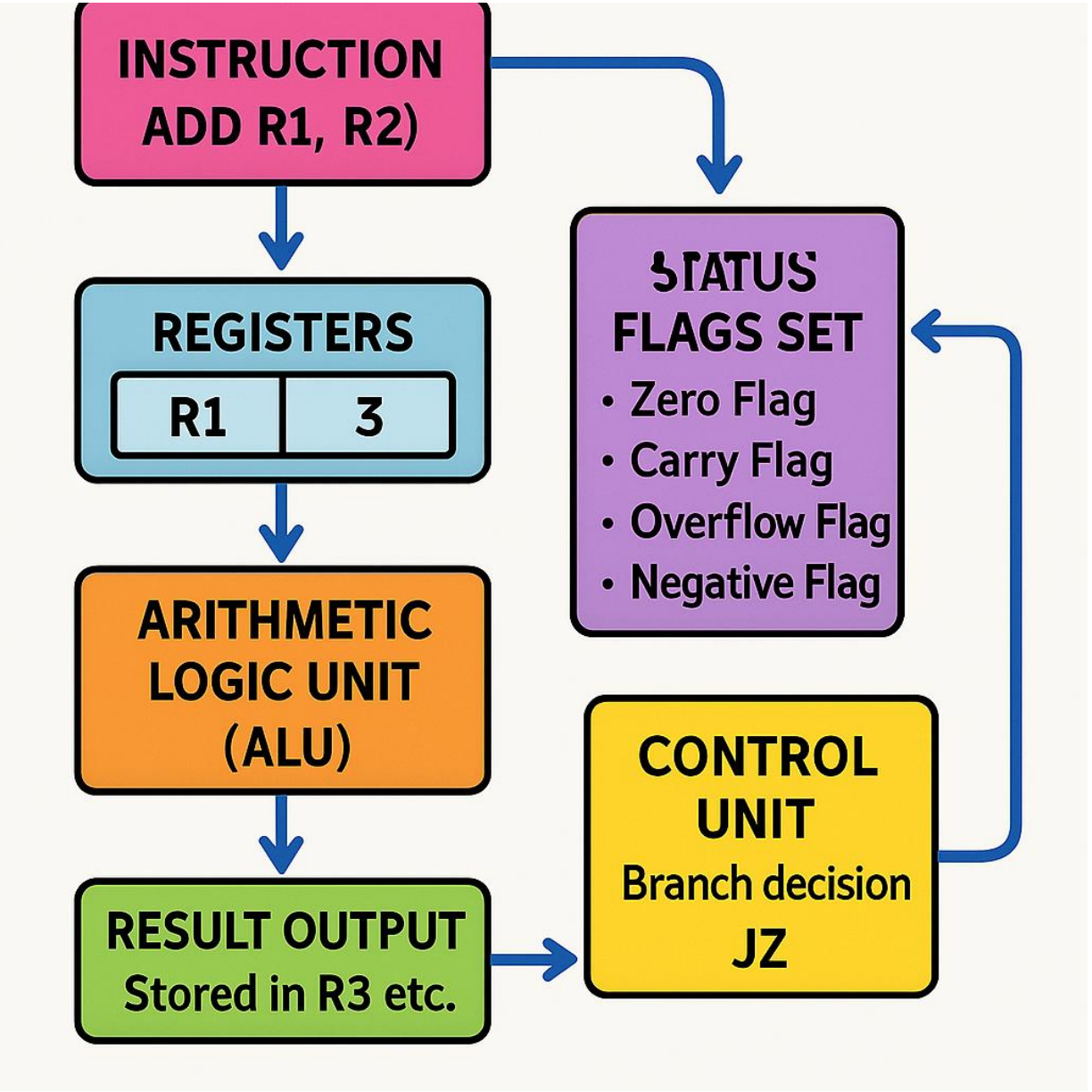
3. Flag Setting:

- After each operation, the ALU sets **status flags** (also called condition codes):
  a) **Zero flag (Z)** – set if the result is zero.
  b) **Carry flag (C)** – set if there is a carry out from the operation (useful in addition).
  c) **Overflow flag (O)** – set if there is a signed overflow.
  d) **Negative flag (N)** – set if the result is negative (in 2's complement form).
- These flags are used by the control unit for **conditional branching** (e.g., if equal, if greater).

Functional Components: ALU

Key Functions and Roles of the ALU:

3. Flag Setting:

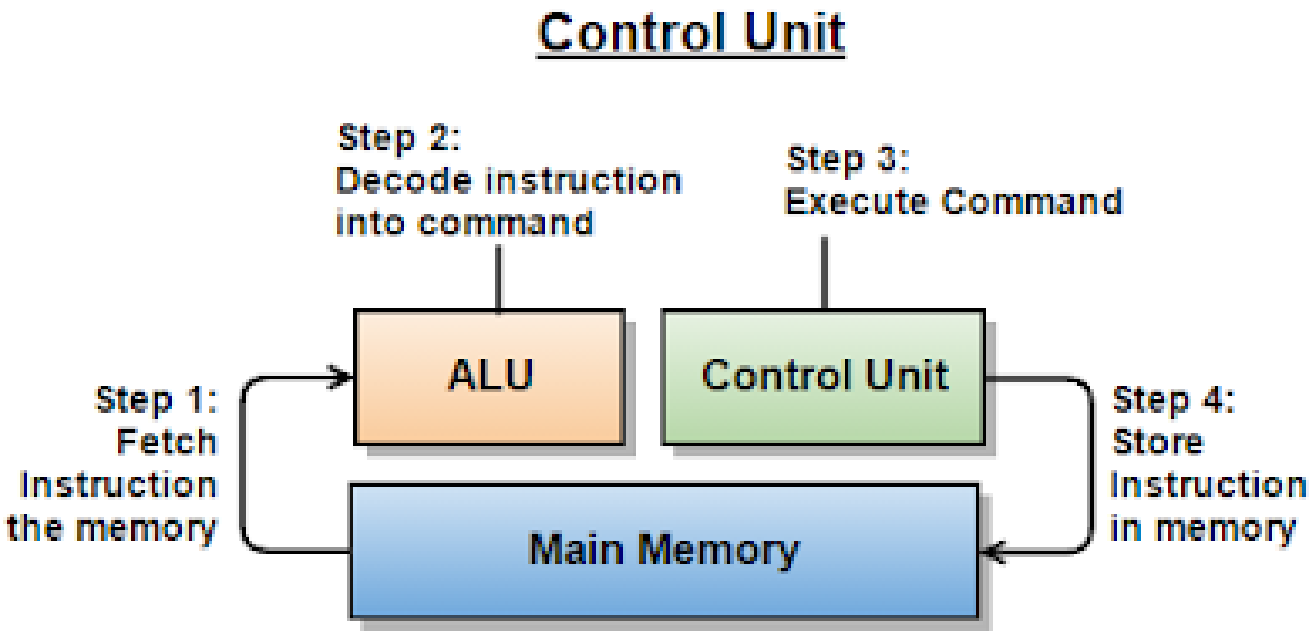


Functional Components: ALU

ALU Control: How Operations are Selected

Controlled by the Control Unit:

The **Control Unit (CU)** of the CPU sends **control signals** to the ALU based on the instruction currently being executed.



Functional Components: ALU

ALU Control: How Operations are Selected

Operation Code (Opcode):

- The **opcode** (operation code) part of a machine instruction determines what operation the ALU should perform.
- Example:
  - ✓ Opcode 0000 → ADD
  - ✓ Opcode 0001 → SUBTRACT
  - ✓ Opcode 0010 → AND
  - ✓ Opcode 0011 → OR

Functional Components: ALU

ALU Control: How Operations are Selected

ALU Control Signals:

- The CU decodes the instruction and translates it into **ALU control lines**.
- These control lines configure the ALU circuitry to perform a specific operation.

MUX Usage Inside ALU:

- Internally, **multiplexers (MUXes)** are often used in the ALU to select the desired operation circuitry depending on control inputs.

Functional Components: ALU

ALU Control: How Operations are Selected

Example:

Let’s say we execute the instruction ADD R1, R2, R3:

- **Control Unit** decodes the ADD instruction.
- It sends **control signals** to ALU to perform an addition.
- ALU takes the values in **R2** and **R3**, adds them, and stores the result in **R1**.
- Flags are updated accordingly (e.g., zero if result is 0, carry if there's an overflow).

Functional Components: Registers

Registers are small, high-speed storage units inside the CPU that temporarily hold data, instructions, or addresses during processing.

Types of Registers

1. Data Registers
2. Address Registers
3. General Purpose Registers (GPRs)
4. Special Purpose Registers
5. Control Registers
6. Segment Registers
7. Memory Buffer Register (MBR)

Functional Components: Registers

Types of Registers

1. Data Registers

Hold operands and intermediate results of operations.

- **Accumulator (ACC)**: Central to many CPUs for arithmetic/logic results.

Functional Components: Registers

Types of Registers

2. Address Registers

Store memory addresses.

- **Memory Address Register (MAR)**: Holds address to be accessed in RAM.
- **Base Register**: Holds the starting address for memory access.
- **Index Register**: Used for indexed addressing in loops or arrays.

Functional Components: Registers

Types of Registers

3. General Purpose Registers (GPRs)

Used for general data storage and manipulation.

- Example: R0–R15 in ARM, EAX/EBX in x86.

Functional Components: Registers

Types of Registers

4. Special Purpose Registers

- **Program Counter (PC)**: Holds the address of the next instruction.
- **Instruction Register (IR)**: Contains the currently executing instruction.
- **Status/Flag Register**: Stores condition flags (Zero, Carry, Overflow, Sign, etc.).
- **Stack Pointer (SP)**: Points to the top of the stack in memory.
- **Frame Pointer (FP)**: Used for stack frame management in function calls.

Functional Components: Registers

Types of Registers

5. Control Registers

Used by the CPU to control internal operations.

- **Control Register (CR)**: Used in memory management (e.g., CR0–CR4 in x86).
- **Processor Status Word (PSW)**: Combines status and control bits.

Functional Components: Registers

Types of Registers

6. Segment Registers (specific to segmented architectures like x86)

Hold the base addresses of code, data, stack, and extra segments.

- **CS (Code Segment)**, **DS (Data Segment)**, **SS (Stack Segment)**, **ES (Extra Segment)**.

7. Memory Buffer Register (MBR)

Temporarily holds data read from or written to memory.

Functional Components: Control Unit

- The part of the CPU that manages and coordinates all operations of the computer system.

- The Control Unit acts like a traffic controller, ensuring all parts of the CPU work in harmony.

- The **Control Unit (CU)** does not process data itself but orchestrates how and when data is processed by other components like the ALU and memory.

Functional Components: Control Unit

Key Functions of CU:

1) **Fetch**: Retrieves the next instruction from memory (using the Program Counter).
2) **Decode**: Interprets the fetched instruction to determine the required action.
3) **Execute/Control**: Sends appropriate control signals to:
   - ALU (for arithmetic/logical operations)
   - Registers (for data movement)
   - Memory (for read/write)
   - I/O devices (for input/output operations)

Functional Components: Control Unit

Types of Control Units:

1. Hardwired Control

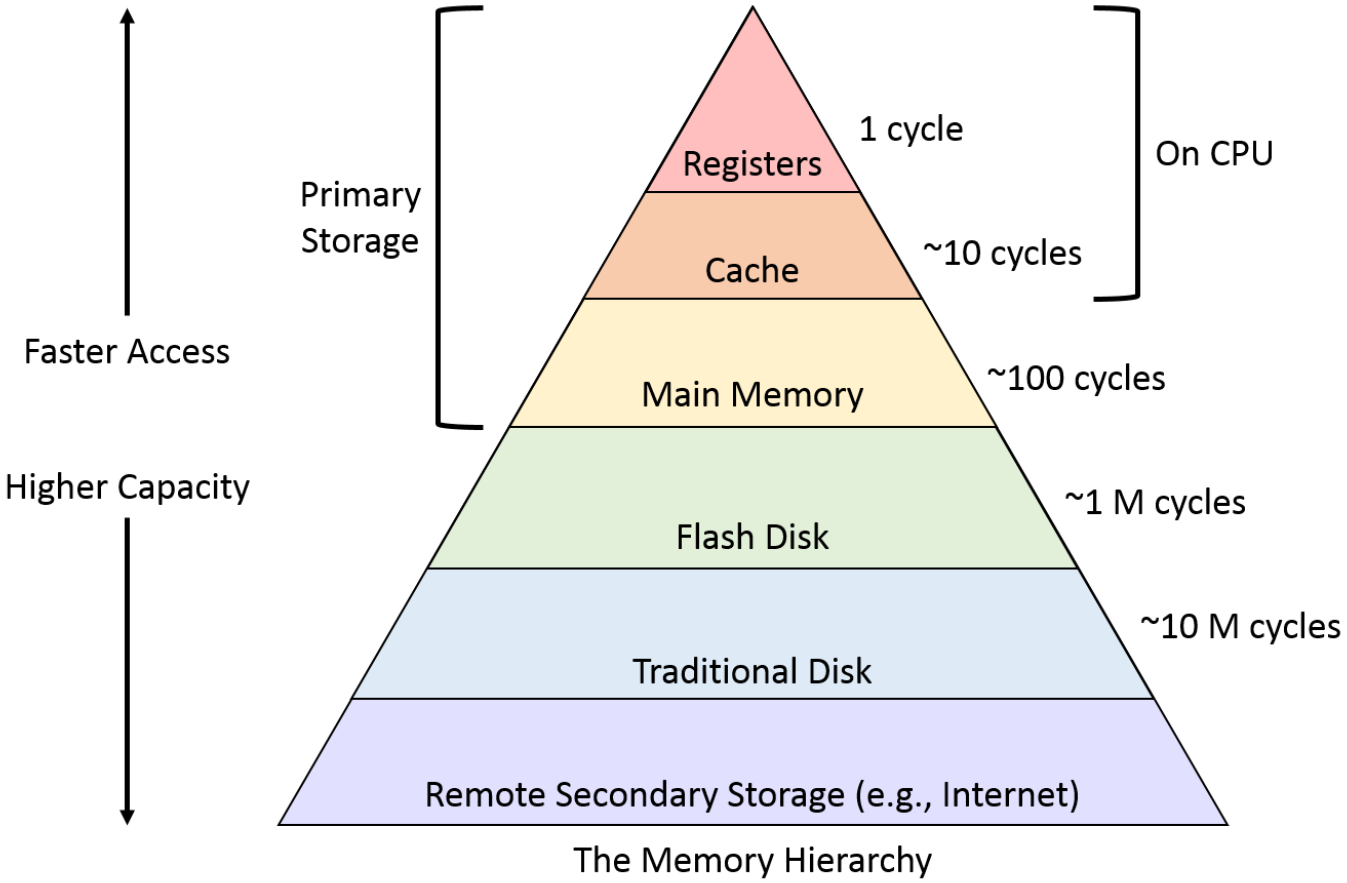
- Uses fixed logic circuits (combinational logic).
- Fast but inflexible.
- Common in RISC (Reduced Instruction Set Computer)processors.

2. Microprogrammed Control

- Uses microinstructions stored in a control memory (firmware).
- Easier to modify and more flexible.
- Common in CISC (Complex Instruction Set Computer) processors.

Functional Components: Memory

- Memory stores data and instructions needed for processing.
- It is organized in a **hierarchy** based on speed, size, and cost.



The Memory Hierarchy

- A **CPU clock cycle** is a single tick of the processor's clock.
- Each cycle is a basic unit of time in which the CPU can perform part of an instruction.

Functional Components: Memory

Memory Hierarchy (Top = Fastest, Most Expensive):

1. **Registers**:
    1. Located inside the CPU.
    2. Extremely fast and small.
    3. Store immediate values for current instructions.
2. **Cache Memory (L1, L2, L3)**:
    1. Stores frequently accessed data/instructions.
    2. L1 is fastest and closest to the CPU; L3 is slower but larger.
    3. Reduces time to access data from RAM.

Functional Components: Memory

Memory Hierarchy (Top = Fastest, Most Expensive):

3. **Main Memory (RAM)**:
    1. Temporarily holds data and programs currently in use.
    2. Volatile (data is lost when power is off).
    3. Organized into **bytes and words**, accessed by unique **addresses**.
4. **Secondary Storage (HDD/SSD)**:
    1. Permanent storage for programs and files.
    2. Much slower than RAM but large in capacity.
    3. Non-volatile (data retained without power).

Functional Components: Memory

Key Concepts:

- **Memory Address Register (MAR)**:

  Holds the address of the memory location to read from or write to.

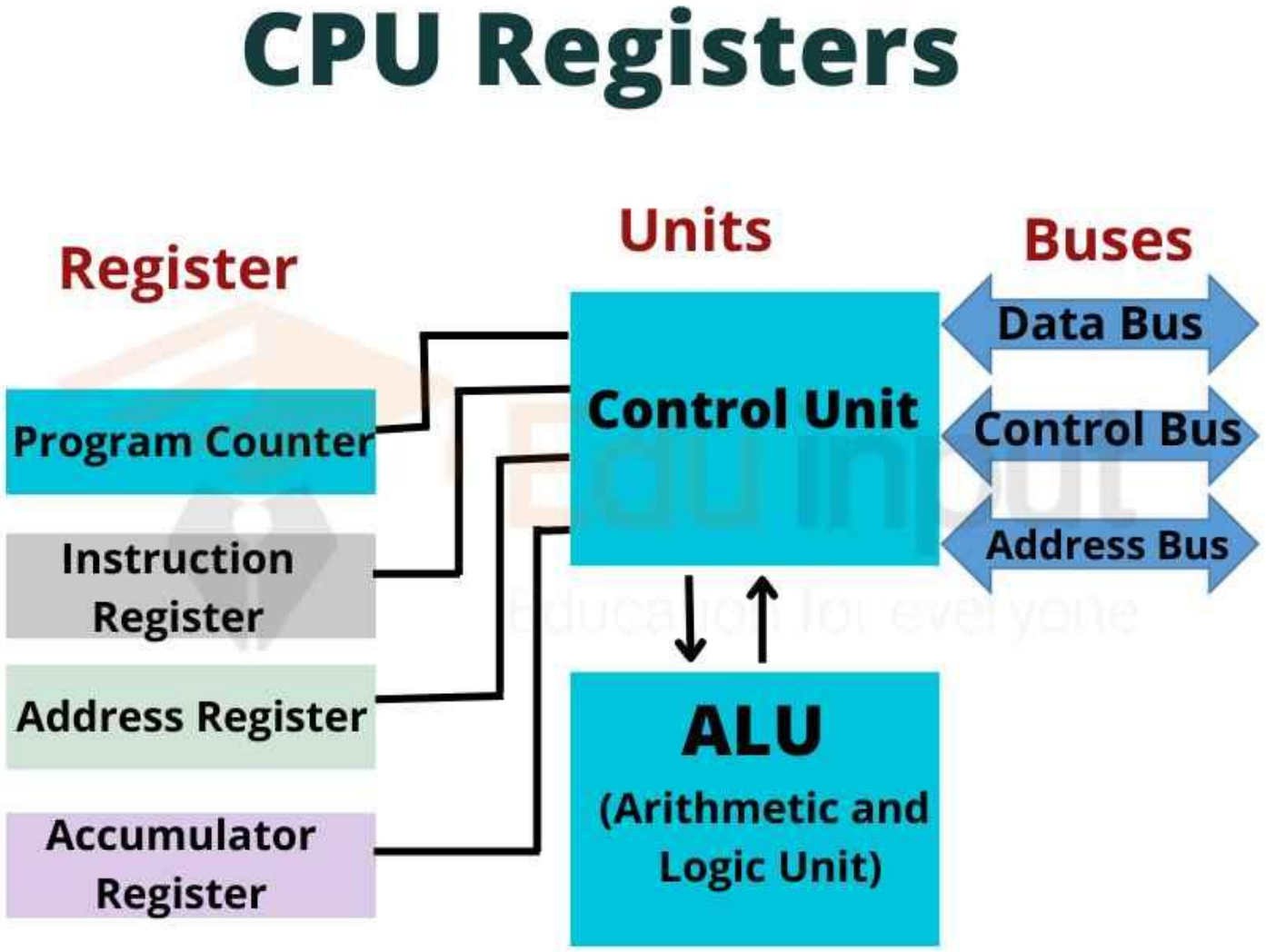
- **Memory Buffer Register (MBR)**:

  Temporarily holds the data being transferred to/from memory.

- **Bus Systems**:

  - ✓ **Data Bus**: Carries data between CPU, memory, and I/O.
  - ✓ **Address Bus**: Carries memory addresses.
  - ✓ **Control Bus**: Carries control signals (e.g., Read/Write).

Functional Components: Memory



Functional Components: Input/Output (I/O)

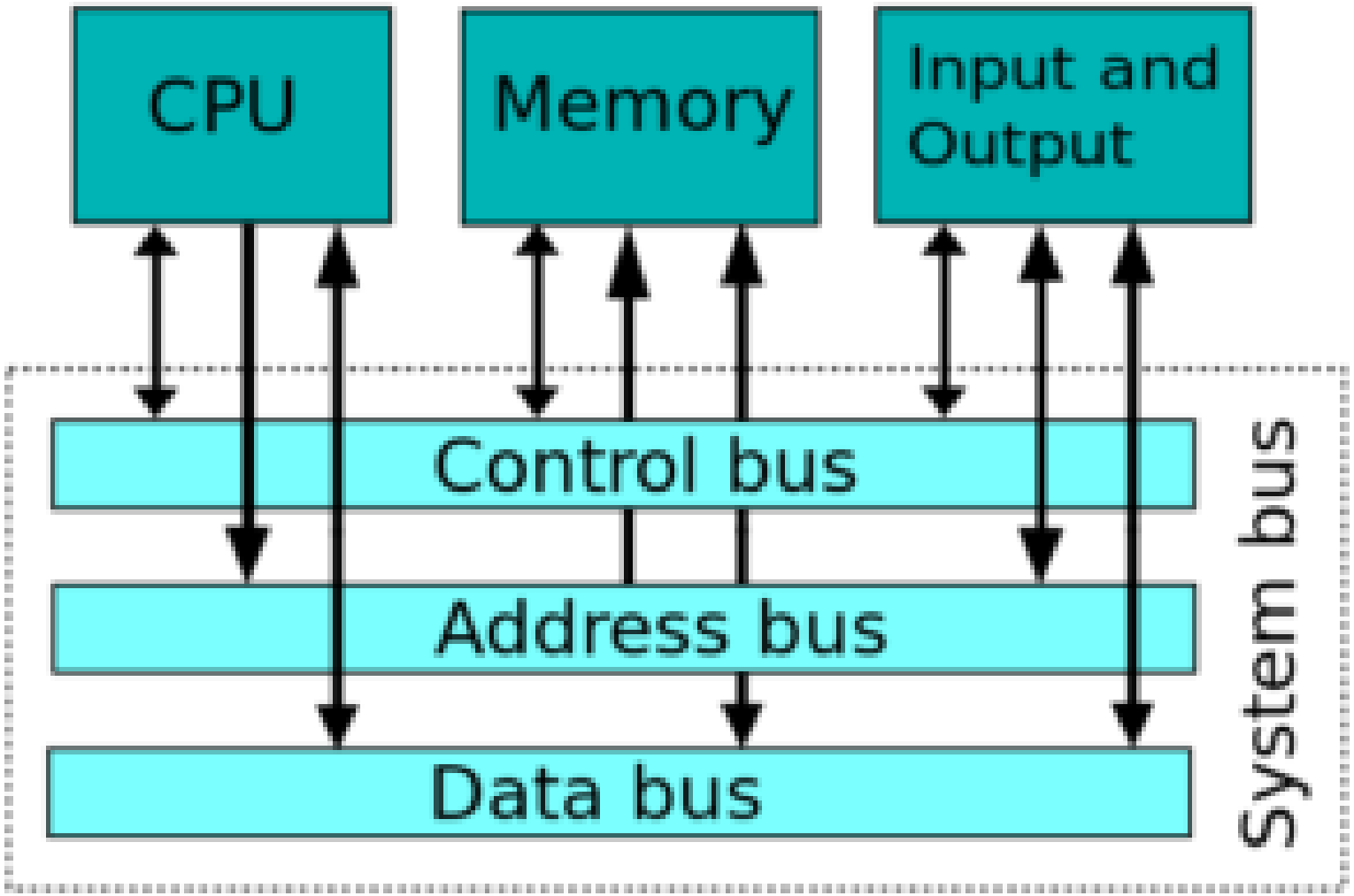
Allows external devices (keyboard, mouse, disk, printer, etc.) to **communicate** with the **CPU and memory**.

Main Components:

- **I/O Modules**:

Act as the **interface** between peripheral devices and the CPU. They manage data transfer, status checking, and command handling.

Functional Components: Input/Output (I/O)



Functional Components: Input/Output (I/O)

Main Components:

- **I/O Buses and Controllers**:
  - ✓ **Bus**: Transfers data, addresses, and control signals between CPU, memory, and I/O devices.
  - ✓ **Controller**: Manages a specific type of device (e.g., disk controller, display controller).

Functional Components: Input/Output (I/O)

I/O Techniques:

1. **Programmed I/O**:
    1. CPU actively waits and controls the entire I/O operation.
    2. Simple but inefficient (CPU is blocked).

2. **Interrupt-Driven I/O**:
    1. CPU issues I/O command, continues other tasks.
    2. Device sends an **interrupt** when ready; CPU handles it.
    3. More efficient than programmed I/O.

Functional Components: Input/Output (I/O)

I/O Techniques:

3.Direct Memory Access (DMA):

1. I/O module transfers data **directly to/from memory** without CPU involvement.
2. CPU is only interrupted once at the end.
3. High-speed and efficient for large data transfers.

Functional Components: Input/Output (I/O)

I/O Addressing:

1. Memory-Mapped I/O:

- I/O devices share the **same address space** as memory.
- Standard load/store instructions are used for I/O access

2. Isolated I/O (Port-Mapped I/O):

- I/O devices have a **separate address space**.
- Requires special instructions like IN and OUT.

AMITY UNIVERSITY BENGALURU

Module 1: Fundamentals of Computer Organization

- Introduction to Computer Systems
- Difference between Organization and Architecture
- Functional components: ALU, registers, control unit, memory, I/O
- **Overview of instruction execution (Fetch, Decode, Execute)**
- Concept of bus systems and data flow

Overview of instruction execution (Fetch, Decode, Execute)

- The **Instruction Cycle** is the fundamental operation cycle of a computer.
- Every program instruction is processed through a repetitive cycle: **Fetch** → **Decode** → **Execute**.
- These steps occur under the control of the **Control Unit (CU)** of the CPU.

Overview of instruction execution (Fetch, Decode, Execute)

a) Fetch Phase

Retrieve the next instruction from memory.

Steps:

- The **Program Counter (PC)** holds the address of the next instruction.
- This address is sent to **Memory**.
- The instruction is fetched and loaded into the **Instruction Register (IR)**.
- PC is incremented to point to the next instruction.

Key Components:

- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Instruction Register (IR)

Overview of instruction execution (Fetch, Decode, Execute)

b) Decode Phase

Interpret the instruction's meaning.

Steps:

- The **Opcode** is extracted from the instruction in IR.
- **Control Unit** interprets the opcode to identify the operation.
- If necessary, operand addresses are determined (direct, indirect, immediate modes).

Process:

- Opcode decoding
- Identifying operand(s)
- Determining addressing mode

Overview of instruction execution (Fetch, Decode, Execute)

c) Execute Phase

Carry out the operation.

Steps depend on instruction type:

- **Arithmetic/Logic:** ALU performs the operation.
- **Data Transfer:** Data moved between CPU and memory/I/O.
- **Control Transfer:** PC updated to new address.

Results may be stored in:

- Registers
- Memory
- Flags (status bits updated: zero, negative, carry, etc.)

Overview of instruction execution (Fetch, Decode, Execute)

Example:

ADD Instruction Execution

Instruction: ADD R1, R2

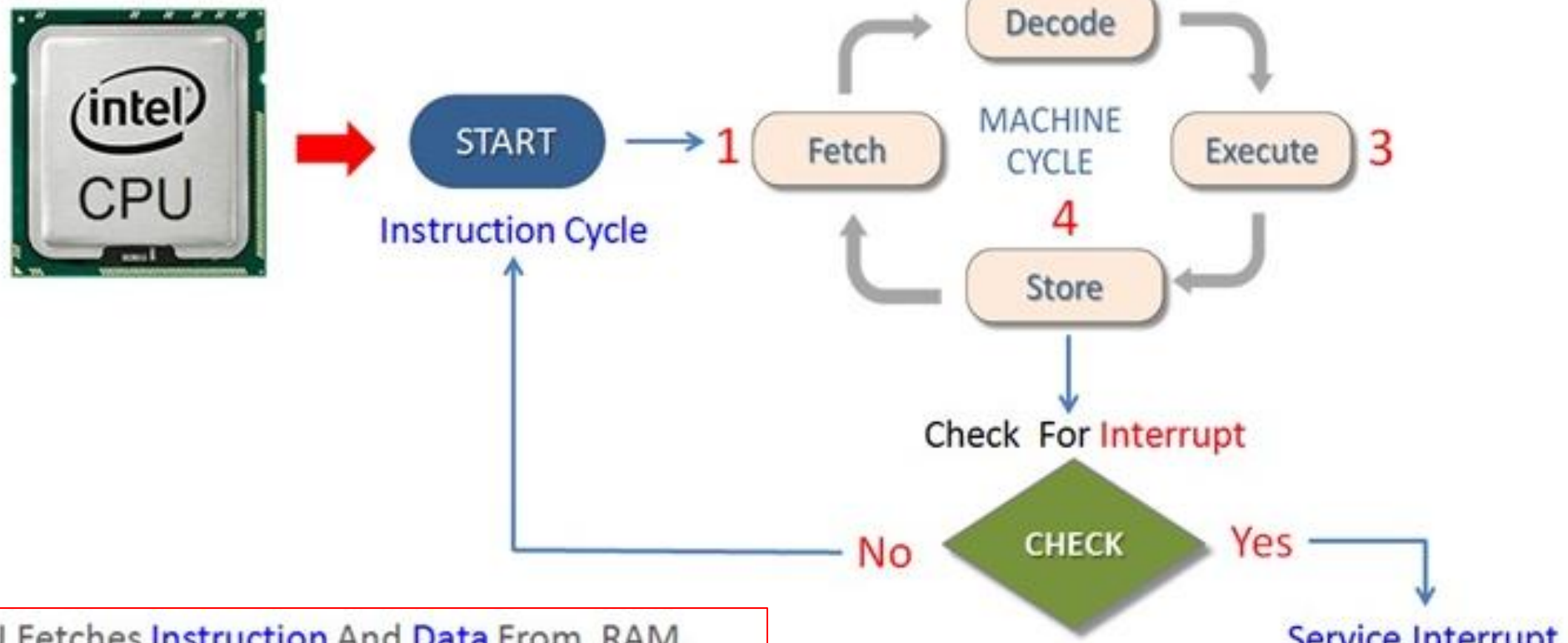
- ✓ Fetch: CPU fetches this instruction from memory.
- ✓ Decode: Opcode “ADD” is identified; operands R1 and R2 located.
- ✓ Execute: Contents of R1 and R2 are added in the ALU; result stored in R1.

Overview of instruction execution (Fetch, Decode, Execute)

Instruction Cycle with Interrupt Consideration

- After **Execute**, the CPU checks for **interrupts**.
- If an interrupt is present, CPU saves current context and services the interrupt before returning to the next instruction cycle.

Overview of instruction execution (Fetch, Decode, Execute)



1st Step - Fetch : The CPU Fetches Instruction And Data From RAM.

2nd Step - Decode : The Control Unit Decodes the Instruction .

3rd Step - Execute : The ALU Executes Instruction & Operates On Data.

4th Step - Store : The Processed Data is Stored In The RAM.

AMITY UNIVERSITY BENGALURU

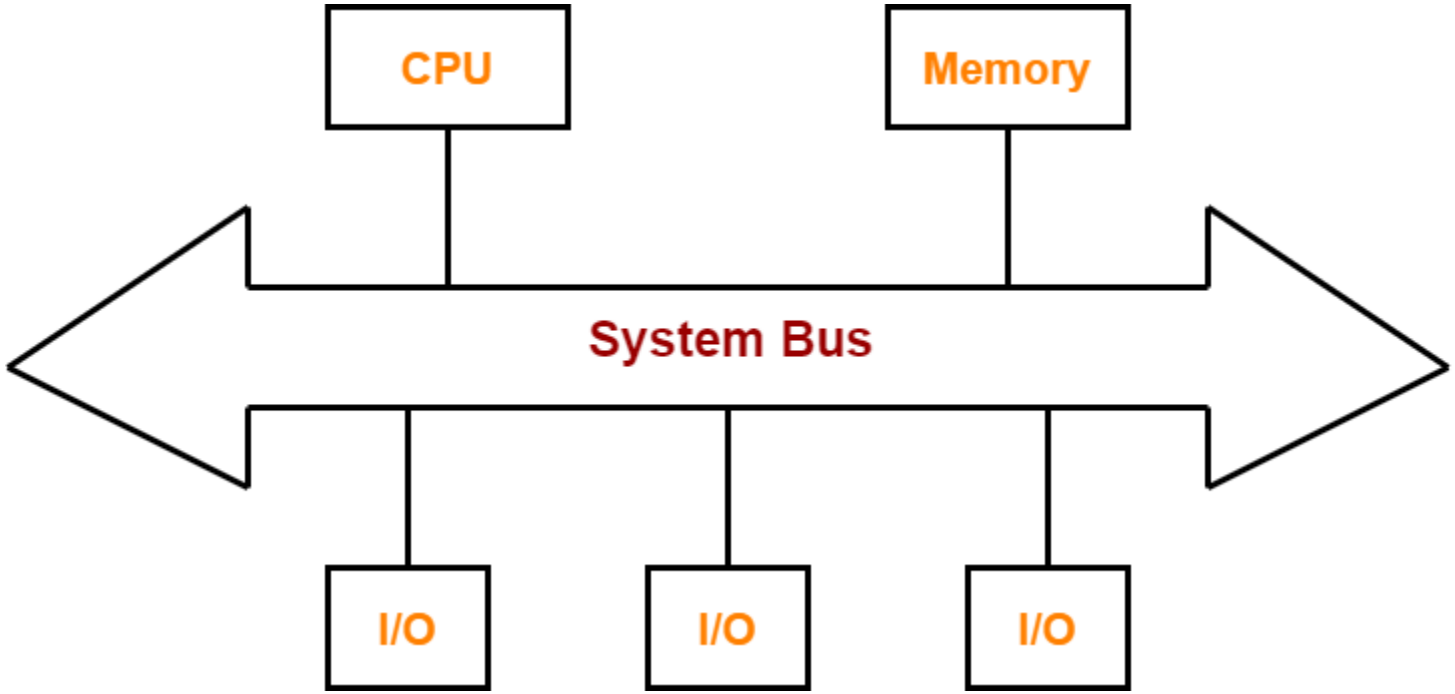
Module 1: Fundamentals of Computer Organization

- Introduction to Computer Systems
- Difference between Organization and Architecture
- Functional components: ALU, registers, control unit, memory, I/O
- Overview of instruction execution (Fetch, Decode, Execute)
- **Concept of bus systems and data flow**

Concept of bus systems and data flow

Definition of Bus

- A **bus** is a communication pathway connecting two or more devices.
- **Shared transmission medium -** only one device can transmit at a time.

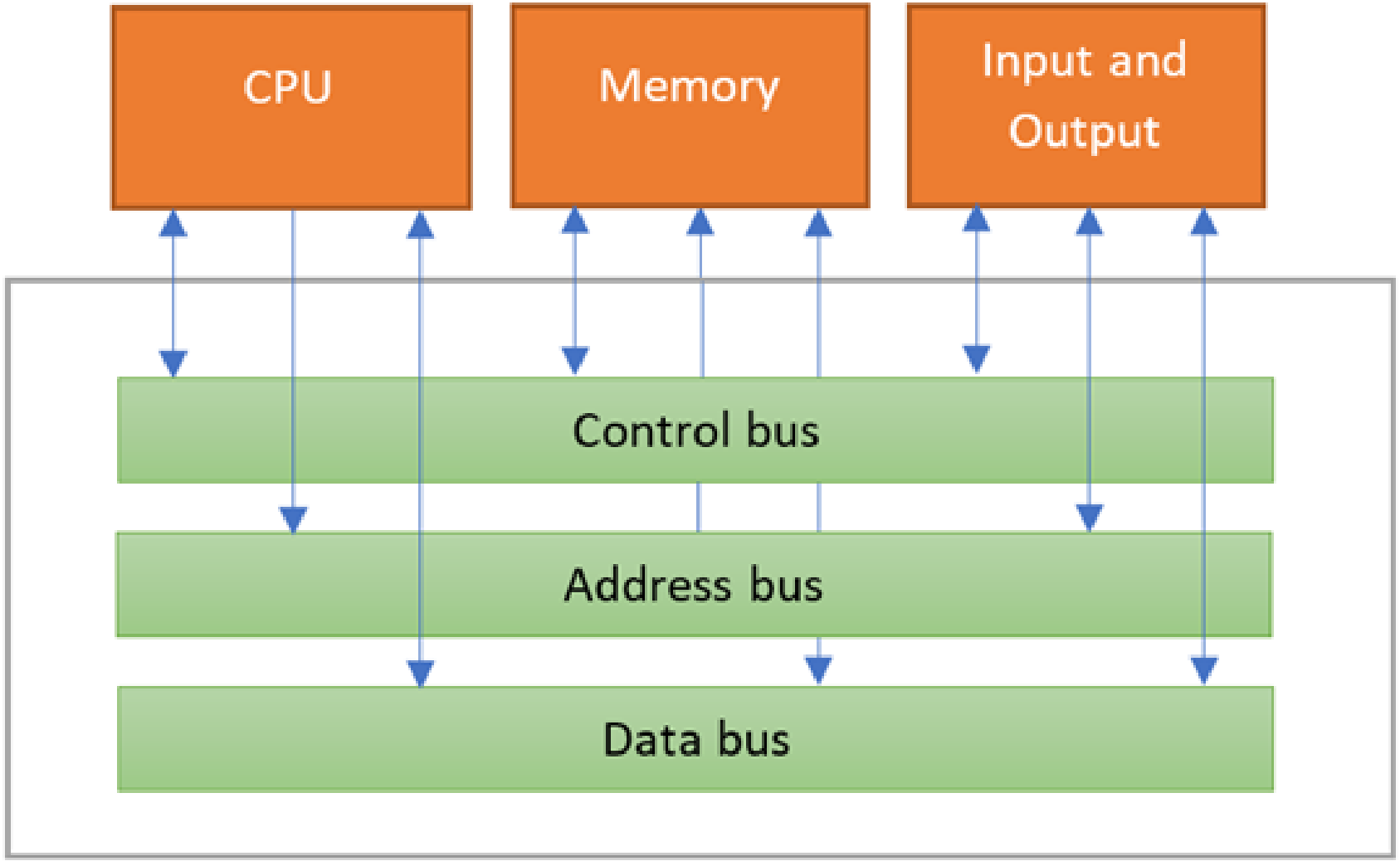


Concept of bus systems and data flow

Types of Buses

- **Data Bus**:
  - ✓ Transfers actual data between components (e.g., CPU, memory, I/O).
  - ✓ Width (e.g., 32-bit, 64-bit) determines the amount of data transmitted.
- **Address Bus**:
  - ✓ Carries the address of the data (not the data itself).
  - ✓ Width determines maximum addressable memory.
- **Control Bus**:
  - ✓ Carries control signals (e.g., read/write, clock, interrupt).
  - ✓ Controls the access and use of the data and address lines.

Concept of bus systems and data flow



Concept of bus systems and data flow

Bus Structure – Single-Bus vs. Multiple-Bus System

Feature	*Single-Bus System*	*Multiple-Bus System*
Architecture	One common bus for all components	Separate buses for CPU-memory, I/O, etc.
Speed	Slower due to bus contention	Faster, supports parallel operations
Cost	Low	High (more hardware)
Complexity	Simple	More complex due to multiple paths
Scalability	Poor (limited performance improvement)	Good (can add more devices easily)
Example	Basic microcontrollers	High-performance computers

Concept of bus systems and data flow

Bus Structure – Synchronous vs. Asynchronous Buses

Feature	*Synchronous Bus*	*Asynchronous Bus*
Timing	Uses a common **clock signal**	Uses **handshaking signals**
Speed	Fast (for short distances)	Moderate (due to handshake overhead)
Flexibility	Less flexible (devices must match clock speed)	Highly flexible (different speed devices can connect)
Complexity	Simple timing design	Complex control logic
Scalability	Poor (clock skew at large scale)	Good (no global clock needed)
Example	Internal CPU buses	USB communication

Concept of bus systems and data flow

Data Flow in Computer Systems

- **Data Path**: The route data follows through components (CPU ⇄ Memory ⇄ I/O).
- **Internal CPU data flow**:
  - ✓ Registers → ALU → Memory (via buses).
- **Instruction Cycle**:
  - ✓ **Fetch**: From memory to CPU.
  - ✓ **Decode and Execute**: Internal CPU data movement.
  - ✓ **Write-back**: Result back to register or memory.

Concept of bus systems and data flow

Bus Arbitration

When **multiple devices** request control of the bus at the **same time**, an **arbitration mechanism** is required to decide **which device gains control**.

Arbitration Techniques

1. Centralized Arbitration
2. Distributed Arbitration

Concept of bus systems and data flow

Bus Arbitration

1. Centralized Arbitration

- **Single controller** (e.g., bus arbiter or CPU) manages all bus access.
- Devices send requests to the central arbiter.
- **Priority-based decision** (e.g., fixed or rotating).
- **Simple & efficient**, but has a **single point of failure**.

Concept of bus systems and data flow

Bus Arbitration

2. Distributed Arbitration

- **All devices participate** in arbitration.
- No single controller; each device has **arbitration logic**.
- Devices decide among themselves based on a **predefined protocol**.
- **More robust**, but **more complex** than centralized.

Concept of bus systems and data flow

Bus Performance

The **performance of a system bus** plays a crucial role in determining the **overall speed and efficiency** of a computer system. It affects **data transfer rate**, system responsiveness, and CPU–I/O interaction.

Factors Affecting Bus Performance

1.Bus Width:

- ✓ Refers to the number of bits transmitted simultaneously.
- ✓ Wider buses (e.g., 64-bit vs. 32-bit) can carry **more data per clock cycle**.
- ✓ Example: A 64-bit data bus can transfer double the amount of data compared to a 32-bit bus in the same time..

Concept of bus systems and data flow

Factors Affecting Bus Performance

2. Bus Speed (Clock Rate):

- ✓ The rate at which data is transmitted, measured in MHz or GHz.
- ✓ Higher clock rates mean **faster data transmission**.
- ✓ Limitation: Speed is often constrained by electrical and physical design limitations (e.g., signal integrity).

3. Bus Contention and Latency:

- ✓ **Contention** occurs when multiple devices request the bus at the same time.
- ✓ **Latency** is the delay before data starts transferring.
- ✓ Both reduce effective bus performance as they **introduce wait times**.

Concept of bus systems and data flow

Direct Memory Access (DMA)

What is DMA?

DMA is a feature that allows **I/O devices** (like disk drives, network cards) to **directly read from or write to main memory without CPU involvement** for every byte of data.

Concept of bus systems and data flow

Direct Memory Access (DMA)

Benefits

- **Frees up CPU**:
  - ✓ CPU does not have to manage each data transfer.
  - ✓ Can perform **other tasks** while data is being transferred.
- **Speeds up Data Movement**:
  - ✓ Transfers are **faster and more efficient** than CPU-managed I/O.
  - ✓ Ideal for **large blocks of data** (e.g., file transfers, video streaming).
- **Efficient Bus Usage**:
  - ✓ DMA controller uses the **system bus independently**, but only after being **granted bus access** (through bus arbitration).

Concept of bus systems and data flow

Direct Memory Access (DMA)

How DMA Works (Step-by-Step)

1. **CPU initializes DMA**:
   - ✓ Sets up source/destination addresses and data length in the DMA controller.
2. **DMA requests bus access**:
   - ✓ When ready, the DMA controller requests control of the bus.
3. **DMA performs transfer**:
   - ✓ Once granted, it **transfers data directly** between I/O device and memory.
4. **DMA completion**:
   - ✓ It raises an **interrupt** to notify the CPU when the transfer is complete.

Concept of bus systems and data flow

Techniques to Improve Bus Performance

1. Bus Hierarchies

1. Multiple buses (local, system, I/O) reduce traffic on the main bus.
2. Example: CPU ↔ Memory uses a fast local bus; peripherals use slower I/O bus.

2. Dedicated High-Speed Buses

1. Devices like GPUs/SSDs get separate high-speed buses.
2. Avoids main bus congestion, boosts speed and responsiveness.

3. Buffers & DMA

1. **Buffers** handle speed mismatch between devices.
2. **DMA** allows direct memory access without CPU, reducing load and speeding transfers.

AMITY UNIVERSITY BENGALURU

Thank You

Dr. Vignesh Ramamoorthy. H

Associate Professor

AIIT - AUB