CSE 4305
Computer Organization and Architecture

Top Level View of Computer

Course Teacher: Md. Hamjajul Ashmafee

Lecturer, CSE, IUT

Email: ashmafee@iut-dhaka.edu

Computer from Top level

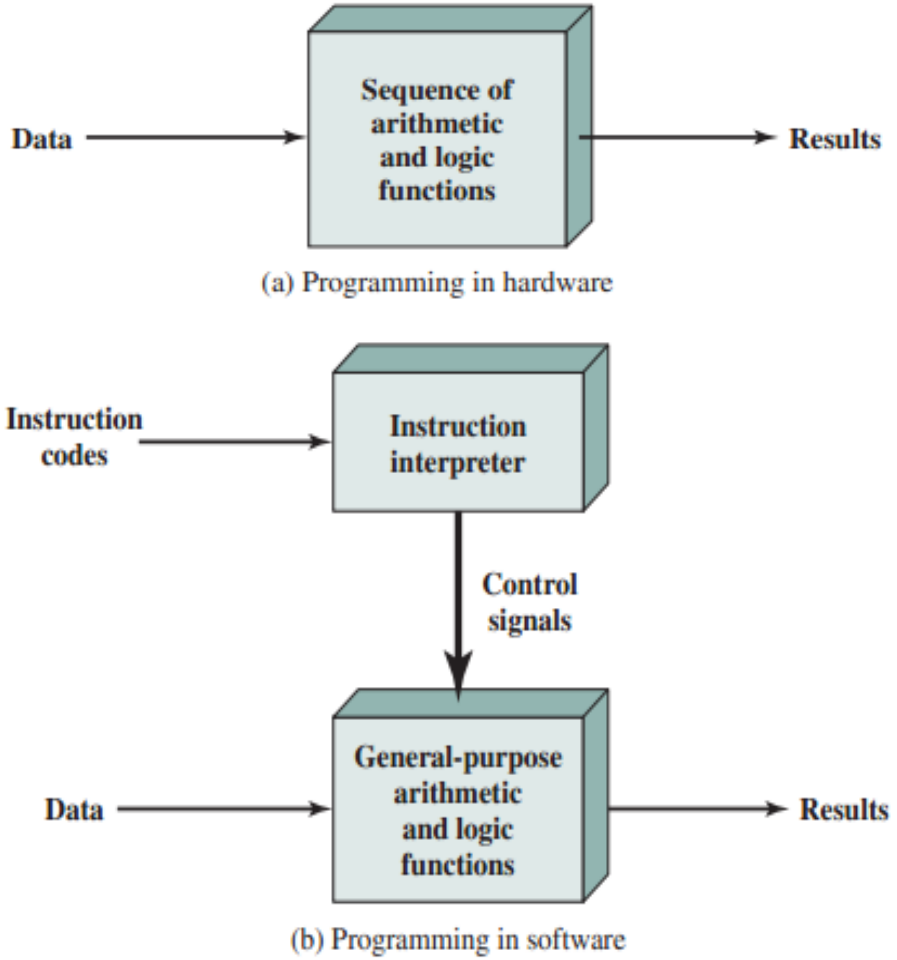
- Computer consists of CPU, Memory, I/O components
- At a top level, we can characterize a **computer system** describing the **behavior** of the **components** and their **interconnection structure**

Key Concepts to Design a Computer

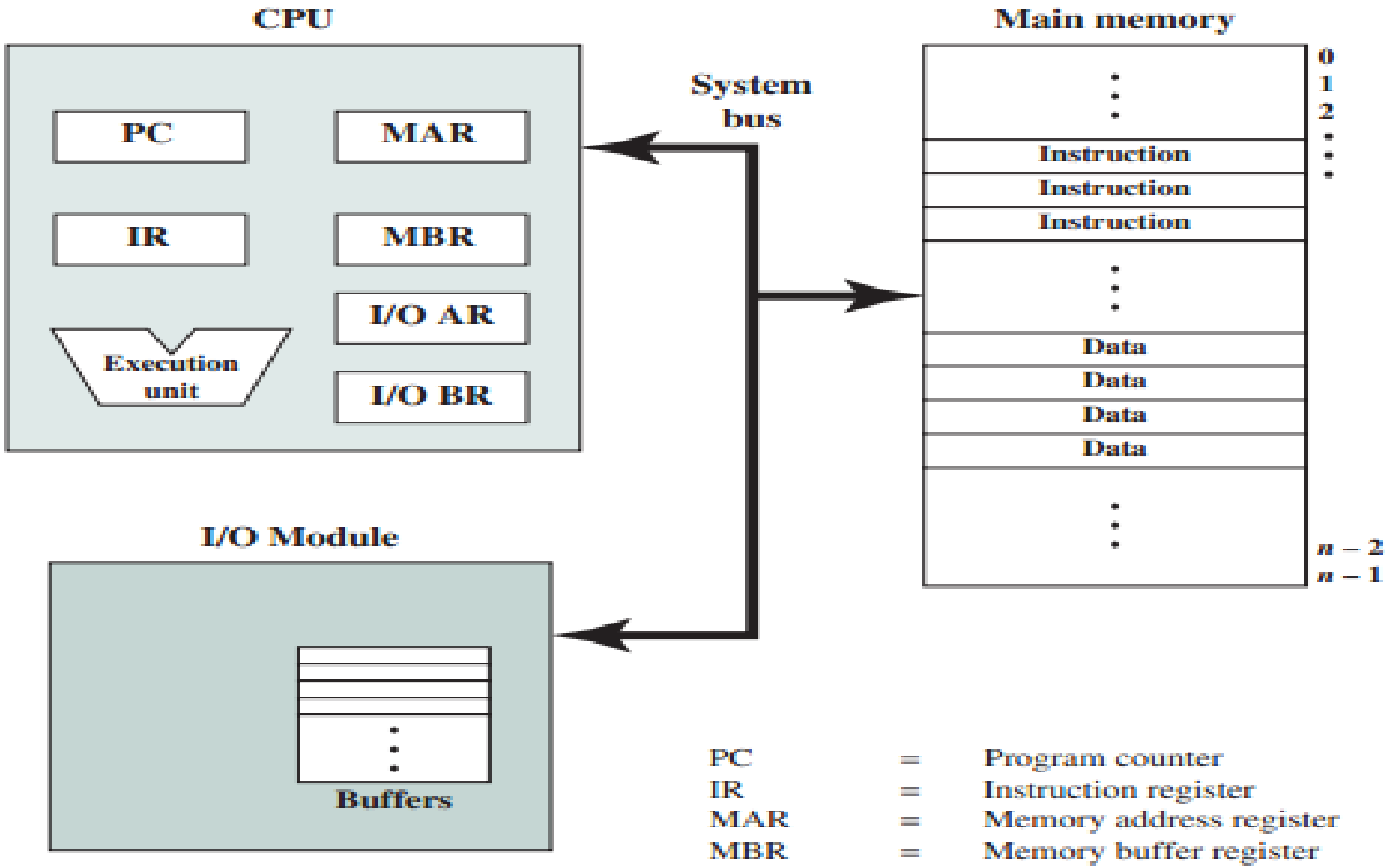
- **Von Neumann Architecture** based on the concepts of:
  - Data and instructions are **stored** in a single read-write memory
  - The contents of the memory are **addressable by location**
  - **Execution** occurs in a sequential fashion from one instruction to next
- **Implementation** of this architecture:
  - Basic logic components – **store** binary data
  - **Arithmetic and logic operation** on that data
  - A particular design of a **configuration of logic components** – to perform computation on that data
- These form of configuration connecting those components regarded as a *programing* – **hardwired programming**

Design a Computer…

- **Alternative Solution:**
  - Construct a **general purpose configuration** of ALU – perform various function based on **control signal (main difference)**
- For customized (dedicated) hardware – system accepts data and produces result (hardware oriented)
- For general-purpose hardware – system accepts data and **control signal** and produces result (program oriented)
- Because of **software**, no need to rewrite the H/W, only provide new codes with the help of I/O components and memory



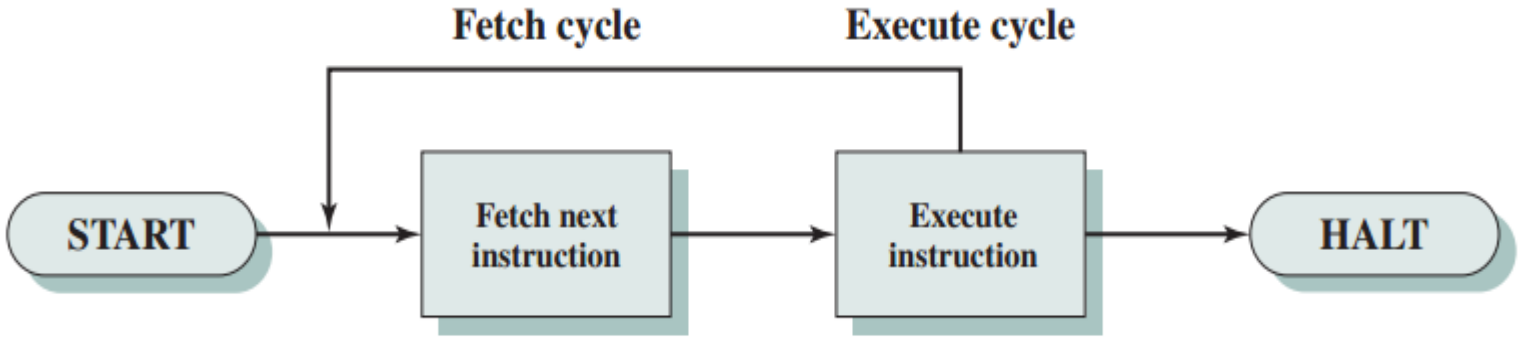
Components and Their Interconnections



PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

Computer Function

- **Main** function of a computer – **Execution of a program** (made of a set of instructions stored in memory) – using processor
- Program execution is consisting of 2 steps **repeatedly** -
  - **Fetch** instructions from memory
  - (Decode +)**Execute** each instructions
- **Instruction cycle** – a **complete processing** required for **a single instruction** – consisting of fetch cycle and execute cycle – halts if machine is turned off, unrecoverable error, HALT instruction



State diagram of an instruction cycle

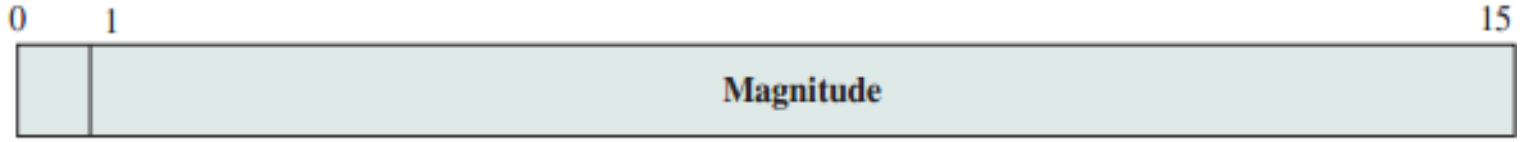
Instruction Fetch and Execute

- At the **beginning** of each instruction cycle, the **processor** **fetches** an instruction **from memory**.
- **Program Counter (PC)** holds the address of the instruction to be fetched next. (usually it increments PC in sequence – next higher address)
- **Instruction Register (IR)** – **Store** the fetched instruction – **specify** the **action** (opcode and address)
- **Types of actions:**
  - **Data transfer:** Processor – memory, Processor – I/O (Load, store, ….)
  - **Data processing:** Arithmetic and Logic Operation on data (add, sub, …)
  - **Control:** Altering the sequence of control (branching, jump, call, …)

Instruction Fetch and Execute…

- **Example: Hypothetical Machine Contains:**

(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory = 1
0010 = Store AC to memory = 2
0101 = Add to AC from memory = 5

(d) Partial list of opcodes

4 bits for opcode = 16 opcodes
12 bits for address = 4K addressable words in memory

Instruction Fetch and Execute…

- **Problem:**

Assume that the program counter is set to memory location 300, where the location address refers to a 16-bit word. The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on.

The program adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.

- **Solution:**

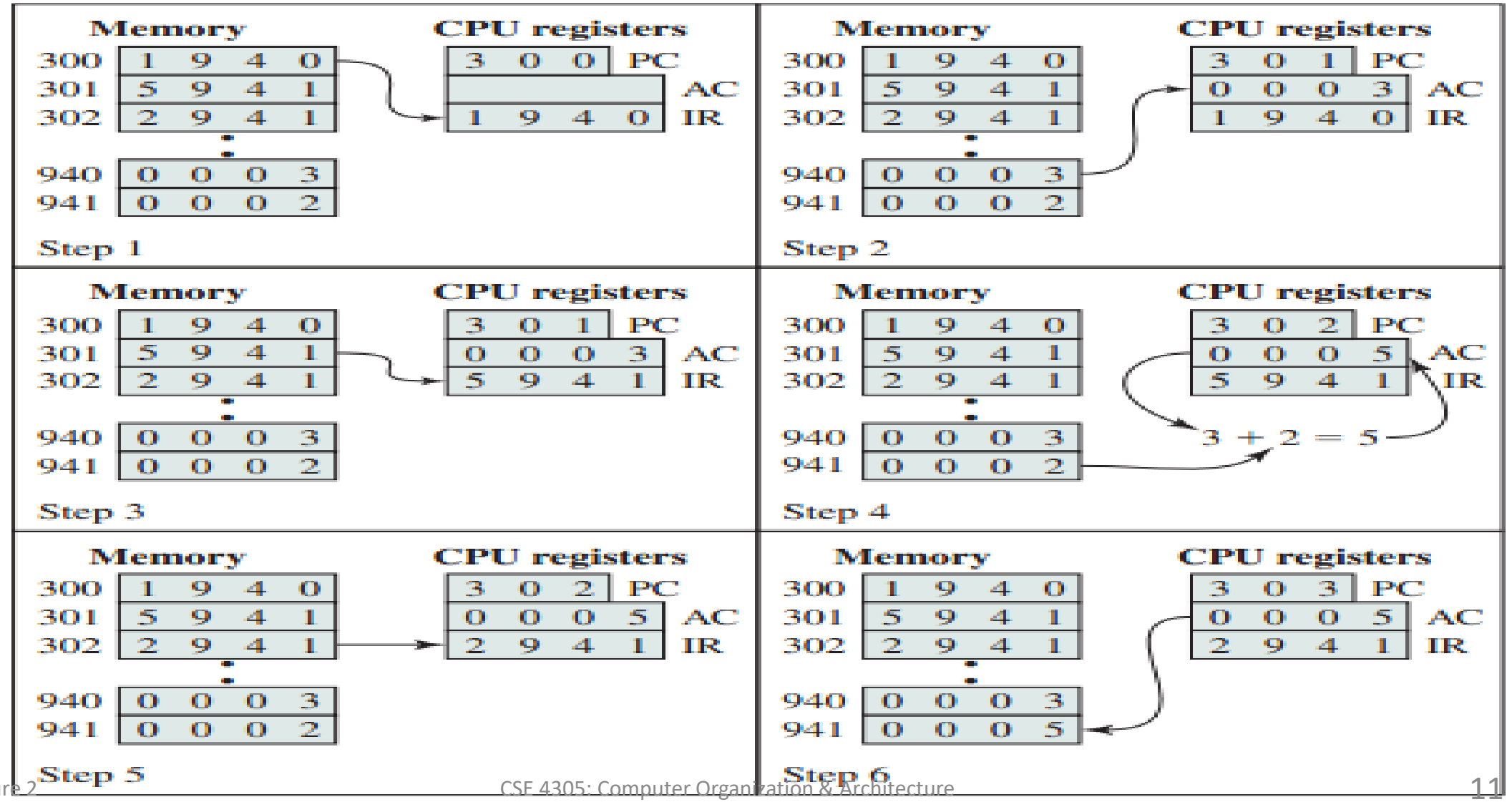
3 instructions – 3 fetch cycles and 3 execute cycles

Instruction Fetch and Execute…

3 instructions can be described as following:

1. PC contains 300 (the address of the first instruction). This instruction is loaded into IR from 300 & PC will be incremented. (Using MAR and MBR) – instruction – **1940**
2. The first Instruction's opcode (4 bits) in IR indicates that AC is to be **loaded**. Remaining 12 bits specify the address from where data are to be loaded
3. The next instruction is fetched from location 301 & PC is incremented - **5941**
4. Old AC and contents of the location 941 are **added** and result is stored in AC
5. The next instruction is fetched from location 302 & PC is incremented - **2941**
6. The contents of the AC are **stored** in location 941

Instruction Fetch and Execute…



Instruction Execution: ADD B, A

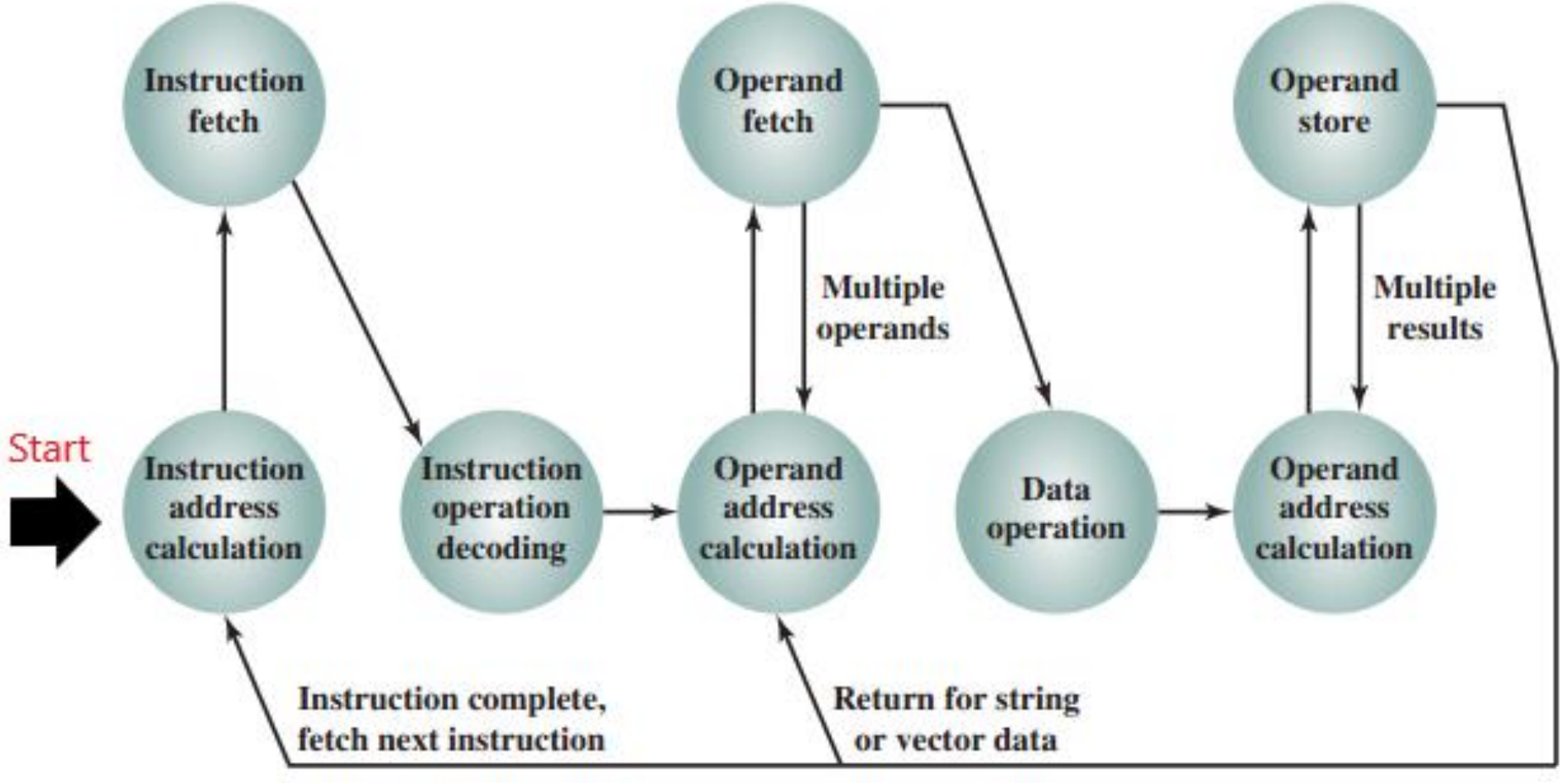
- **ADD B, A:** to store the sum of the contents of memory locations B and A into memory location A.

A single instruction cycle with the following steps occurs:

- Fetch the ADD instruction.
- Read the contents of memory location A into the processor.
- Read the contents of memory location B into the processor. In order that the contents of A are not lost, the processor must have at least two registers for storing memory values, rather than a single accumulator. (using 2 registers)
- Add the two values.
- Write the result from the processor to memory location A.

An instruction or execution cycle may require **more than one memory reference**.

State Diagram of A Basic Instruction Cycle



States' name:
IAC
IF
IOD
OAC
OF
DO
OS

- ADD A, B
- ADD A[10], B
- Binary -> A

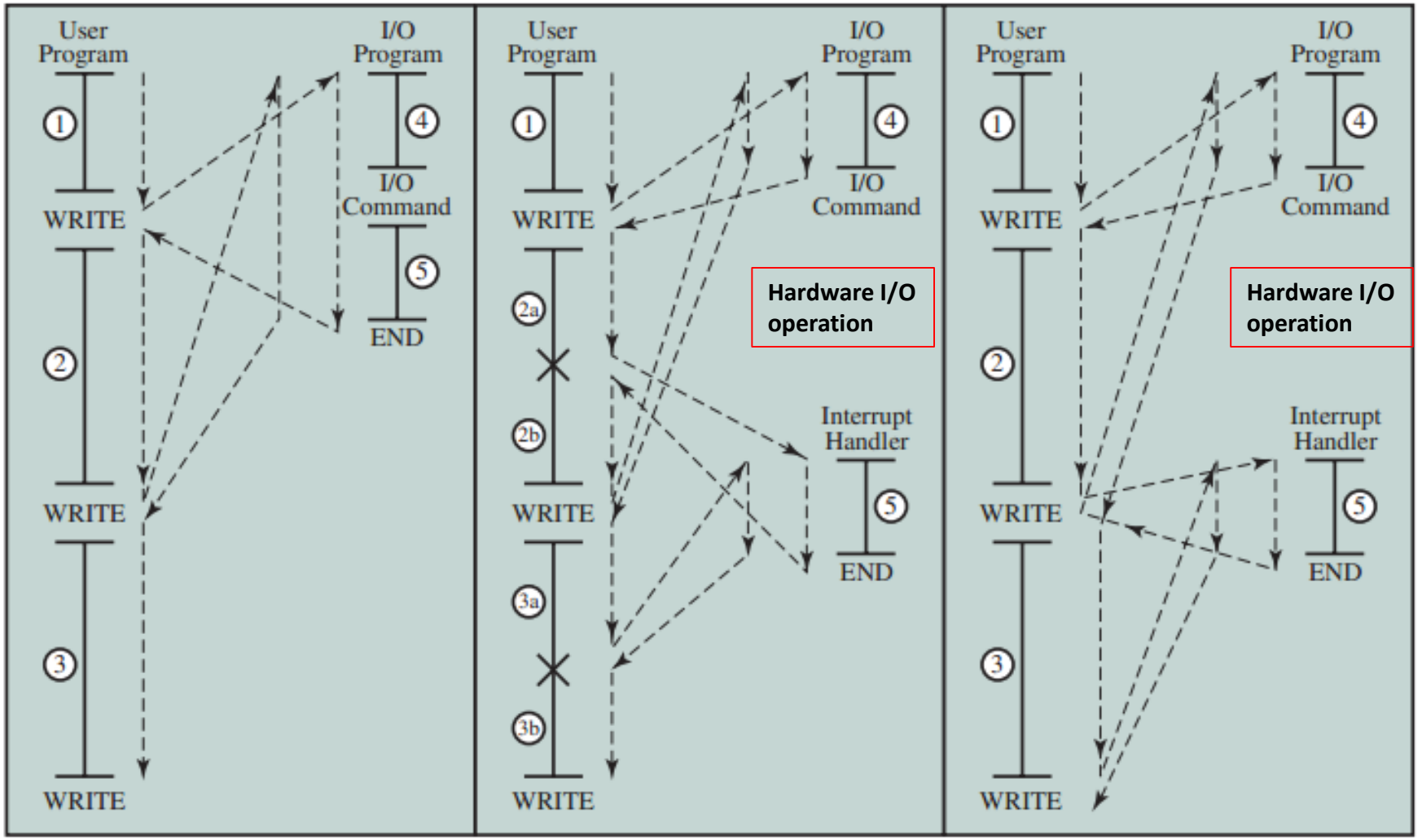
Some states may be null or may be visited multiple times.

Interrupt

- **Interrupt:** A **mechanism** by which **other modules** (like I/O, memory) can **disturb/ break** the normal processing of the processor – to **improve** processing efficiency (like avoiding idle cycles)

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware Failure	Generated by a failure such as power failure or memory parity error.

Program flow with and without Interrupt



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

× = interrupt occurs during course of execution of user program

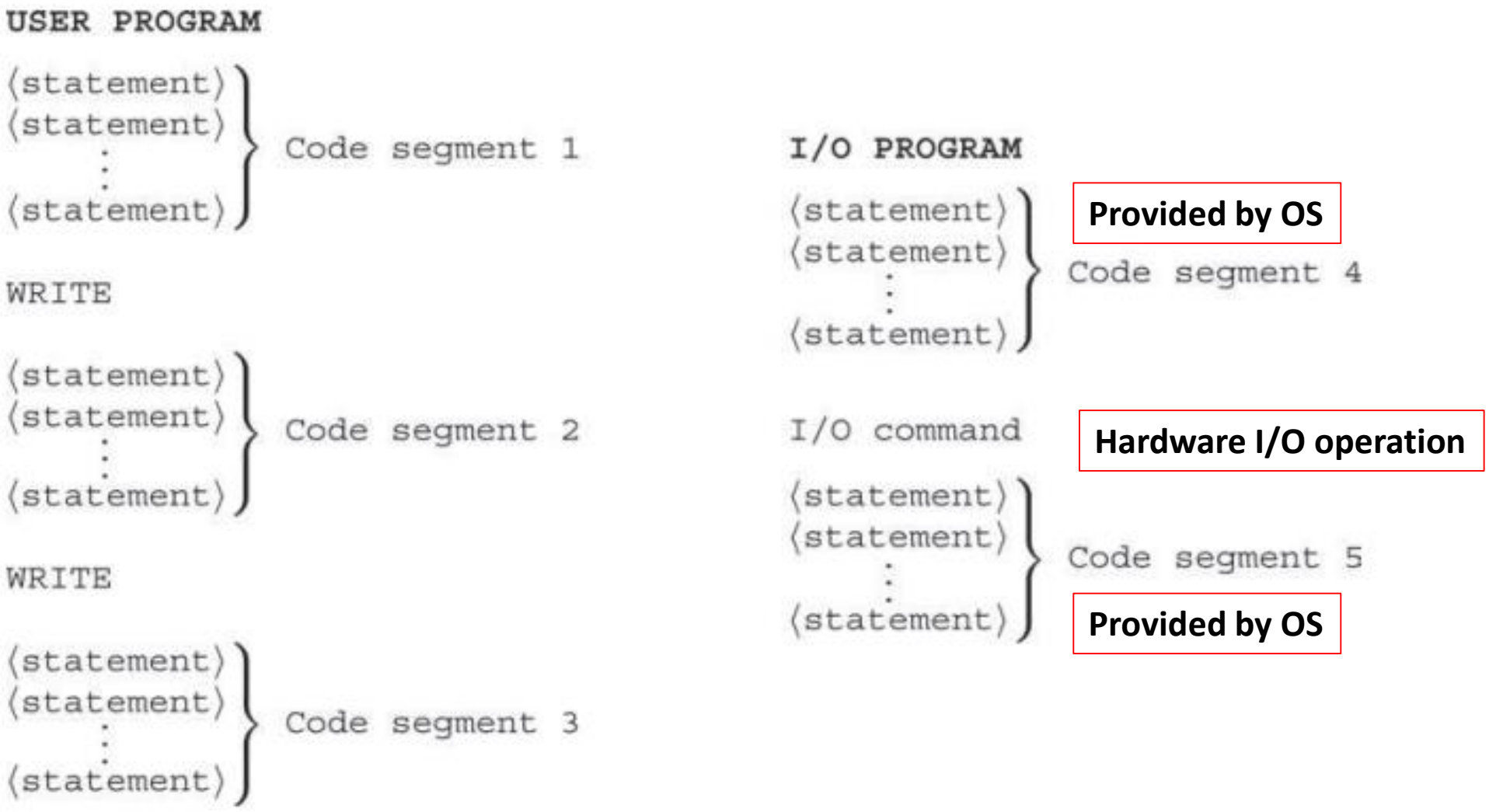
A Simple Interrupting Program

- An interrupting **program** consists of :
  - A sequence of instruction for **preparation** (using processor) – **label 4**
  - The **actual interrupt command** (without processor, wait) – **label break**
  - A sequence of instruction to **indicate success or failure** (using processor) – **label 5**
- An Interrupting **program** may take a **relatively long time** to complete – user program is **stopped** for some considerable period of time.

Interrupt and Instruction Cycle

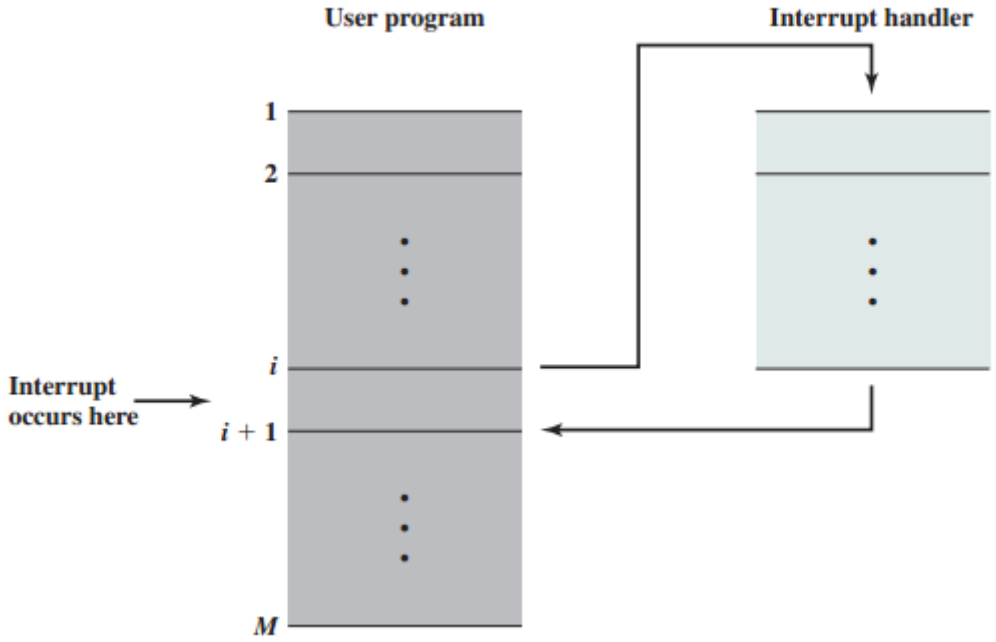
- **Main feature of Interrupt**: with it, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- After initiating the interrupting program & executing few instructions, control returns to the user program – after being **initiated**, the **interrupting program** will perform its own task without disturbing the processor concurrently.
- **Example:** printing text from the main memory
- During performing the external task (from processor), if the interrupting module needs the help from the processor, it sends an interrupt request to processor
- Processor responds that module (interrupt module) branching off to a **program (interrupt handler)** to serve that external device
- Processor resumes its original task after the device is served

User and I/O Program in Details

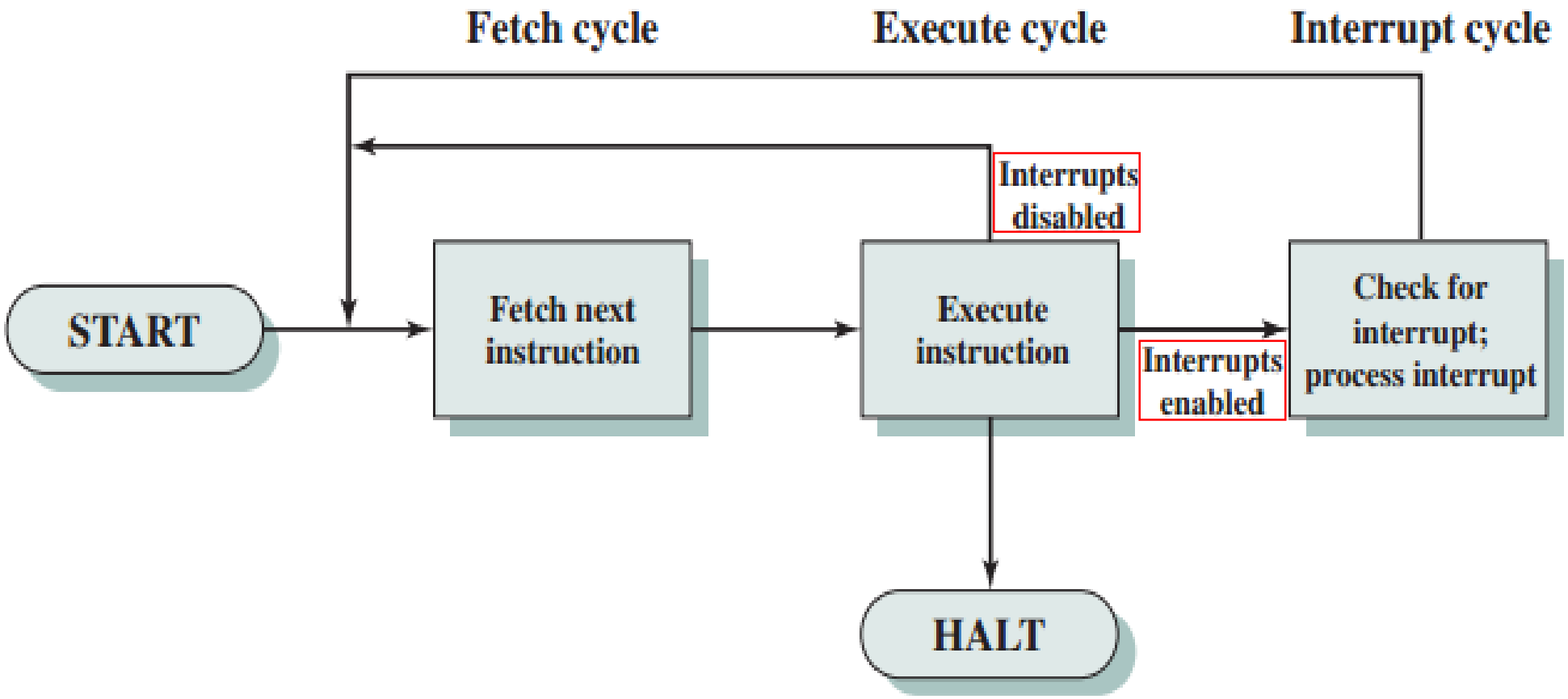


Interrupts…

- An interrupt is just **a normal sequence of execution**- when interrupt processing is completed, main/ user program resumes.
- The **user program** needs not to contain any special code to accommodate interrupt – **OS and processor** are responsible to suspend the user program and resume it at the same point.



Instruction Cycle with Interrupt (Redesign)

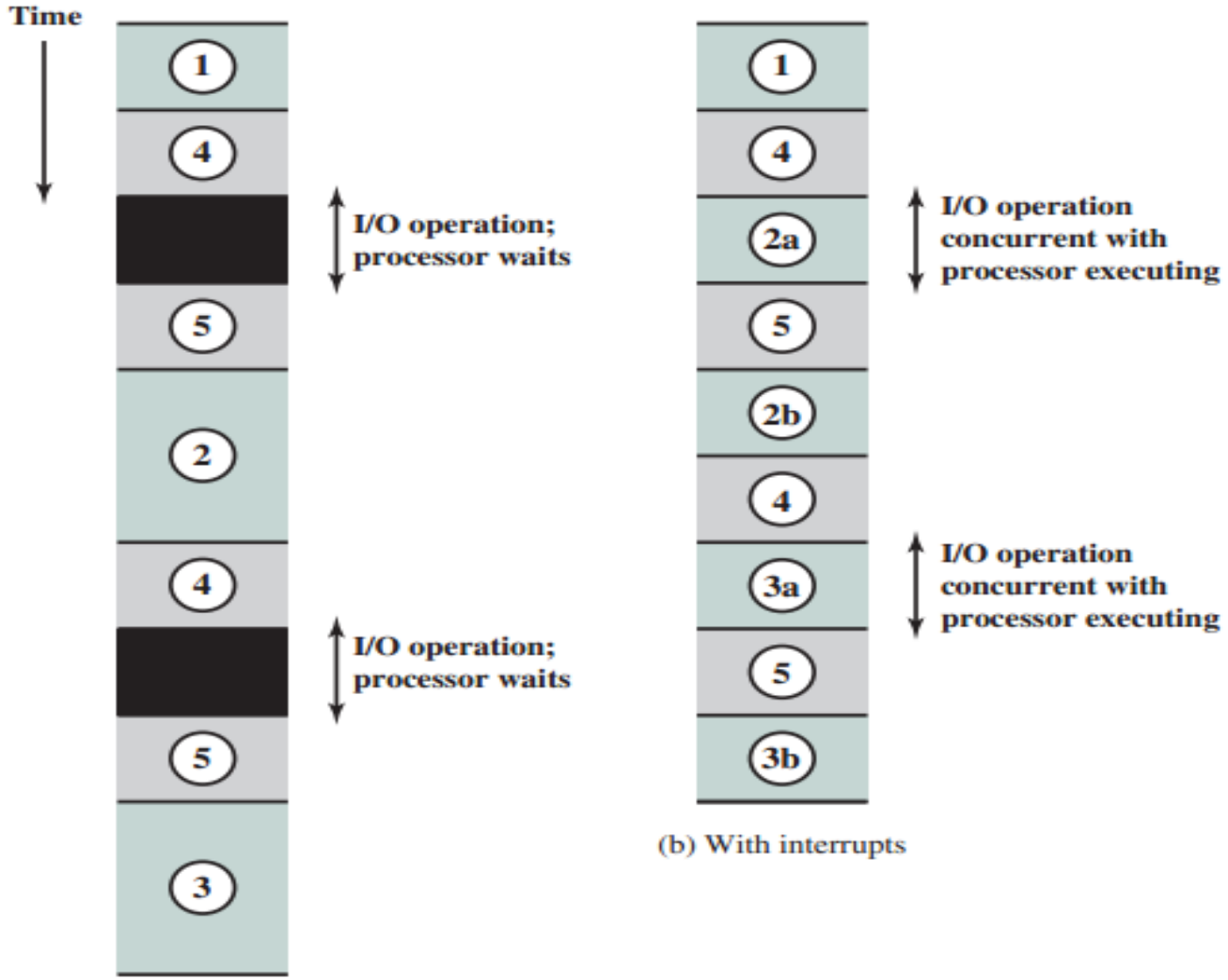


Response to a Pending Interrupt

- The **processor** will perform like this way:
  - **Suspends** execution of the current program & save its context (**save** basic registers) to track its current activity
  - It sets **PC** to the **starting address of an interrupt handler routine** (part of OS)
- **Interrupt handler routine**: determines the **nature of interrupt** and **perform** the required actions – also known as Interrupt Service Routine (ISR)

Interrupt – Gain in Efficiency – Timing Diagram

Short I/O wait

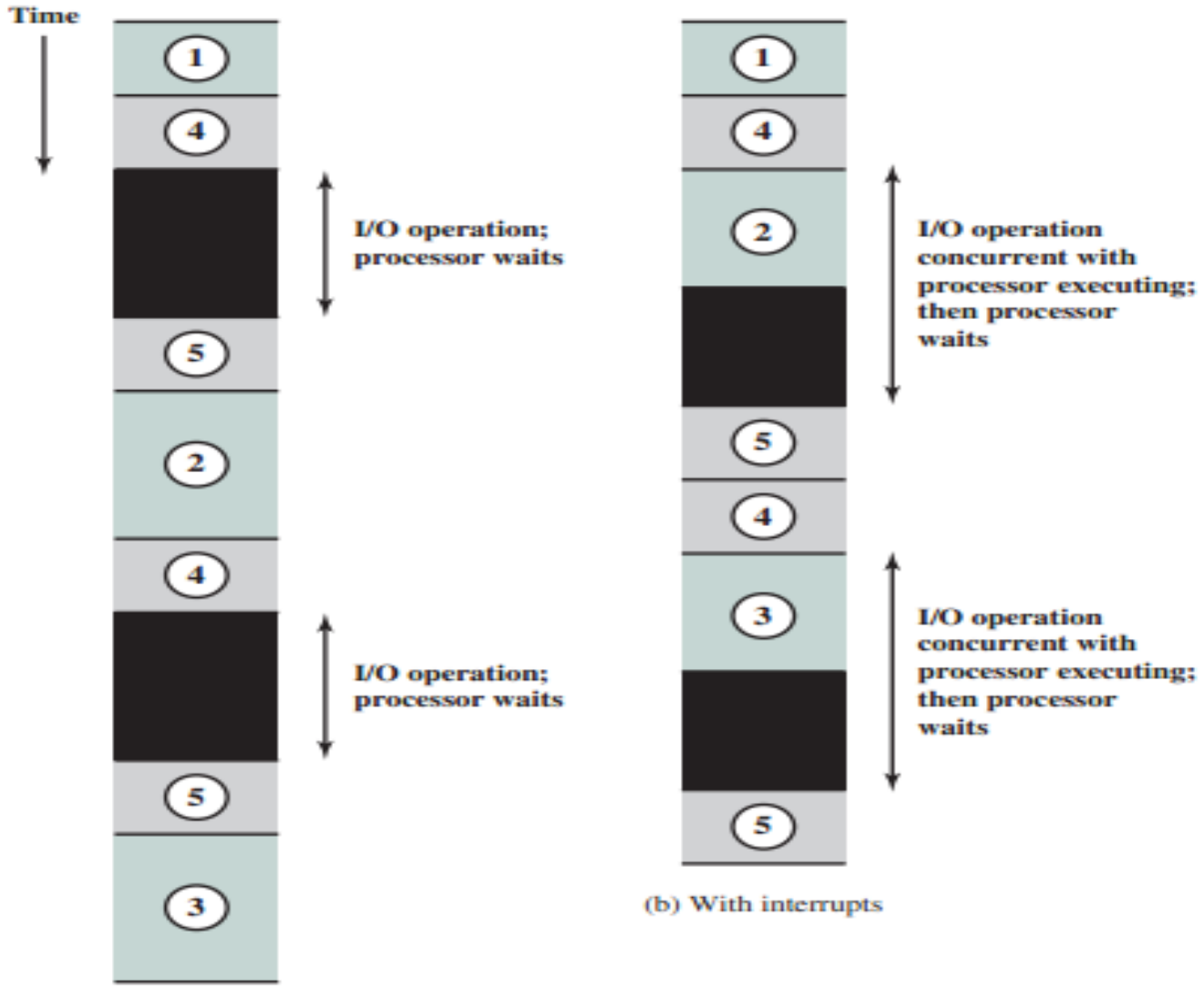


(a) Without interrupts

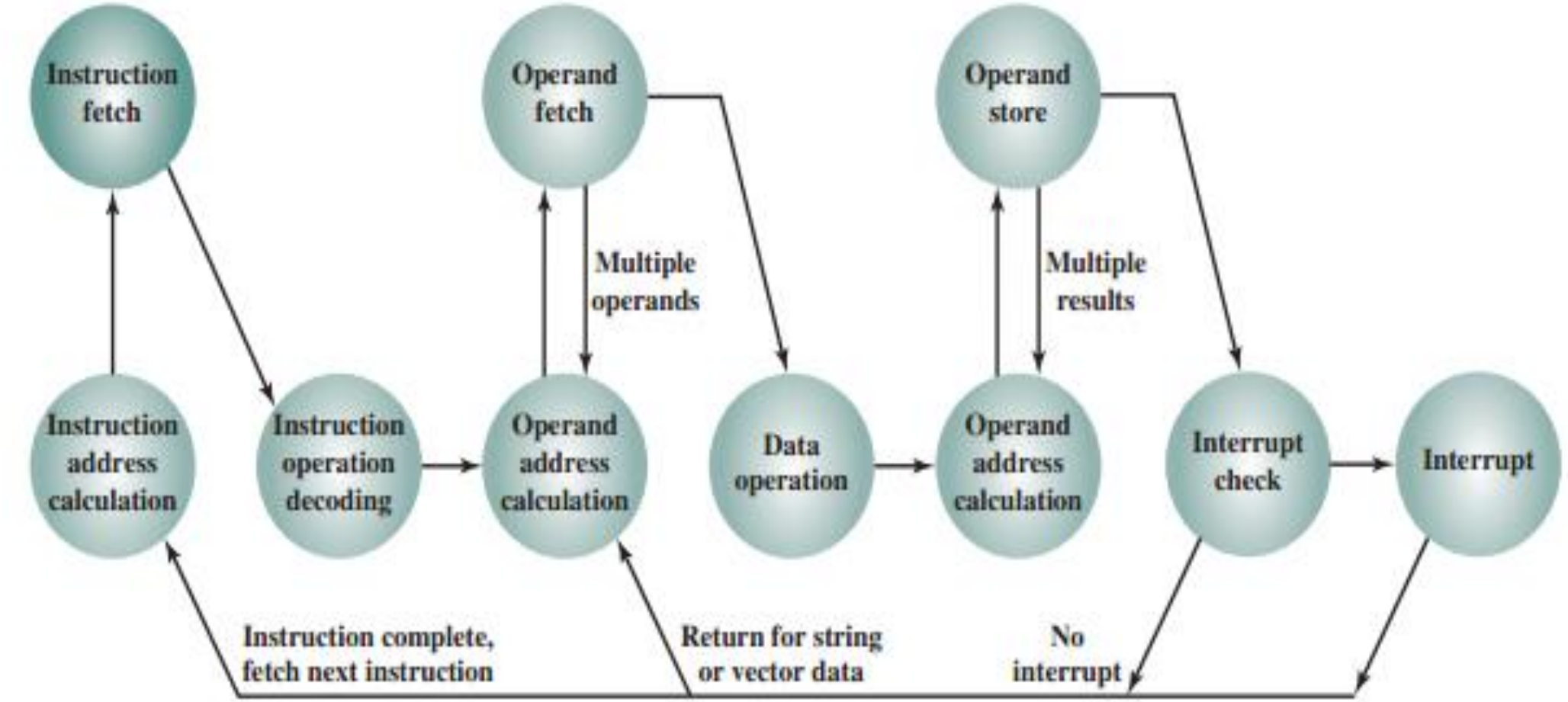
(b) With interrupts

Interrupt – Gain in Efficiency – Timing Diagram…

Long I/O wait



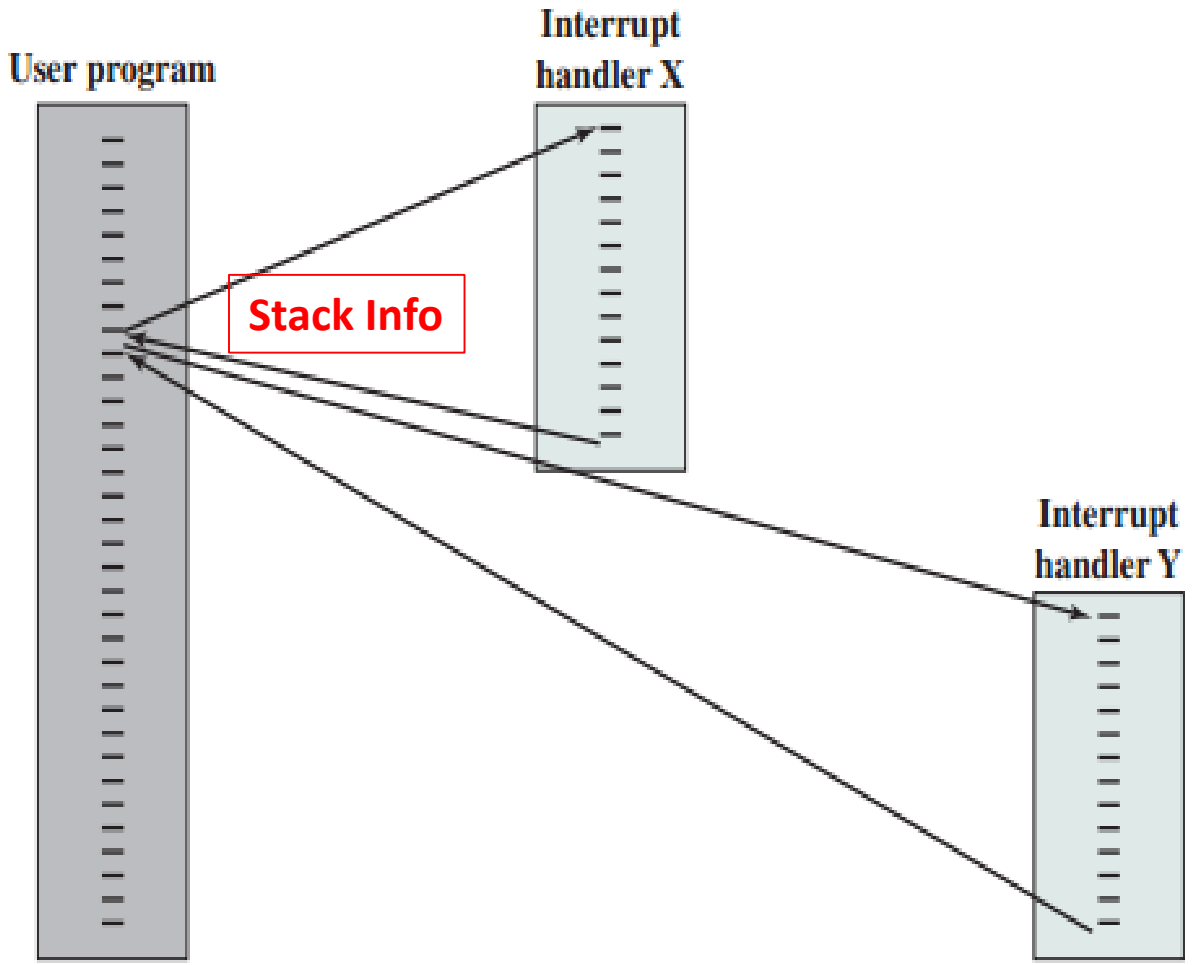
State Diagram of Instruction Cycle with Interrupt



Multiple Interrupts

- **Example:** Receiving data from communication line and printing results
- **Two approaches** to deal with multiple interrupts
- **First approach** –
  - **Disable the interrupt** while processing an interrupt handler (**Disable Interrupt**)
  - Done by simply **ignoring other interrupt requests**
  - If any interrupt requests, **OS will keep it as pending**
  - Pending interrupts will be **checked when user program resumes its execution**
  - **Nice and simple** – handling interrupts in **strict sequential order**
  - **Drawback** – **not recognize** relative **priority** or **time critical needs** (rapid response)

Sequential Interrupt Processing

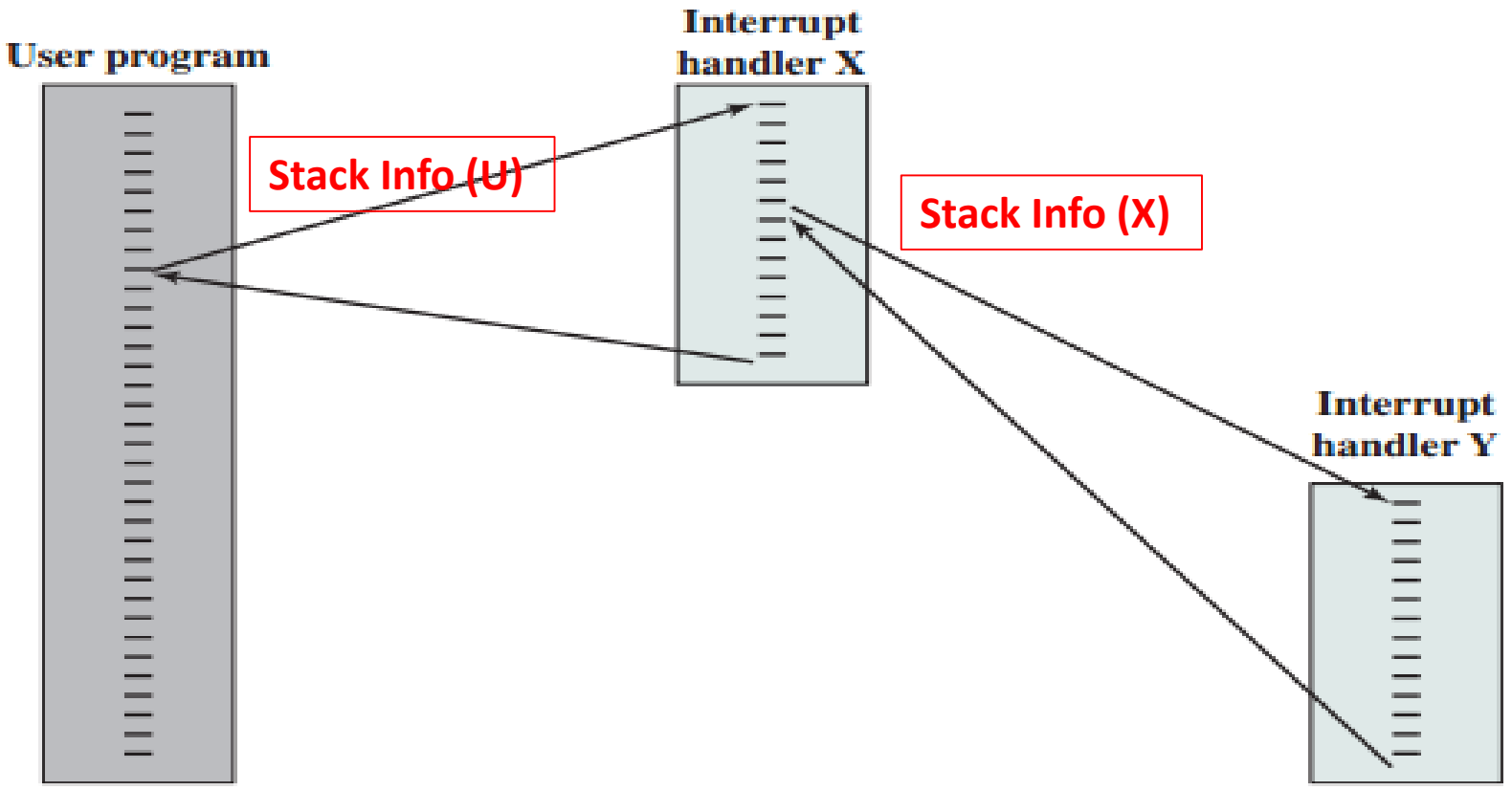


(a) Sequential interrupt processing

Multiple Interrupts…

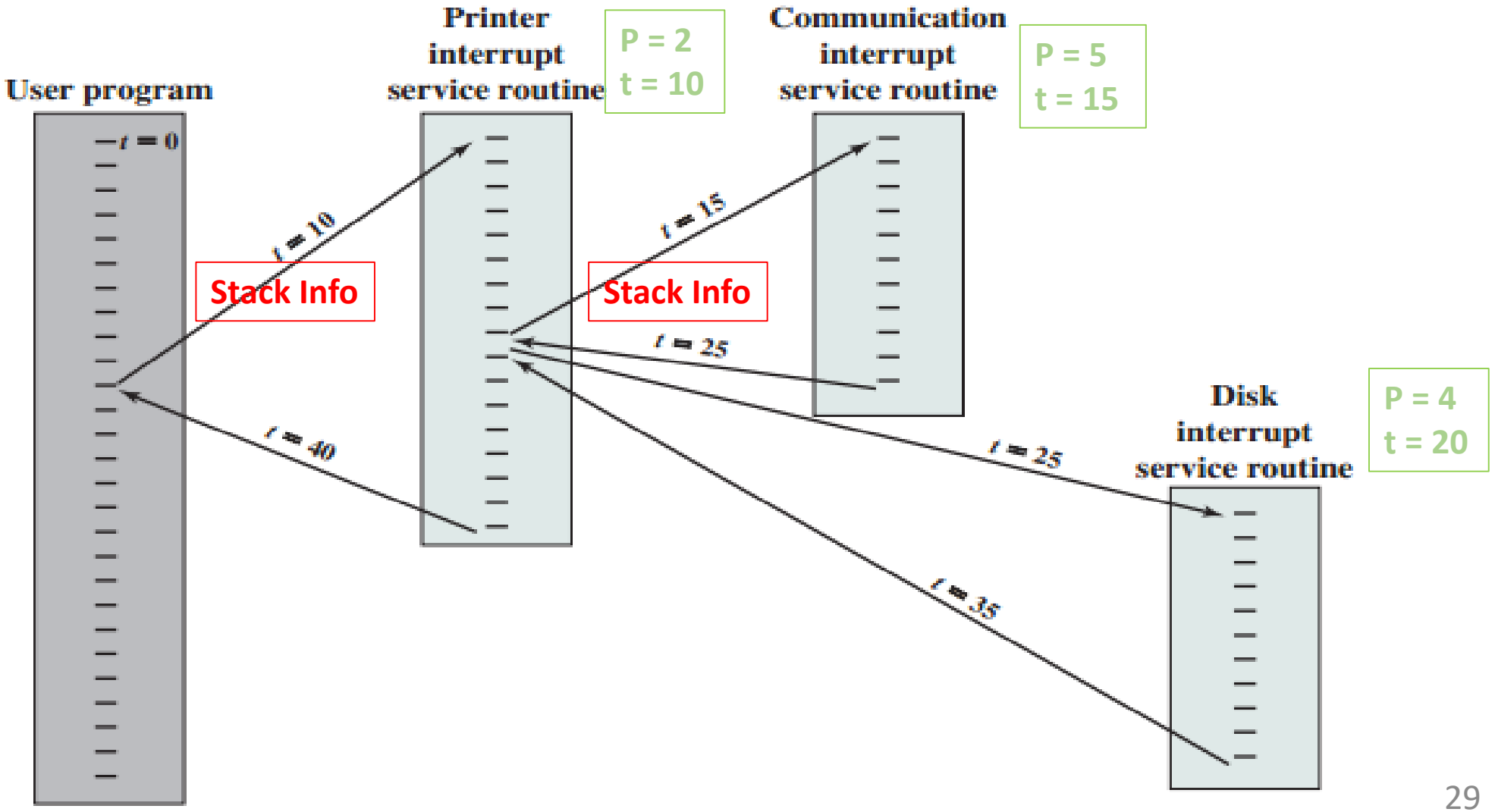
- **Second approach**:
  - Define the **priorities** for the interrupts – higher to lower
  - If Lower priority handler ISR is in service and gets interrupt from higher priority interrupt, the previous ISR will be interrupted o/w it continues its task.
  - **Honor the higher priority ISR before executing even a single instruction in lower priority ISR**
  - One kind of **nested approach**.

Nested Interrupt Processing



(b) Nested interrupt processing

Multiple Interrupts…



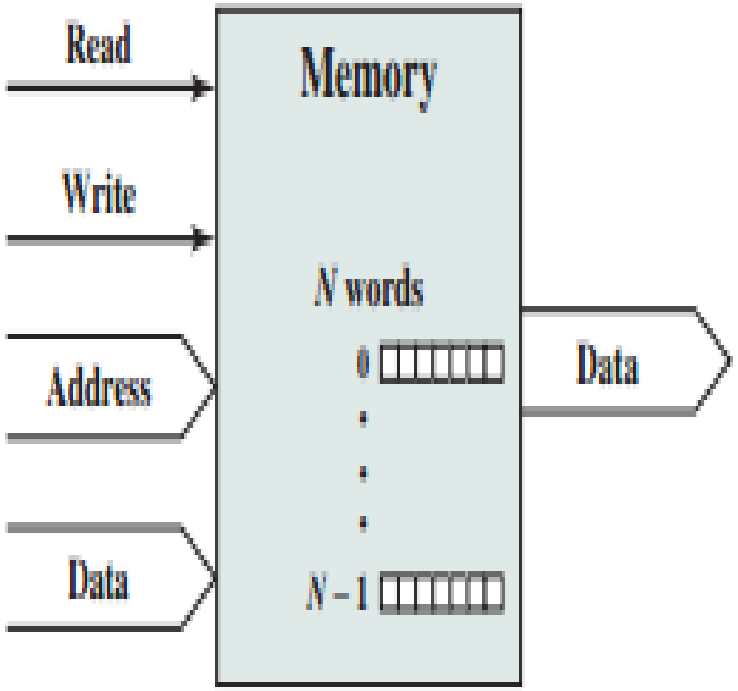
I/O Function

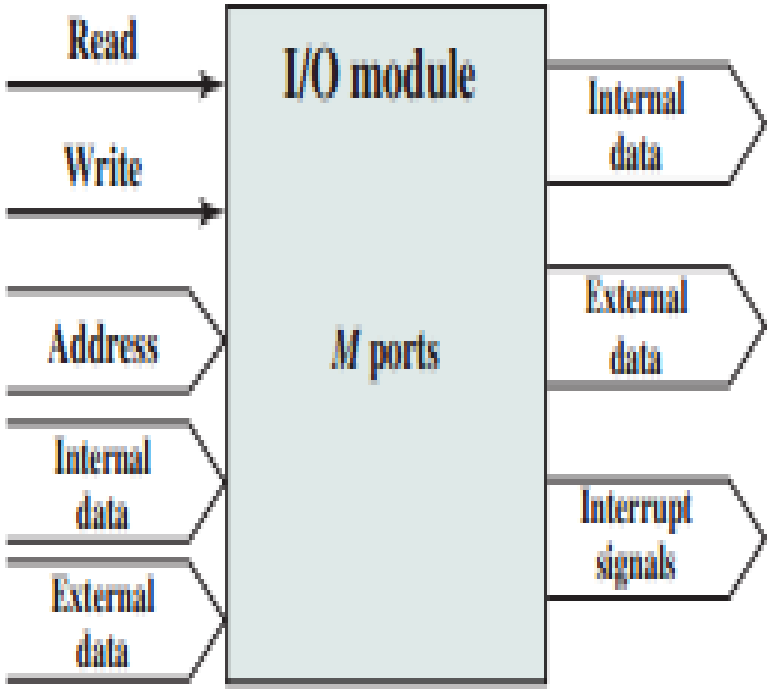
- An **I/O module** (e.g., a disk controller) can exchange data directly with the **processor** – (Chapter 7)
- The **processor** can also read data from or write data to an **I/O module** designating the **port address** – like **memory** and its **memory location**.
- **In some cases**, it is desirable to allow I/O exchanges to occur directly with memory.
- In such a case, the **processor grants** to **an I/O module** the authority to read from or write to memory, so that the I/O-memory transfer can occur without tying up the processor. (Example: storing a large file from I/O device to memory)
  - This operation is known **as direct memory access (DMA)**

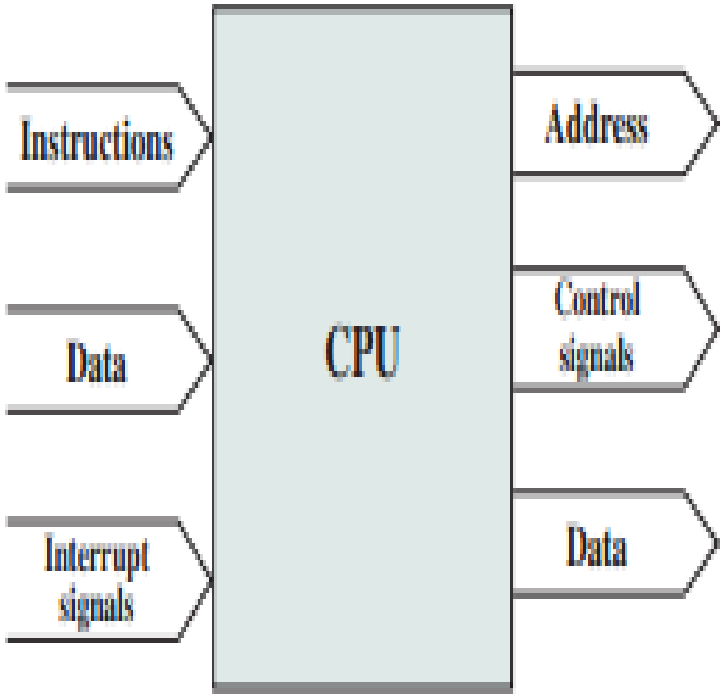
Interconnection Structures

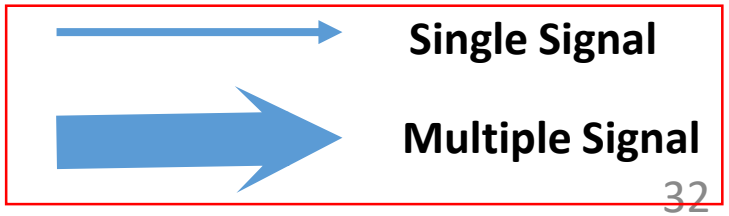
- **Computer components** (like CPU, memory, I/O) **communicate with each other** – computer is a **network of basic modules through paths**
- **Collection of paths** connecting various modules build **interconnection structure** – established based on the **exchange**
- **Basic modules** and their **exchange types**:
  - **Memory** – a data word can be **read** from or **written** to the **memory specified by an address**
  - **I/O model –** same as a memory; **read and write operation** – **controlled by external devices** – through an **interface** (port; address) – **dedicated** input-output data path with I/O devices – also able **to send interrupt**
  - **CPU – read** data and instruction before execution and **write** data after data processing on a specific **address/ location** – also generates control signal to control others – **receives interrupt**

Basic Modules and Types of Exchanges









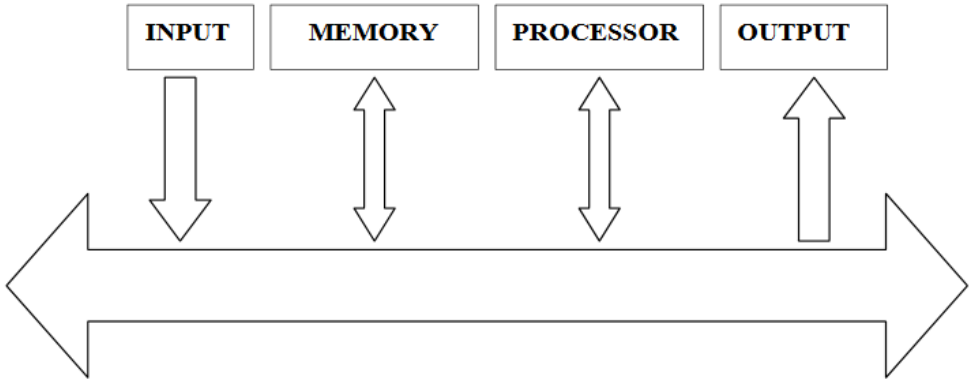
Types of Data Transfer

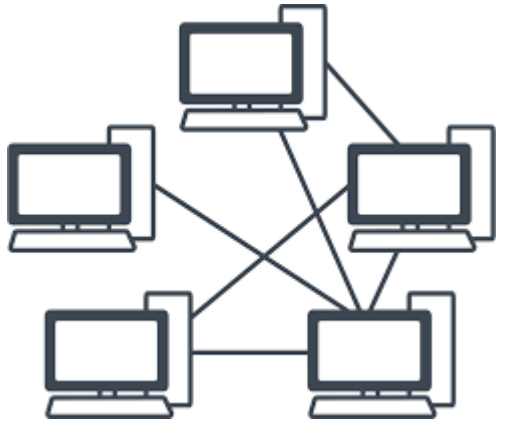
- Memory to processor
- Processor to memory
- I/O to Processor
- Processor to I/O
- I/O to memory
- Memory to I/O

They have different data format, data rate, transfer protocols, mediums.

Different Interconnection structures

- **BUS** interconnection structures
- **Point-to-point** interconnection structures





Bus Interconnection

- **BUS** – **dominant means** of computer system components **interconnection** – communication pathway connecting two or more devices
- **But** gradually **given way** to point to point interconnection
- **Used** in *embedded system, microcontrollers*
- **Main feature: Shared transmission medium** - Multiple devices connect to the bus
- A signal transmitted by any one device is available for reception by all other devices attached to the bus.
- **Drawback:** if two devices transmit during the same time period, their signals will overlap and become illogical - only **one device** at a time can successfully transmit – controlled by **BUS master**

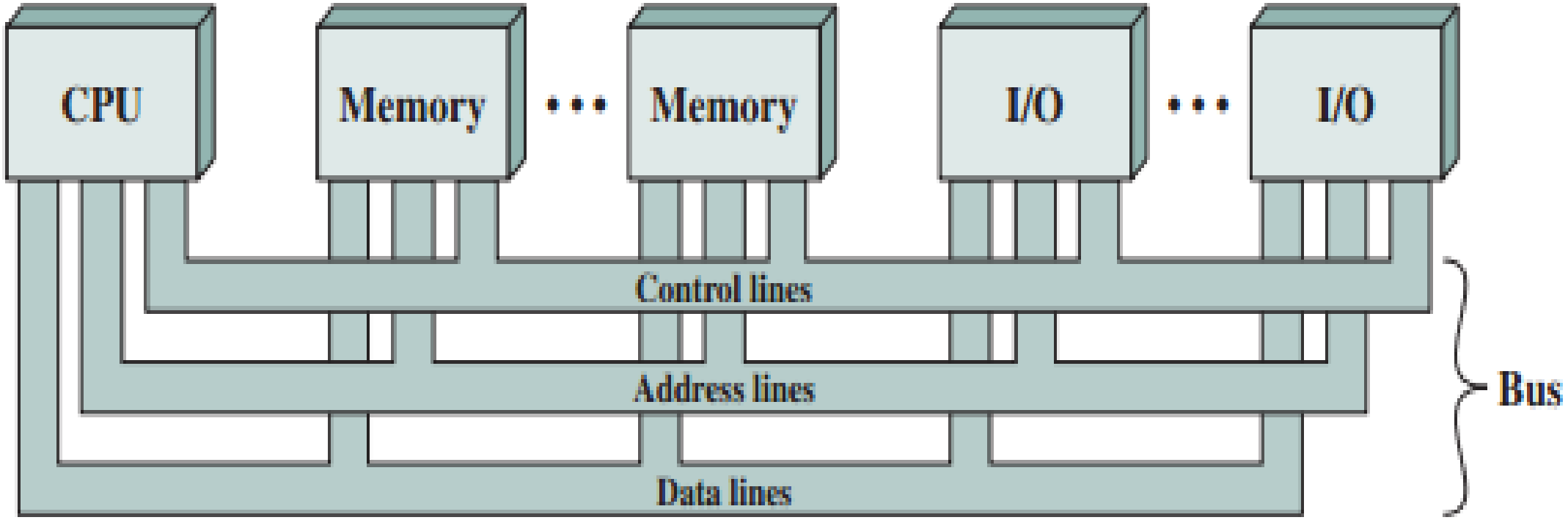
Bus Interconnection…

- BUS consists of **multiple** communication **pathways or lines** – **each** of them capable of transmitting **binary signals** (1 or 0)
- A **sequence of binary digits** can be transmitted across a **single line**
- **Several lines** of a BUS used to transmit **simultaneously/ in parallel**
- Different buses at **various levels** of system hierarchy
- **System Bus** – **connect major components** – consists of **50-100 lines**; each assigned for a **particular function**
- BUS lines are **grouped** into **data, address and control lines** – additionally power distribution lines

Bus Interconnection…

- **Data lines** – **move data** among system modules – collectively called as Data Bus – 32, 64, 128 or even more lines in width of data bus – **key factor** in system performance
- **Address lines** – **designate** the **source and destination of data** on data bus – width of address bus determines the **maximum memory capacity** – also used **to address I/O ports** – higher order bits for module selection and lower order bits for locations
- **Control lines** – to **control** access, data transfer and address selection – because **data and address bus are shared** by all components – **transmit command** (for operation) and **time** (for validity) – memory read/ write, I/O read/ write, Transfer ACK, BUS request/ grant, Interrupt request/ACK , clock, reset

Bus Interconnection…



Bus Operation

- To **send data** to another module:
  - **Obtain** the use of the bus
  - **Transfer** data via the bus
- To **request data** from another module:
  - **Obtain** the use of the bus
  - **Transfer a request** to the other module over the appropriate control and address lines - **wait** for that second module to send the data.

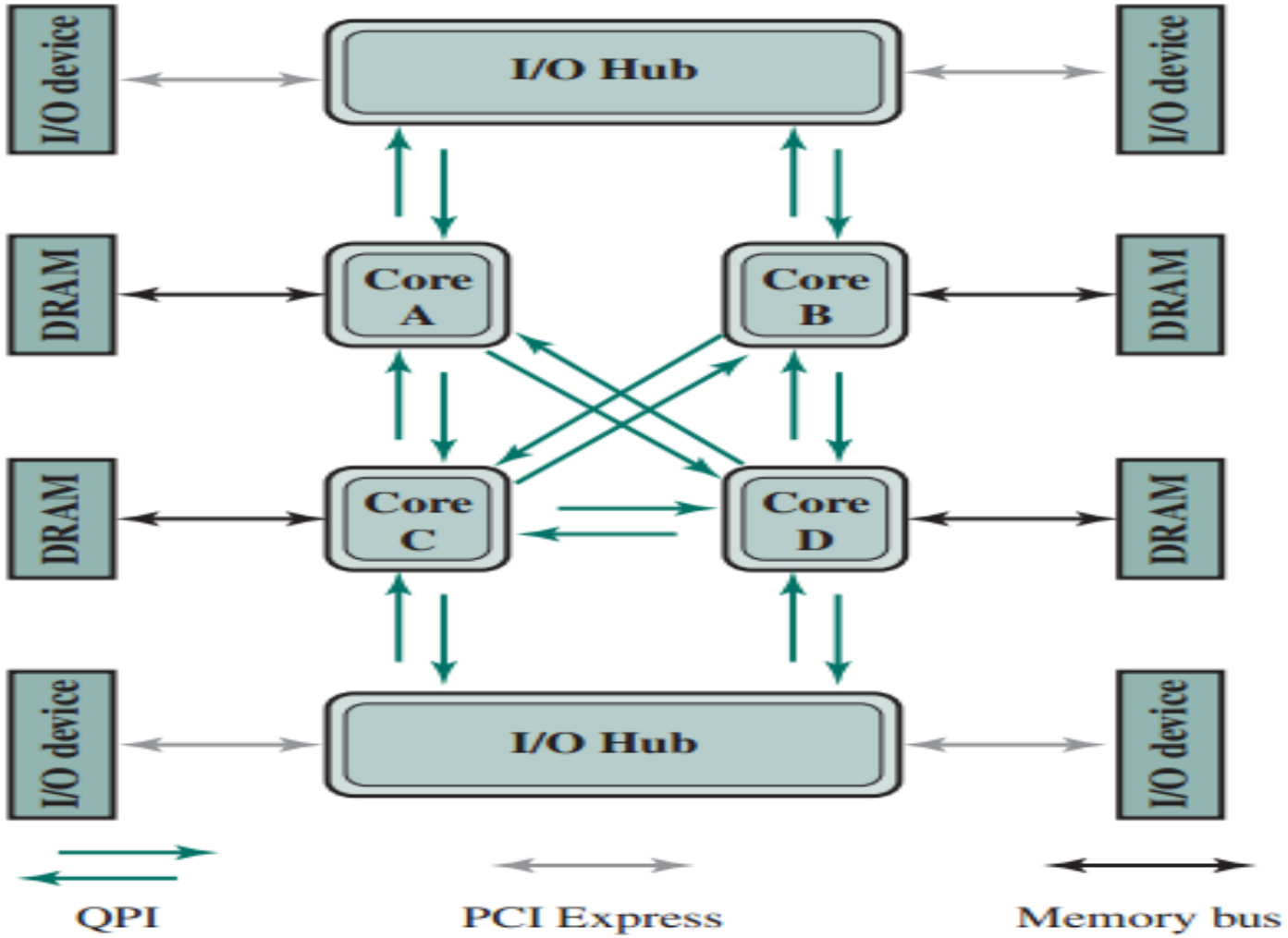
Point-to-Point Interconnection

- Bus **was** standard approach for decades – but **modern systems** rely on point-to-point interconnection
- **Main reason** – **electrical constraints** because of increasing **wider** synchronous bus day by day – **difficult** for synchronization and arbitration in time
- Things get **worse** in the advent of **multicore processor** and significant **memory on a single chip** – data rate was reduced and BUS latency was increased
- Point-to-point interconnection has lower BUS latency and higher data rate; better scalability
- Representative approach: Intel's QuickPath Interconnect (QPI)

Main Characteristics: QPI P2P Interconnection

- **Multiple Direct Connections**: Multiple components enjoy direct **pairwise** connections with other components – **eliminate arbitration**
- **Layered Protocol Architecture**: Use layered protocol architecture rather than simple use of signal
- **Packetized Data Transfer**: Data are not sent in bit stream rather sent as **a sequence of packets** – packets include **headers** and **error control codes**

QPI P2P Interconnection: in Multicore System

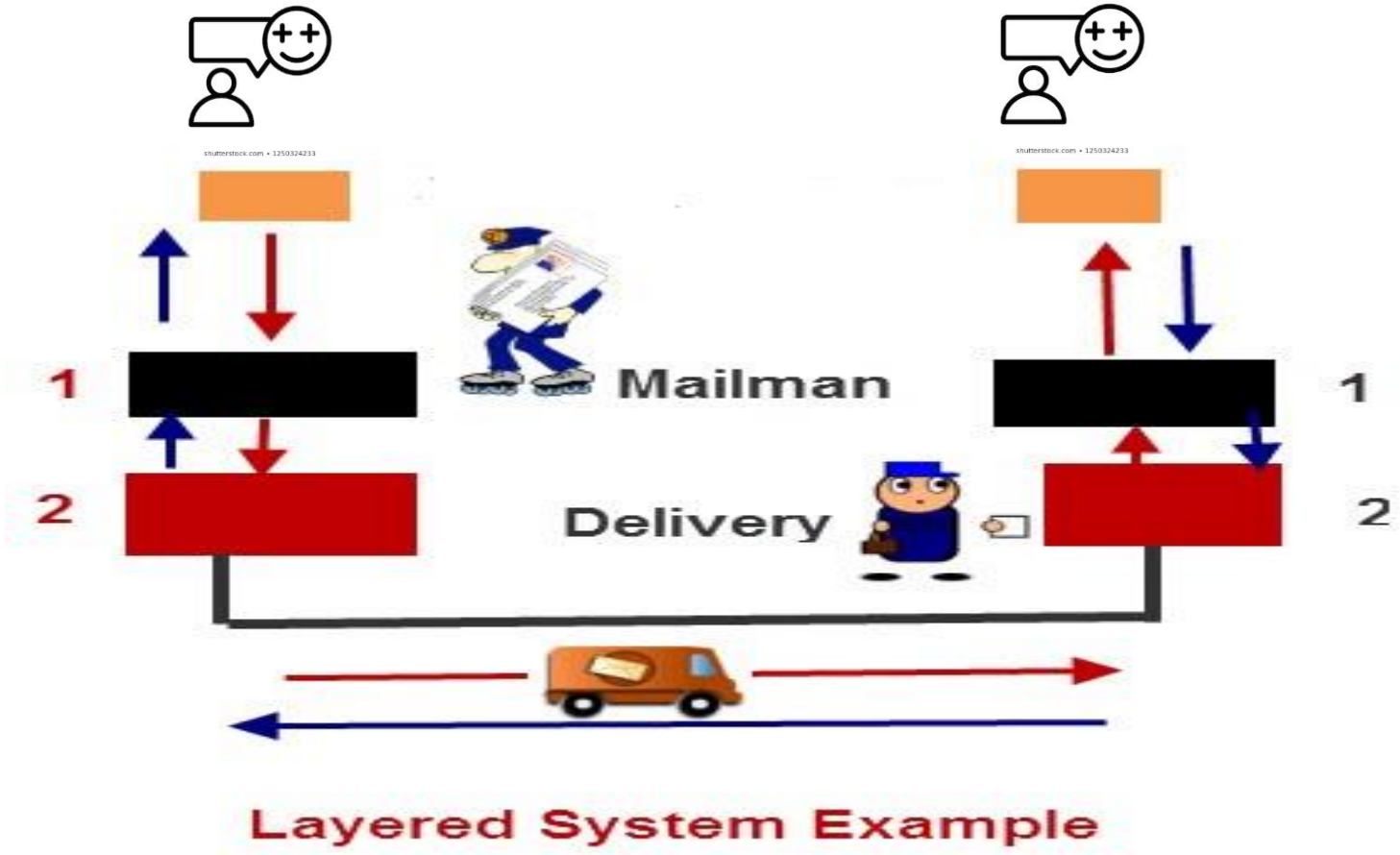


QPI P2P Interconnection in I/O Device Controller

- QPI is used also **to connect I/O module** – through **I/O Hub (IOH)** – which acts as a **switch** directing traffic to and from I/O devices
- Newer technology **PCI express** (PCIe) used **to link IOH and I/O device** controllers – IOH **translates** protocols and formats between **QPI** and **PCIe**

QPI P2P Interconnection: QPI Layers

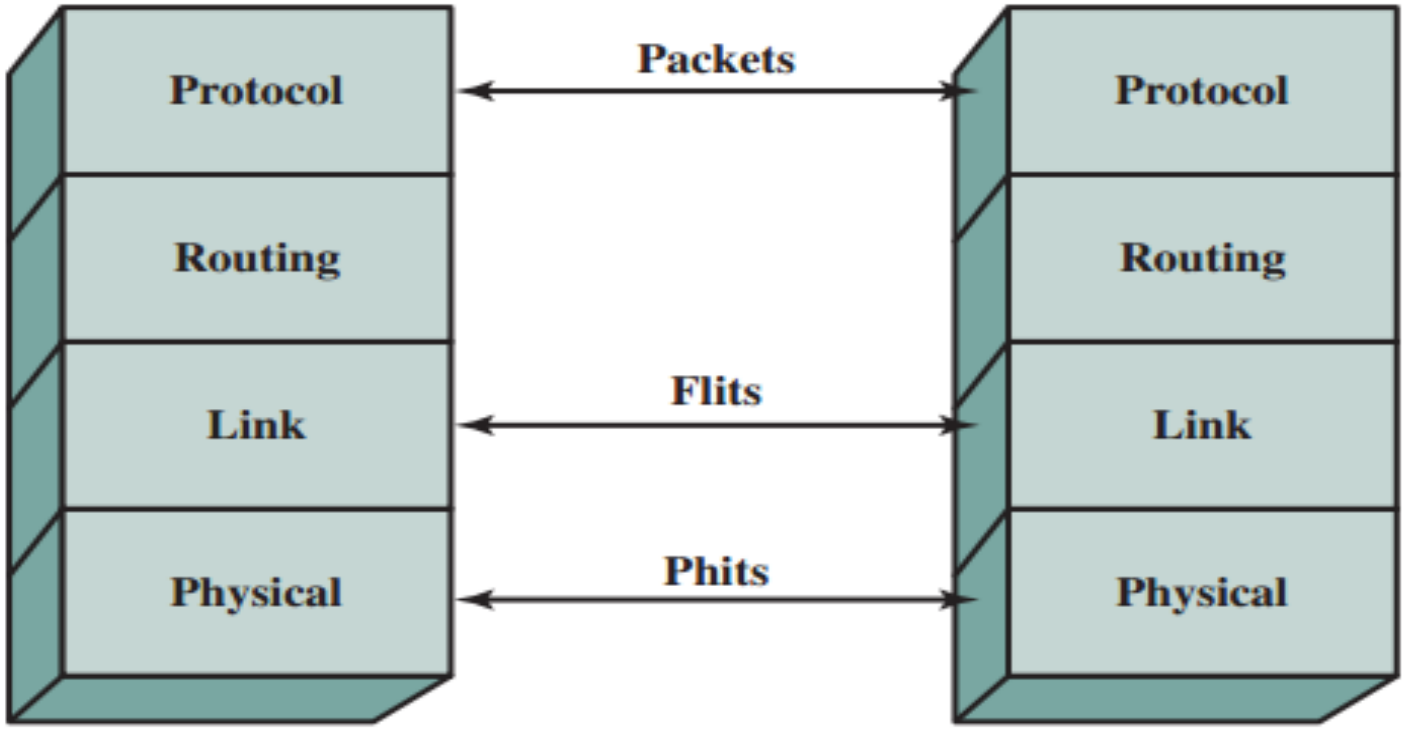
Layer???



QPI P2P Interconnection: QPI Layers…

- QPI is defined as a **four layer protocol** architecture: Like a product shipment from one place to another place
  - **Physical**: consisting **actual wires to carry signal** – also have **circuit and logic to support extra features** to transmit and receive data – unit of data transfer is 20 bits (**Phit – physical unit**)
  - **Link**: for **reliable transmission and flow control** – to avoid data loss, error, corruption – unit of data transfer is 80 bits **(Flit – flow control unit)**
  - **Routing:** provide **framework** to **direct packets through the fabric**
  - **Protocol:** A high level **sets of rule** – used for **exchanging packets of data between devices** – where a packet consists of integer number of Flits

QPI P2P Interconnection: QPI Layers…



PCI Express

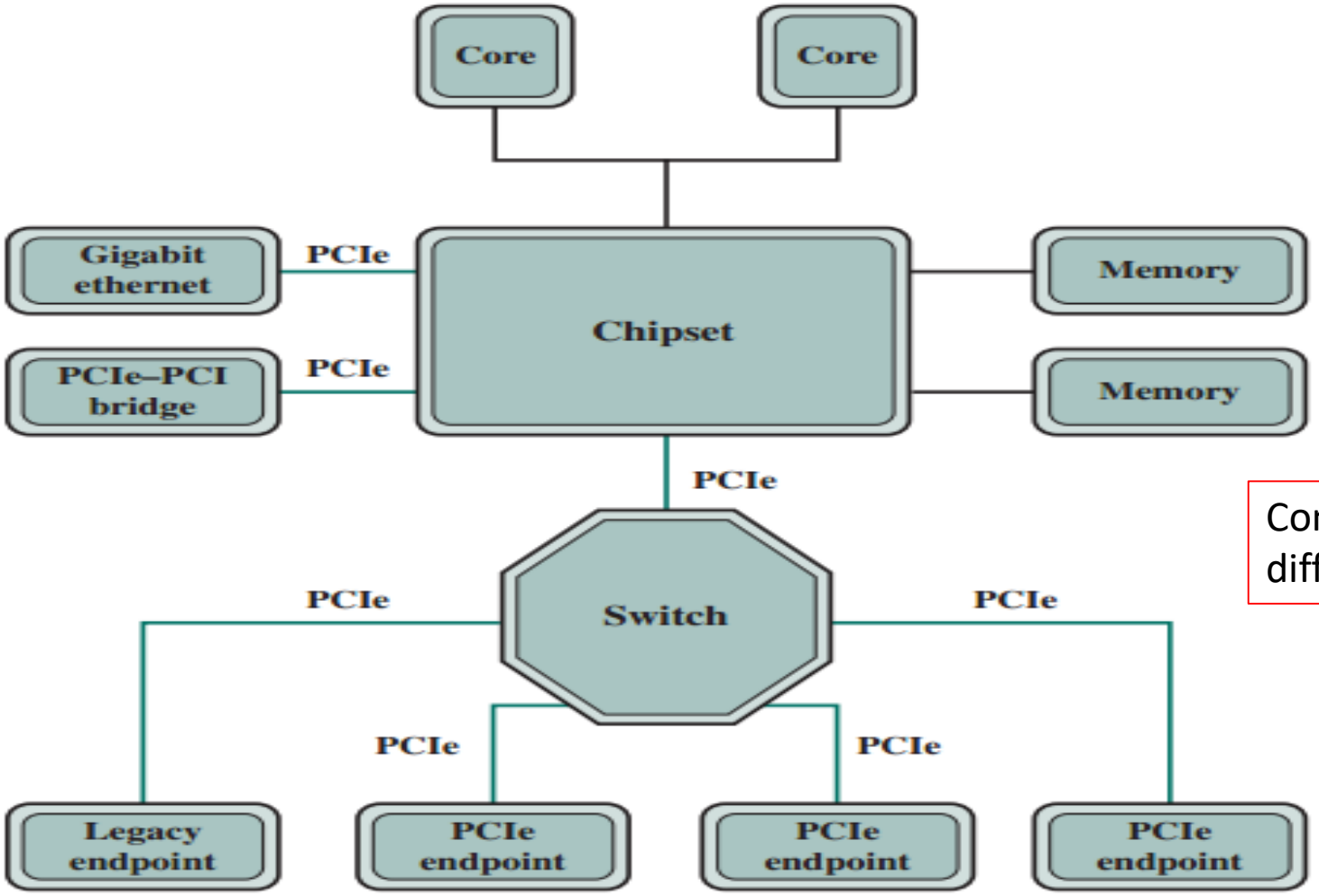
- **P**rinciple **C**omponent **I**nterconnection – **high bandwidth processor independent bus** – **used as peripheral bus** – for **I/O subsystems** (graphics adapter, network controller, disk controller) – **Intel** product
- As it is **BUS based** scheme – lower data rate (shared) – **PCI Express** (PCIe) is developed – a **point-to-point interconnection** scheme as QPI
- PCIe **ensures** supporting **high data rate** with I/O devices and **time dependent** data streaming (real time video/ audio streaming)

All data can not be treated as equal – prioritized with data tag to check which one is more/ most important

PCI Physical Architecture

- **Chipset**
  - Also named as **host bridge**
  - **Connects** processor, memory with PCIe switch fabric
  - Acts as a **buffering device** to support different data rate devices
  - A **translator** between processor, memory and I/O modules
  - Has **multiple PCIe ports** to connect PCIe device/endpoint or switch
  - May attach with –
    - **Switch:** to support multiple PCIe
    - **PCIe endpoint:** An I/O device that implements PCIe
    - **Legacy endpoint**: a device that has been migrated to PCIe and also allows legacy behavior
    - **PCI/PCIe bridge**: to allow older PCI devices

PCI Physical Architecture: Configuration

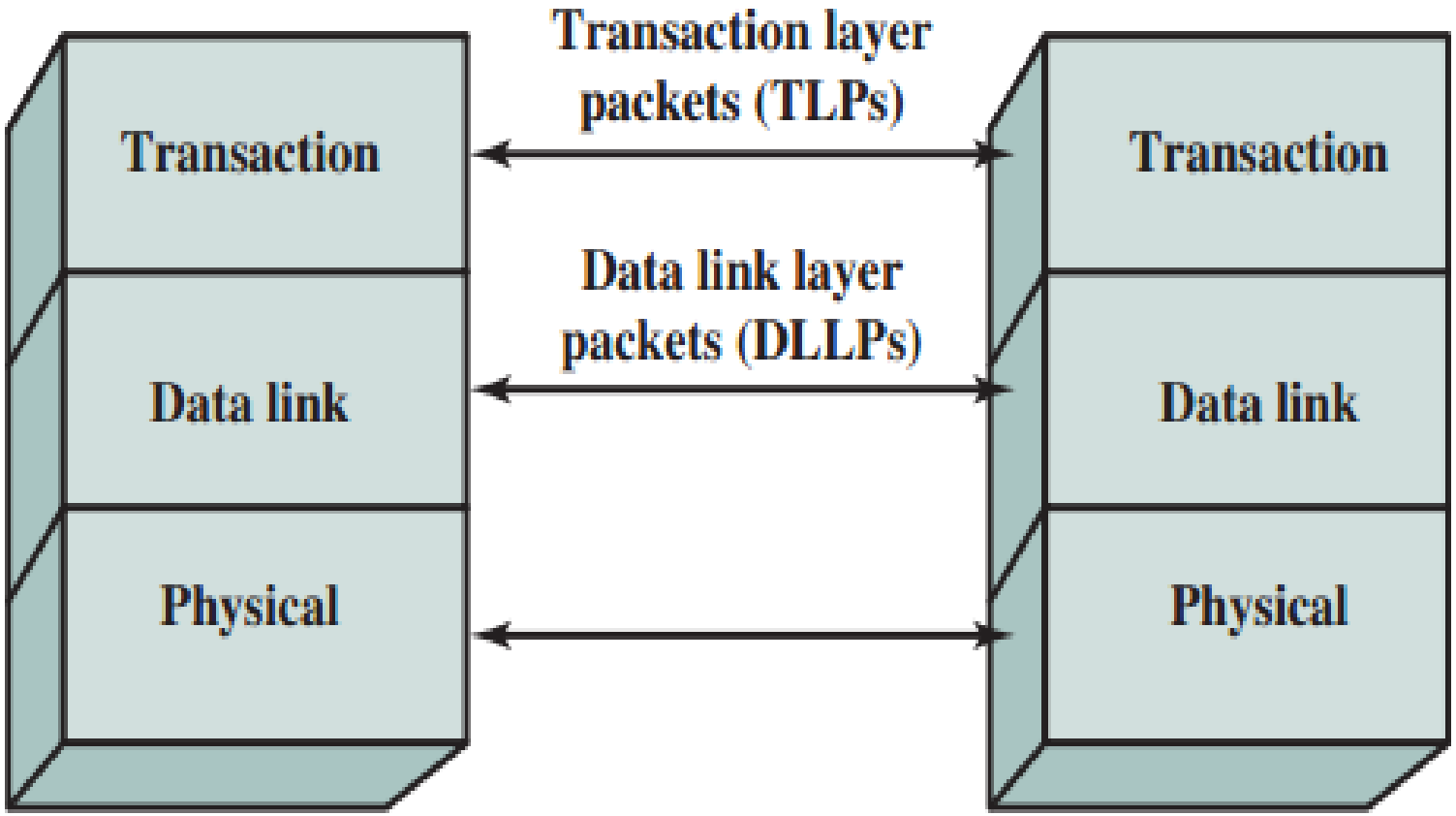


Connection established in different layers - not directly

PCI Logical Architecture

- Defined as a **protocol architecture** having following layers:
  - **Physical:** consists of **actual wires and circuitry** to carry signal and support necessary features
  - **Data Link**: responsible for **reliable transmission and flow control** – to avoid data loss, error, corruption
  - **Transaction**: **generates and consumes data packets** to implement load and store – packets are called Transaction Layer Packets (TLPs)
- Above them, there is **software layer** that generates read/ write request

PCI Logical Architecture: Protocol Layers



New Slide

- asdsad