

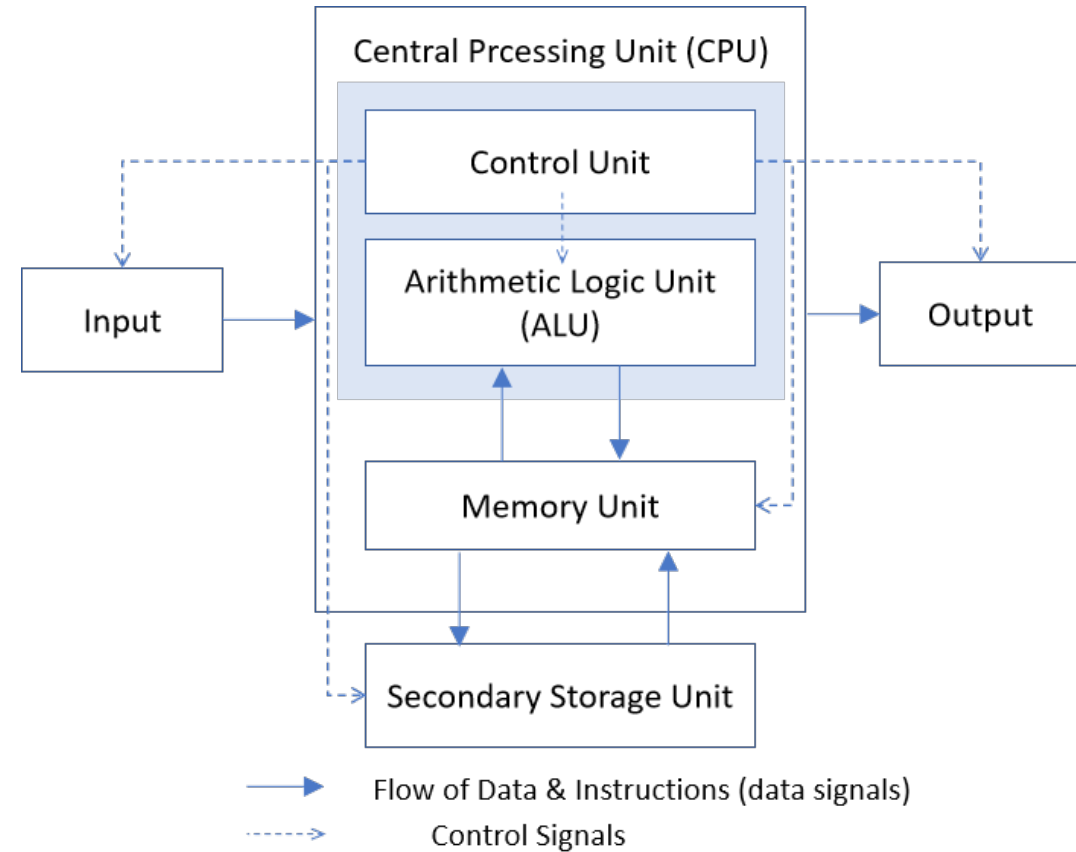
# Fundamentals of Computer Organization & Digital Electronics

# What you will learn?

- Basic Organization of Computers
- Classification Micro, Mini, Mainframe and Super Computer
- System Bus and Interconnection
- PCI, Computer Function, I-Cycle
- Interrupt and Class of Interrupts

# Basic Organization of Computers

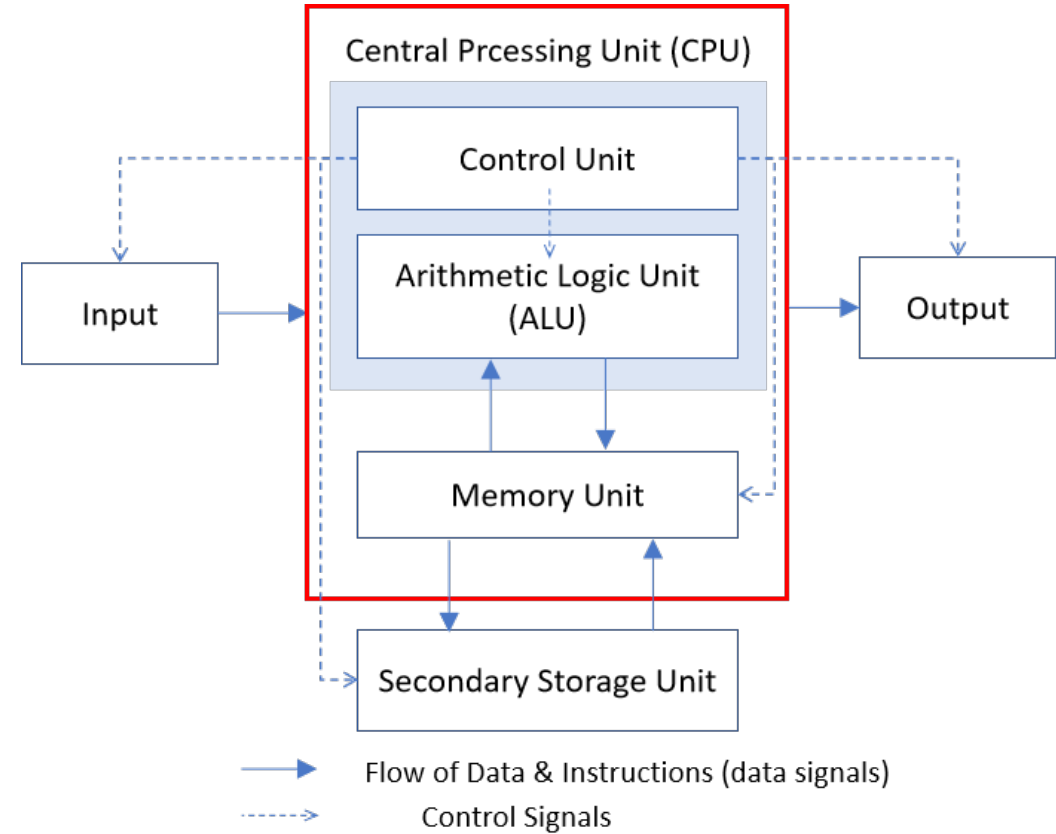
The organization of a computer system involves its fundamental components and their interconnections, which work together to perform various computational tasks. This organization can be broadly categorized into hardware and software components.



# Core Components of a Computer System

## Central Processing Unit (CPU)

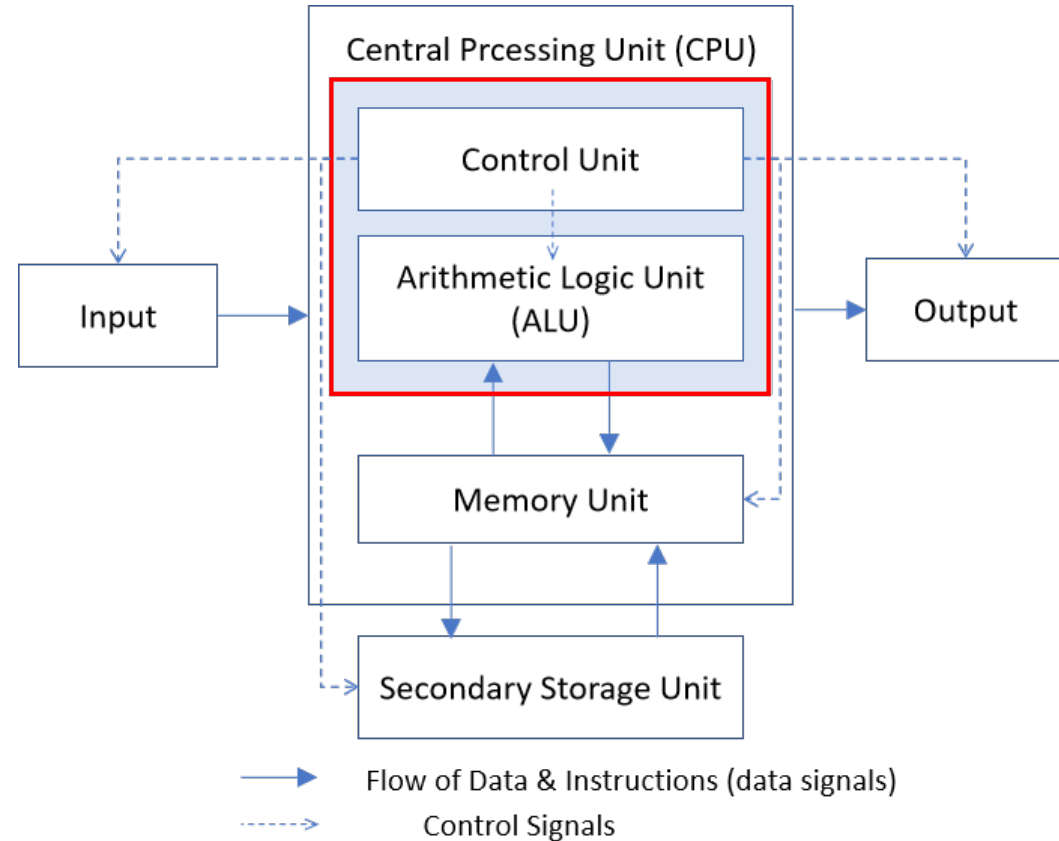
The CPU is often referred to as the "brain" of the computer. It consists of two primary subcomponents:



# Core Components of a Computer System

**Arithmetic Logic Unit (ALU):** Responsible for performing arithmetic and logical operations.

**Control Unit (CU):** Directs the operation of the processor and coordinates the activities of all other components by fetching and interpreting instructions from memory



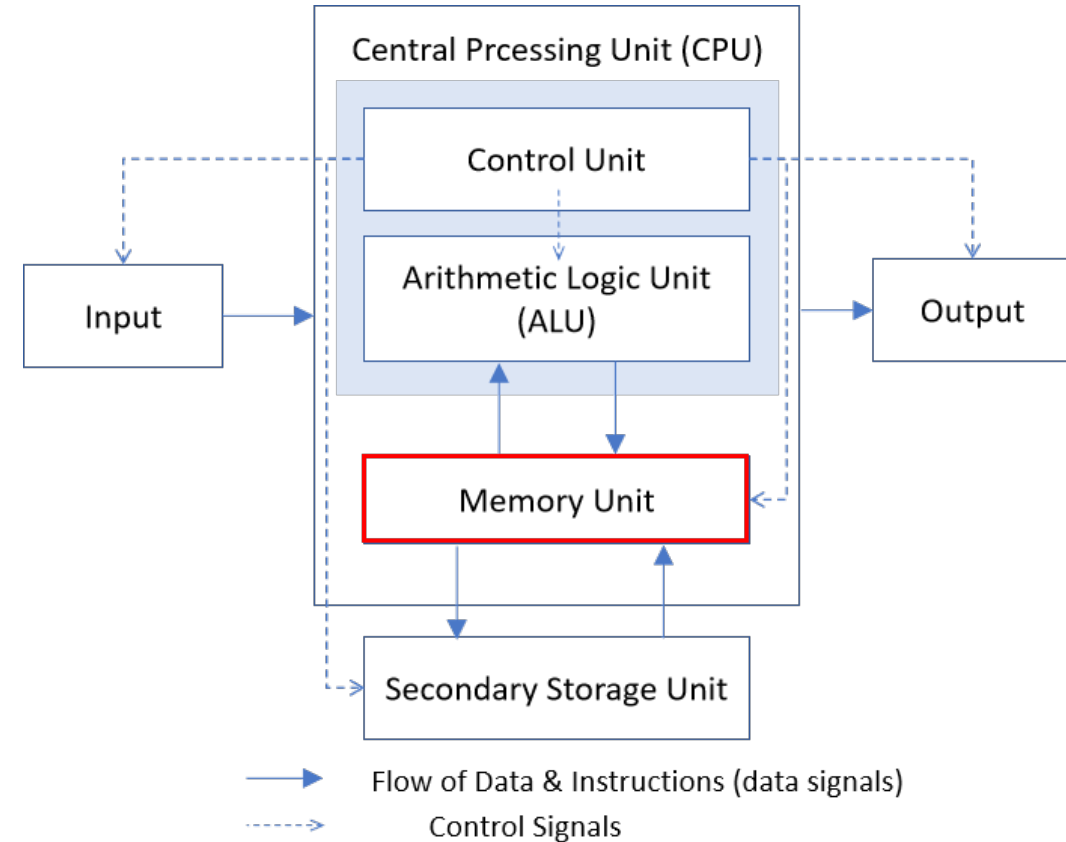
# Core Components of a Computer System

## Main Memory/ Memory Unit / Primary Memory

Main memory, commonly known as Random Access Memory (RAM), temporarily holds data and instructions that the CPU uses during processing.

It is organized as a sequence of memory locations, each identified by a unique address<sup>13</sup>.

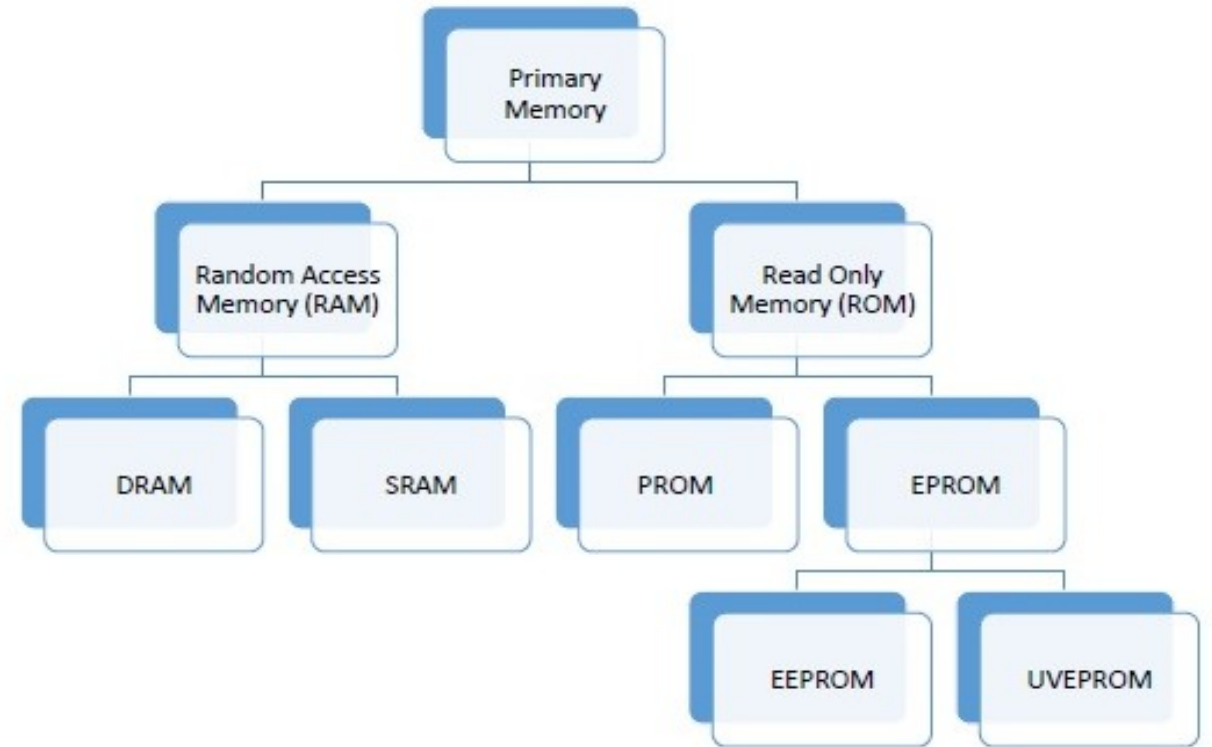
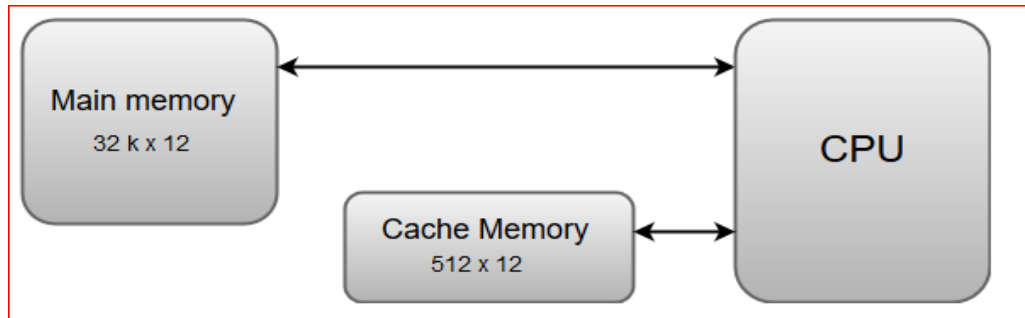
Memory can be byte-addressable, meaning each address corresponds to one byte of data



# Core Components of a Computer System

Basically, Computer memory is categorized into two parts:

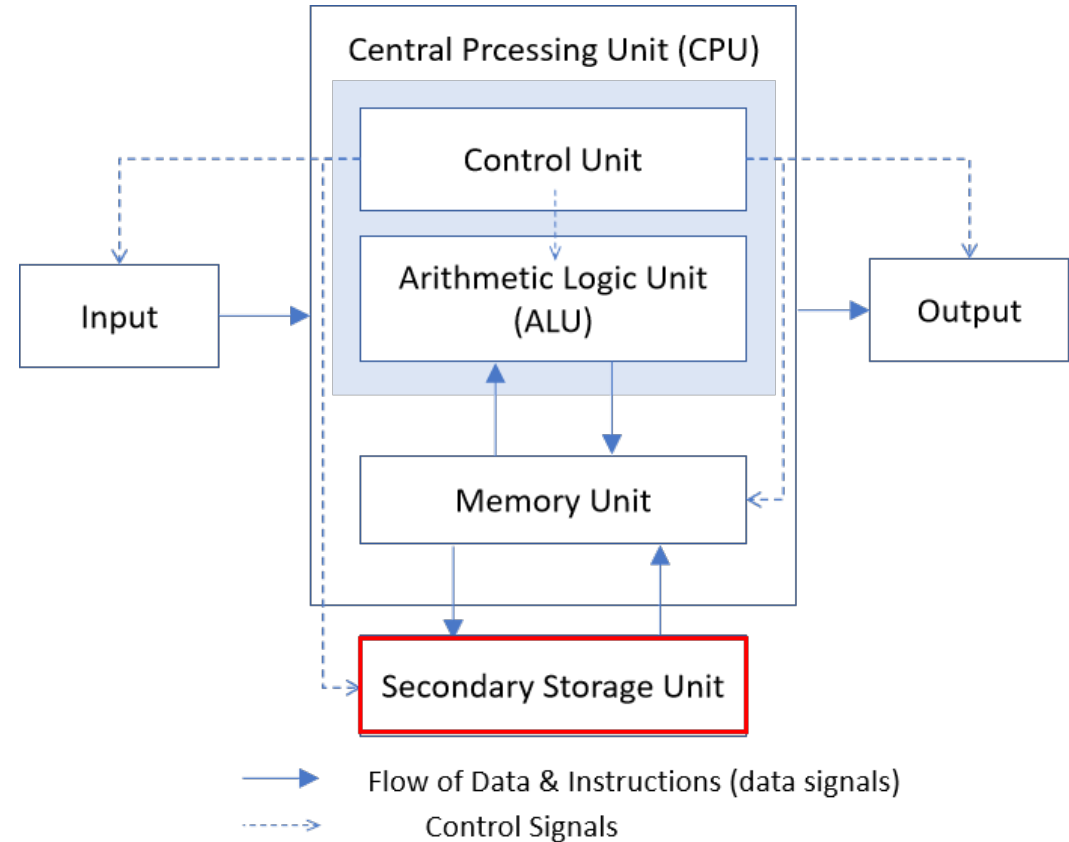
- (i) Primary Memory
- (ii) Secondary Memory



# Core Components of a Computer System

## Secondary Memory Devices

- If we need to store large amount of data or programs permanently, we need a cheaper and permanent memory. Such memory is called secondary memory.
- Characteristics of secondary memory:-
  - It is non-volatile, i.e. it retains data when power is switched off
  - It is large capacities to the tune of terabytes
  - It is cheaper as compared to primary memory

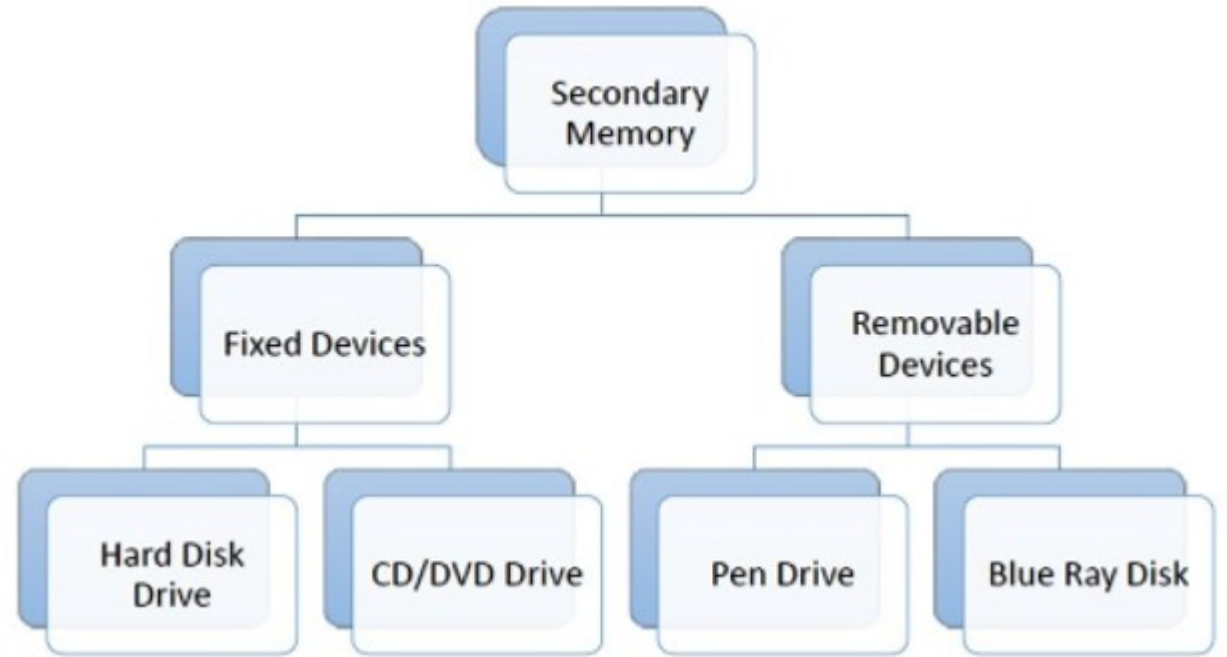




# Core Components of a Computer System

## Secondary Memory Devices

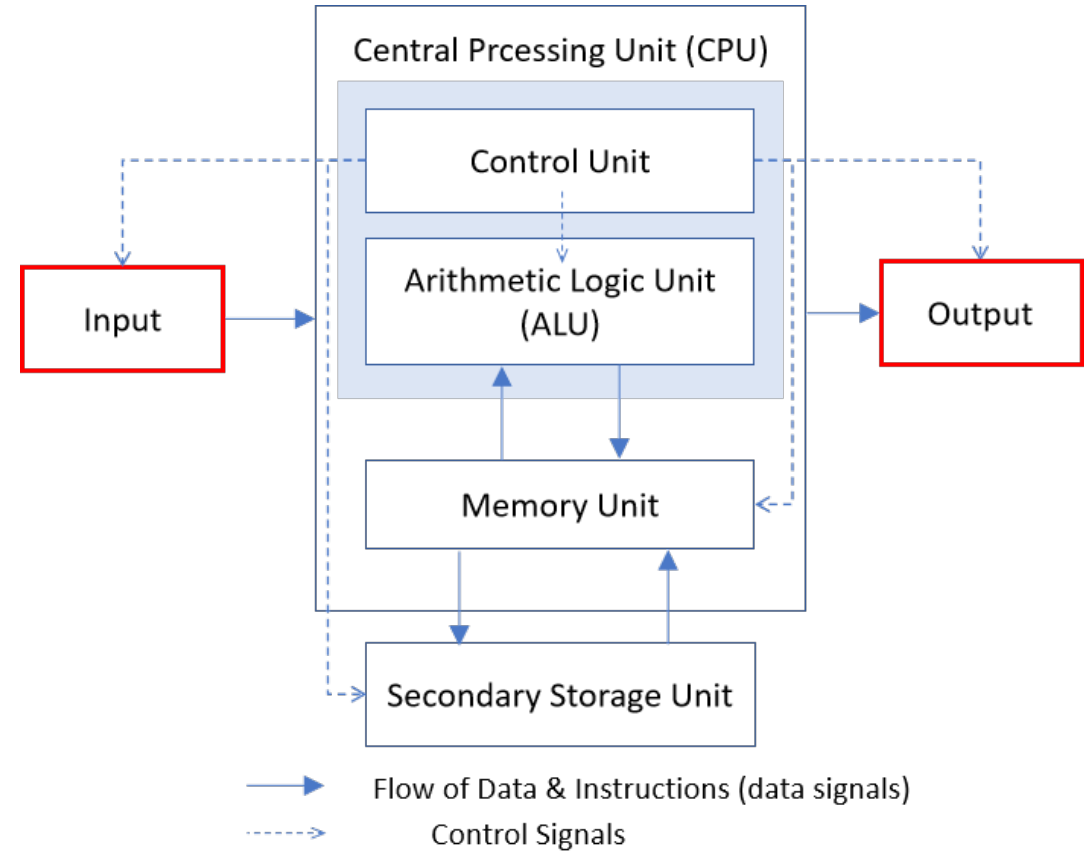
- If we need to store large amount of data or programs permanently, we need a cheaper and permanent memory. Such memory is called secondary memory.
- Characteristics of secondary memory:-
  - It is non-volatile, i.e. it retains data when power is switched off
  - It is large capacities to the tune of terabytes
  - It is cheaper as compared to primary memory



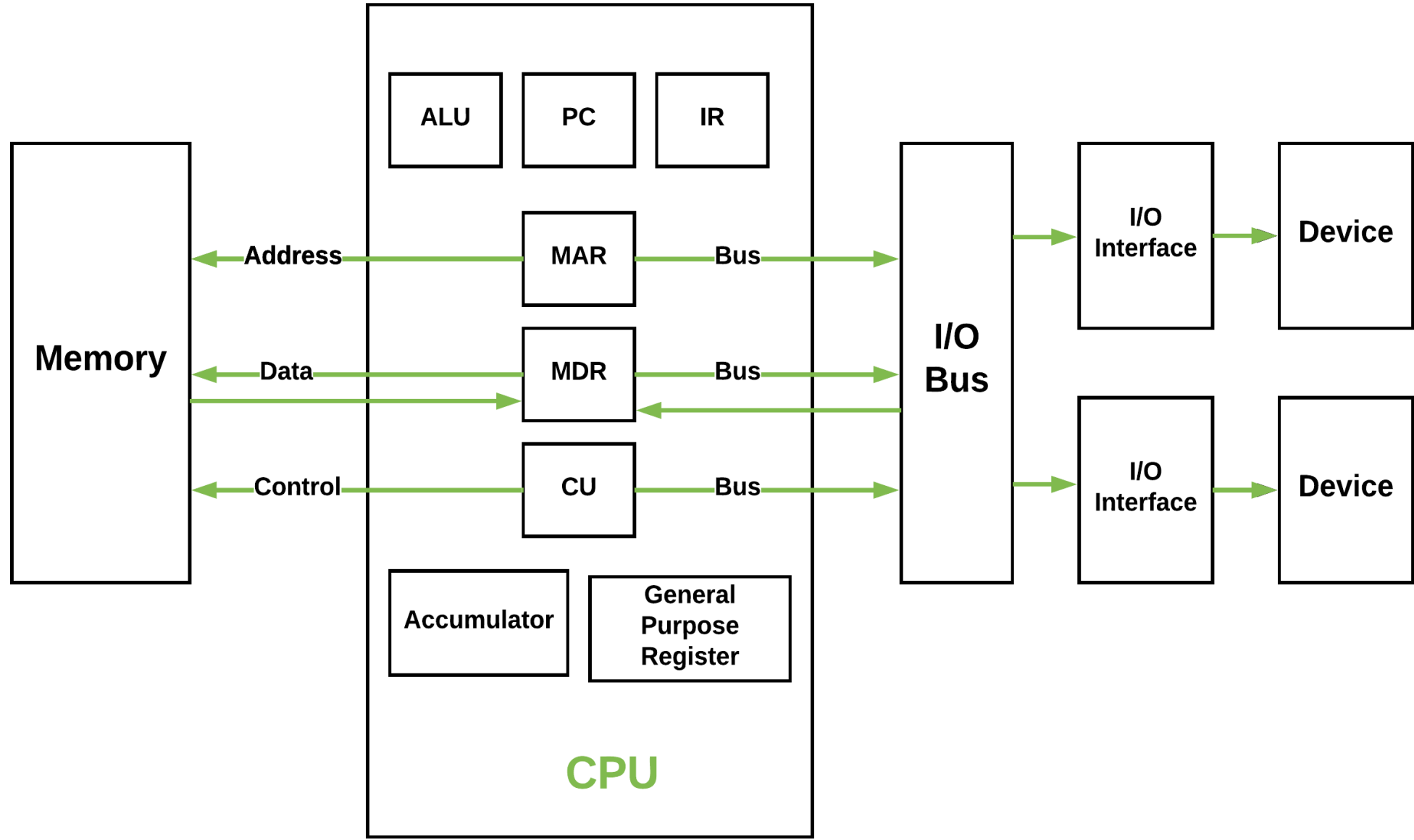
# Core Components of a Computer System

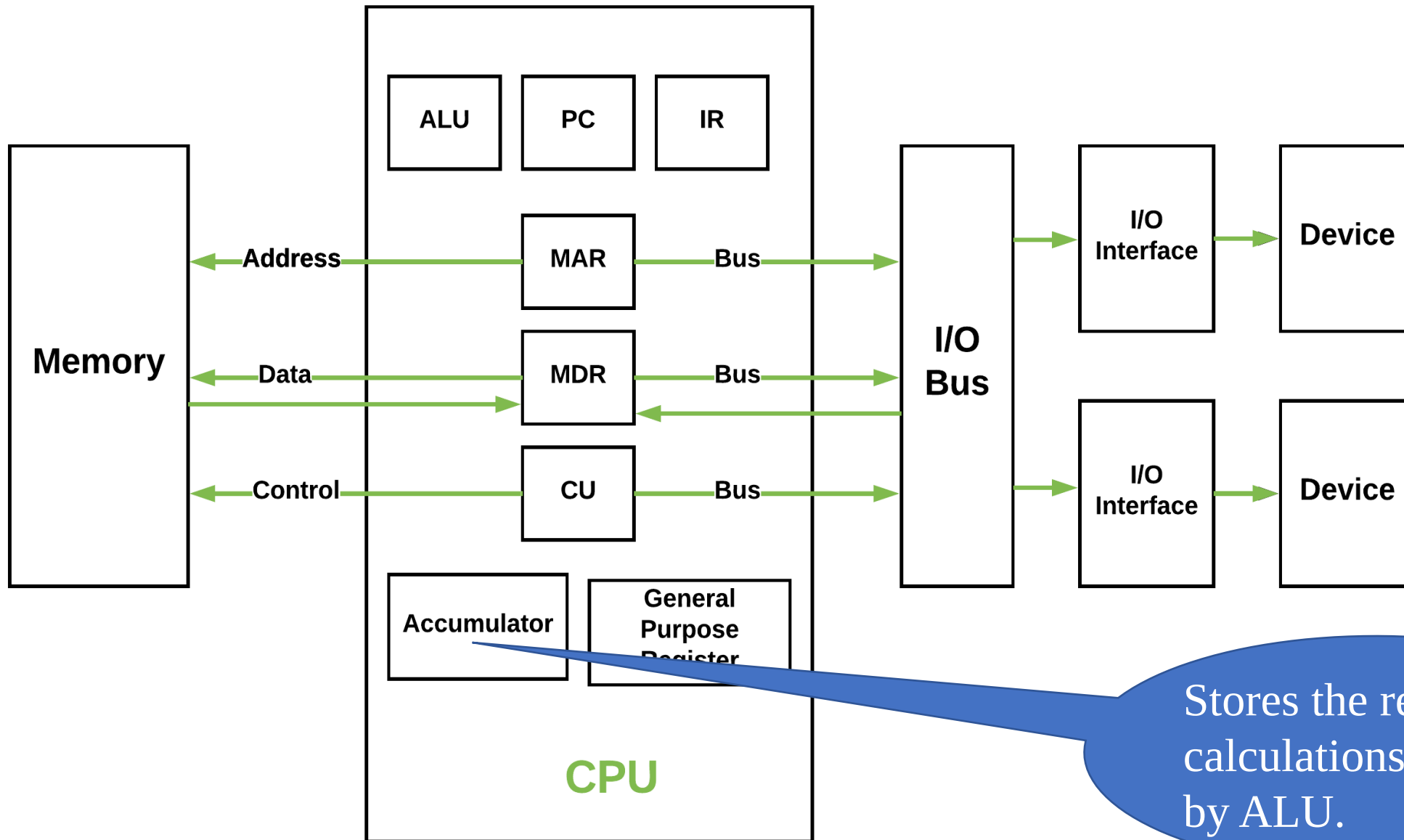
- Input/Output Devices

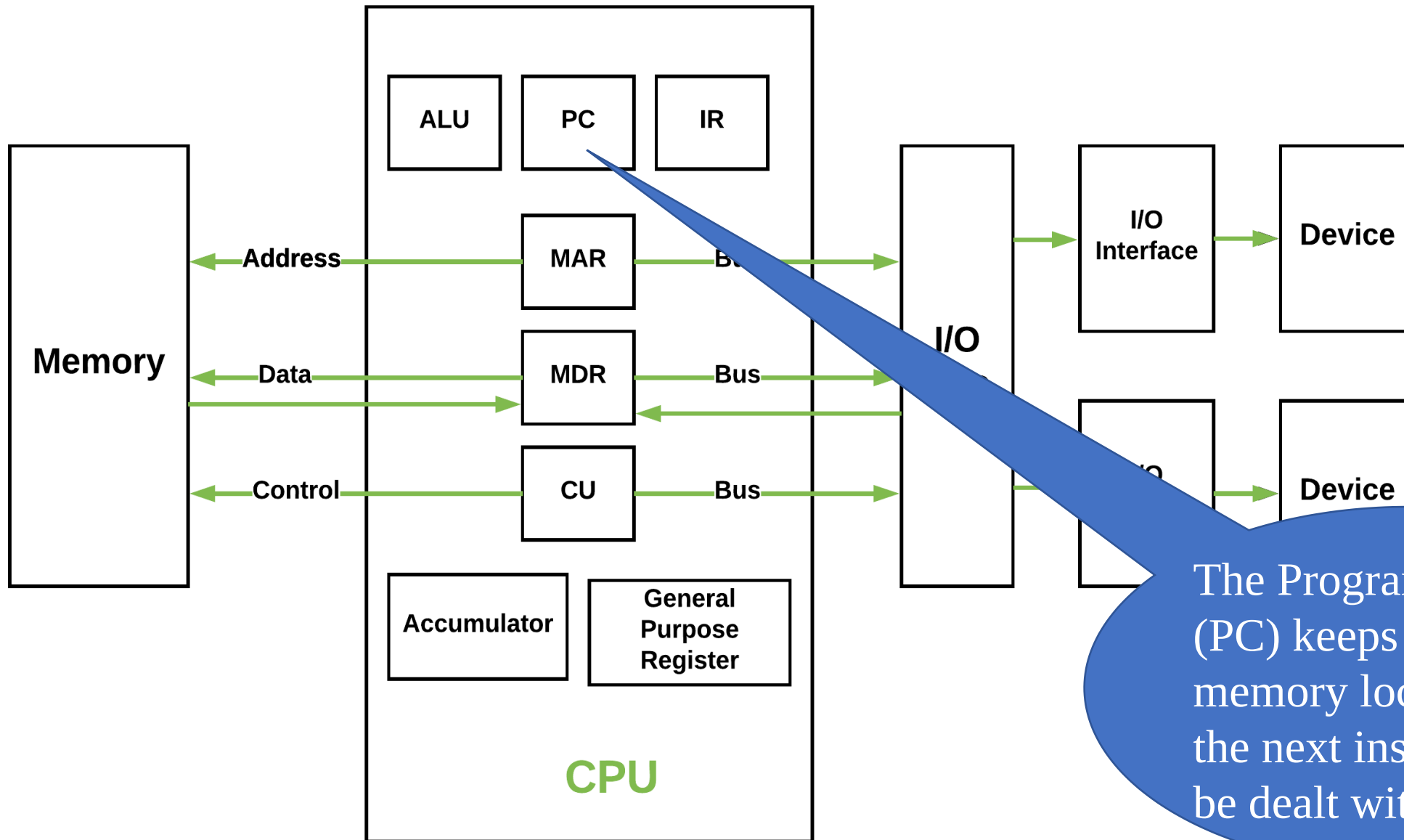
These devices facilitate communication between the computer and the external environment. Input devices (like keyboards and mice) allow users to enter data, while output devices (like monitors and printers) present processed data



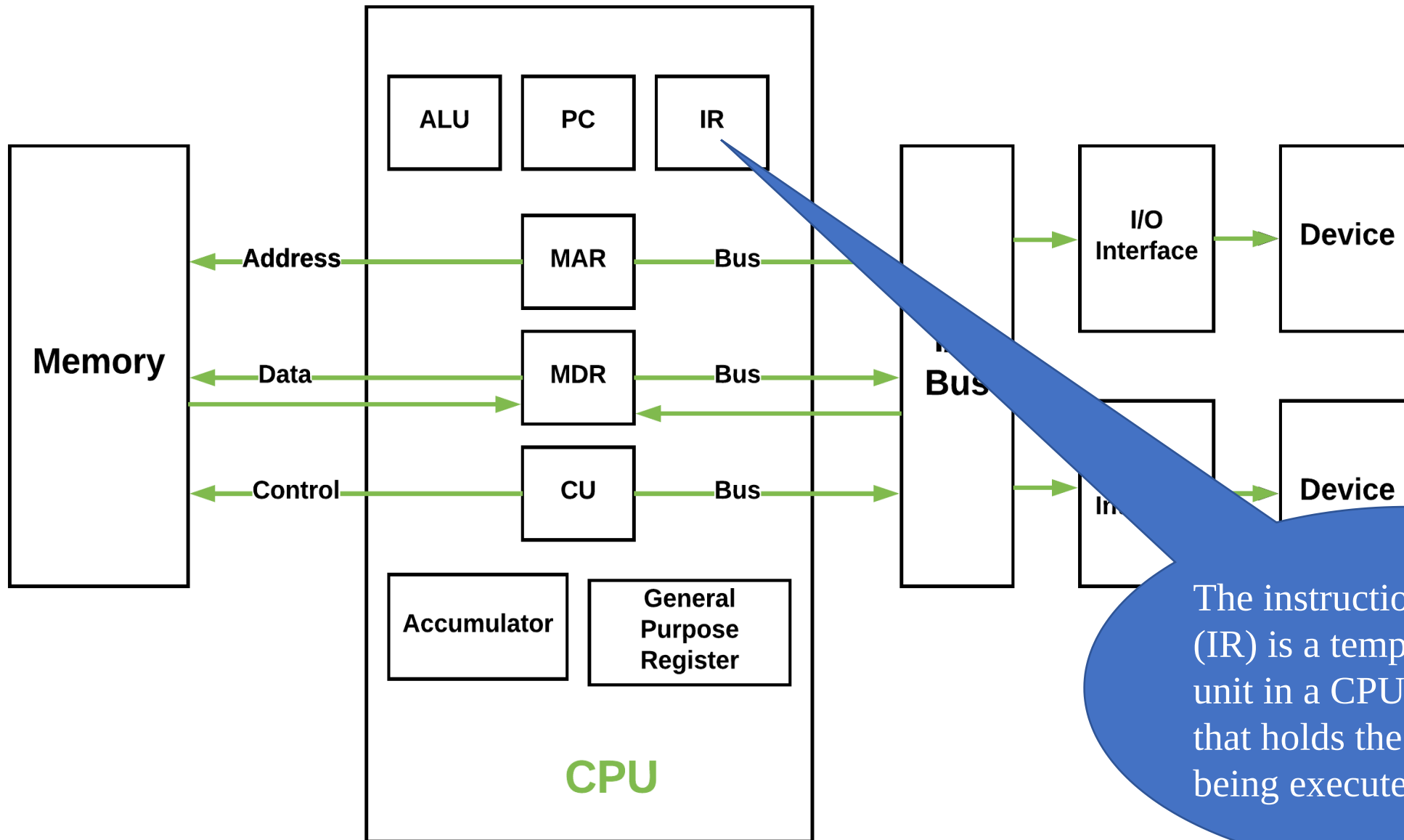
# Basic CPU structure, illustrating ALU



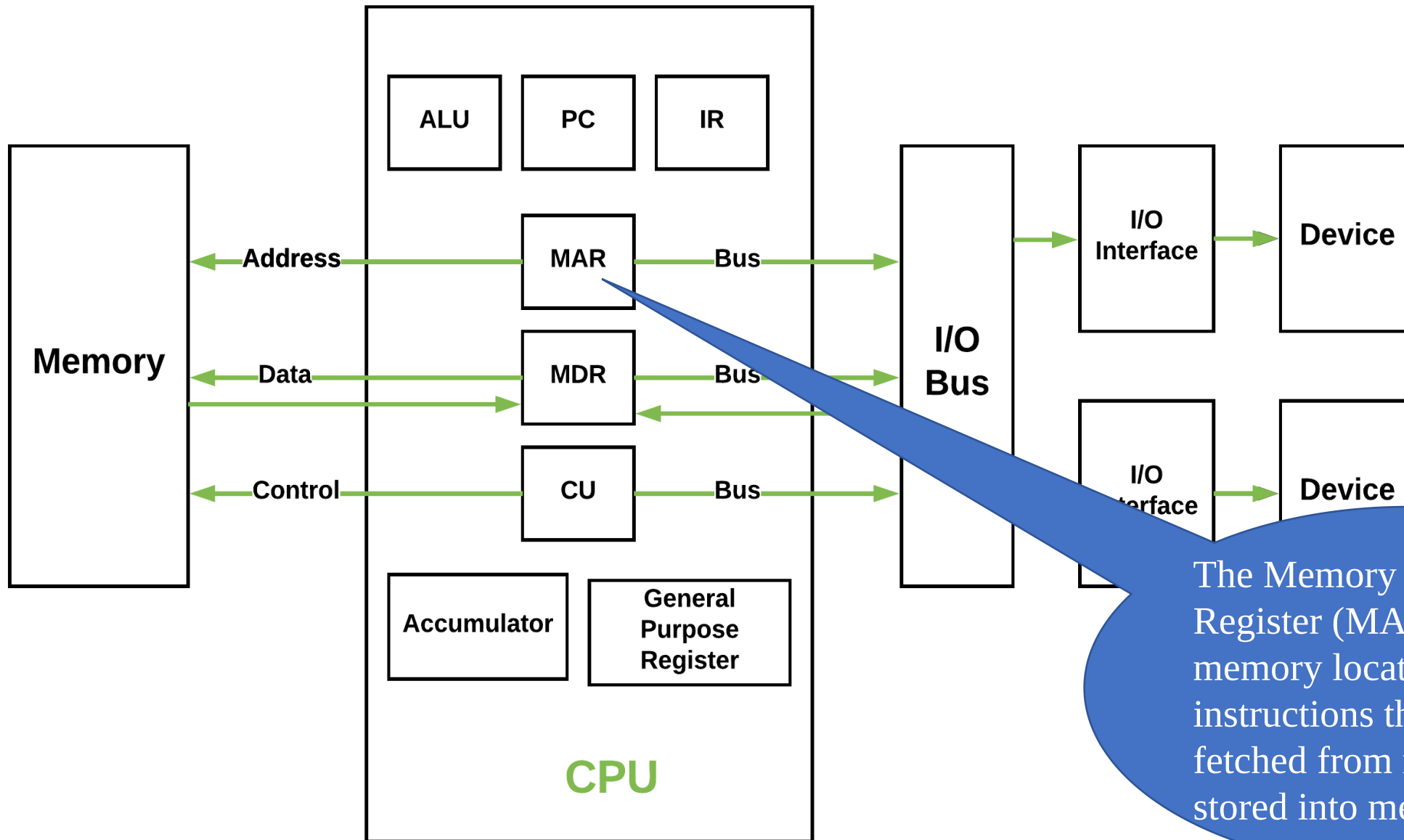




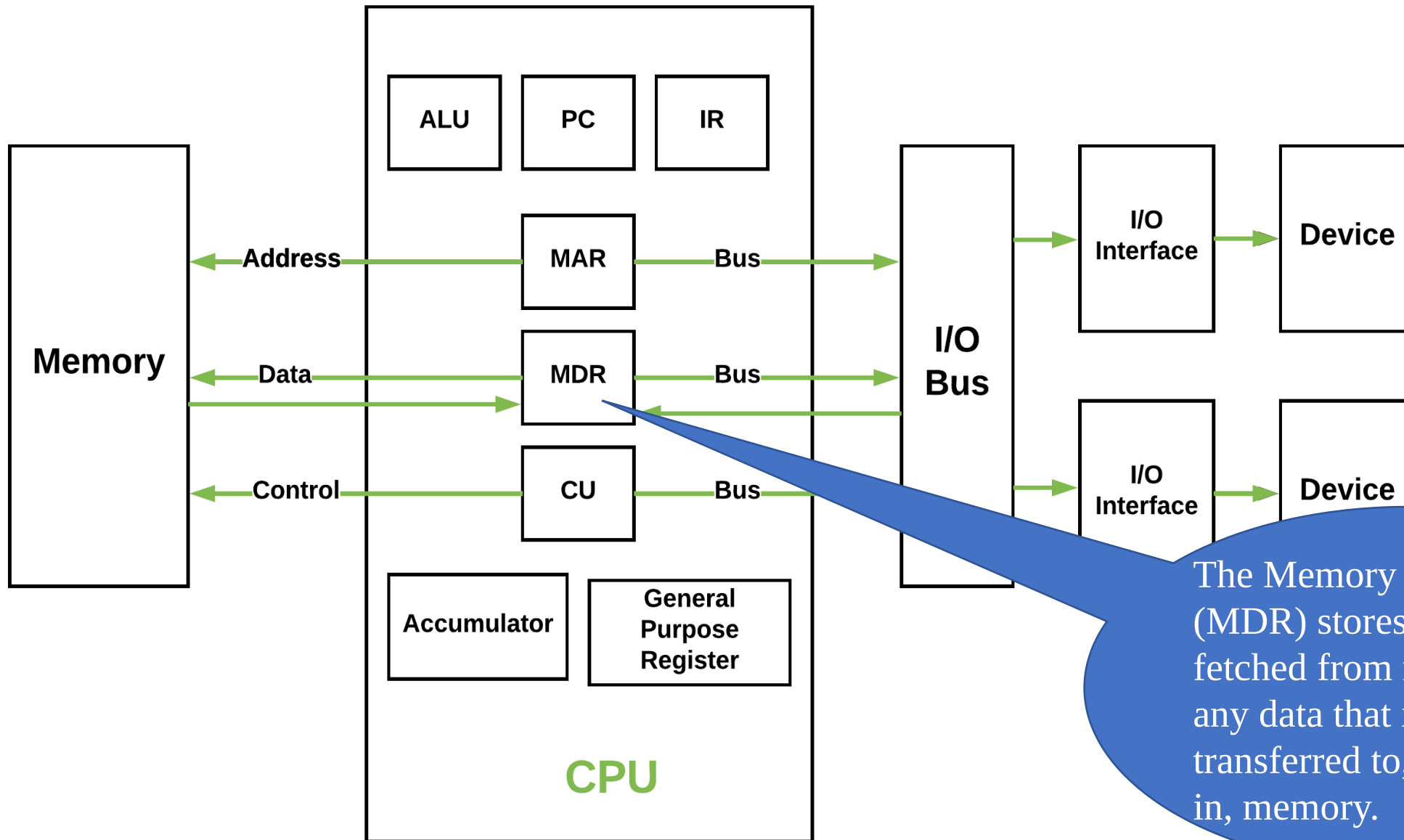
The Program Counter (PC) keeps track of the memory location of the next instructions to be dealt with.



The instruction register (IR) is a temporary storage unit in a CPU's control unit that holds the instruction being executed or decoded

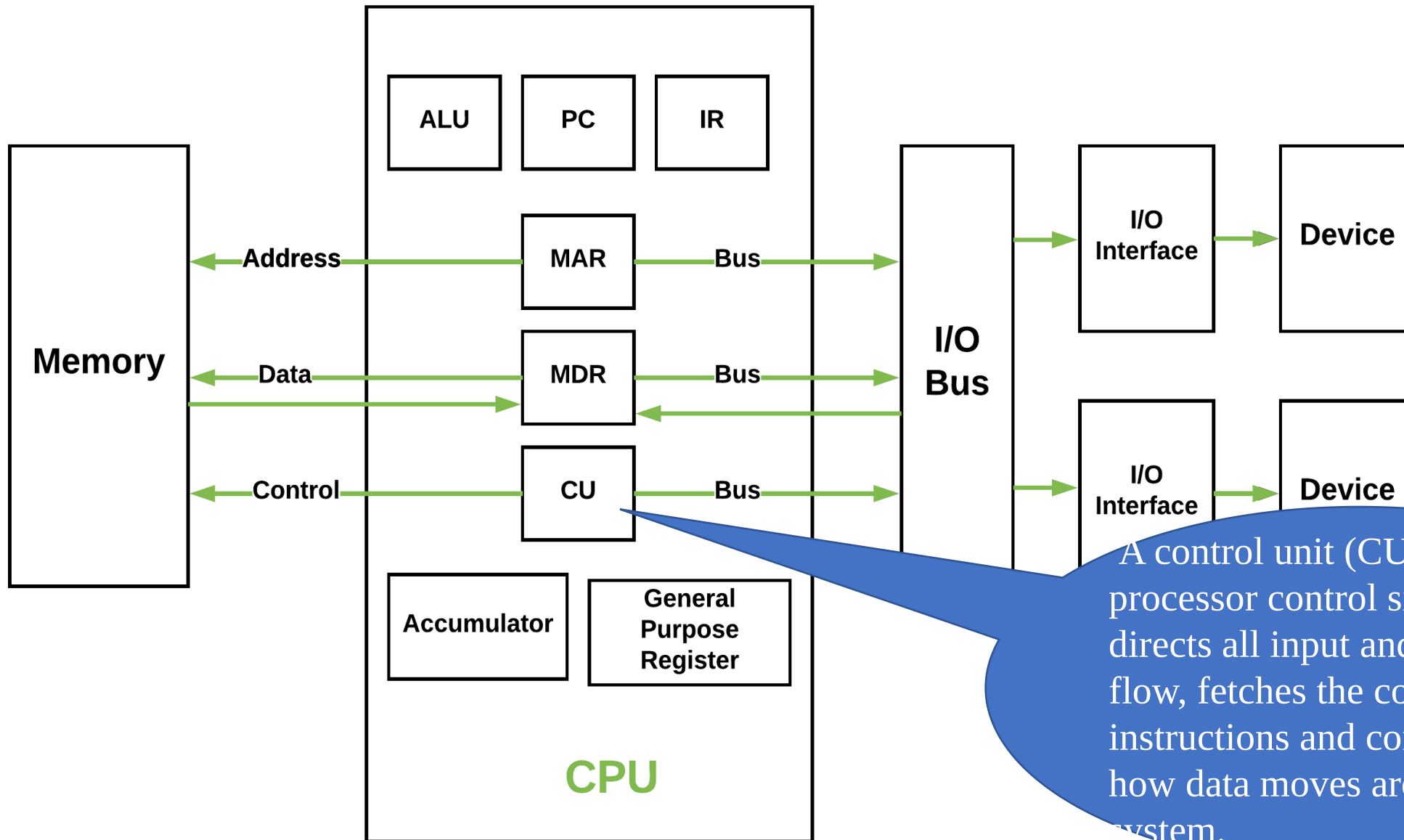


The Memory Address Register (MAR) stores the memory locations of instructions that need to be fetched from memory or stored into memory.

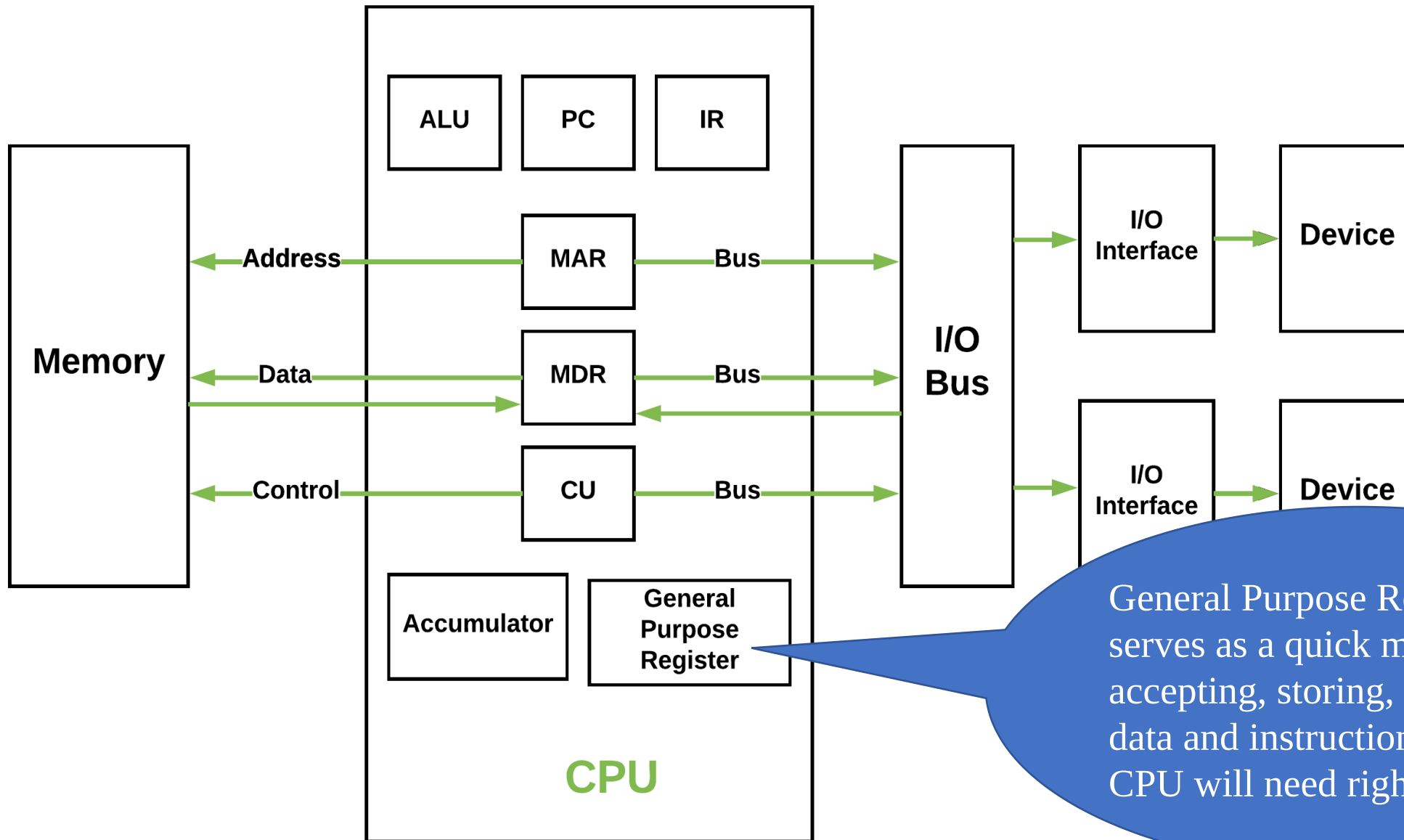


The Memory Data Register (MDR) stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.





A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches the code for instructions and controlling how data moves around the system.



# What is the difference between Microcontroller ( $\mu$ C) and Microprocessor ( $\mu$ P)?

- A programmable device that takes in input performs some arithmetic and logical operations over it and produces the desired output.
- A Microprocessor is a digital device on a chip that can fetch instructions from memory, decode and execute them, and give results.
- A microcontroller (MCU) is a small computer on a single integrated circuit that is designed to control specific tasks within electronic systems.
- It combines the functions of a central processing unit (CPU), memory, and input/output interfaces, all on a single chip.

Microcontroller	Microprocessor
A microcontroller is a specialized form of a microprocessor	The microprocessor is designed to be general-purpose.
It is cost-effective.	It is a silicon chip
It is self-sufficient.	It is a dependent unit
The microcontroller is used to perform a particular tasks.	The Microprocessor is used to perform a certain task.
Its power consumption is low.	Its power consumption is high.
It contains <a href="#">CPU</a> , RAM, ROM, Registers, Timer and input/output ports.	It requires a combination of timers, controllers memory chips.
Its size is smaller.	Its size is larger.
It is a chip which is called single chip computer.	It is a general purpose device which is called a CPU.

Microcontroller	Microprocessor
Microcontroller have no advantage of designing RAM, ROM, I/O port.	It have advantages of versatility such that designer can decide the amount of RAM, ROM, I/O port as needed.
Its microprocessors processing power is lower than microprocessor.	Its processing power is higher.
It uses Harvard Architecture.	It uses Von Neumann Architecture.
It's system cost is low.	It's system cost is high.
Each instruction needs an internal operation.	Each instruction needs an external operation.
For Example- Television.	For Example- Personal Computers.

Feature	Microcontroller (μC)	Microprocessor (μP)
Purpose	Designed for specific <a href="#">embedded system</a> applications	Designed for general-purpose computing applications
Architecture	Single-chip computer system with on-board memory, peripherals, and I/O interfaces	CPU with minimal on-board memory, peripherals, and I/O interfaces
Integration level	Highly integrated	Less integrated
System architecture	Single-chip system	CPU + support chips
Processing power	Lower power	Higher power
Instruction set	Fixed instruction set	More flexible

Feature	Microcontroller (μC)	Microprocessor (μP)
On-board memory	On-chip memory	No on-board memory
Input/Output (I/O)	More I/O ports	Fewer I/O ports
Peripheral devices	On-board peripherals	External peripherals
Cost	Lower cost	Higher cost
Power consumption	Lower power	Higher power
Applications	Embedded systems	General-purpose
Development	Integrated development environment (IDE) provided by manufacturers, with specialized programming languages and tools	Standard development tools and languages such as C, C++, and assembly
Clock speed	Lower clock speed, typically less than 100 MHz	Higher clock speed, typically greater than 1 GHz

# Basic Operations of a Computer

Computers perform four fundamental operations known as IPOS:

- Input: Receiving data from input devices.
- Processing: Performing calculations or logical operations on the data.
- Output: Sending processed data to output devices.
- Storage: Saving data in memory or secondary storage for future use



# CPU Operation Cycle

The CPU operates in a cycle consisting of four main steps:

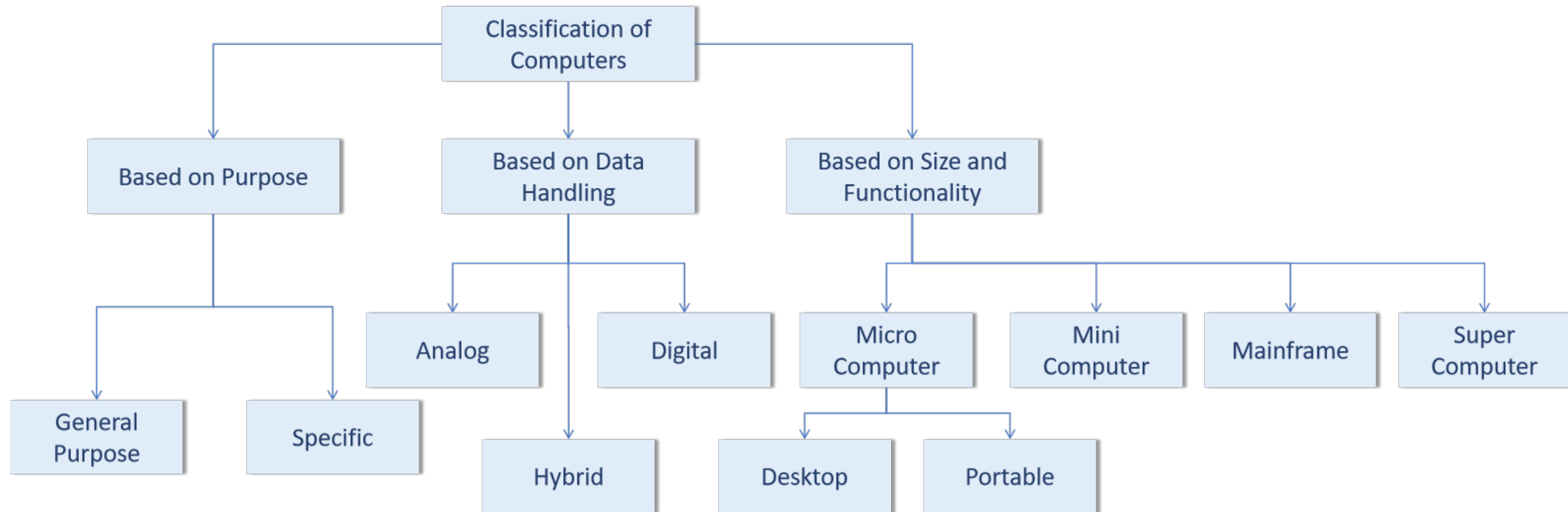
- Fetch: Retrieve an instruction from memory using the Program Counter (PC), which keeps track of the next instruction's address.
- Decode: Interpret the fetched instruction to determine what action is required.
- Execute: Carry out the instruction using the ALU or other components.
- Write Back: Store the result back in memory or registers

# Memory Organization

Memory in computers is structured into various types:

- Registers: Small, fast storage locations within the CPU used to hold temporary data and instructions.
- Cache Memory: A smaller, faster type of volatile memory that provides high-speed access to frequently used data and instructions.
- Main Memory: As mentioned earlier, this is where active programs and data reside during execution.

# Classification Micro, Mini, Mainframe and Super Computer



<b>Classification Criteria</b>	<b>Microcomputers</b>	<b>Minicomputers</b>	<b>Mainframe Computers</b>	<b>Supercomputers</b>
<b>Based on Purpose</b>	General-purpose; personal use	Specific-purpose; small to medium businesses	Specific-purpose; Critical applications for large organizations	Specific-purpose; High-performance tasks in scientific research
<b>Based on Data Handling</b>	Digital data processing	Digital data processing	Digital data processing	Digital data processing
<b>Based on Size</b>	Small	Medium	Large	Very Large
<b>Based on Functionality</b>	Single-user systems	Multi-user systems	Multi-user systems	Extremely high-performance systems
<b>Examples</b>	Desktops, laptops, tablets	DEC PDP-11, VAX	IBM zSeries, Unisys ClearPath	IBM Summit, Fujitsu Fugaku
<b>Typical Applications</b>	Office work, gaming, web browsing	Billing, inventory management	Banking, telecom sectors	Scientific research, simulations

Feature	Microcomputers	Minicomputers	Mainframe Computers	Supercomputers
Size	Small (e.g., desktops, laptops)	Medium (fits in a single rack/box)	Large (occupies entire rooms)	Very Large (complex systems)
Processing Power	Low to Moderate	Medium	High	Extremely High
Storage Capacity	Low to Moderate	Medium	High	Very High
Cost	Low (affordable for personal use)	Moderate	High (1 to 5 million dollars)	Very High (10 to 30 million dollars)
Usage	Personal and small-scale applications	Small to medium-sized organizations	Large organizations and government agencies	High-performance scientific and research applications
User Capacity	Single-user	Multi-user (supports several users)	Hundreds to thousands of users	Specialized teams for research
Examples	PCs, laptops, smartphones	DEC PDP-11, AS/400, VAX	IBM zSeries, Unisys ClearPath	IBM Summit, Fujitsu Fugaku
Typical Applications	Office work, personal computing	Business applications (financial, engineering)	Transaction processing, scientific simulations	Climate modeling, genomics research

# DEC PDP-11



# IBM AS/400





IBM zSeries



# El Capitan



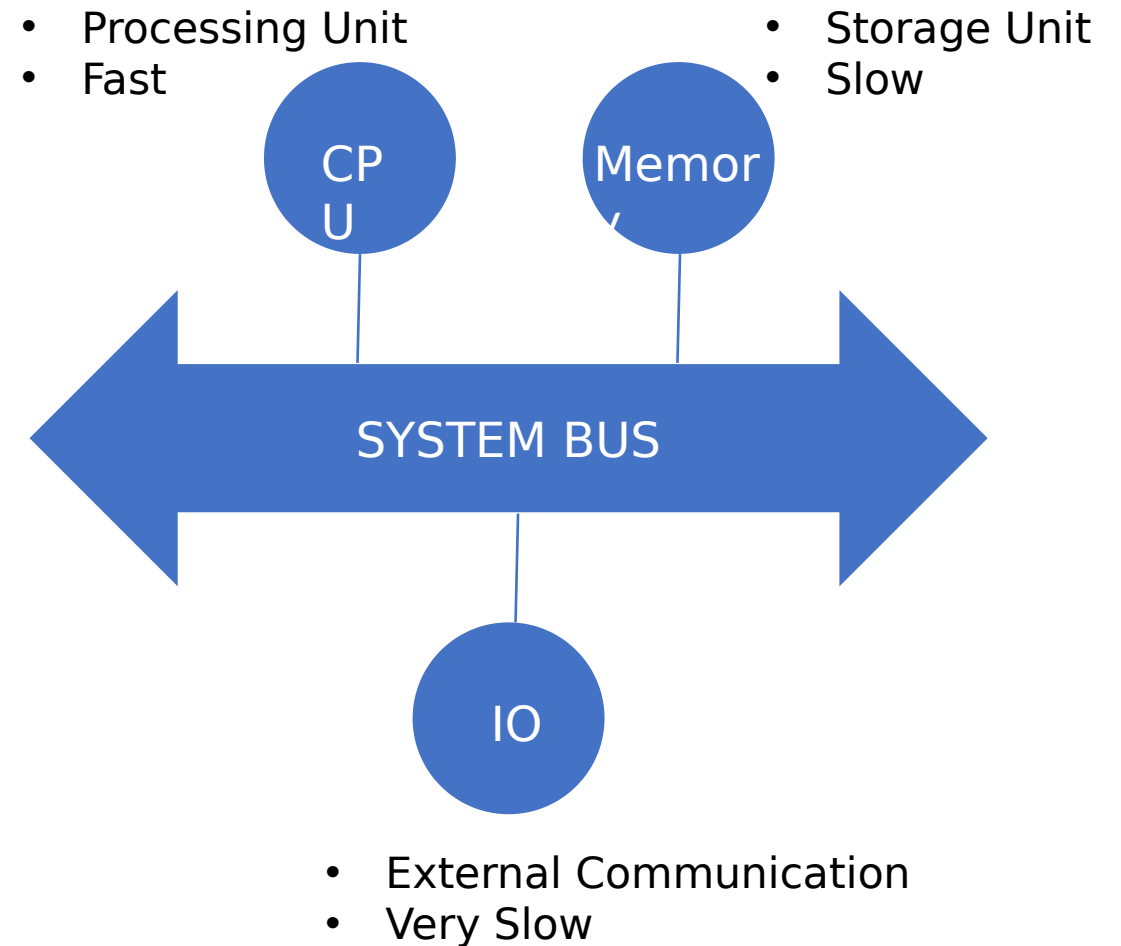
As of November 2024, the world's most powerful supercomputer is El Capitan



# System Bus and Interconnection

A system bus is a crucial component in computer architecture, serving as a communication pathway that connects the major components of a computer system, including the CPU, memory, and input/output (I/O) devices.

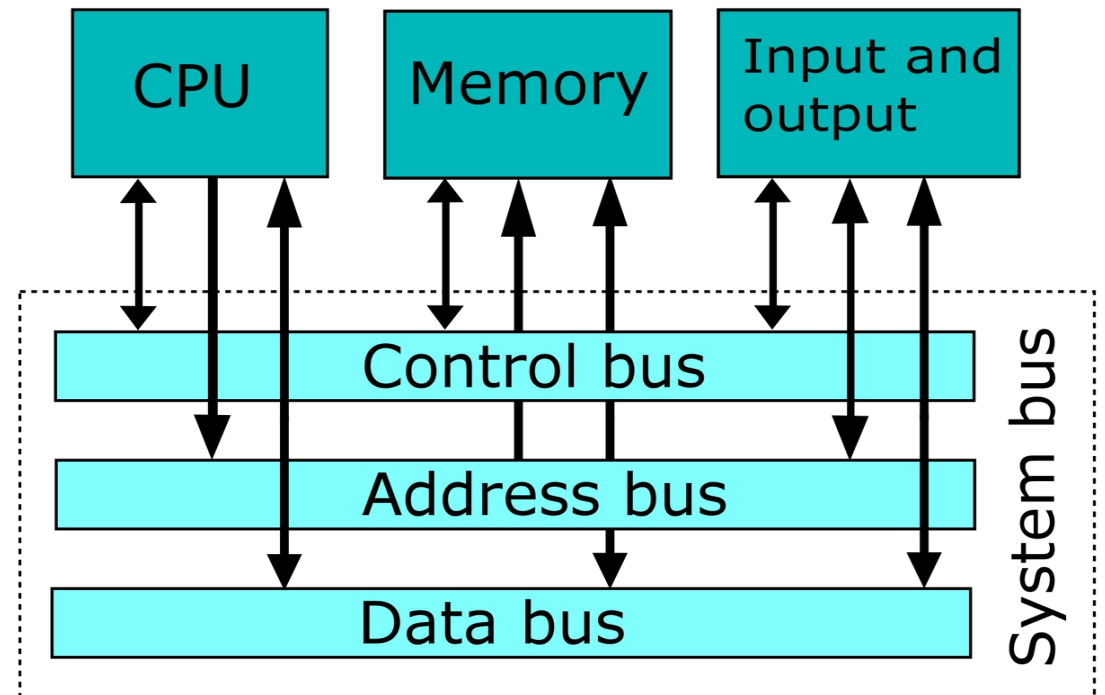
It facilitates the transfer of data, addresses, and control signals between these components.



# Components of a System Bus

The system bus typically consists of three main components:

1. Data Line/Bus
2. Address Line/Bus
3. Control Line/Bus

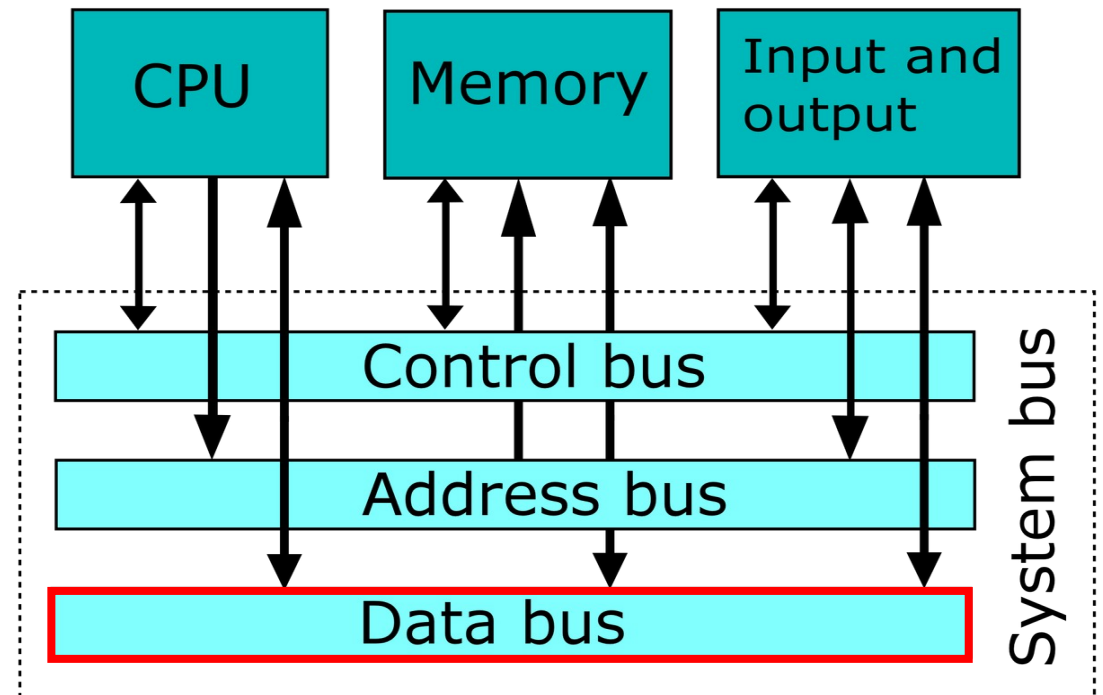


# Components of a System Bus - Data Line/Bus

Function: Carries actual data between the CPU, memory, and I/O devices.

Direction: **Bi-directional**, allowing data to flow both to and from the CPU.

Characteristics: The width of the data bus (number of wires) determines how much data can be transferred simultaneously (e.g., 8-bit, 16-bit, 32-bit).

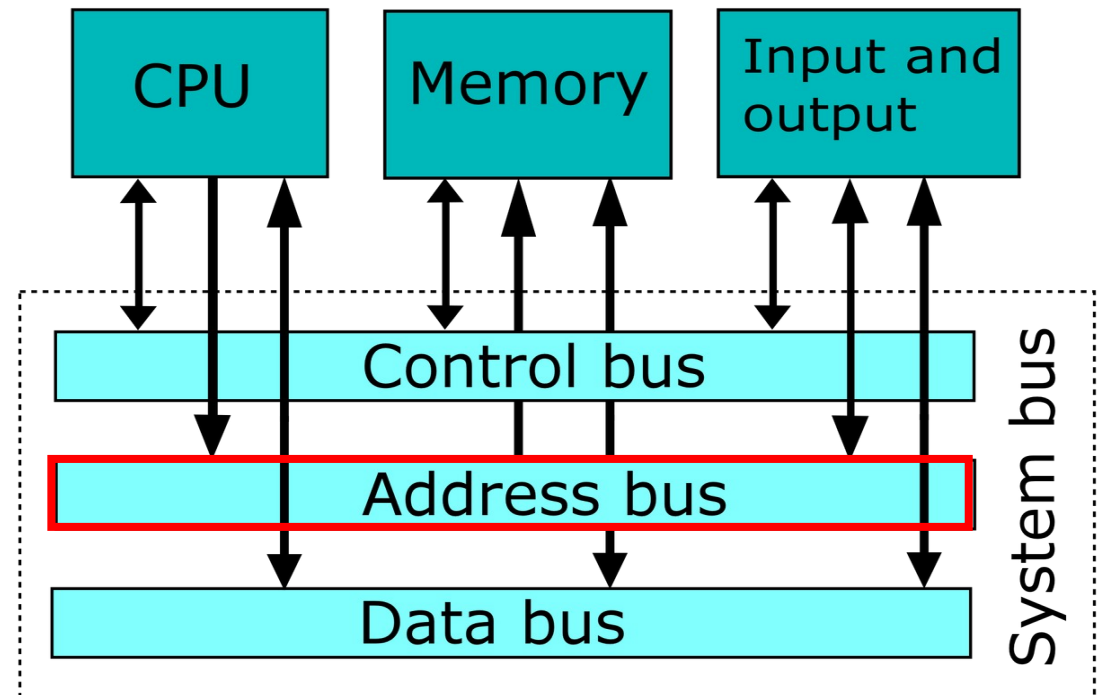


# Components of a System Bus - Address Line/Bus

Function: Carries the addresses of data (not the data itself) to specify where data should be sent or read from in memory.

Direction: **Unidirectional**; it only sends addresses from the CPU to other components.

Characteristics: The width of the address bus determines how much memory can be addressed (e.g., a 20-bit address bus can address  $2^{20}$  locations).

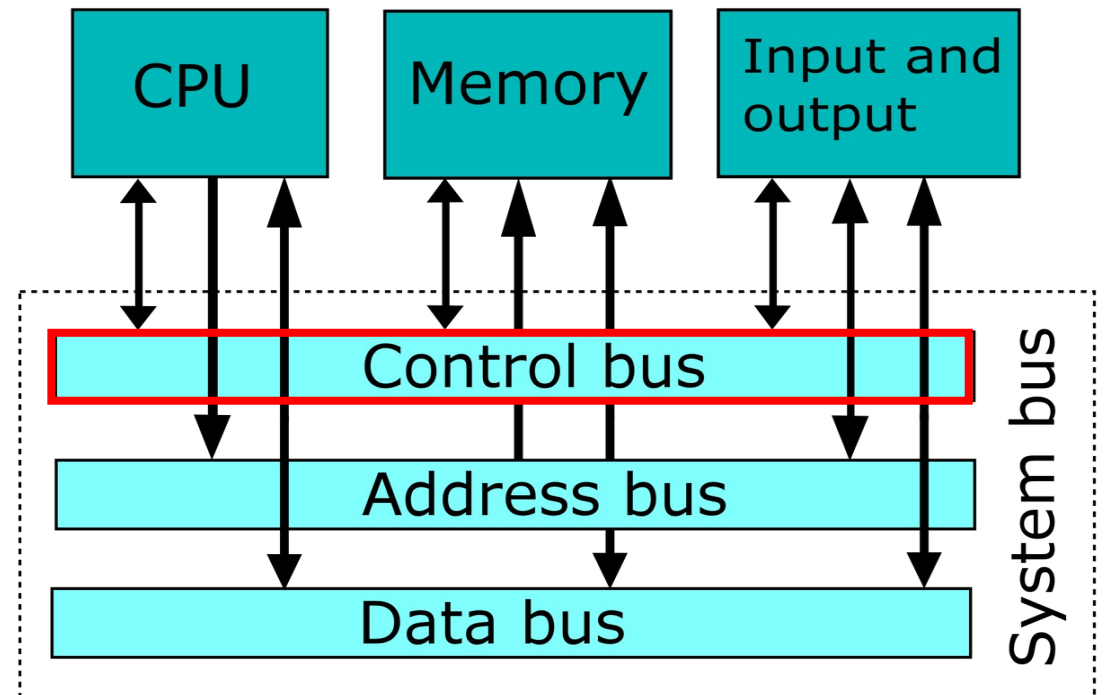


# Components of a System Bus - Control Line/Bus

Function: Transmits control signals that manage the operations of the CPU and other components.

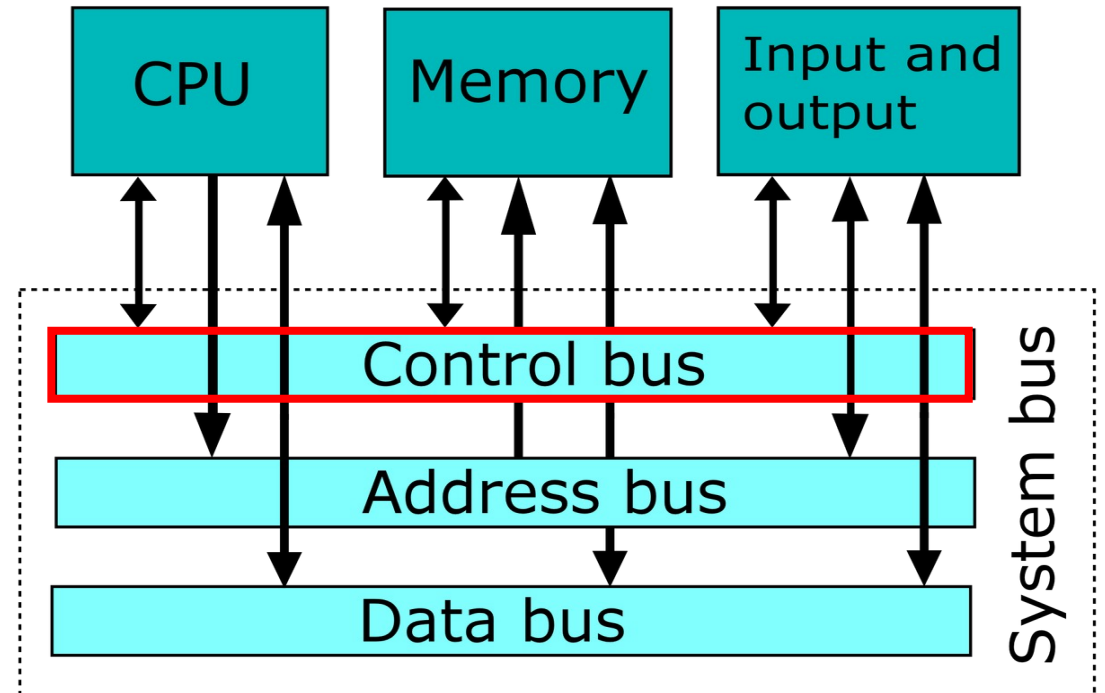
Direction: **Bi-directional**; it carries signals that coordinate activities among various devices.

Characteristics: Ensures that data transfers occur without collisions by managing timing and control signals.



# Components of a System Bus - Control Line/Bus

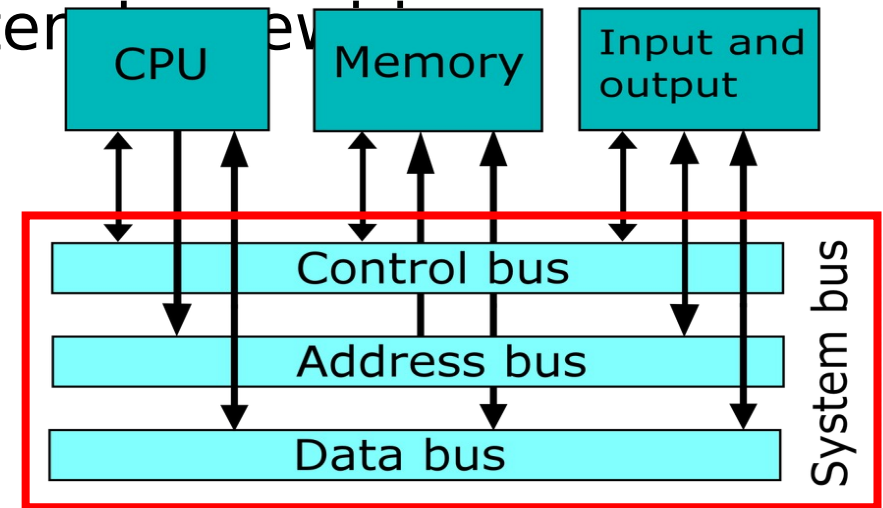
- Used to control the access to and the use of the data and address lines
- Since the data and the address line shared by all the components, there must be a means of controlling their use
- Control signal transmit both commands and timing information between the modules
- Typical control lines includes; Memory write , Memory read, I/O write, I/O read, Clock, Reset, Bus request, Bus grant, Interrupt request



Bus Type	Direction	Function	Characteristics	Examples
Address Bus	Unidirectional	Carries address information for data locations	<ul style="list-style-type: none"><li>- Only sends addresses from CPU to memory/I/O devices</li><li>- Determines how much memory can be accessed based on width (n lines = <math>2^n</math> addresses)</li></ul>	Used in all computer systems
Data Bus	Bidirectional	Transfers actual data between components	<ul style="list-style-type: none"><li>- Allows data to flow both ways</li><li>- Width determines how much data can be transferred simultaneously (e.g., 8-bit, 32-bit)</li></ul>	System bus in PCs, PCI bus
Control Bus	Typically Unidirectional	Carries control signals to manage operations	<ul style="list-style-type: none"><li>- Sends control signals from CPU to other components</li><li>- Coordinates activities and prevents data</li></ul>	Used in all computer systems

# Importance of the System Bus

- The system bus is essential for enabling communication between independent components within a computer system. It simplifies interconnections by providing a common pathway rather than requiring direct connections between all components, which would be complex and costly.
- It allows for modular design in computer systems, facilitating upgrades and modifications without external reconfiguration.





# How the System Bus Works

When the CPU needs to access data from memory:

- It places the address of the desired data on the address bus.
- The control bus sends signals indicating whether to read or write data.
- The data bus carries the actual data to or from memory based on these signals.

# Peripheral Component Interconnect (PCI)

Peripheral Component Interconnect (PCI) is a computer bus **standard** developed in the early 1990s for attaching peripheral devices to a computer's motherboard.

It facilitates data transfer between the central processing unit (CPU) and connected devices, enhancing the system's capabilities by allowing the addition of various hardware components such as graphics cards, network cards, and sound cards.

# Peripheral Component Interconnect (PCI)

Peripheral Component Interconnect (PCI) is a computer bus **standard** developed in the early 1990s for attaching **peripheral devices** to a computer's motherboard.

It facilitates data transfer between the central processing unit (CPU) and connected devices, enhancing the system's capabilities by allowing the addition of various hardware components such as graphics cards, network cards, and sound cards.

**A peripheral device is** a device that connects to a computer and works with it in some way.

# Key Features of PCI

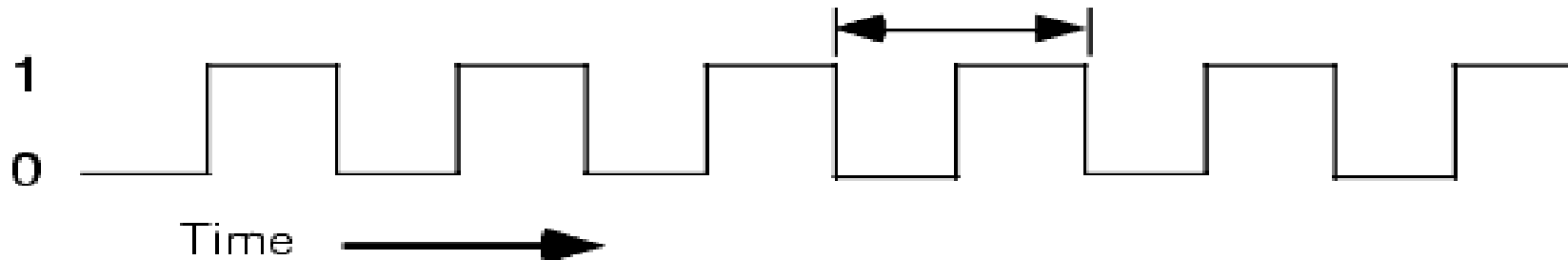
- Data Transfer Method: PCI is a parallel bus that operates synchronously with a **single bus clock**. It supports both 32-bit and 64-bit data paths, enabling data transfer rates of up to 133 MBps for 32-bit configurations and 266 MBps for 64-bit configurations.

What is a clock signal?

# What is a clock signal?

a steady  
pulse

- A clock signal is a particular type of signal that oscillates between a high and low state. With the signal acting as a metronome, the digital circuit follows in time to coordinate its sequence of actions.
- Digital circuits rely on clock signals to know when and how to execute the functions that are programmed.
- If the clock in a design is like the heart of an animal, then clock signals are the heartbeats that keep the system in motion.



# Key Features of PCI

- **Data Transfer Method:** PCI is a parallel bus that operates synchronously with a single bus clock. It supports both 32-bit and 64-bit data paths, enabling data transfer rates of up to 133 MBps for 32-bit configurations and 266 MBps for 64-bit configurations.
- **Bus Mastering:** PCI supports bus mastering, allowing devices to initiate data transfers without involving the CPU, which improves efficiency.
- **Automatic Configuration:** PCI automatically configures connected devices to work with other PCI cards in the system, reducing conflicts that were common with older standards like ISA (Industry Standard Architecture).
- **Compatibility:** PCI has been largely superseded by PCI Express (PCIe), which offers higher speeds and improved performance but maintains backward compatibility with some PCI devices.

# Evolution of PCI

- Versions: Over time, various versions of PCI have emerged, including PCI-X (an extended version) and PCI Express (PCIe), which uses a serial communication method rather than parallel. PCIe provides lower latency and higher data transfer rates compared to traditional PCI.
- Transition: While PCI was prevalent from the early 1990s until about 2007, it has gradually been replaced by PCIe in modern computer architectures due to its superior performance characteristics.

# Evolution of PCI

- Versions: Over time, various versions of PCI have emerged, including PCI-X (an extended version) and PCI Express (PCIe), which uses a serial communication method rather than parallel. PCIe provides lower latency and higher data transfer rates compared to traditional PCI.
- Transition: While PCI was prevalent from the early 1990s until about 2007, it has gradually been replaced by

**PCIe** in modern computer architectures due to its superior performance characteristics. PCIe stands for Peripheral Component Interconnect Express, a high-speed serial interface that connects a computer's motherboard to expansion cards.



# Applications

PCI is primarily used in desktop computers and servers to connect various expansion cards. Common applications include:

- Graphics Cards: Enhancing video output capabilities.
- Network Interface Cards (NICs): Providing network connectivity.
- Sound Cards: Improving audio output quality.
- Storage Controllers: Managing connections to hard drives or SSDs.

<b>Feature</b>	<b>Description</b>
<b>Standard Name</b>	Peripheral Component Interconnect (PCI)
<b>Introduction Year</b>	Early 1990s
<b>Data Transfer Method</b>	Parallel bus
<b>Data Path Widths</b>	32-bit and 64-bit
<b>Transfer Speeds</b>	Up to 133 MBps (32-bit), 266 MBps (64-bit)
<b>Bus Mastering Support</b>	Yes
<b>Automatic Configuration</b>	Yes
<b>Successors</b>	PCI-X, PCI Express (PCIe)
<b>Common Uses</b>	Graphics cards, network cards, sound cards, etc.

# How PCI Works

Peripheral Component Interconnect (PCI) is a local computer bus standard that allows various peripheral devices to connect to a computer's motherboard, facilitating communication between the central processing unit (CPU) and these devices. Here's a detailed overview of how PCI operates:

## 1. Architecture

- **Bus Structure:** PCI uses a parallel bus architecture, meaning multiple bits of data can be transmitted simultaneously across multiple wires. It typically supports 32-bit and 64-bit data paths.
- **Slots and Cards:** The motherboard has PCI slots where expansion cards (like graphics cards, network cards, etc.) can be inserted. Each card connects to the PCI bus, allowing it to communicate with the CPU and memory.

# How PCI Works Cont...

## 2. Data Transfer

- Bus Mastering: PCI supports bus mastering, which allows devices to initiate data transfers without CPU intervention. This improves efficiency by reducing CPU workload.
- Data Flow: When a device needs to send or receive data, it sends a request over the PCI bus. The CPU or another device acting as the bus master can then control the data transfer process.

## 3. Addressing

- Address Space: Each device connected to the PCI bus is assigned an address within the processor's address space. This addressing allows the CPU to identify and communicate with specific devices.
- Memory Mapping: Devices can be memory-mapped, meaning they can be accessed as if they were part of the system's memory, simplifying programming and data handling.

# How PCI Works Cont...

## 4. Interrupt Handling

- Interrupt Requests (IRQs): PCI includes multiple interrupt lines (INTA# through INTD#), allowing devices to signal the CPU when they need attention. Devices share IRQs, which helps manage multiple devices without conflicts.
- Level-triggered Interrupts: The PCI standard uses level-triggered interrupts, which are more reliable for shared lines compared to edge-triggered interrupts.

## 5. Configuration

- Plug-and-Play: PCI supports plug-and-play capabilities, allowing the operating system to automatically detect and configure new devices as they are added to the system without requiring manual setup.
- Configuration Space: Each device has a configuration space that stores information about its capabilities and settings, which is used during initialization.

# PCI Bus Architecture

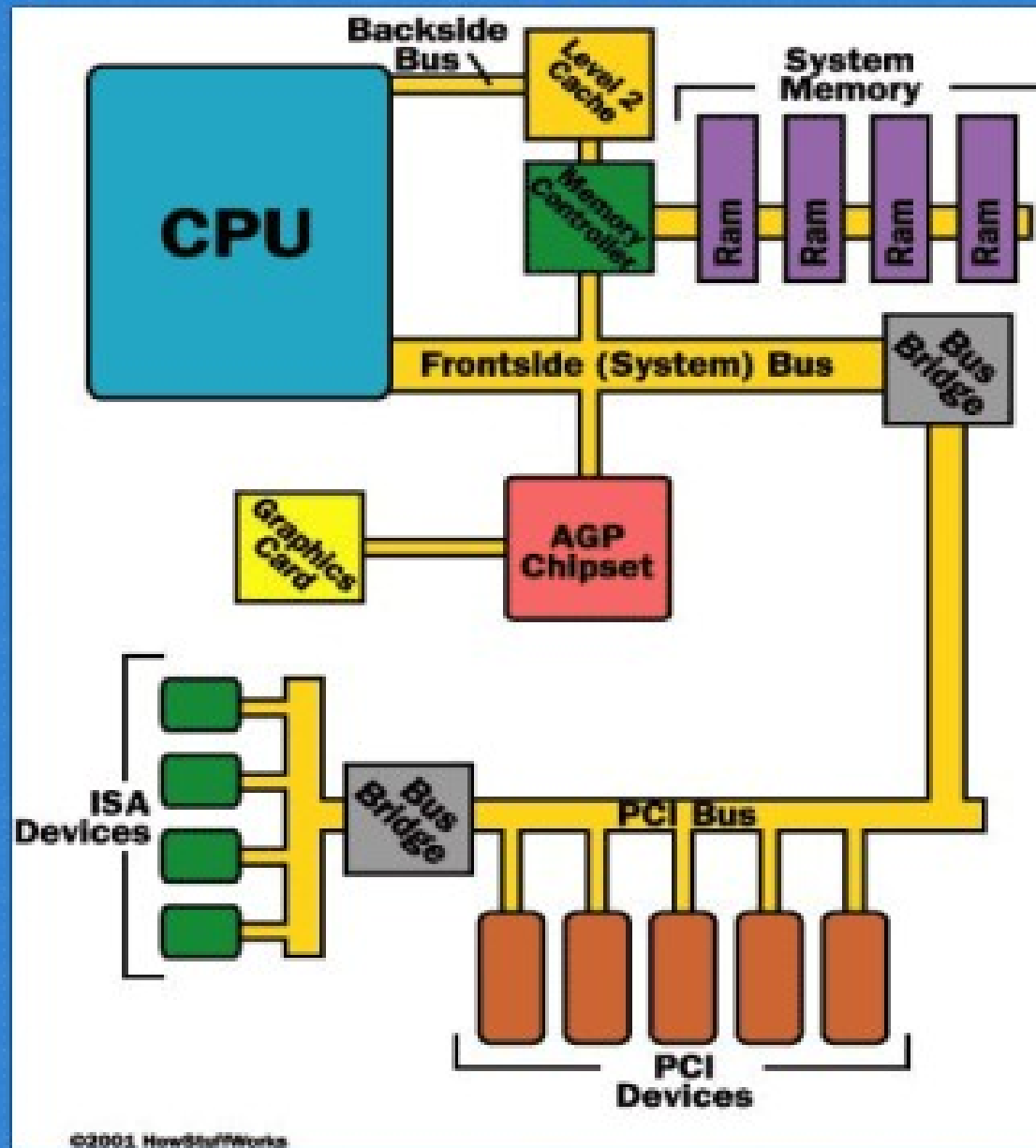
A computer bus is used to transfer data from one location or device on the motherboard to the central processing unit where all calculations take place.

Two different parts of a Bus

- Address bus-transfers information about where the data should go
- Data bus-transfers the actual data

# PCI Bus Architecture

- PCI(Peripheral Component Interconnect) bus is based on ISA (Industrial Standard Architecture) Bus and VL (VESA Local) Bus.
- Introduced by Intel in 1992
- Revised twice into version 2.1 which is the 64-bit standard that it is today.
- Great feature of PCI Bus was that it was invented as an industry standard
- PCI provides direct access to system memory for the devices that are connected to the bus which is then connected through a bridge that connects to the front side bus.
- This configuration allowed for higher performance without slowing down the processor



- The PCI Bus was originally 33Mhz and then changed to 66Mhz.
- PCI Bus became big with the release of Windows 95 with “Plug and Play” technology
- “Plug and Play” utilized the PCI bus concept.



# PCI System Bus Performance

- What makes the PCI bus one of the fastest I/O bus used today?
- Three features make this possible:
  - Burst Mode: allows multiple sets of data to be sent (Kozierok, 2001a)
  - Full Bus Mastering: the ability of devices on the PCI bus to perform transfers directly (Kozierok, 2001c)
  - High Bandwidth Options: allows for increased speed of the PCI (Kozierok, 2001a)

# Instruction Cycle

Registers Involved In Each Instruction Cycle:

Memory address registers(MAR) : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.

Memory Buffer Register(MBR) : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.

Program Counter(PC) : Holds the address of the next instruction to be fetched.

Instruction Register(IR) : Holds the last instruction fetched.

In computer organization, an instruction cycle, also known as a fetch-decode-execute cycle, is the basic operation performed by a central processing unit (CPU) to execute an instruction.

The instruction cycle consists of several steps, each of which performs a specific function in the execution of the instruction. The major steps in the instruction cycle are:

1. **Fetch:** In the fetch cycle, the CPU retrieves the instruction from memory. The instruction is typically stored at the address specified by the program counter (PC). The PC is then incremented to point to the next instruction in memory.
2. **Decode:** In the decode cycle, the CPU interprets the instruction and determines what operation needs to be performed. This involves identifying the opcode and any operands that are needed to execute the instruction.
3. **Execute:** In the execute cycle, the CPU performs the operation specified by the instruction. This may involve reading or writing data from or to memory, performing arithmetic or logic operations on data, or manipulating the control flow of the program.

## Additional steps

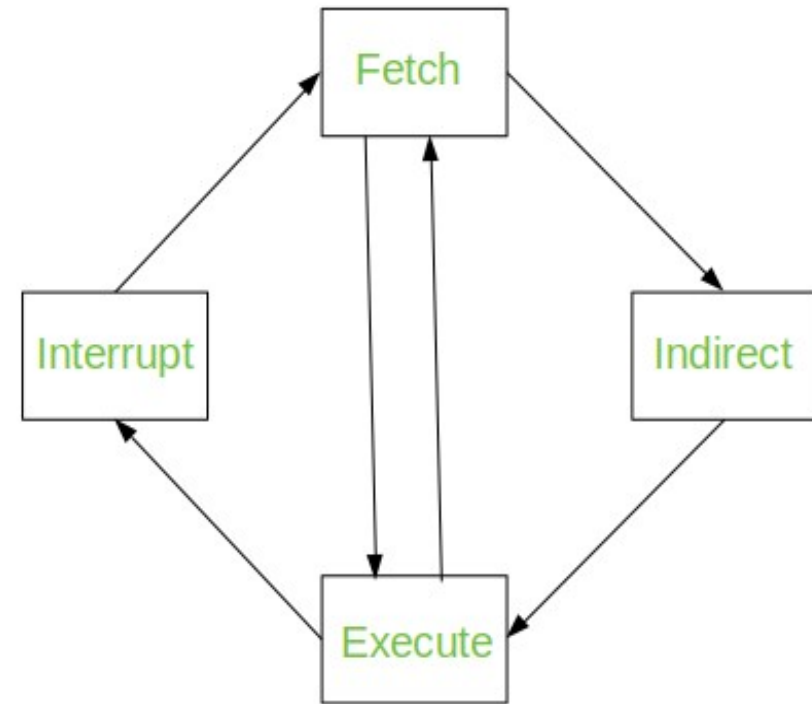
**Fetch operands:** In some CPUs, the operands needed for an instruction are fetched during a separate cycle before the execute cycle. This is called the fetch operands cycle.

**Store results:** In some CPUs, the results of an instruction are stored during a separate cycle after the execute cycle. This is called the store results cycle.

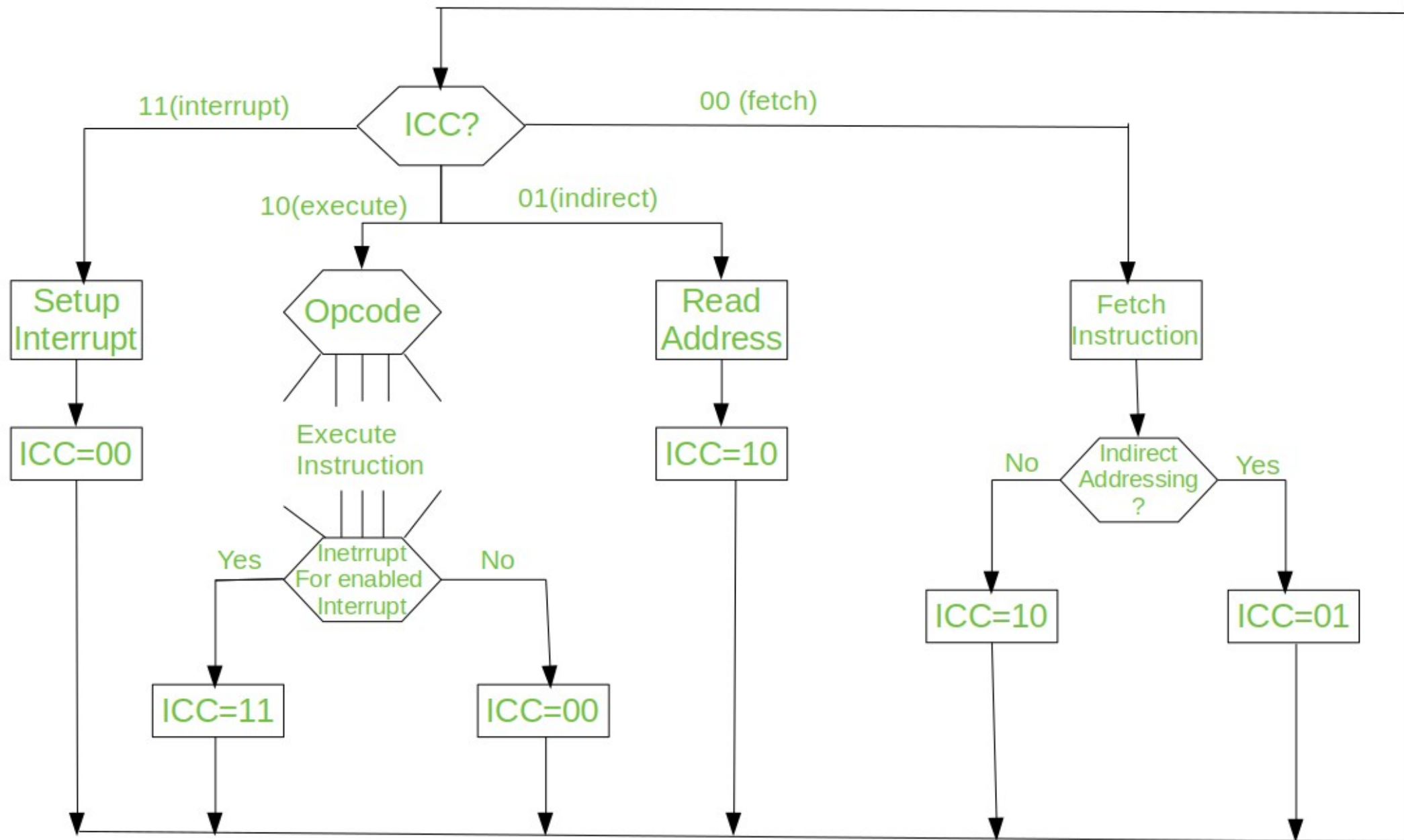
**Interrupt handling:** In some CPUs, interrupt handling may occur during any cycle of the instruction cycle. An interrupt is a signal that the CPU receives from an external device or software that requires immediate attention. When an interrupt occurs, the CPU suspends the current instruction and executes an interrupt handler to service the interrupt.

# Instruction Cycle

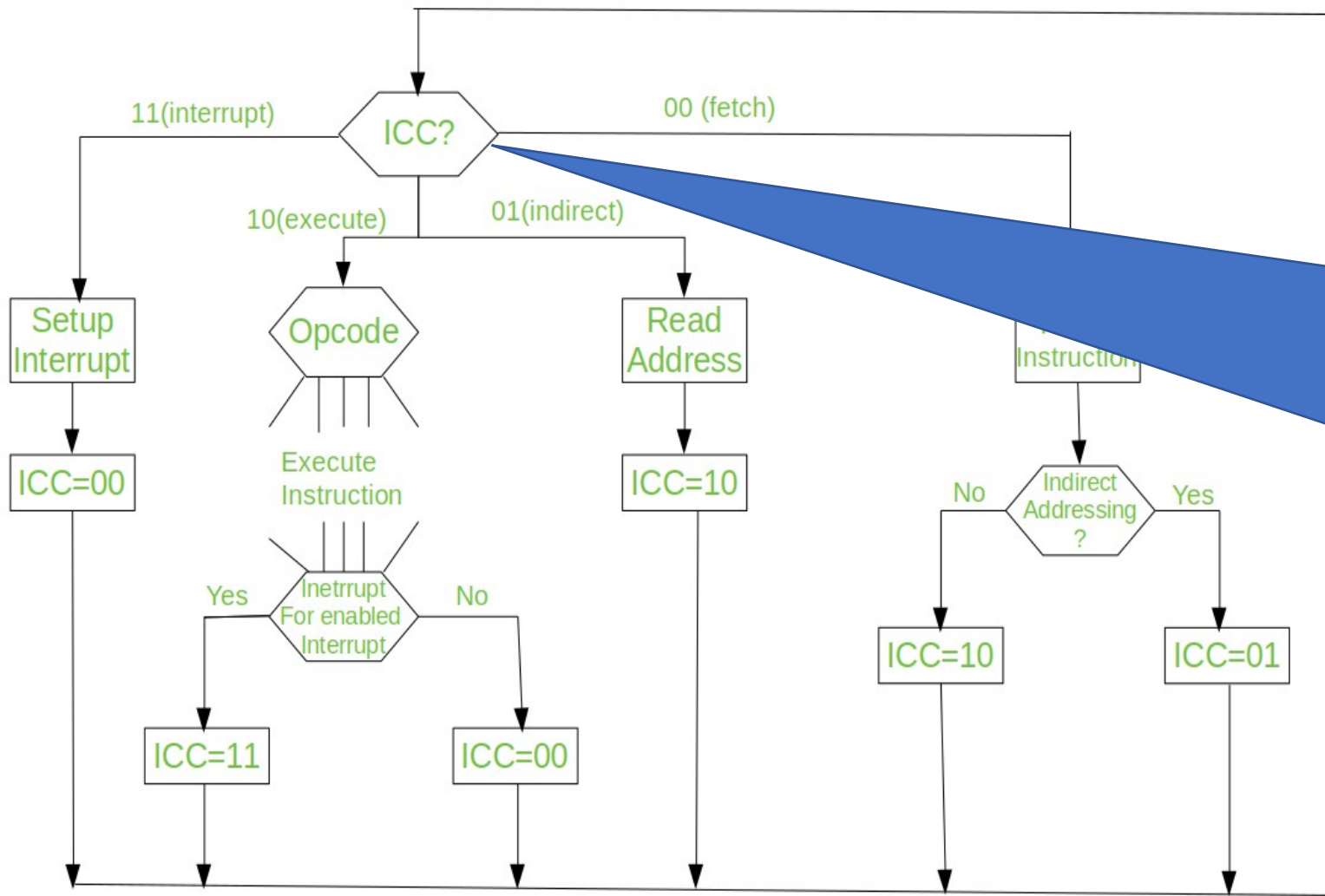
- Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations.
- The Indirect Cycle is always followed by the Execute Cycle. The Interrupt Cycle is always followed by the Fetch Cycle. For both fetch and execute cycles, the next cycle depends on the state of the system



The Instruction Cycle



Flowchart for Instruction Cycle

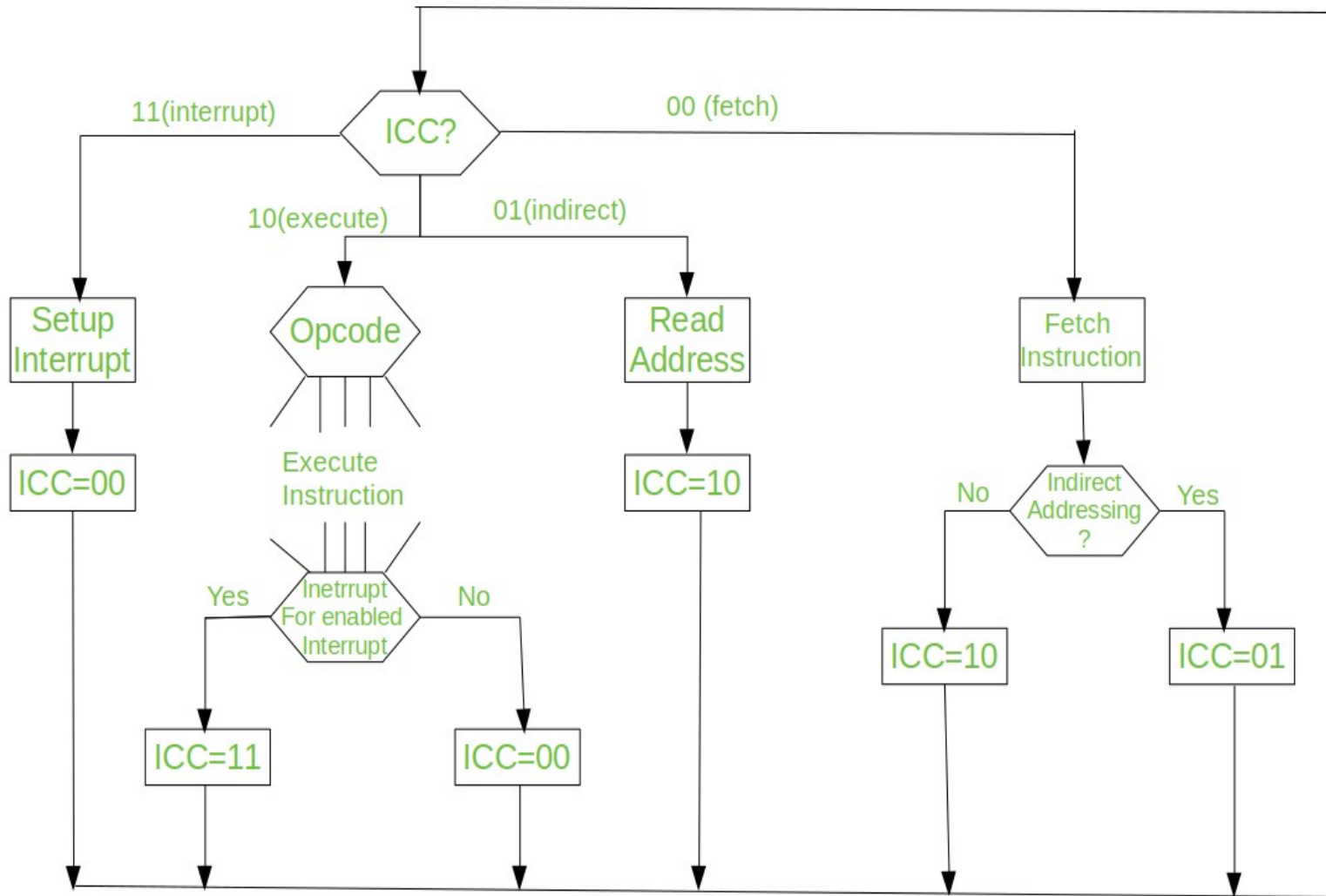


Flowchart for Instruction Cycle

We assumed a new 2-bit register called Instruction Cycle Code (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:-

- 00 : Fetch Cycle
- 01 : Indirect Cycle
- 10 : Execute Cycle
- 11 : Interrupt Cycle

At the end of the each cycles, the ICC is set appropriately.



Flowchart for Instruction Cycle

The flowchart of *Instruction Cycle* describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern (this is a simplified example). The operation of the processor is described as the performance of a sequence of micro-operation.



- The Fetch Cycle –

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).

MAR	
MBR	
PC	0000000001100100
IR	
AC	

BEGINNING

- Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.

MAR	0000000001100100
MBR	
PC	0000000001100100
IR	
AC	

FIRST STEP

- Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction. (These two action can be performed simultaneously to save time)

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	
AC	

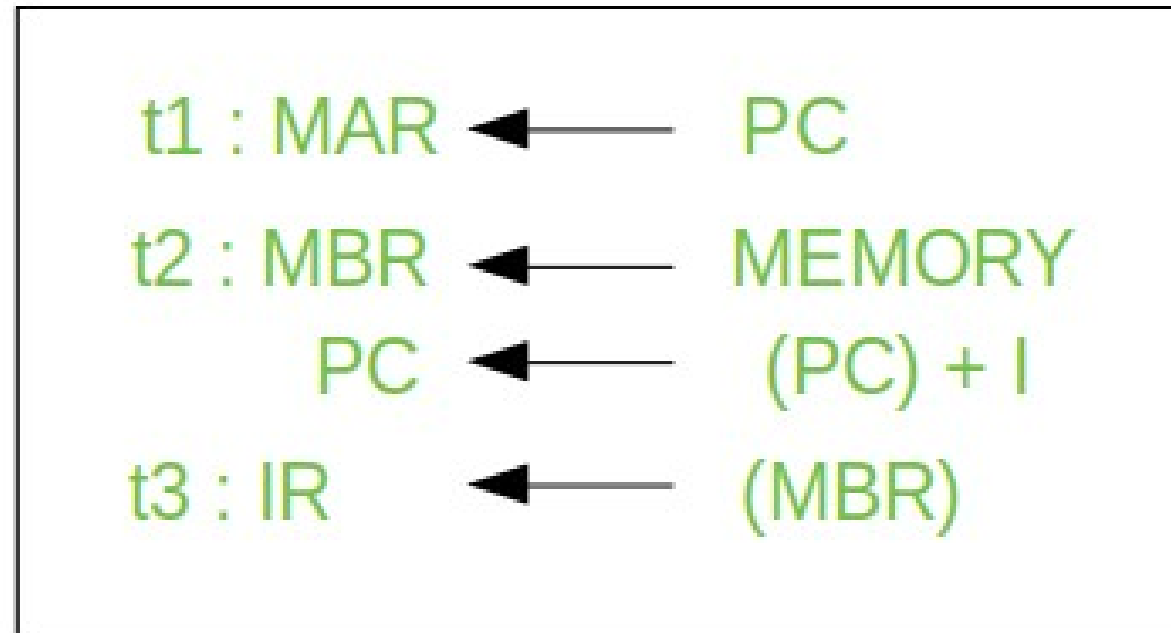
SECOND STEP

- Step 3: The content of the MBR is moved to the instruction register(IR).

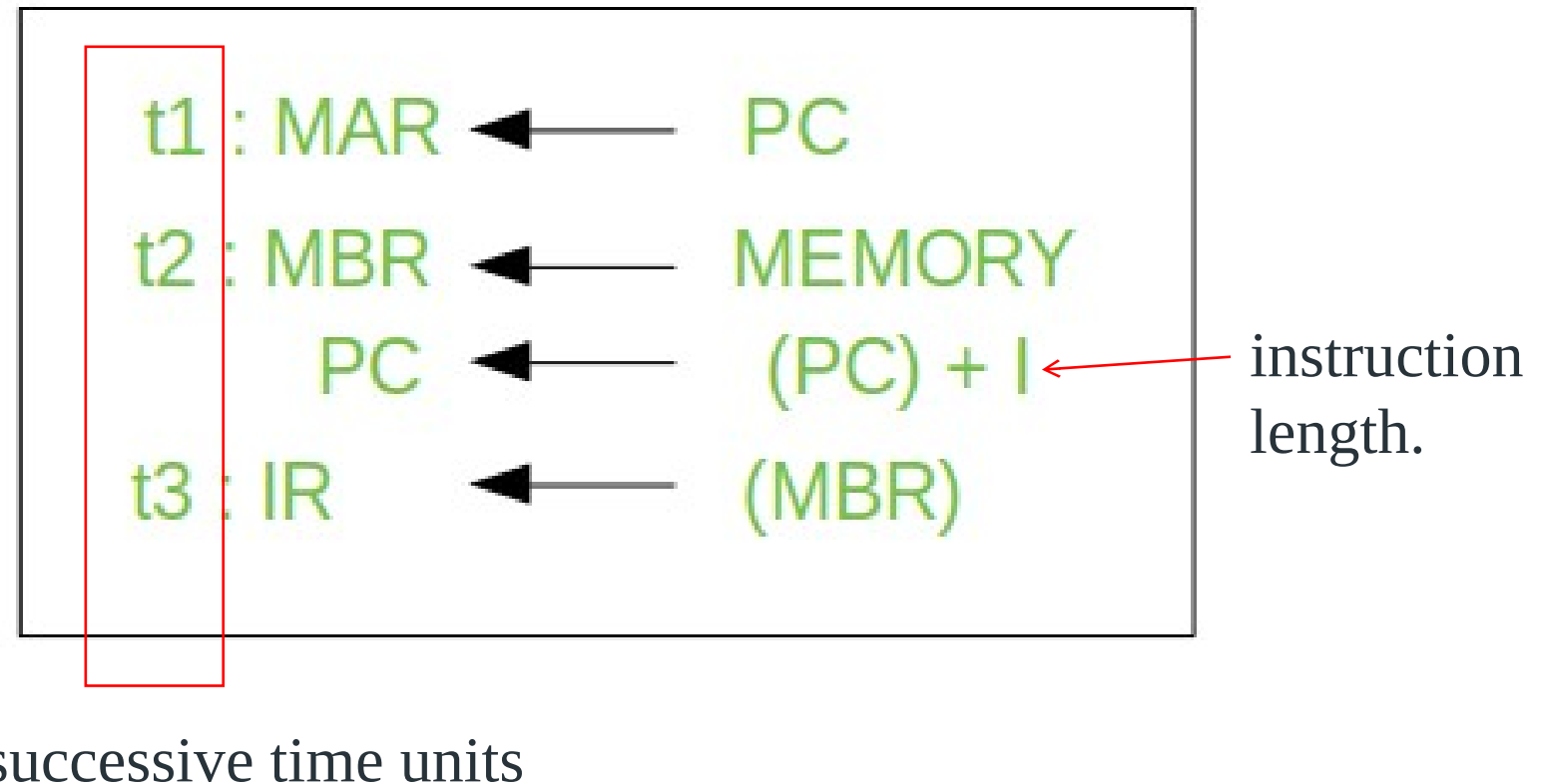
MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100100
IR	0001000000100000
AC	

FIRST STEP

- Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



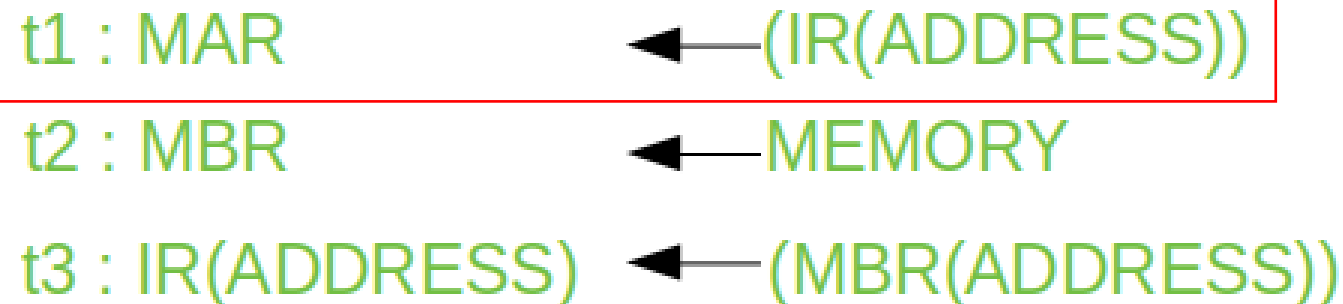
- Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



- **The Indirect Cycles –**

Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing( it can be fetched by any [addressing mode](#), here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-

Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.



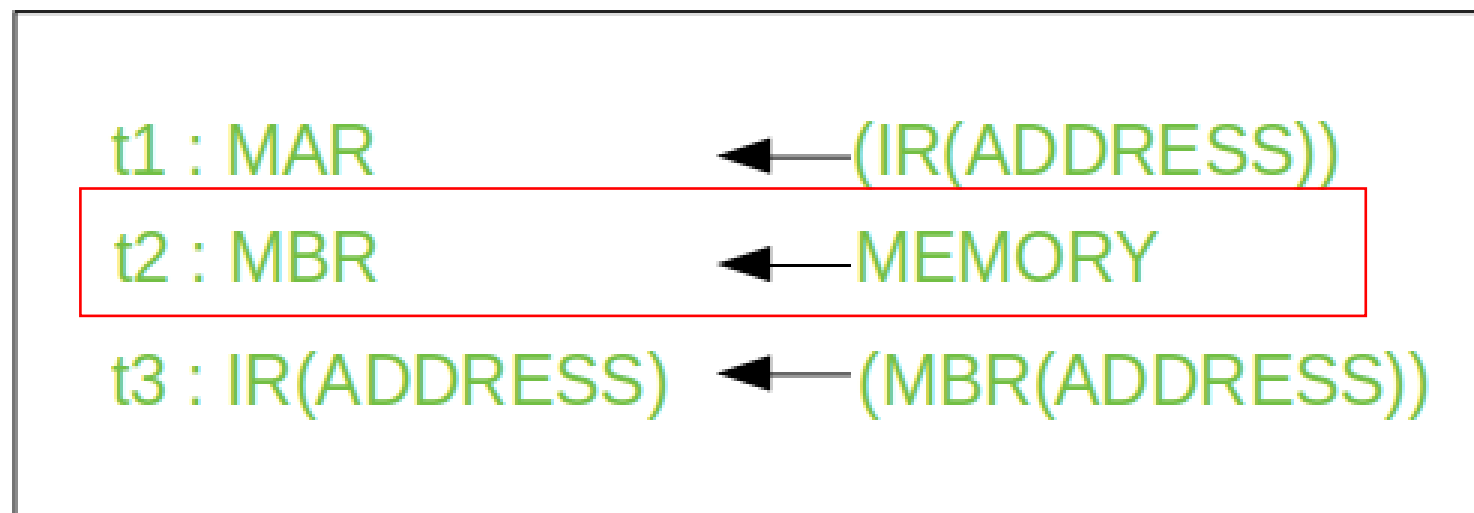
The diagram illustrates the first step of indirect addressing. It shows three micro-operations: t1, t2, and t3. t1 is highlighted with a red box. t1: MAR ← (IR(ADDRESS)) shows the address from the Instruction Register (IR) being loaded into the Memory Address Register (MAR). t2: MBR ← MEMORY shows the data from memory being loaded into the Memory Buffer Register (MBR). t3: IR(ADDRESS) ← (MBR(ADDRESS)) shows the data from the MBR being stored back into the IR at the address specified in the original instruction.

```
t1 : MAR ← (IR(ADDRESS))
t2 : MBR ← MEMORY
t3 : IR(ADDRESS) ← (MBR(ADDRESS))
```

- **The Indirect Cycles –**

Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing( it can be fetched by any [addressing mode](#), here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-

Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)





- **The Indirect Cycles –**

Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing( it can be fetched by any [addressing mode](#), here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-

Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

```
graph LR; t1["t1 : MAR"] --> IR1["(IR(ADDRESS))"]; t2["t2 : MBR"] --> MEM["MEMORY"]; t3["t3 : IR(ADDRESS)"] --> MBR1["(MBR(ADDRESS))"];
```

t1 : MAR ← (IR(ADDRESS))

t2 : MBR ← MEMORY

t3 : IR(ADDRESS) ← (MBR(ADDRESS))

- **The Execute Cycle**

The other three cycles(*Fetch, Indirect and Interrupt*) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.

Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.

Lets take an hypothetical example :-

consider an add instruction:

**ADD R , X**

- Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

t1 : MAR	← (IR(ADDRESS))
t2 : MBR	← MEMORY
t3 : R	← (R) + (MBR)

Step 1: The address portion of IR is loaded into the MAR.

- Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

t1 : MAR	← (IR(ADDRESS))
t2 : MBR	← MEMORY
t3 : R	← (R) + (MBR)

Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.

- Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

t1 : MAR	← (IR(ADDRESS))
t2 : MBR	← MEMORY
t3 : R	← (R) + (MBR)

Step 3: Now, the contents of R and MBR are added by the ALU.

- **The Interrupt Cycle:**

At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another.

Lets take a sequence of micro-operation:-

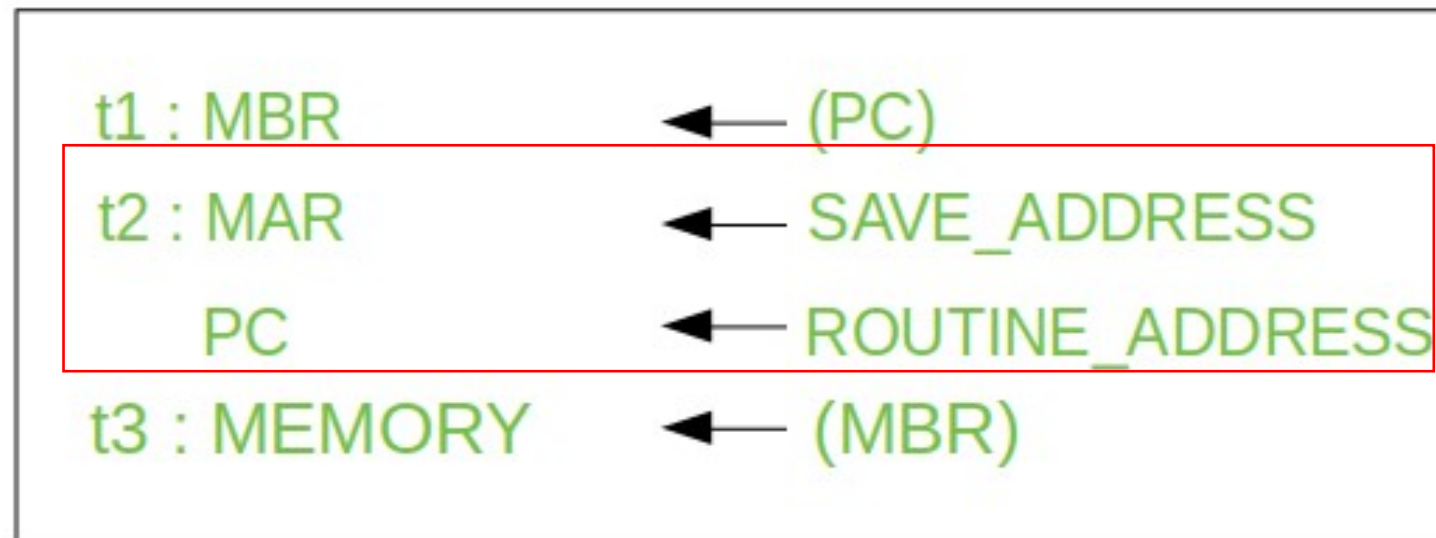
t1 : MBR	← (PC)
t2 : MAR	← SAVE_ADDRESS
PC	← ROUTINE_ADDRESS
t3 : MEMORY	← (MBR)

Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.

- **The Interrupt Cycle:**

At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another.

Lets take a sequence of micro-operation:-



Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.

PC is loaded with the address of the start of the interrupt-processing routine.

- **The Interrupt Cycle:**

At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another.

Lets take a sequence of micro-operation:-

t1 : MBR	← (PC)
t2 : MAR	← SAVE_ADDRESS
PC	← ROUTINE_ADDRESS
t3 : MEMORY	← (MBR)

Step 3: MBR, containing the old value of PC, is stored in memory.



# Uses of Different Instruction Cycles

- Fetch cycle: This cycle retrieves the instruction from memory and loads it into the processor's instruction register. The fetch cycle is essential for the processor to know what instruction it needs to execute.
- Decode cycle: This cycle decodes the instruction to determine what operation it represents and what operands it requires. The decode cycle is important for the processor to understand what it needs to do with the instruction and what data it needs to retrieve or manipulate.
- Execute cycle: This cycle performs the actual operation specified by the instruction, using the operands specified in the instruction or in other registers. The execute cycle is where the processor performs the actual computation or manipulation of data.
- Store cycle: This cycle stores the result of the operation in memory or in a register. The store cycle is essential for the processor to save the result of the computation or manipulation for future use.

# Advantages

- **Standardization:** The instruction cycle provides a standard way for CPUs to execute instructions, which allows software developers to write programs that can run on multiple CPU architectures. This standardization also makes it easier for hardware designers to build CPUs that can execute a wide range of instructions.
- **Efficiency:** By breaking down the instruction execution into multiple steps, the CPU can execute instructions more efficiently. For example, while the CPU is performing the execute cycle for one instruction, it can simultaneously fetch the next instruction.
- **Pipelining:** The instruction cycle can be pipelined, which means that multiple instructions can be in different stages of execution at the same time. This improves the overall performance of the CPU, as it can process multiple instructions simultaneously.

# Disadvantages

- Overhead: The instruction cycle adds overhead to the execution of instructions, as each instruction must go through multiple stages before it can be executed. This overhead can reduce the overall performance of the CPU.
- Complexity: The instruction cycle can be complex to implement, especially if the CPU architecture and instruction set are complex. This complexity can make it difficult to design, implement, and debug the CPU.
- Limited parallelism: While pipelining can improve the performance of the CPU, it also has limitations. For example, some instructions may depend on the results of previous instructions, which limits the amount of parallelism that can be achieved. This can reduce the effectiveness of pipelining and limit the overall performance of the CPU.

# Issues of Different Instruction Cycles

- Pipeline hazards: Pipelining is a technique used to overlap the execution of multiple instructions by breaking them into smaller stages. However, pipeline hazards occur when one instruction depends on the completion of a previous instruction, leading to delays and reduced performance.
- Branch prediction errors: Branch prediction is a technique used to anticipate which direction a program will take when encountering a conditional branch instruction. However, if the prediction is incorrect, it can result in wasted cycles and decreased performance.
- Instruction cache misses: Instruction cache is a fast memory used to store frequently used instructions. Instruction cache misses occur when an instruction is not found in the cache and needs to be retrieved from slower memory, resulting in delays and decreased performance.

# Issues of Different Instruction Cycles

- Instruction-level parallelism limitations: Instruction-level parallelism is the ability of a processor to execute multiple instructions simultaneously. However, this technique has limitations as not all instructions can be executed in parallel, leading to reduced performance in some cases.
- Resource contention: Resource contention occurs when multiple instructions require the use of the same resource, such as a register or a memory location. This can lead to delays and reduced performance if the processor is unable to resolve the contention efficiently.

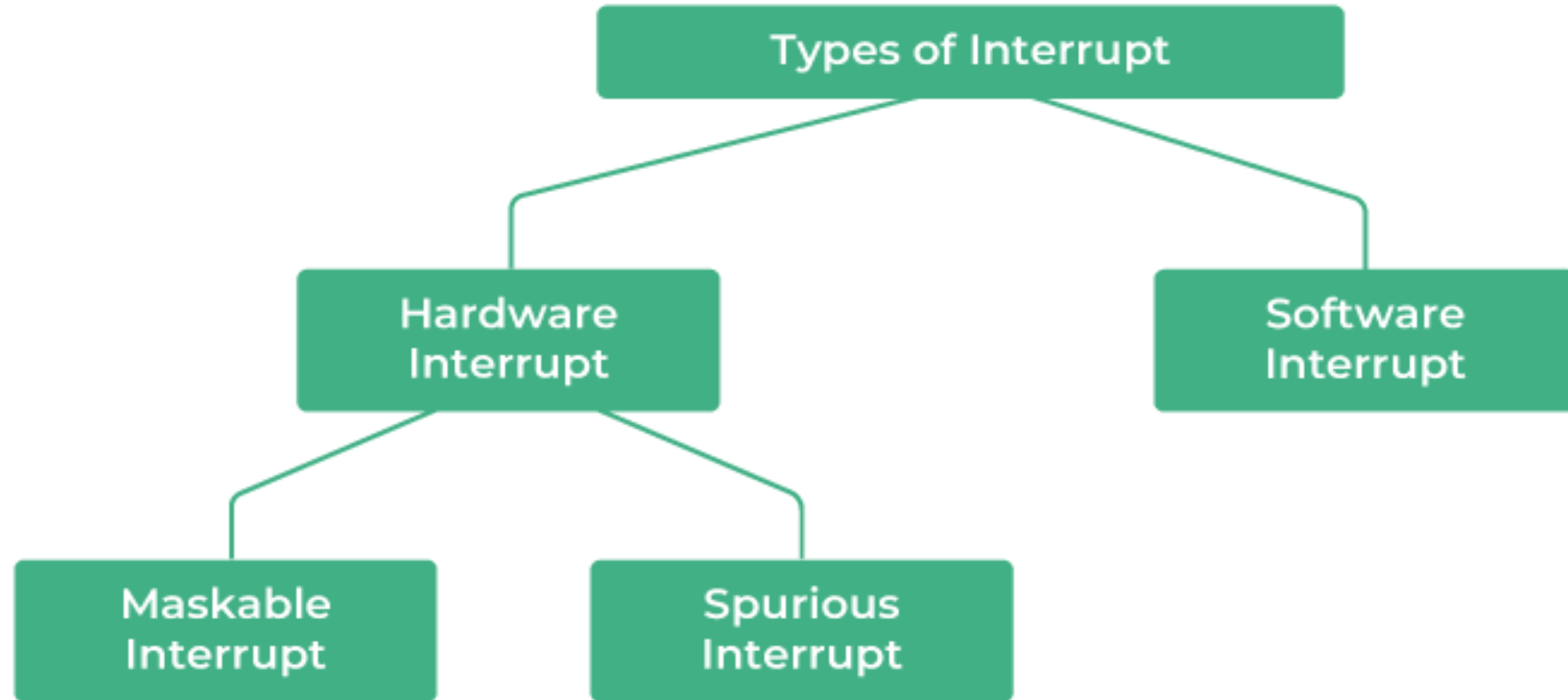
# Interrupt and Class of Interrupts

# Interrupt

- A signal emitted by hardware or software when a process or an event needs immediate attention.
- It alerts the processor to a high-priority process requiring interruption of the current working process.
- In I/O devices one of the bus control lines is dedicated for this purpose and is called the Interrupt Service Routine (ISR).

When a device raises an interrupt at let's say process  $i$ , i.e., the processor first completes the execution of instruction  $i$ . Then it loads the [Program Counter \(PC\)](#) with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process  $i+1$ .

# Types of Interrupt

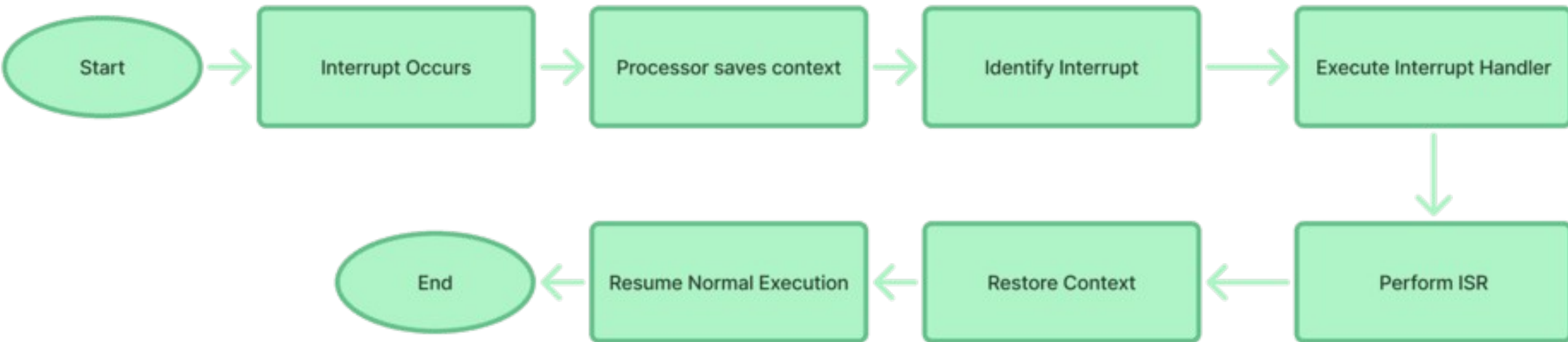


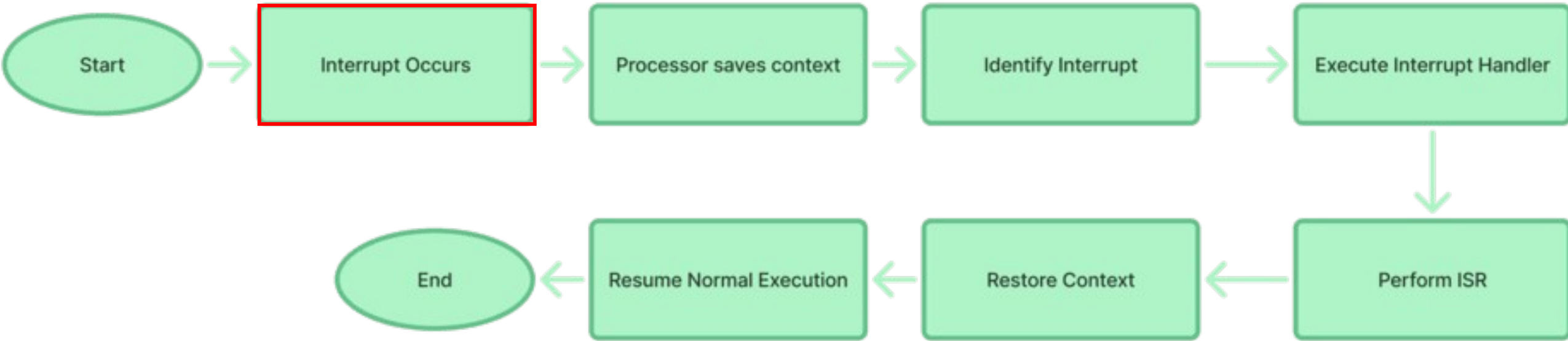


# Sequences of Events Involved in Handling an IRQ(Interrupt Request)

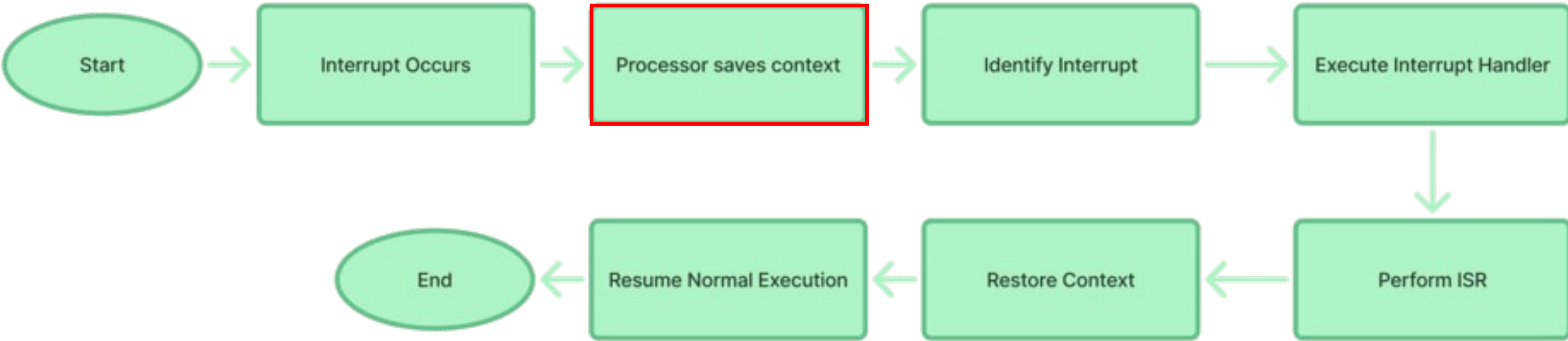
1. Devices raise an IRQ.
2. The processor interrupts the program currently being executed.
3. The device is informed that its request has been recognized and the device deactivates the request signal.
4. The requested action is performed.
5. An interrupt is enabled and the interrupted program is resumed.

# Flowchart of Interrupt Handling Mechanism

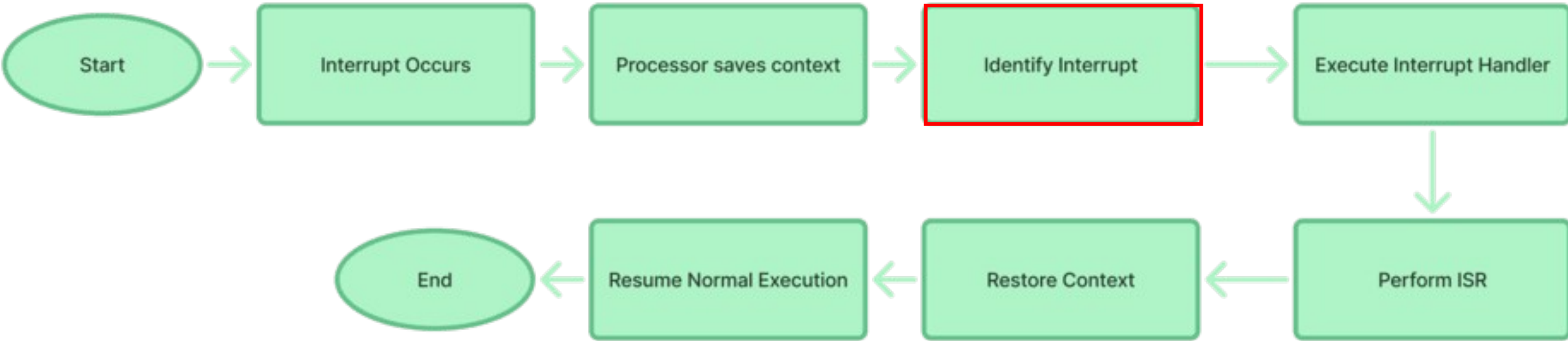




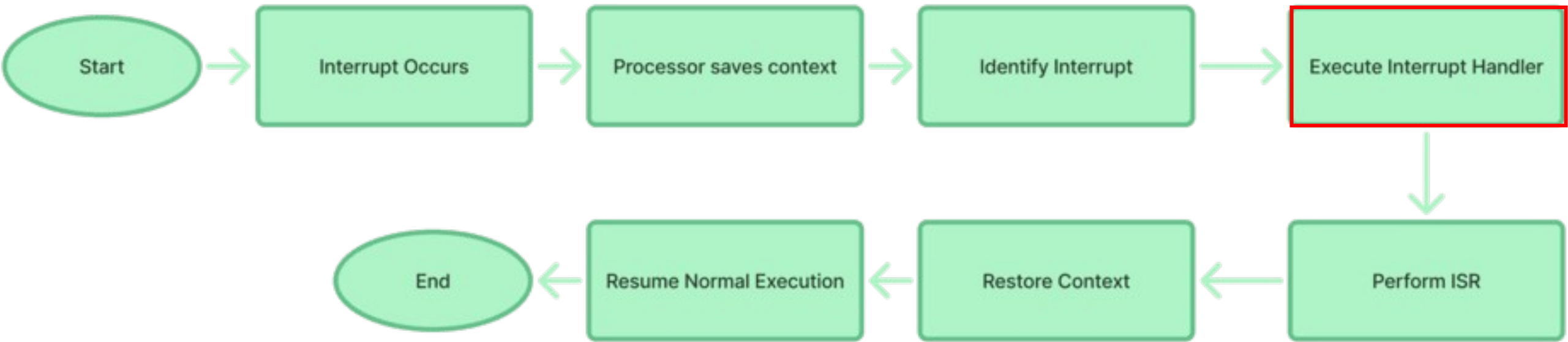
Step 1:- Any time that an interrupt is raised, it may either be an I/O interrupt or a system interrupt.



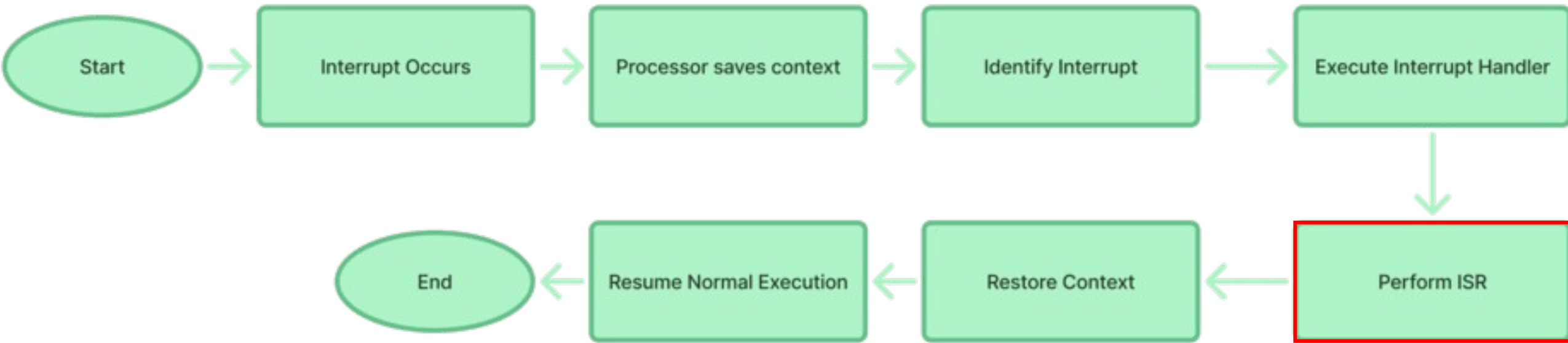
Step 2:- The current state comprising registers and the program counter is then stored in order to conserve the state of the process.



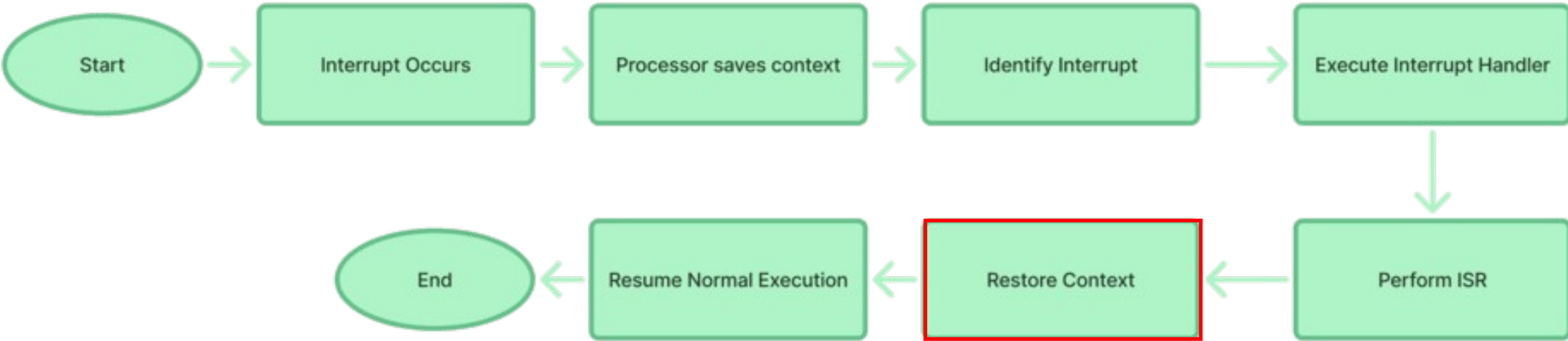
Step 3:- The current interrupt and its handler is identified through the interrupt vector table in the processor.



Step 4:- This control now shifts to the interrupt handler, which is a function located in the kernel space.

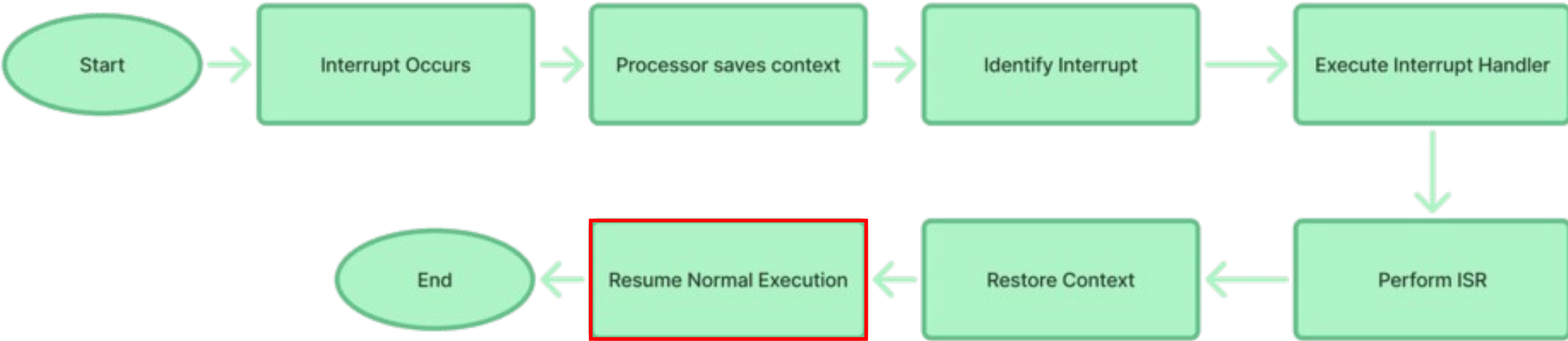


Step 5:- Specific tasks are performed by Interrupt Service Routine (ISR) which are essential to manage interrupt.



Step 6:- The status from the previous session is retrieved so as to build on the process from that point.





Step 7:- The control is then shifted back to the other process that was pending and the normal process continues.

# Managing Multiple Interrupts

- Polling: In polling, the first device encountered with the IRQ bit set is the device that is to be serviced first. Appropriate ISR is called to service the same. It is easy to implement but a lot of time is wasted by interrogating the IRQ bit of all devices.
- Vectored Interrupts: In vectored interrupts, a device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus. This enables the processor to identify the device that generated the interrupt. The special code can be the starting address of the ISR or where the ISR is located in memory and is called the interrupt vector.
- Interrupt Nesting: In this method, the I/O device is organized in a priority structure. Therefore, an interrupt request from a higher-priority device is recognized whereas a request from a lower-priority device is not. The processor accepts interrupts only from devices/processes having priority.

Q.

- Draw a block diagram of basic components of a computer system. Explain each component in detail.  
[10M]
- Explain the instruction cycle in detail.[7M]
- Difference between microprocessor and microcontroller.  
[8]