20CS2009
Computer Architecture
Lecture #3

Interrupts :

Virtually all computers provide a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor

The interrupt is **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

Classes of Interrupts

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.

Following Diagram Shows an program execution (Flow control) with interrupt and without interrupt

Interrupt Handler:

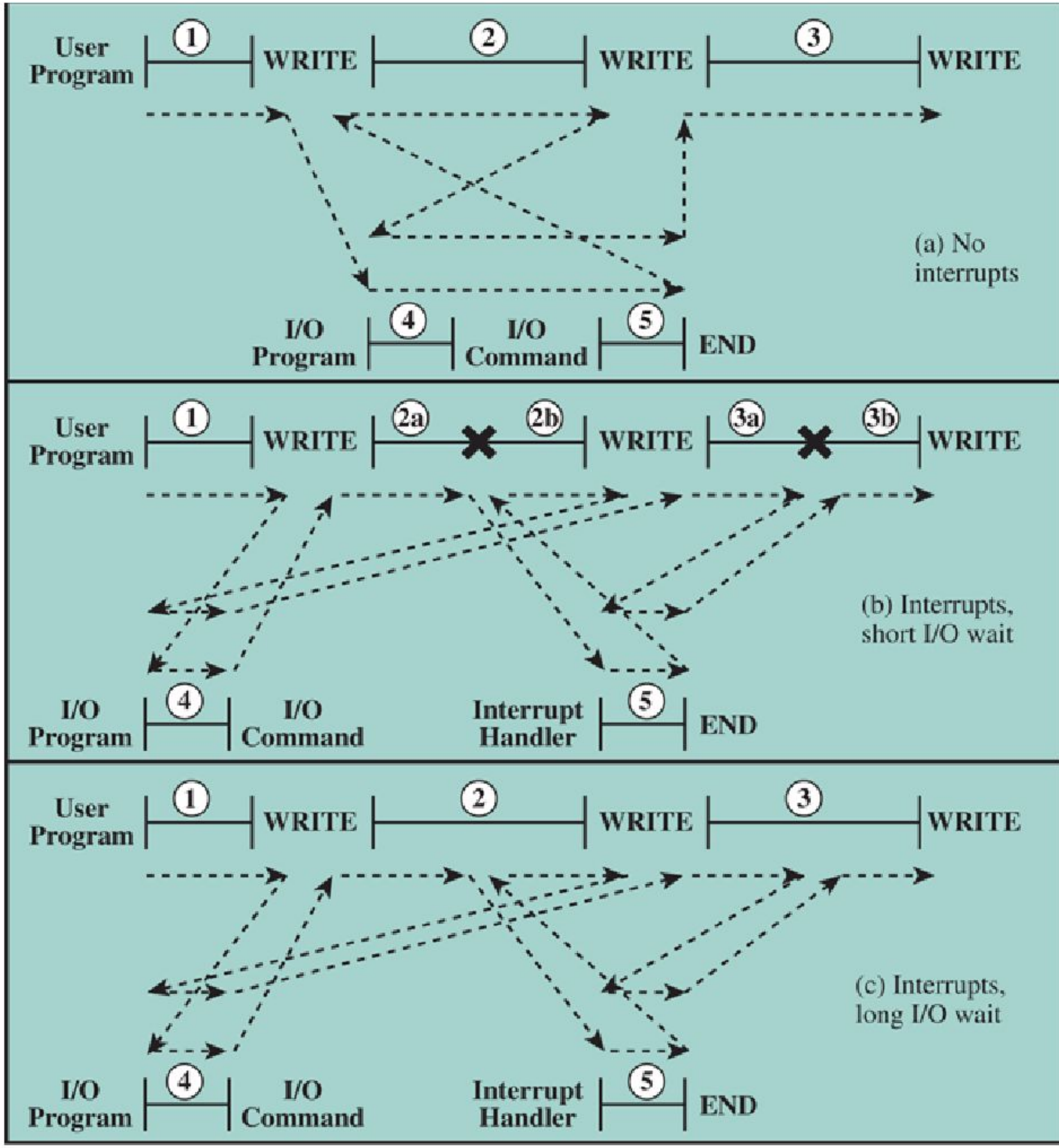
(a) No Interrupt

(b) Interrupt with Short I/O wait

(c) Interrupt with Long I/O wait

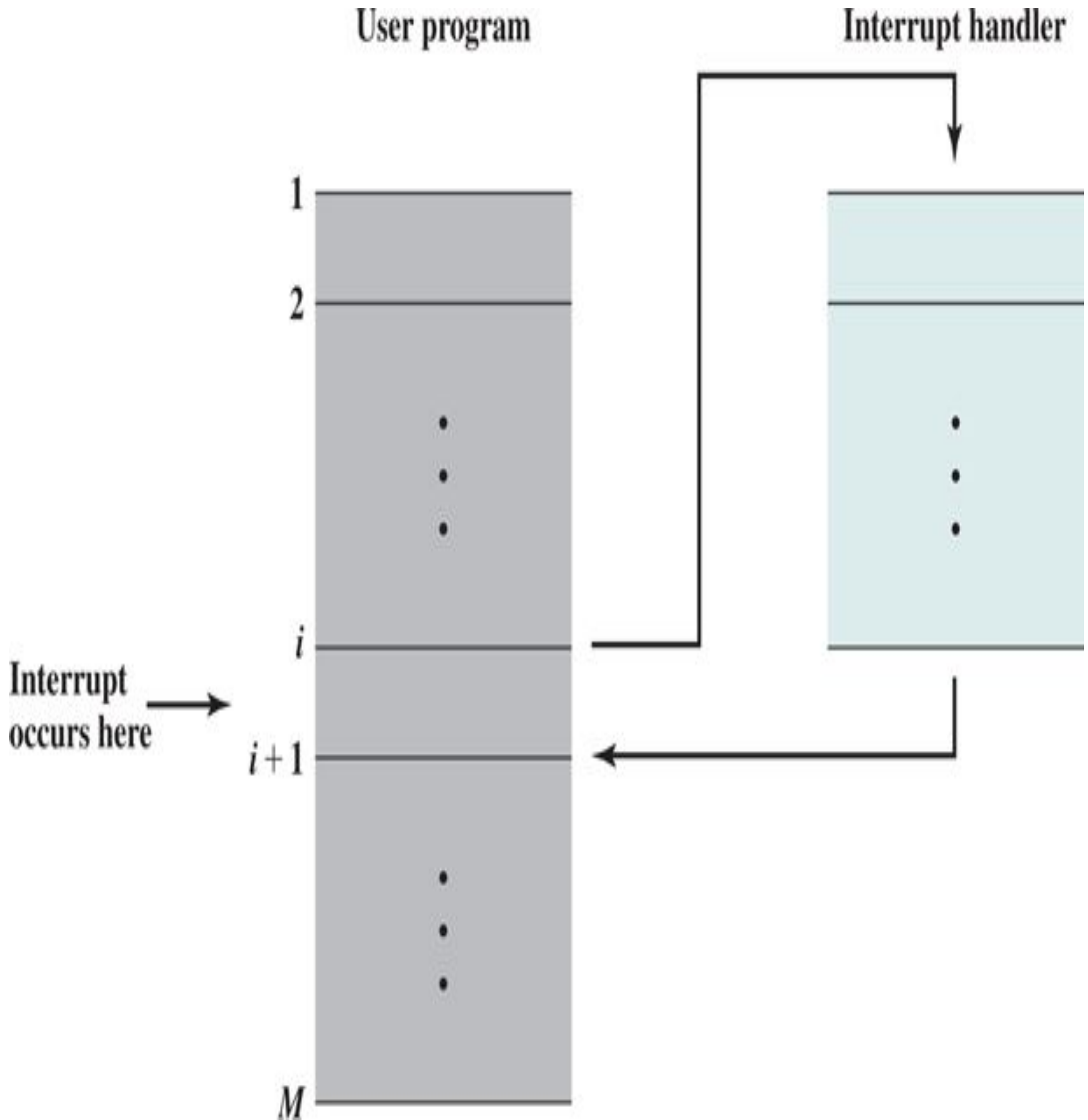
The actual I/O command.

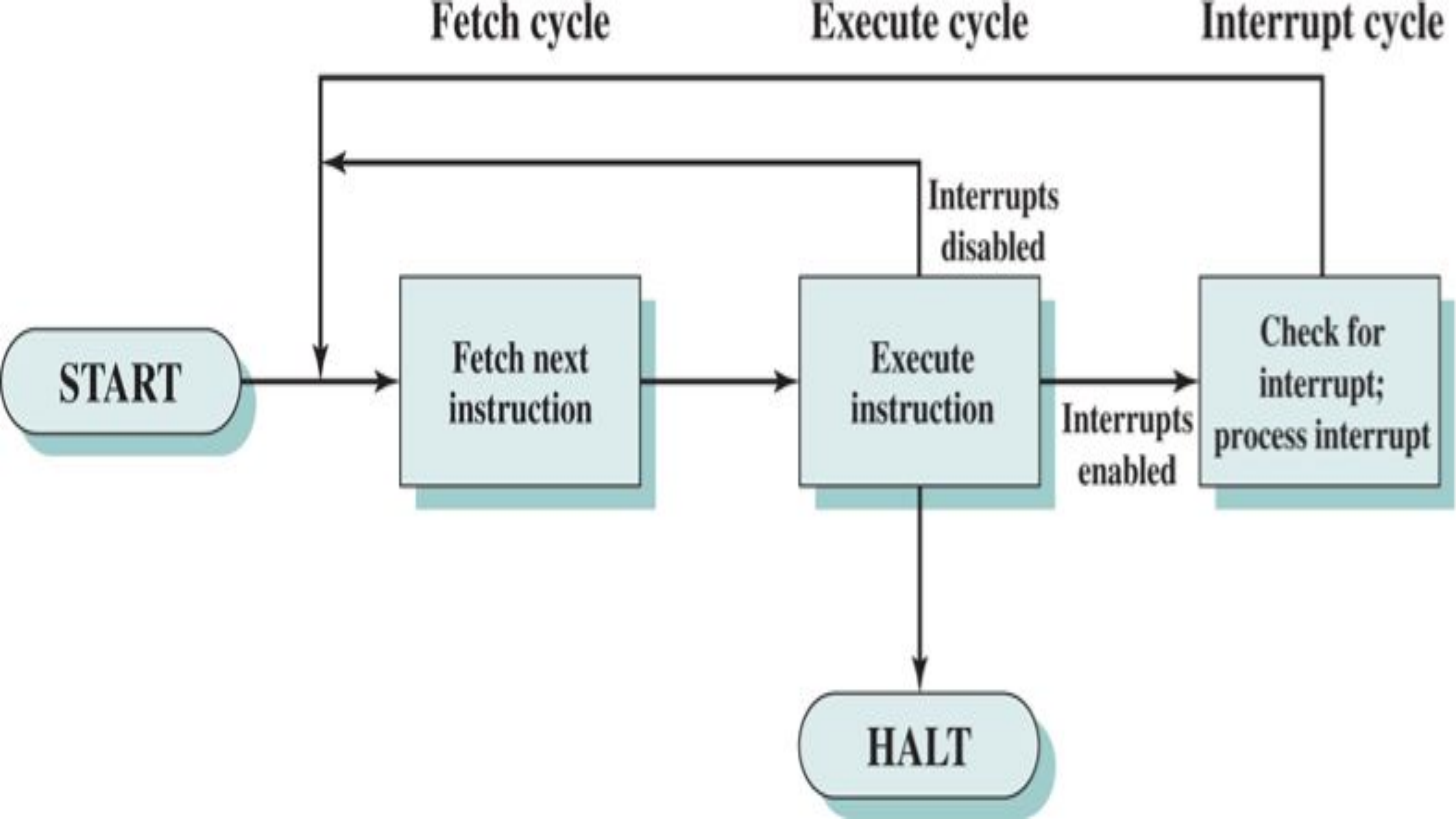
Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically poll the device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done



- *INTERRUPTS AND THE INSTRUCTION CYCLE*

- Work of a Interrupt Handler
- Current Execution may stopped
- Details will be saved in specific location
- Control will be given to interrupt handler
- After Interrupt handler – control will be given to regular program execution




Fetch cycle
Execute cycle
Interrupt cycle
Interrupts disabled
START
Fetch next instruction
Execute instruction
Check for interrupt; process interrupt
Interrupts enabled
HALT

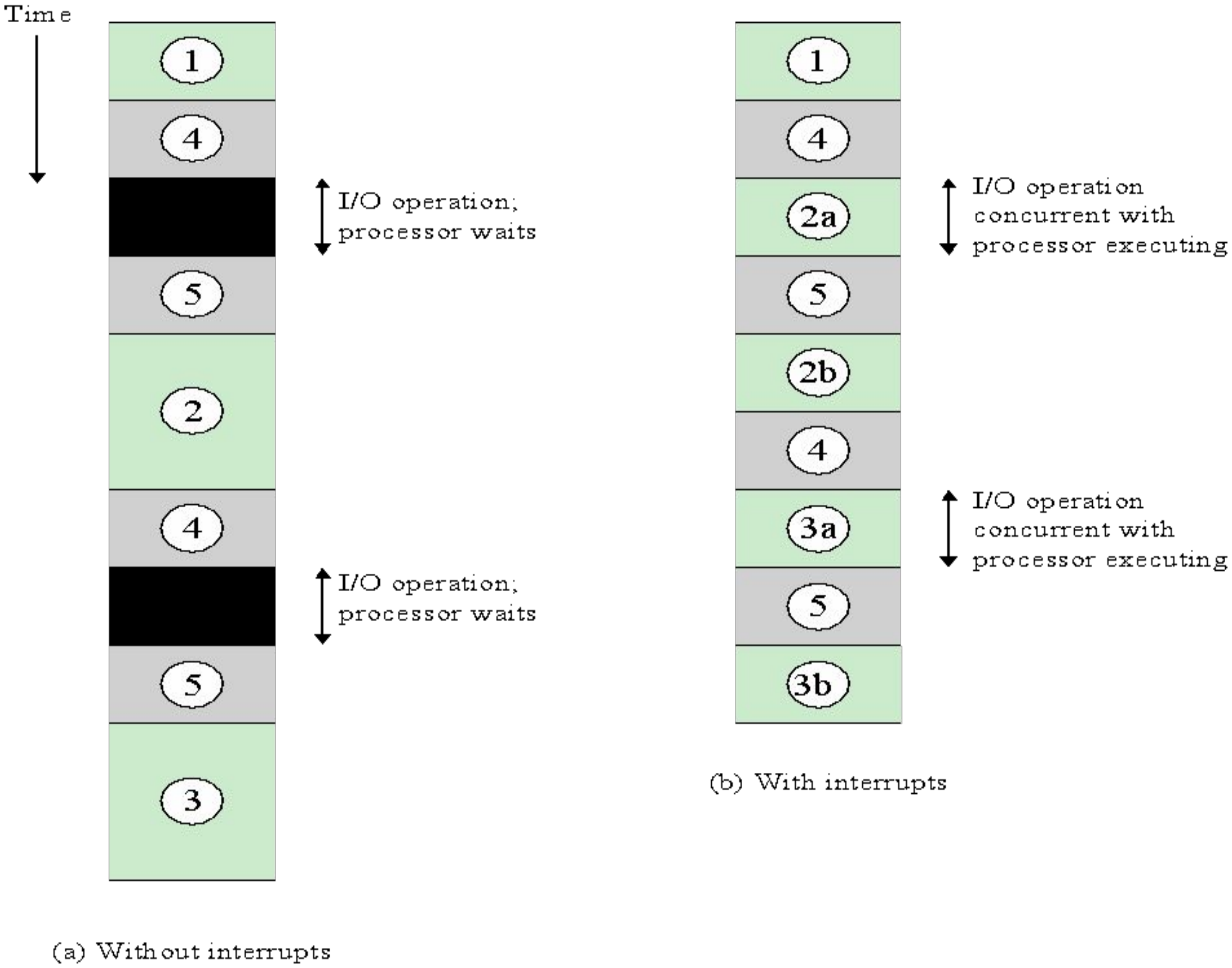

Time
1
4
I/O operation; processor waits
5
2
4
I/O operation; processor waits
5
3
(a) Without interrupts
1
4
2a
I/O operation concurrent with processor executing
5
2b
4
3a
I/O operation concurrent with processor executing
5
3b
(b) With interrupts

Figure 3.10 Program Timing: Short I/O Wait

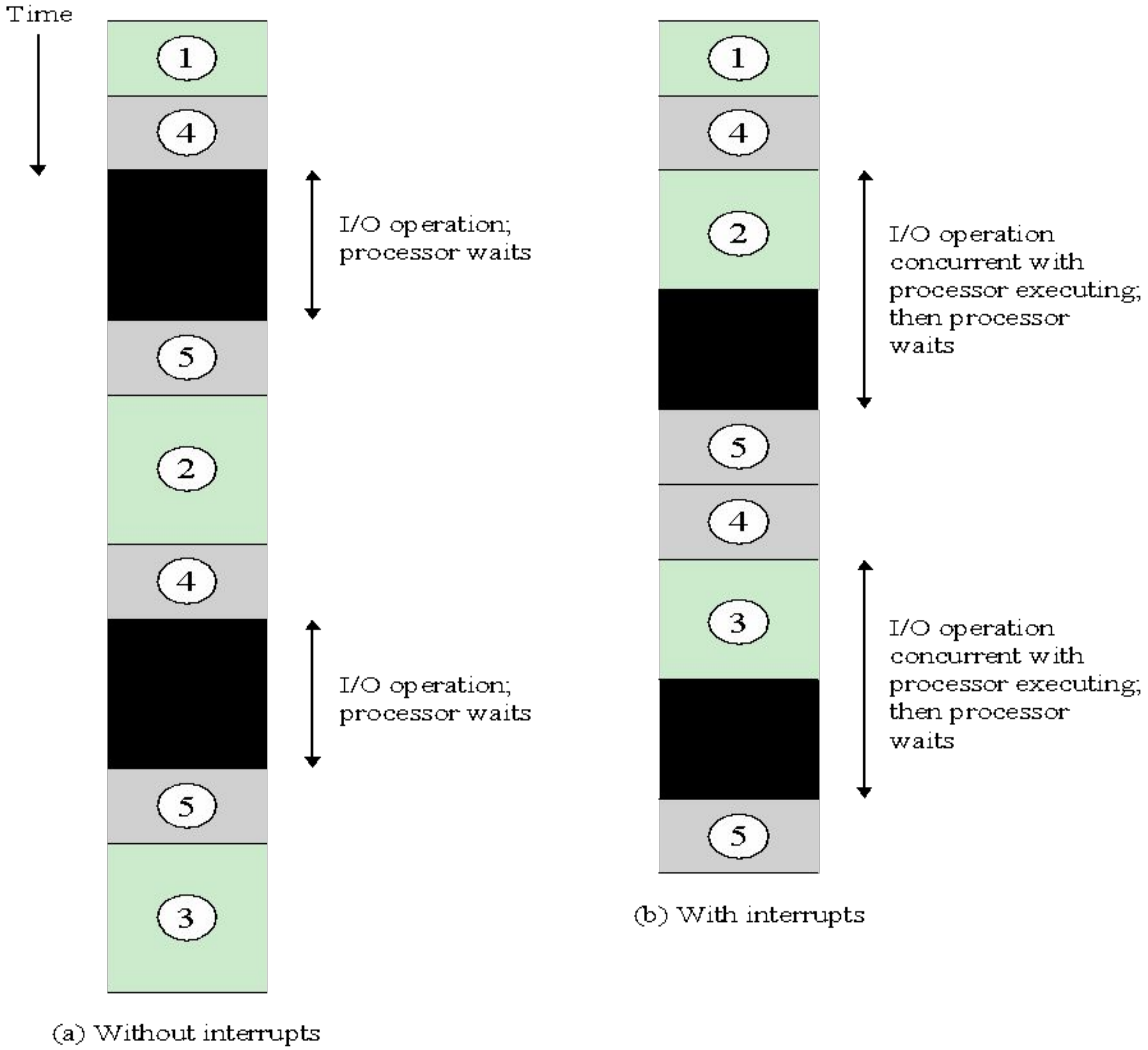

Time
1
4
I/O operation; processor waits
5
2
4
I/O operation; processor waits
5
3
(a) Without interrupts
1
4
2
I/O operation concurrent with processor executing; then processor waits
5
4
3
I/O operation concurrent with processor executing; then processor waits
5
(b) With interrupts

Figure 3.11 Program Timing: Long I/O Wait

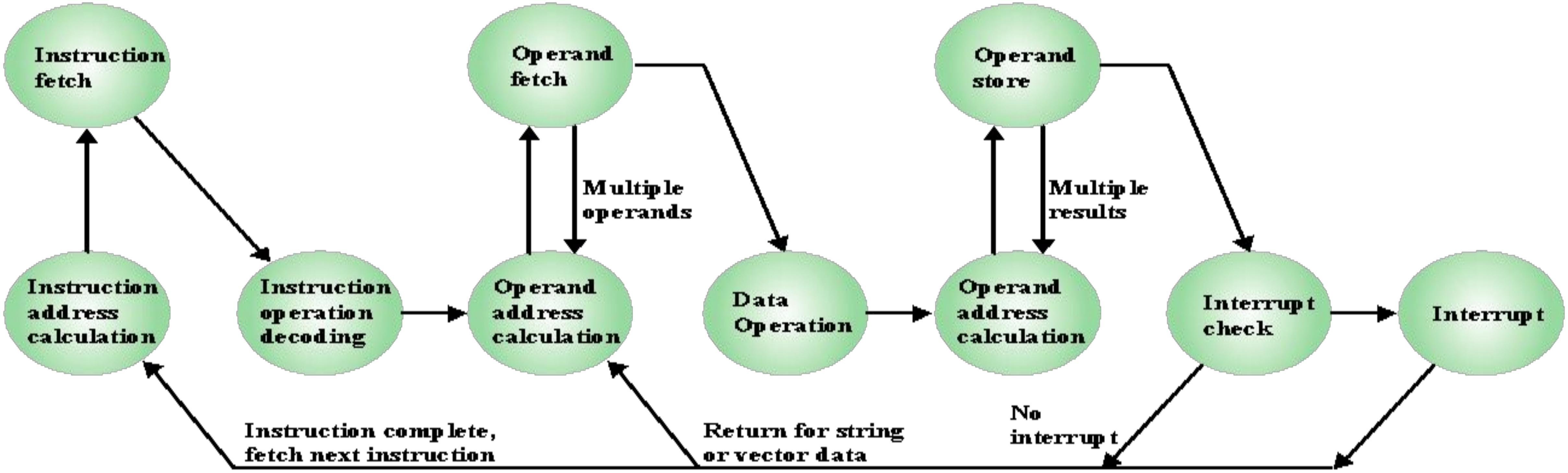
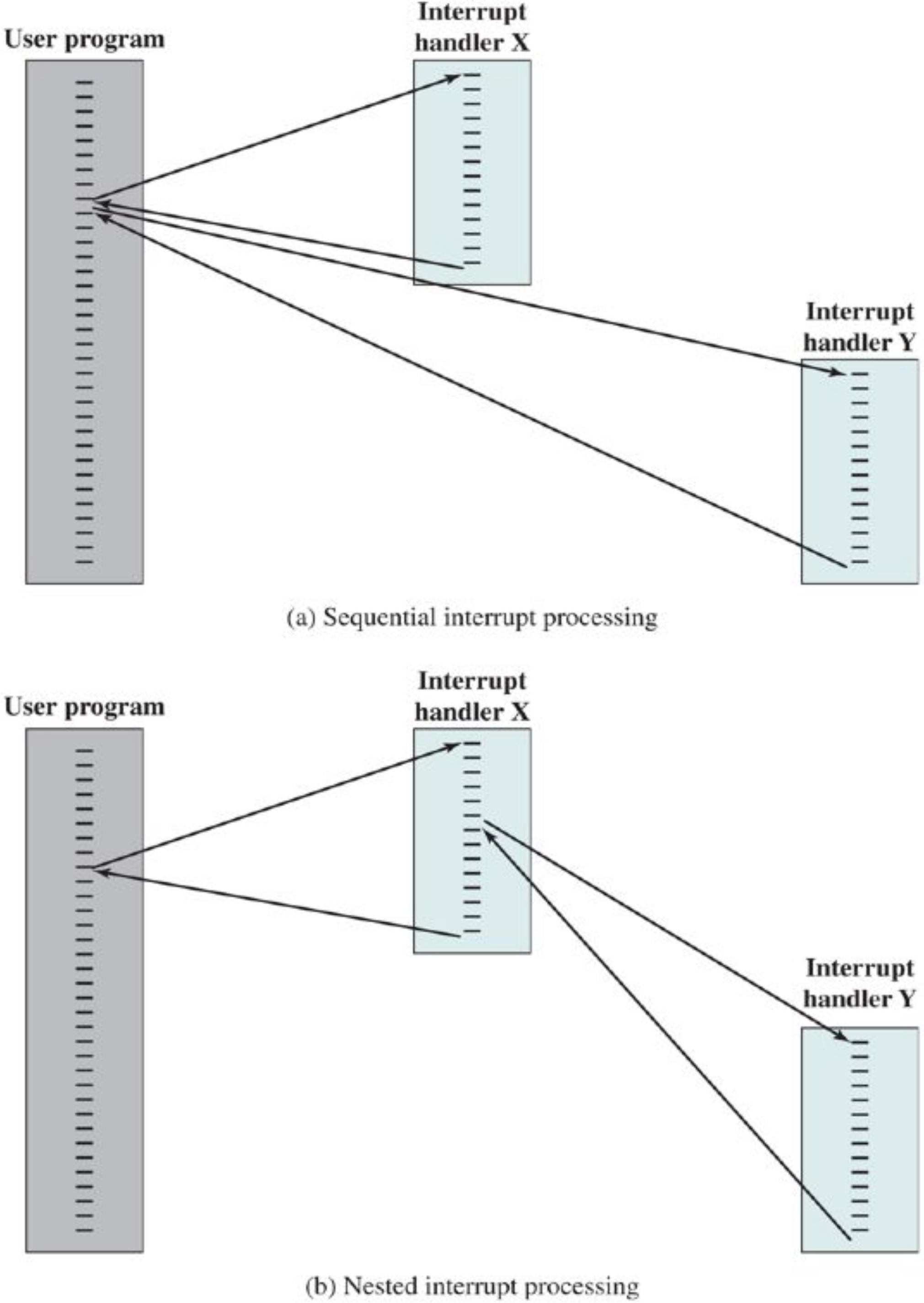

Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
No interrupt
Instruction complete, fetch next instruction
Return for string or vector data

Figure 3.12 Instruction Cycle State Diagram, With Interrupts

MULTIPLE INTERRUPTS:

- Multiple interrupts can occur. For example, a program may be receiving data from a communications line and printing results. The printer will generate an interrupt every time it completes a print operation. The communication line controller will generate an interrupt every time a unit of data arrives. The unit could either be a single character or a block, depending on the nature of the communications discipline. In any case, it is possible for a communications interrupt to occur while a printer interrupt is being processed.



(a) Sequential interrupt processing

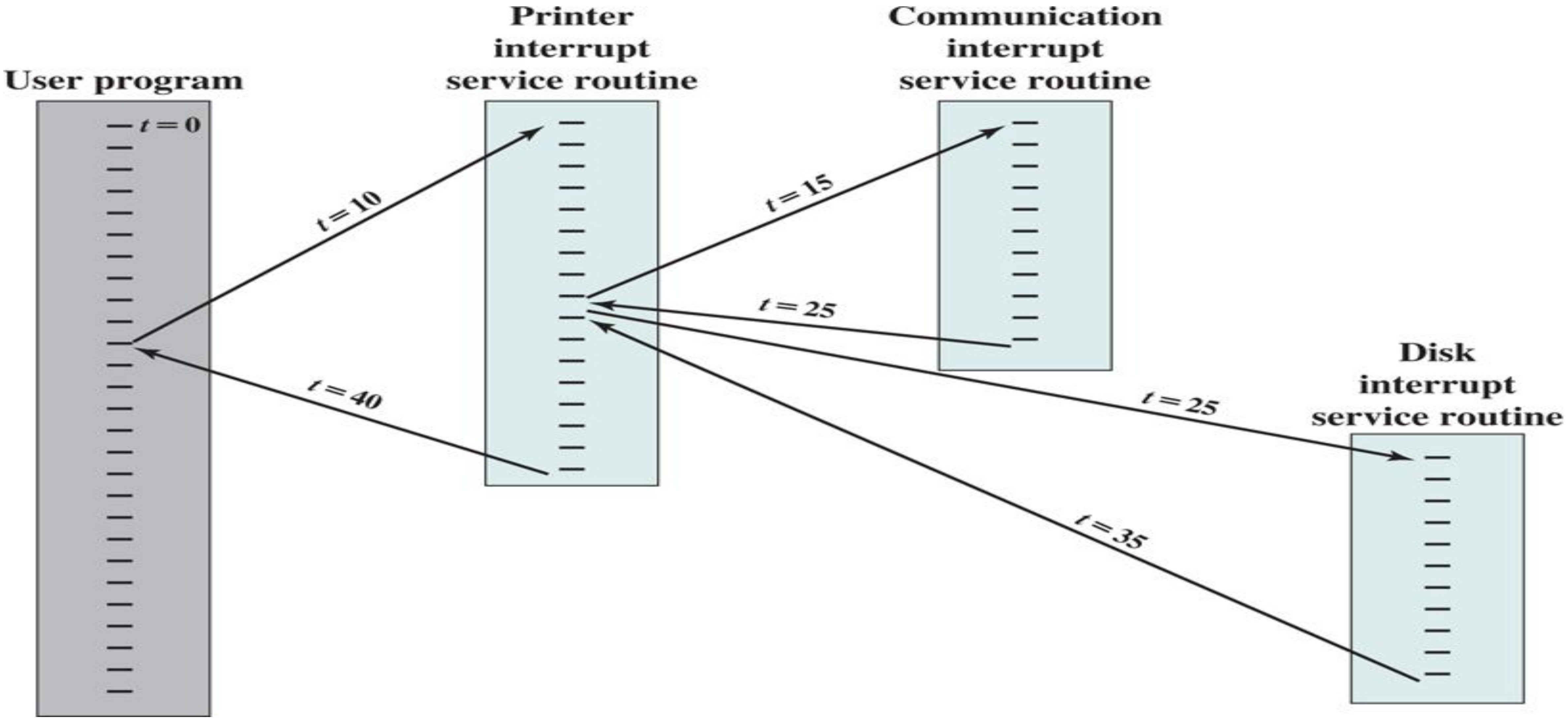
(b) Nested interrupt processing

Two approaches can be taken to dealing with multiple interrupts.

- The **first** is to disable interrupts while an interrupt is being processed. A **disabled interrupt** simply means that the processor can and will ignore that interrupt request signal. If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has enabled interrupts. Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately. After the interrupt handler routine completes, interrupts are enabled before resuming the user program, and the processor checks to see if additional interrupts have occurred.

- A **second approach** is to **define priorities** for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to itself be interrupted

Interrupt Service Routine (ISR).


User program
t = 0
Printer interrupt service routine
Communication interrupt service routine
Disk interrupt service routine
t = 10
t = 15
t = 25
t = 25
t = 35
t = 40

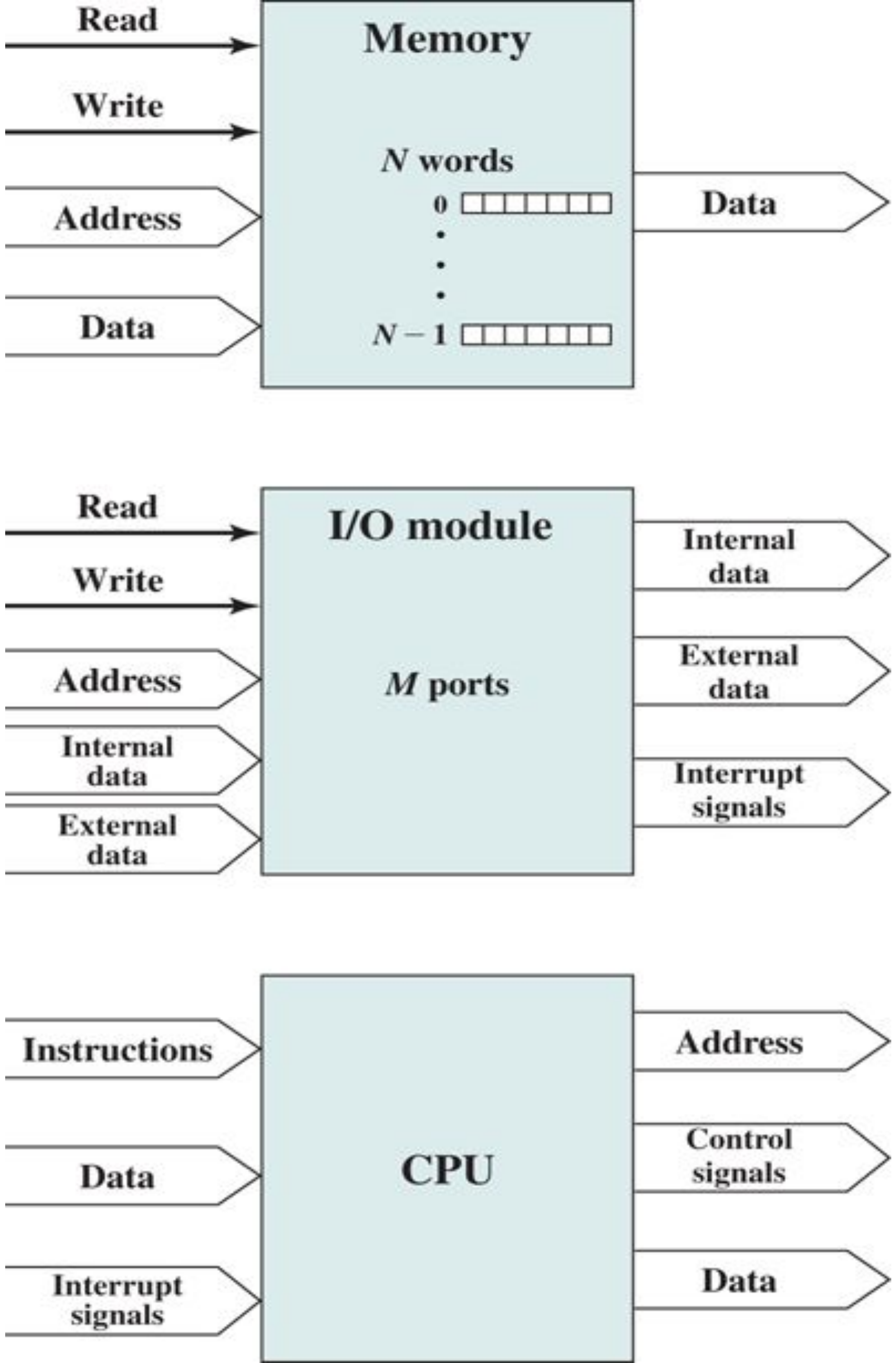
Interconnection Structures:

A computer consists of a **set of components** or modules of three basic types **(processor, memory, I/O)** that communicate with each other.

This diagram shows the types of exchanges that are needed by indicating the major forms of input and output for each module type

The **wide arrows represent multiple signal lines** carrying multiple bits of information in parallel.

Each **narrow arrow represents a single signal line.**

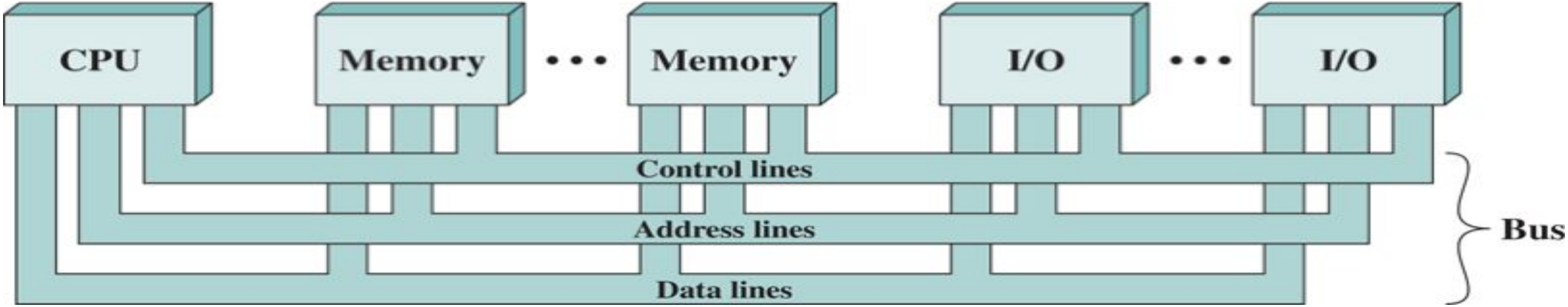


The **interconnection structure** **must support** the following types of transfers:

- **Memory to processor:** The processor reads an instruction or a unit of data from memory.
- **Processor to memory:** The processor writes a unit of data to memory.
- **I/O to processor:** The processor reads data from an I/O device via an I/O module.
- **Processor to I/O:** The processor sends data to the I/O device.
- **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access.

Bus Interconnection :

- The **bus was the dominant** means of computer system component interconnection for decades. For general-purpose computers, it has gradually **given way to various point-to-point interconnection structures**, which now dominate computer system design. However, bus structures are still commonly used for embedded systems, particularly microcontrollers.
- A bus is a communication pathway **connecting two or more devices**.
- A key characteristic of a bus is that it is a **shared transmission medium**.
- **Multiple devices** connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, **only one device at a time can successfully transmit**.



- A bus that connects major computer components (processor, memory, I/O) is called a **system bus**
- In any bus the lines can be classified into three functional groups : **data, address, and control lines.**
- The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the **data bus** . The data bus may consist of 8,16,32, 64, 128, or even more separate lines, the number of lines being referred to as the *width* of the data bus. Because each line can carry only one bit at a time, the number of lines determines how many bits can be transferred at a time.

- The **address lines** are used to designate the source or destination of the data on the data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the **address bus** determines the **maximum possible memory capacity of the system.**

- The **control lines** are used to **control the access** to and the **use of the data and address lines.** Because the data and address lines are shared by all components, there must be a means of controlling their use. **Control signals transmit both command and timing information among system modules. Timing signals** indicate the validity of data and address information. **Command signals** specify operations to be performed.

Typical control lines include:

- **Memory write:** causes data on the bus to be written into the addressed location.
- **Memory read:** causes data from the addressed location to be placed on the bus.
- **I/O write:** causes data on the bus to be output to the addressed I/O port.
- **I/O read:** causes data from the addressed I/O port to be placed on the bus.
- **Transfer ACK:** indicates that data have been accepted from or placed on the bus.
- **Bus request:** indicates that a module needs to gain control of the bus.
- **Bus grant:** indicates that a requesting module has been granted control of the bus.
- **Interrupt request:** indicates that an interrupt is pending.
- **Interrupt ACK:** acknowledges that the pending interrupt has been recognized.
- **Clock:** is used to synchronize operations.
- **Reset:** initializes all modules.