Introduction

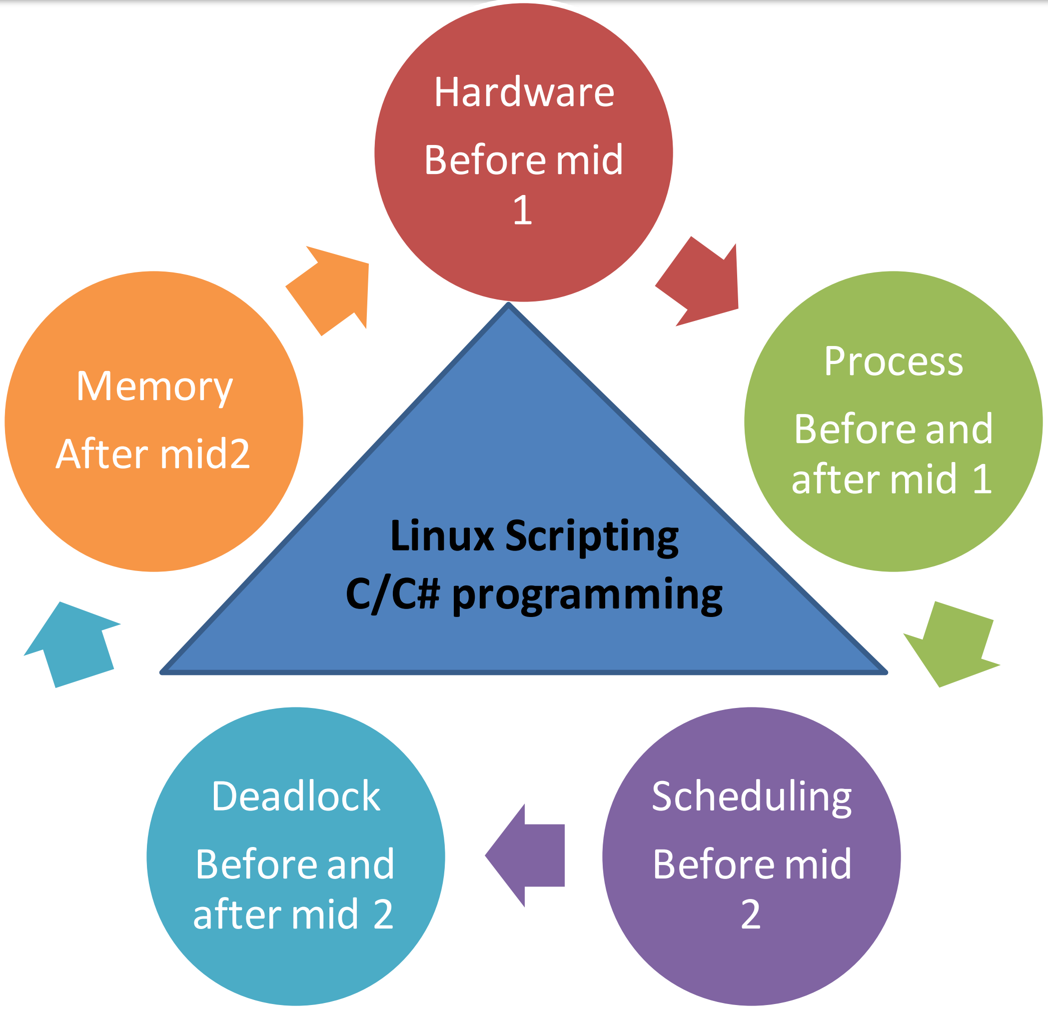
Operating Systems (CS-220)

Fall 2020, FAST NUCES

COURSE SUPERVISOR: ANAUM HAMID

anaum.hamid@nu.edu.pk

Course Outline



Evaluation Instruments and Marks Distributions

- Quizzes = 3 (5 marks – Best 2)
- Assignments = 1 (5 marks)
- Project = 1 (10 marks)
- Midterm = 30 marks (15 for each)
- Final Term = 50 marks
- Total = 100 Marks

Course Recourses

TEXTBOOK:

- Operating Systems Concepts, 9th edition, by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne.

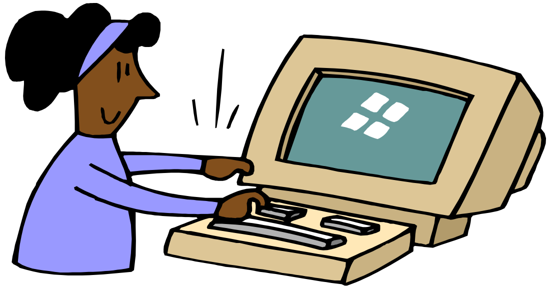
REFERENCE BOOKS:

- Operating Systems – Internals and Design Principles, 8th edition, by William Stallings.

Internals and Design Principles

What we learn this week.

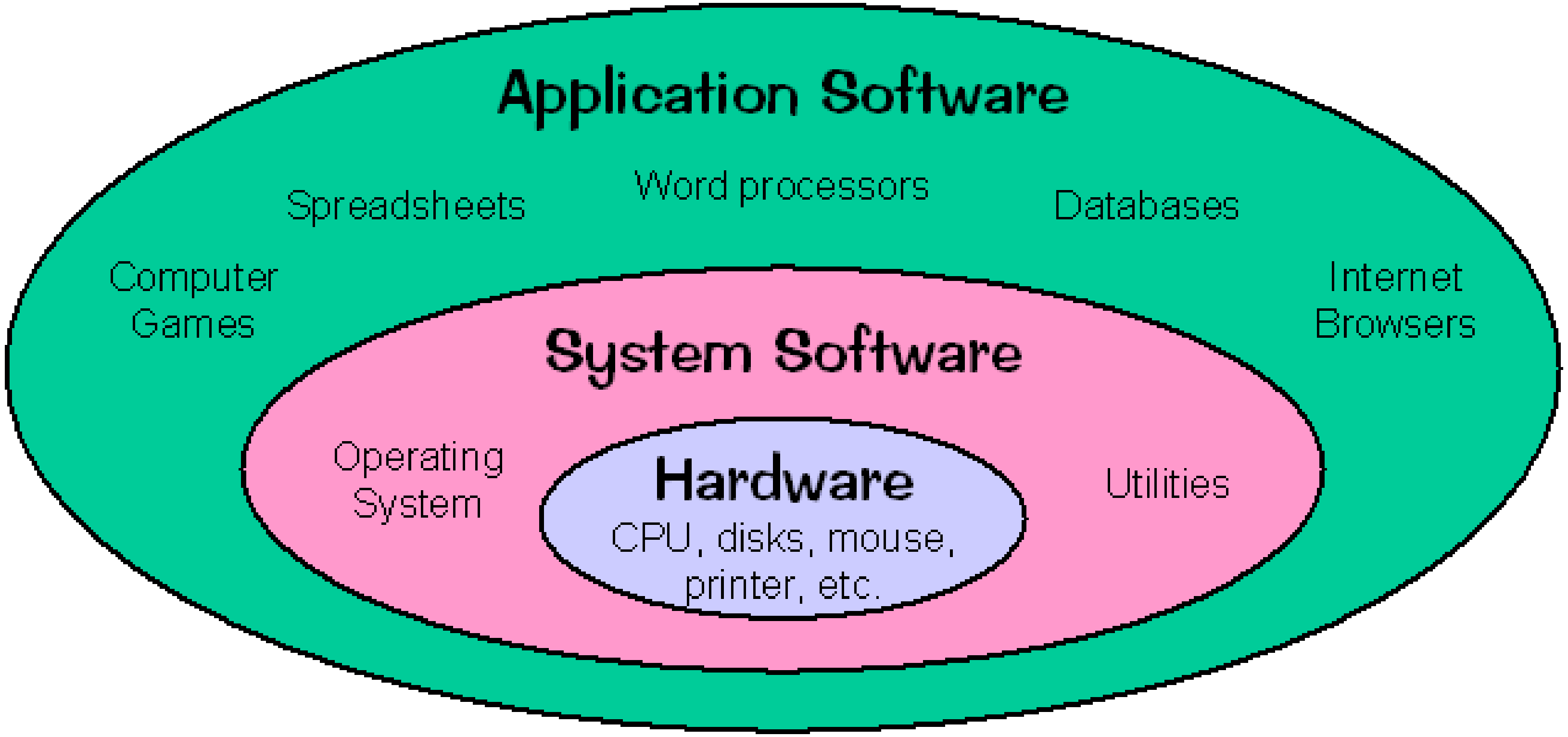
- ✓ What is an Operating System?
- ✓ Basic Elements.
- ✓ Evolution of Microprocessors.
- ✓ Instruction Execution.
- ✓ Interrupt
- ✓ I/O
- ✓ Memory

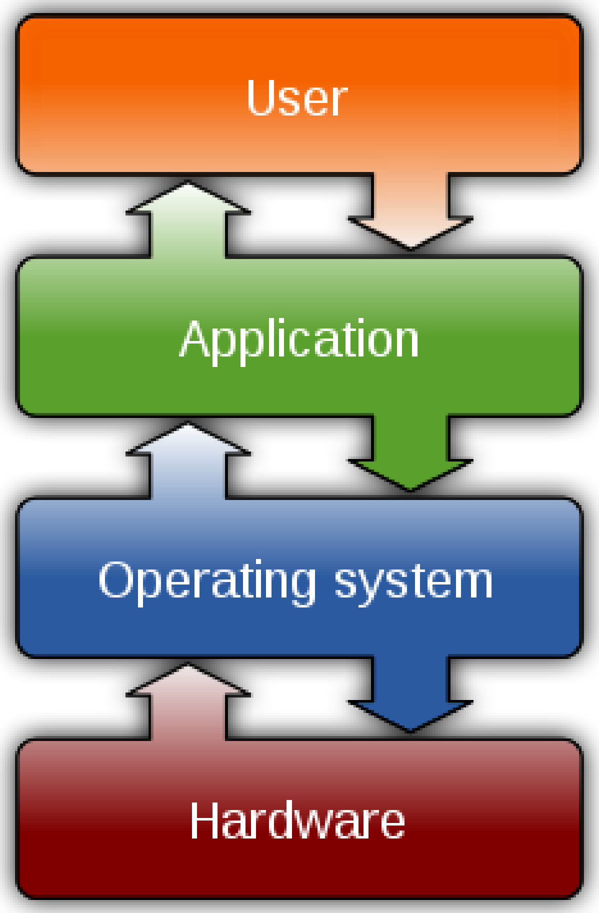


Operating System: Overview

- **Operating system (OS):** a **system software** that exploits the **hardware resources** to provide a set of services to system users.
- The OS manages the processor(s), memory and input/output (I/O) devices on behalf of its users.
- ***Note:*** it is important to have some fundamental understanding of basic data structures, computer organization and a high-level programming language, such as C or Java, before examining topics related to operating systems.

Operating System: Overview (Cont.)



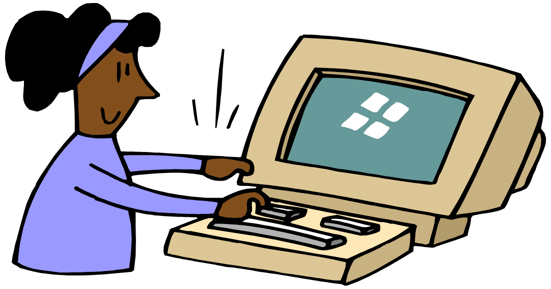


Relationship between application software and system software

Basic Elements

Processor

I/O Modules



Main Memory

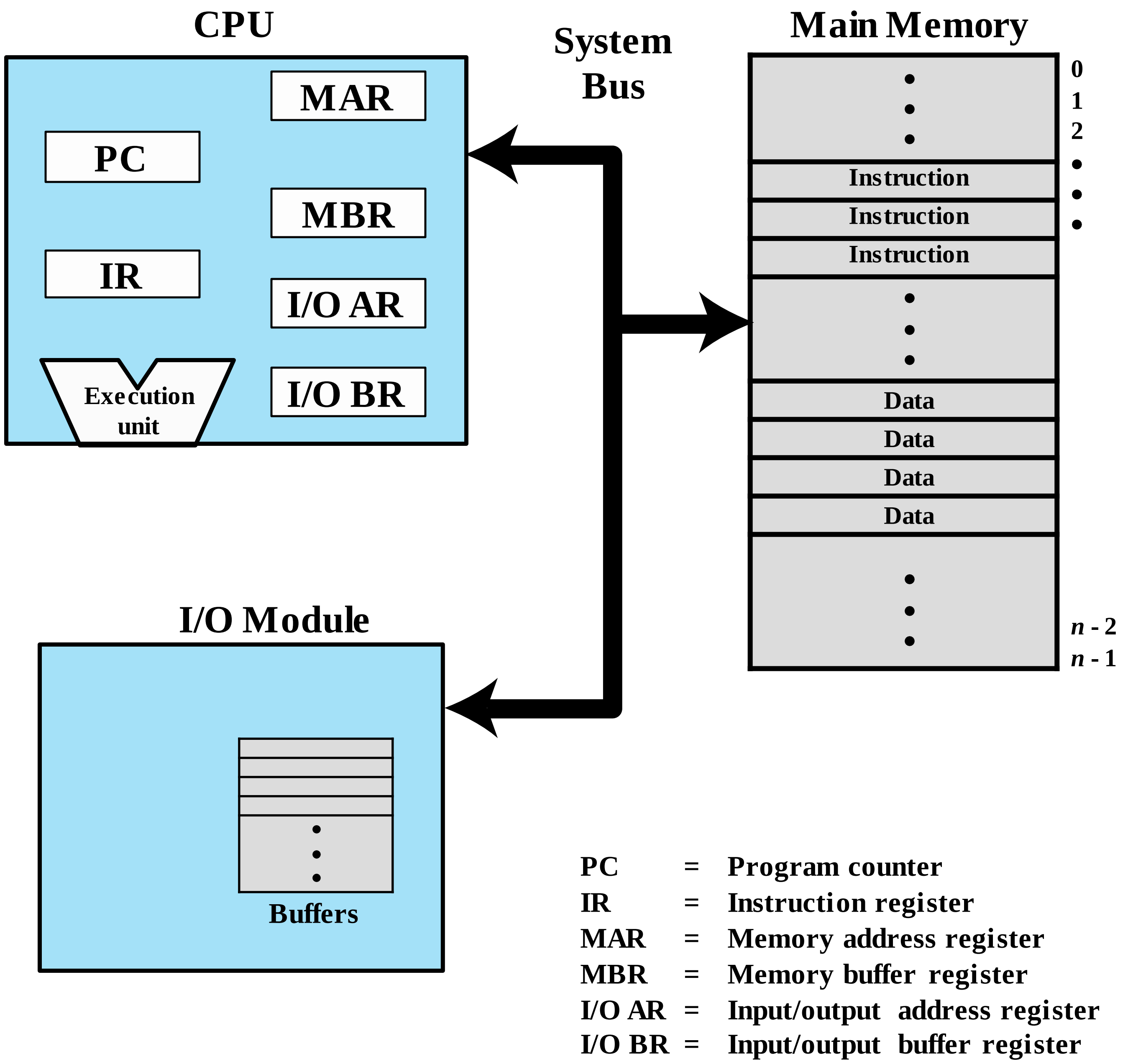
System Bus

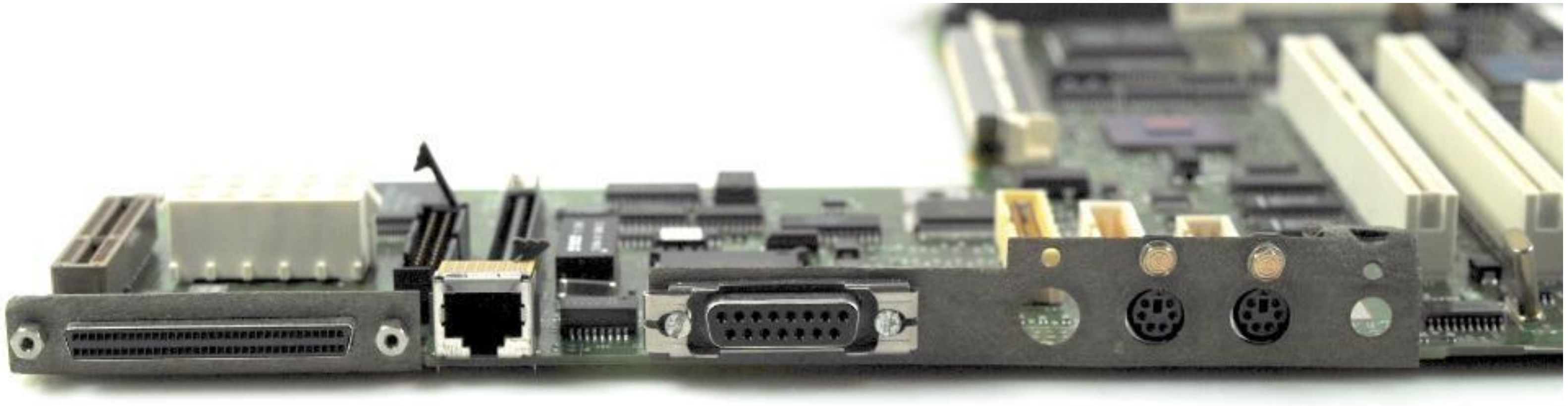
Basic Elements of Computer System

In a computer system, there are four main structural elements:

1. **Processor:** controls operation of the computer and performs its data processing functions.
2. **Main memory (primary memory):** stores data and programs.
   - Main memory is typically volatile.
   - Disk memory is nonvolatile.
3. **I/O modules:** Move data between the computer and its **external environment**, e.g. storage (harddisk).
4. **System bus:** Provides communication among processors, main memory and I/O modules.

Computer Components top-level view






intel


WD1000

Microprocessor

- Invention that brought about desktop and handheld computing
- Processor on a single chip
- Fastest general-purpose processor
- Multiprocessors
- Each chip (socket) contains multiple processors (cores)

Graphical Processing Units (GPUs)

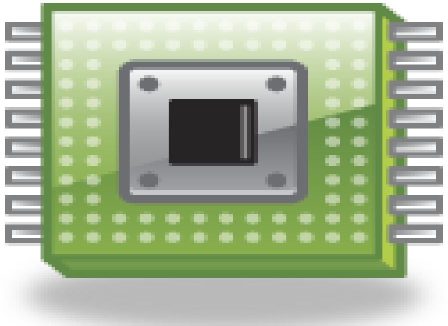
- –Provide efficient computation on arrays of data using Single-Instruction Multiple Data (SIMD) techniques
- –Used for general numerical processing
- –Physics simulations for games
- –Computations on large spreadsheets

Digital Signal Processors (DSPs)

- Deal with streaming signals such as audio or video
- Used to be embedded in devices like modems
- Encoding/decoding speech and video (codecs)
- Support for encryption and security

System on a Chip (SoC)

- To satisfy the requirements of handheld devices, the microprocessor is giving way to the SoC
- Components such as DSPs, GPUs, codecs and main memory, in addition to the CPUs and caches, are on the same chip



Instruction Execution

- A **program** consists of a **"set of instructions"** that are stored in **memory** before being executed by a **processor**.
- Instruction processing consists of two steps ***(simplest form)***:
  a. The **processor** reads **(fetches)** instructions from **memory** one at a time.
  b. The **processor** executes each instruction.
- Program execution consists of repeating the process of instruction **fetch** and instruction **execution**.

Instruction Execution (Cont.)

- **Instruction cycle:** the processing required for a "single instruction", depicted below by a two-step description.

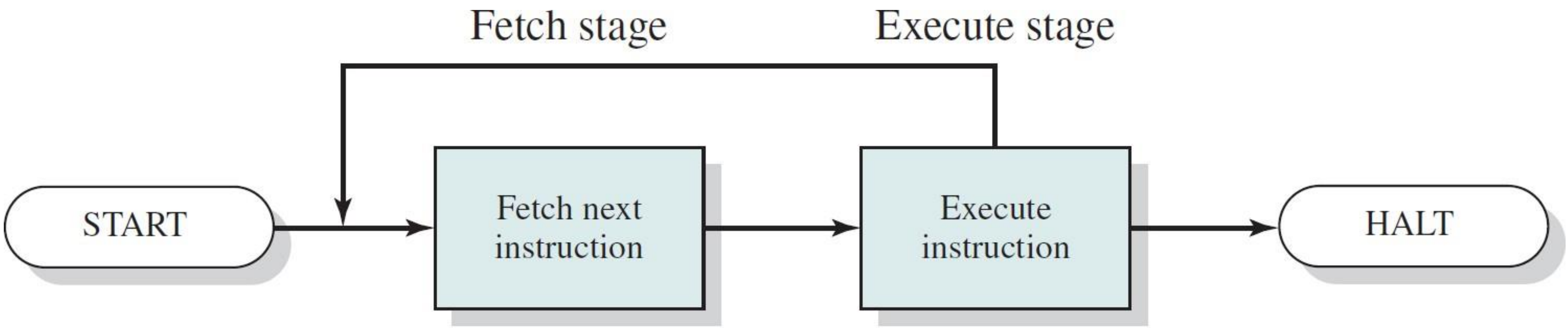


Figure 1.2 **Basic Instruction Cycle**

Instruction Execution (Cont.)

Steps of Instruction Execution:

1. At beginning of each instruction cycle, the **processor** fetches an instruction from **memory**.
2. **Program counter (PC)** holds address of **next** instruction to fetch.
3. Unless instructed, processor increments PC after each instruction fetch so that it will fetch the next instruction in sequence.
4. Fetched instruction is loaded into the **instruction register (IR)**.
5. Instruction contains bits that specify action a processor is to take.
6. Processor interprets the instruction and performs **required action**.

Instruction Execution (Cont.)

- **Program execution halts only if:**

1. Processor is turned off
2. Unrecoverable error occurs
3. Program instruction that halts the processor is encountered

Instruction Execution (Cont.)

In general, the actions a processor takes to an instruction:

1. **Processor-memory:** data may be transferred from processor to memory or from memory to processor.
2. **Processor-I/O:** data may be transferred to or from a peripheral device by transferring between processor and an I/O module.
3. **Data processing:** the processor may perform some arithmetic or logic operation on data.
4. **Control:** an instruction may specify that the sequence of execution be altered.

Characteristics of a Hypothetical Machine

0 – 3	4 – 15
Opcode	Address

(a) Instruction format

0	1 – 15
S	Magnitude

(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
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

Interrupts

- **Interrupt:** a mechanism by which other modules (I/O, memory) may interrupt the ***"normal sequencing of the processor"***.
- Most common classes of Interrupts are:

I/O	**Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.**
Program	**Generated by a condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.**
Timer	**Generated by a timer within the processor.**
Hardware failure	**Generated by a failure, such as memory error.**

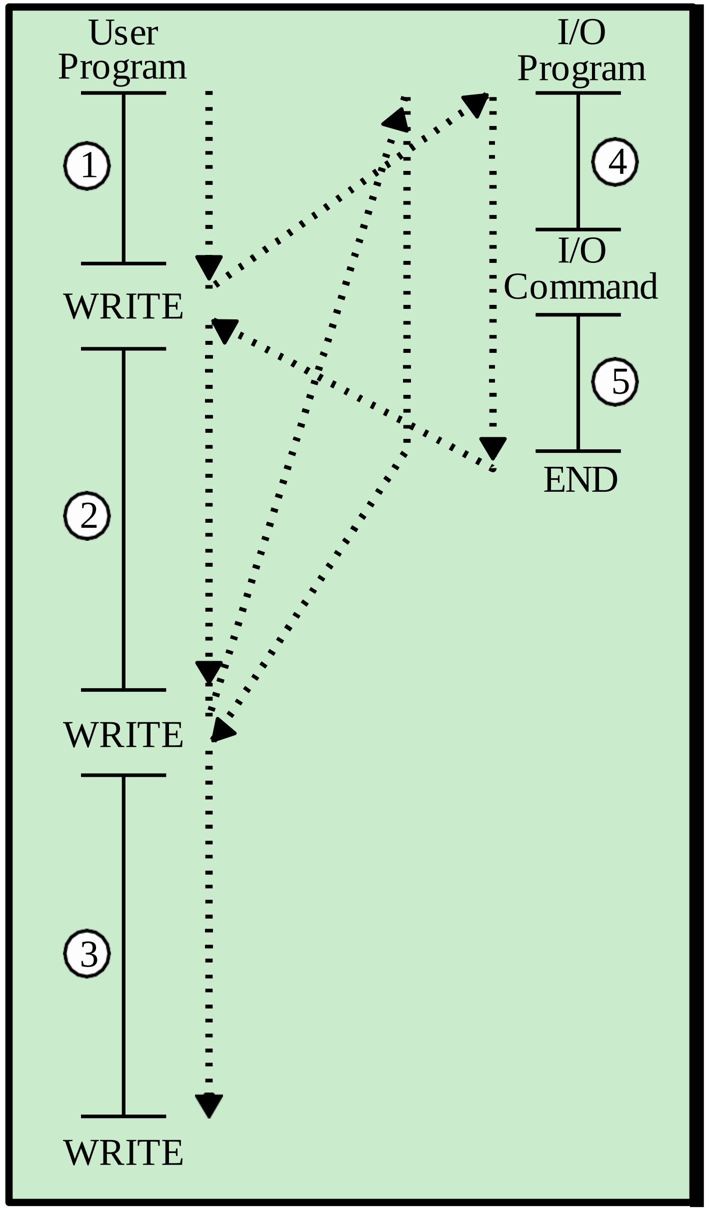
Interrupts (Cont.)

- **Main benefit;** interrupts are provided primarily as a way to improve **processor utilization**. This is because that most I/O devices are much slower than the processor.

- **Example;** suppose a processor is transferring data to a printer. After each **WRITE** operation, the processor must pause and remain idle until the printer catches up. Length of this pause may be in order of many thousands or even millions of **instruction cycles**.

No Interrupts

- **User program** performs a series of **WRITE** calls interleaved with processing.
- The **WRITE** calls are to an **I/O program** that will perform the actual I/O operation.
- In **I/O program**, a sequence of instructions is executed to “prepare” for actual I/O operation.
- Once **actual I/O command** is issued, the **I/O program** must wait for I/O device to perform the requested function.



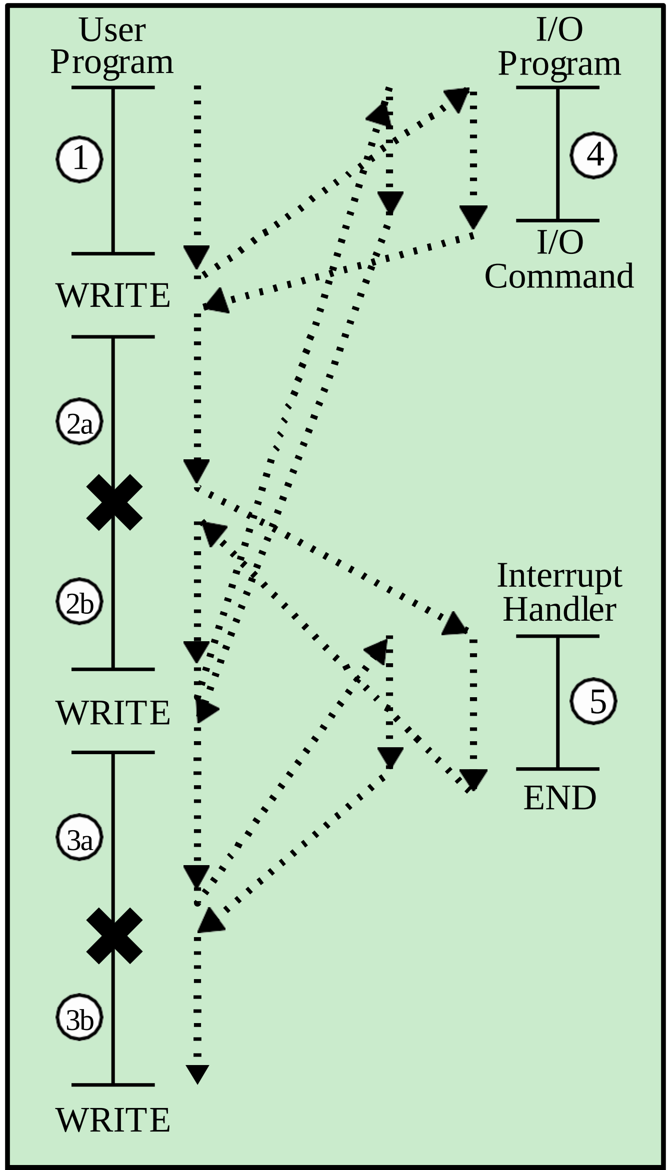
(a) No interrupts

No Interrupts (Cont.)

- After the first **WRITE** instruction is encountered, **user program** is suspended and execution continues with **I/O program**.
- Since I/O operation may take a relatively long time to complete, **I/O program** is hung up waiting for the operation to complete.
- The **user program** is stopped at the point of the **WRITE** call for some considerable period of time.
- After **I/O program** execution is complete, execution resumes in the **user program** immediately following the **WRITE** instruction.

I/O Wait Interrupts

- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- The **I/O program** that is invoked in this case consists only of the **preparation code** and the actual **I/O command**.
- The I/O operation is conducted concurrently with the execution of instructions in **user program**.



(b) Interrupts; short I/O wait

I/O Wait Interrupts (Cont.)

- When the external device becomes ready to accept more data from the processor, the I/O module for that external device sends an ***interrupt request*** signal to the processor.
- The processor responds by **suspending operation** of current program; branching off to a routine to service that particular I/O device, known as an **interrupt handler**; and resuming the original execution after the device is serviced.
- **Note:** an interrupt can occur at any point in the main program, not just at one specific instruction.

I/O Wait Interrupts (Cont.)

- For the **user program**, an interrupt **suspends** the normal sequence of execution. When the interrupt processing is completed, execution resumes.
- Because of the relatively large amount of time that would be wasted by simply waiting on an I/O operation, the processor can be employed much more efficiently with use of interrupts.

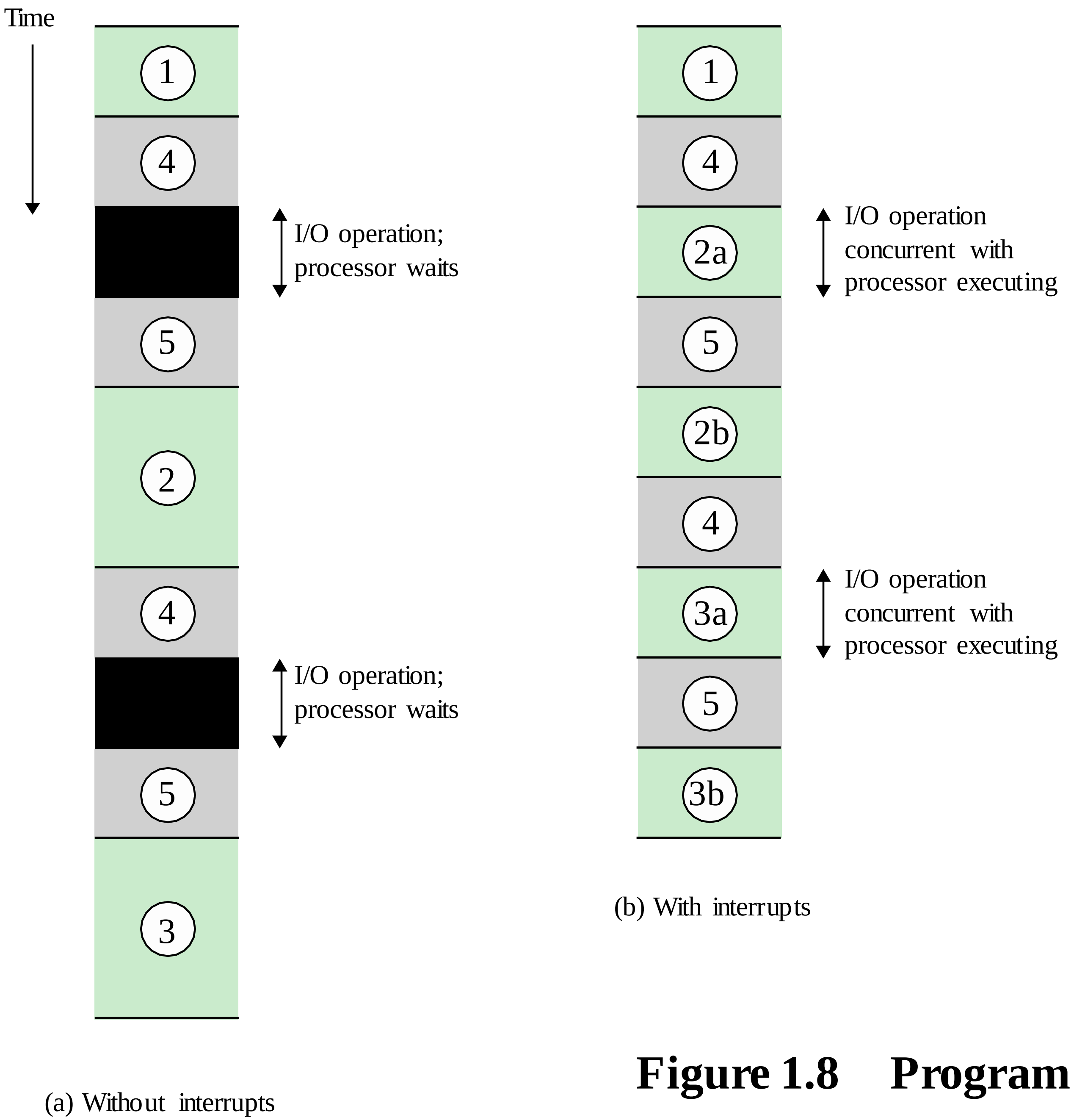


Figure 1.8 Program Timing: Short I/O Wait

Interrupts and Instruction Cycle

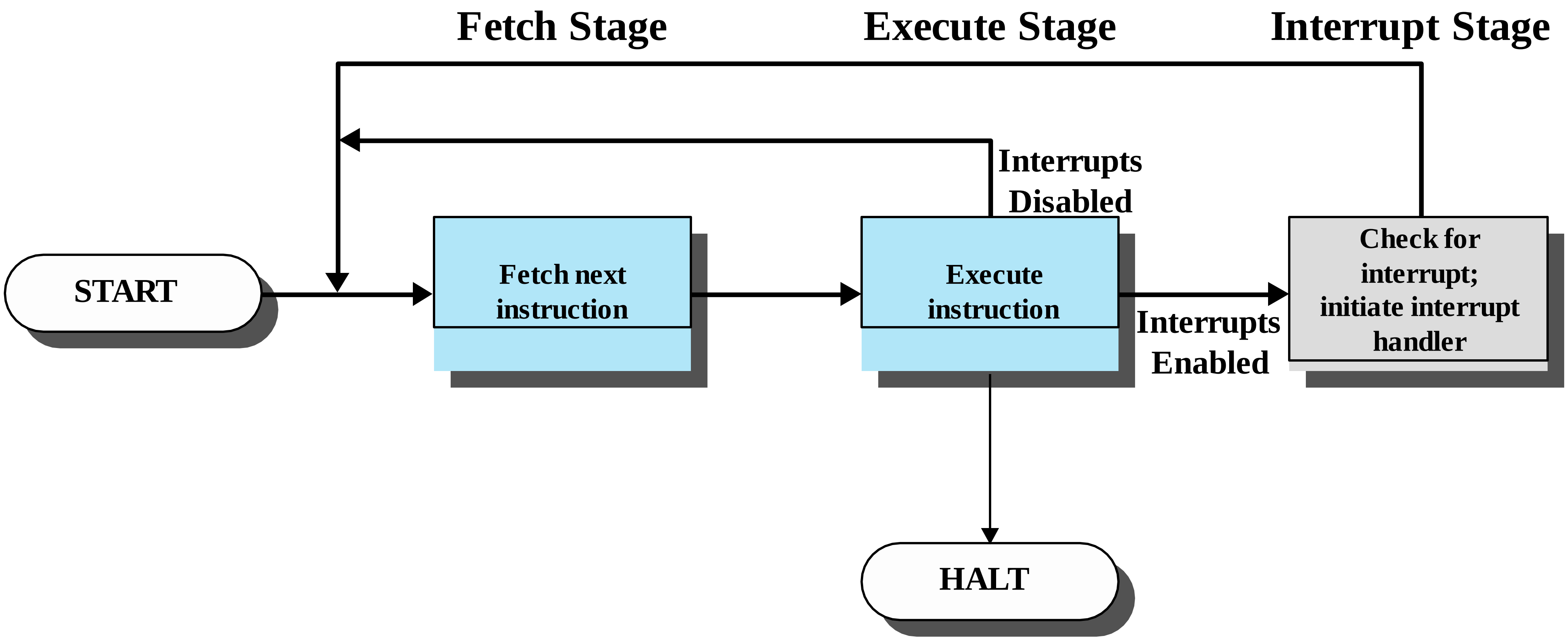


Figure 1.7 Instruction Cycle with Interrupts

Transfer of Control via Interrupts

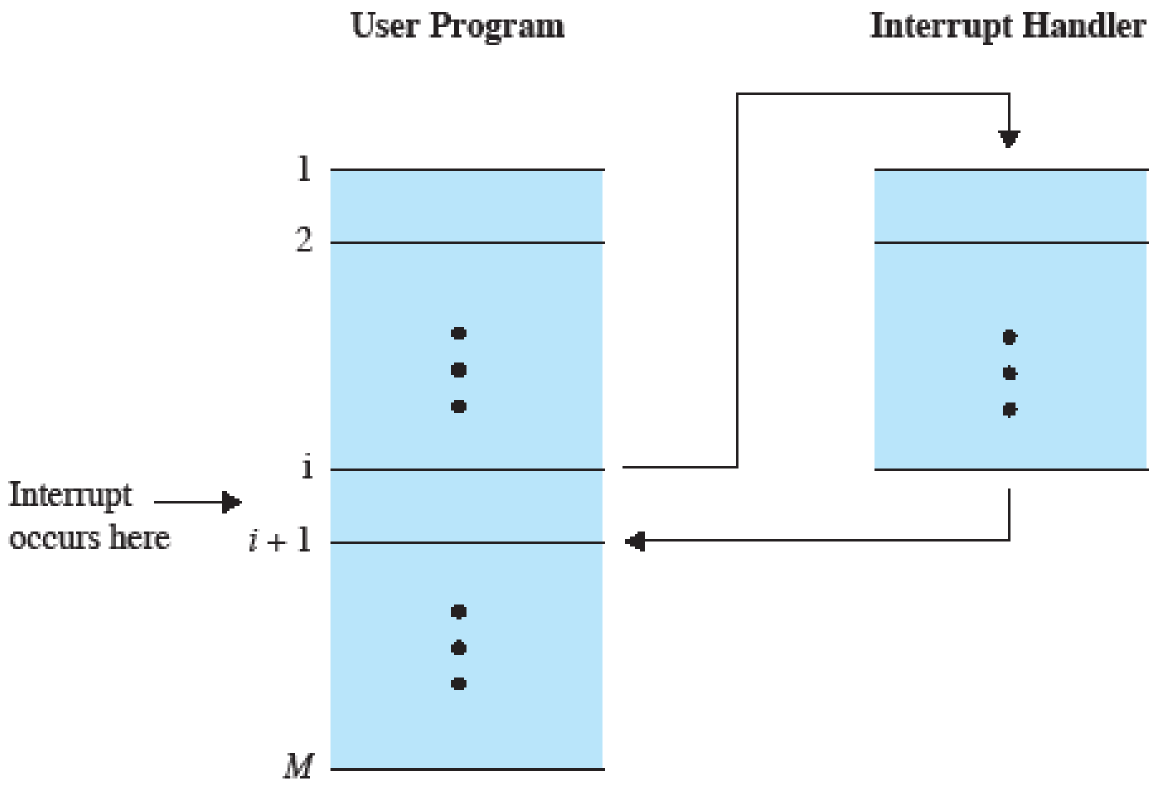
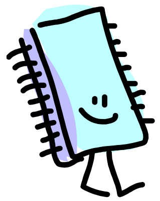


Figure 1.6 Transfer of Control via Interrupts



Multiple Interrupts

- Suppose, that **one or more** interrupts can occur while an interrupt is being processed.
- E.g., a program may be receiving data from a **communications line** and **printing** results at the same time.
- It is possible for a communications interrupt to occur while a printer interrupt is being processed.

Multiple Interrupts

An interrupt occurs while another interrupt is being processed

- e.g. receiving data from a communications line and printing results at the same time

Two approaches:

- disable interrupts while an interrupt is being processed (sequential)
- use a priority scheme (nested)

Multiple Interrupts Approaches

Approach#01:

- Interrupts are handled in **strict sequential order**.
- Drawback is the lack of considering relative priority or time-critical needs.

Approach#02:

- Define **priorities** for interrupts and allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted.

Multiple Interrupts

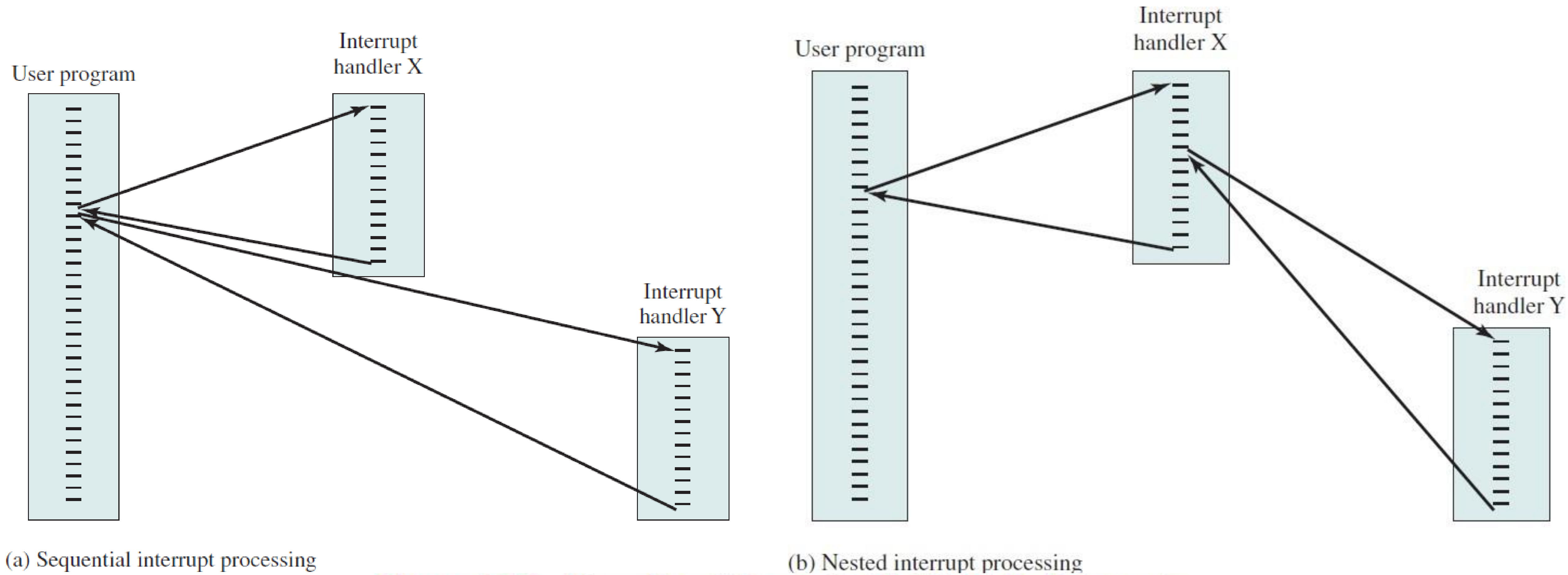


Figure 1.12 Transfer of Control with Multiple Interrupts

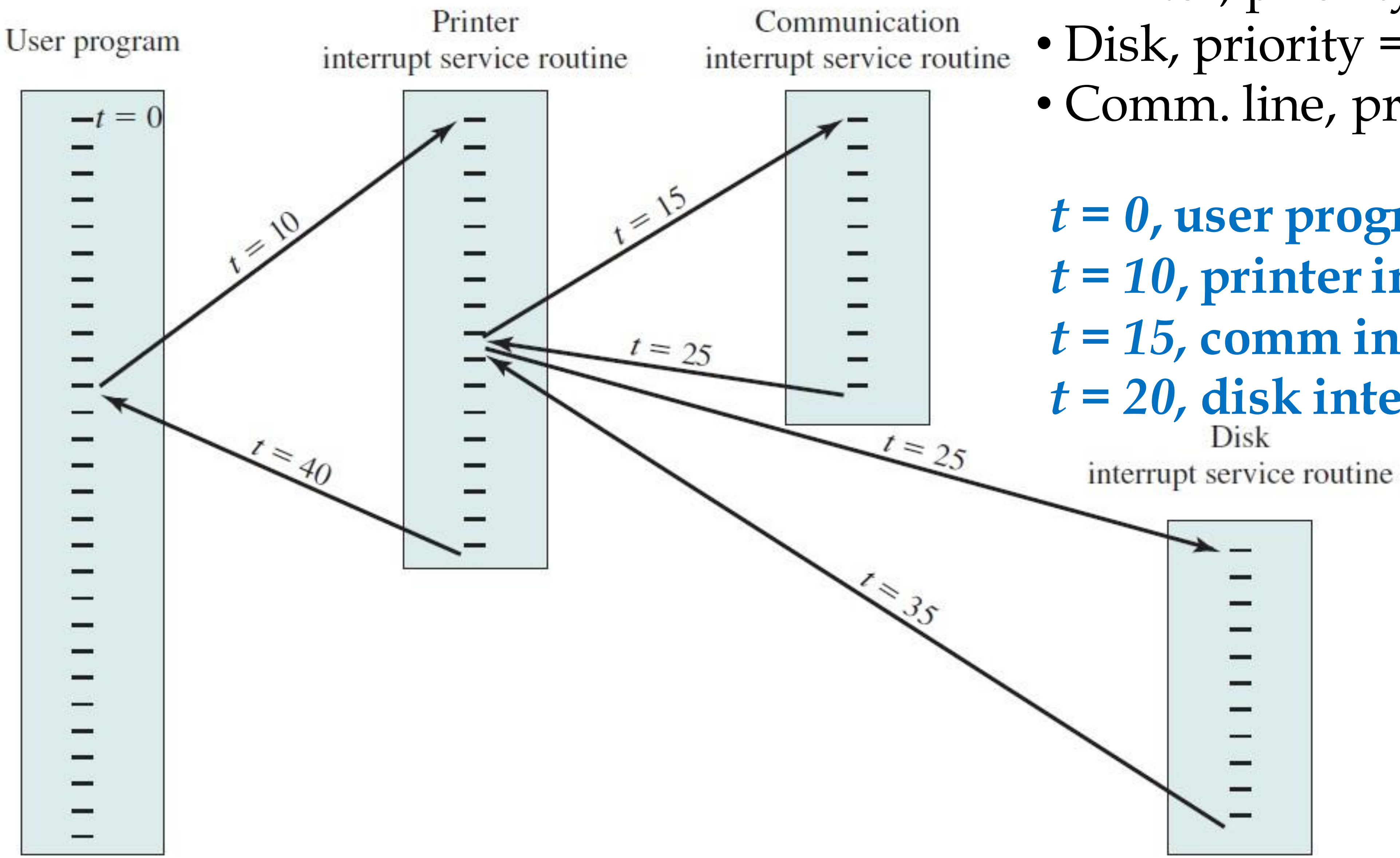


Figure 1.13 **Example Time Sequence of Multiple Interrupts**

- Printer, priority = 2.
- Disk, priority = 4.
- Comm. line, priority = 5.

***t* = *0*, user program begins.**
***t* = *10*, printer interrupt.**
***t* = *15*, comm interrupt.**
***t* = *20*, disk interrupt.**

Storage Definitions and Notation Review

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers, it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.
A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1{,}024^2$ bytes
a **gigabyte**, or **GB**, is $1{,}024^3$ bytes
a **terabyte**, or **TB**, is $1{,}024^4$ bytes
a **petabyte**, or **PB**, is $1{,}024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

Storage Structure

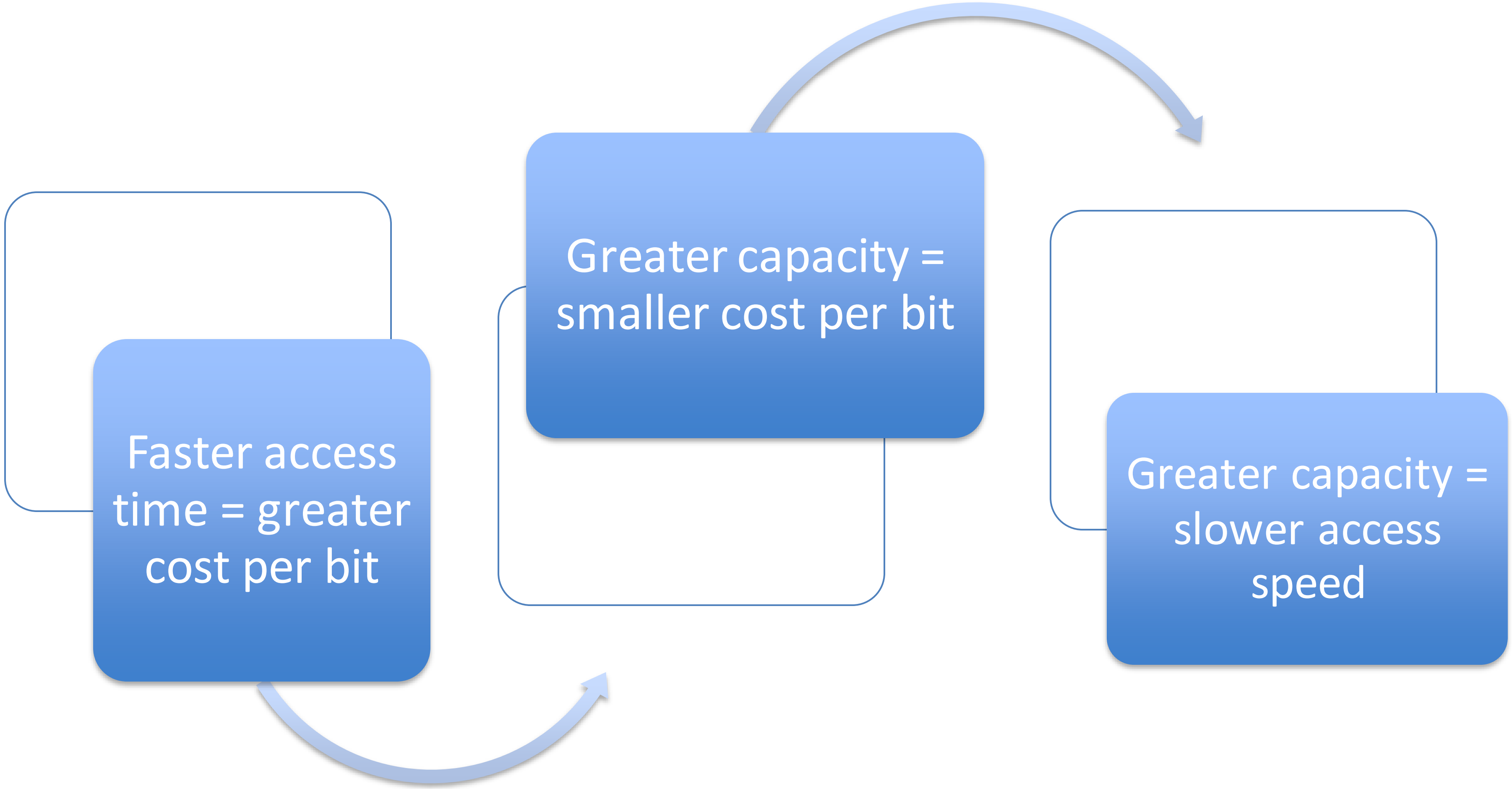
- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Hard disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer
- **Solid-state disks** – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular

The Memory Hierarchy

- Design constraints on a computer's memory:
- **capacity**
- **access time and cost.**
- There is a trade-off among the three key characteristics of memory, where the following "relationships" hold:

1. **Faster access time, greater cost per bit.**
2. **Greater capacity, smaller cost per bit.**
3. **Greater capacity, slower access speed.**

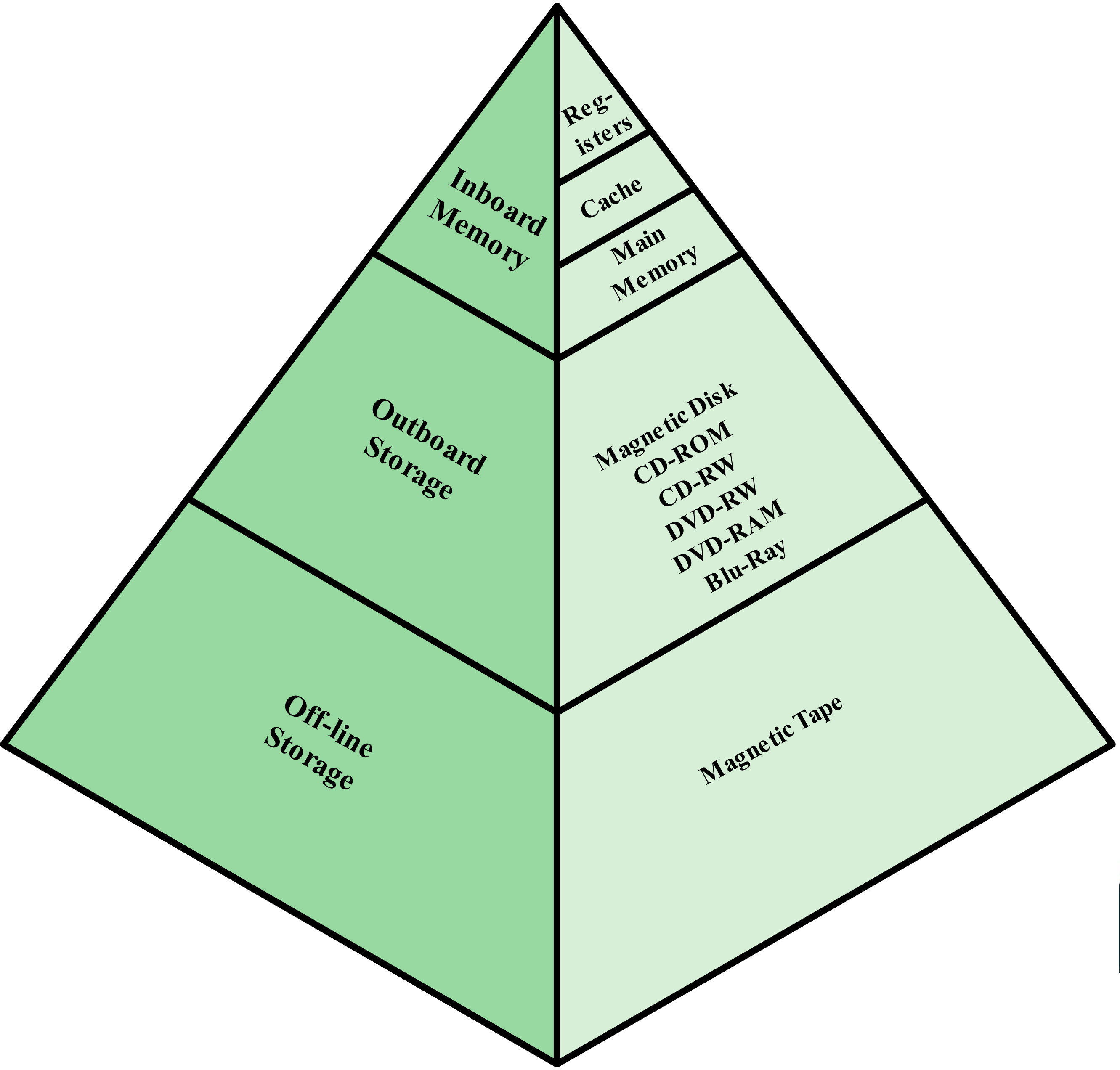
Memory Relationships

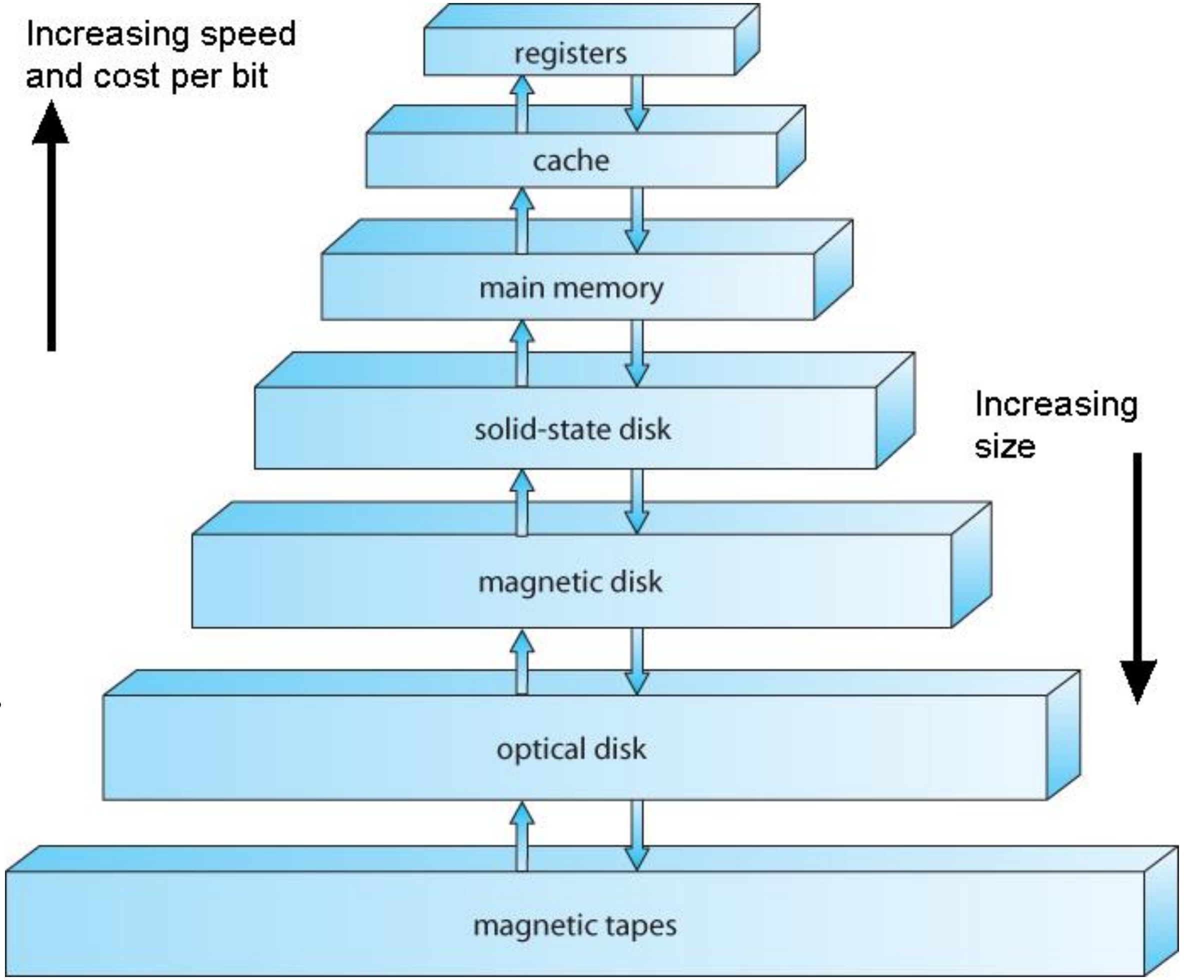


The Memory Hierarchy (Cont.)

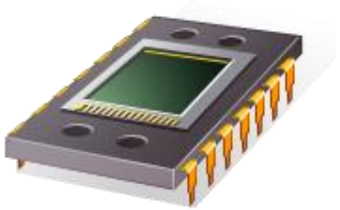
- A variety of technologies are used to implement memory systems.
- The way to meet both large capacity and performance is to employ a memory hierarchy. Going down the hierarchy, the following occurs:

1. **Decreasing cost per bit.**
2. **Increasing capacity.**
3. **Increasing access time.**
4. **Decreasing frequency of access to memory by processor.**


Reg-isters
Cache
Main Memory
Inboard Memory
Outboard Storage
Magnetic Disk
CD-ROM
CD-RW
DVD-RW
DVD-RAM
Blu-Ray
Off-line Storage
Magnetic Tape


Increasing speed and cost per bit
registers
cache
main memory
solid-state disk
magnetic disk
optical disk
magnetic tapes
Increasing size

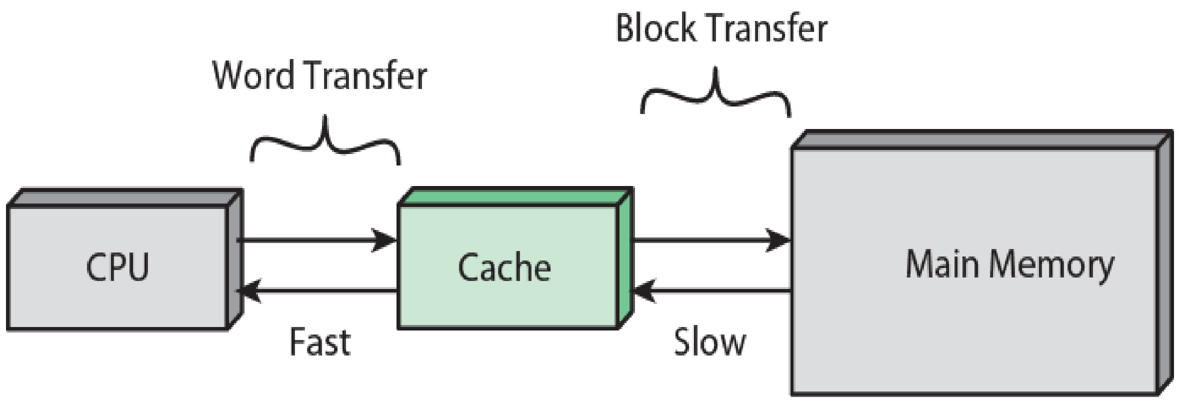
Cache Memory



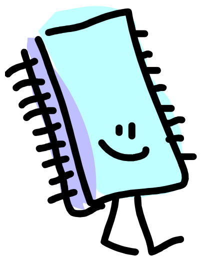
- Invisible to the OS
- Interacts with other memory management hardware
- Processor must access memory at least once per instruction cycle.
- Contains a copy of a portion of main memory
- Processor first checks cache

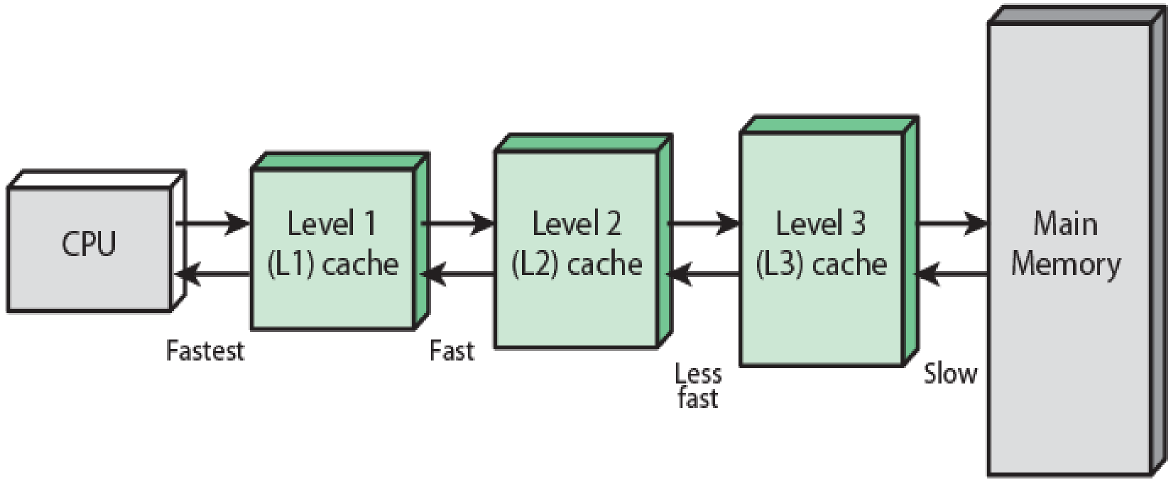
– If not found, a block of memory is read into cache

Cache and Main Memory



(a) Single cache





(b) Three-level cache organization

Cache/Main-Memory Structure

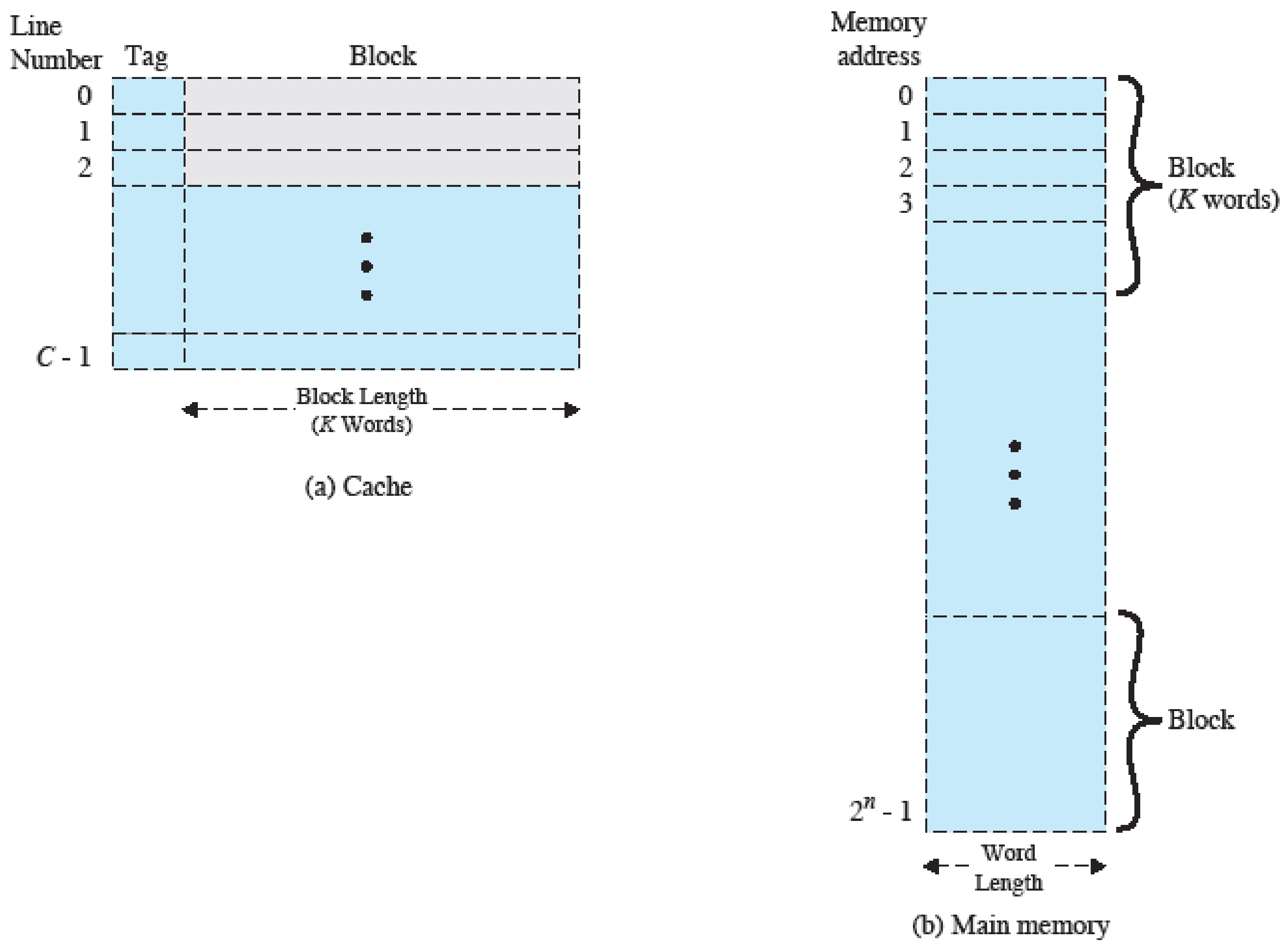
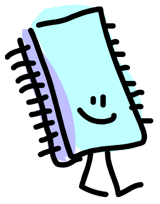
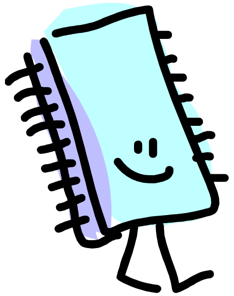


Figure 1.17 Cache/Main-Memory Structure



Cache Read Operation



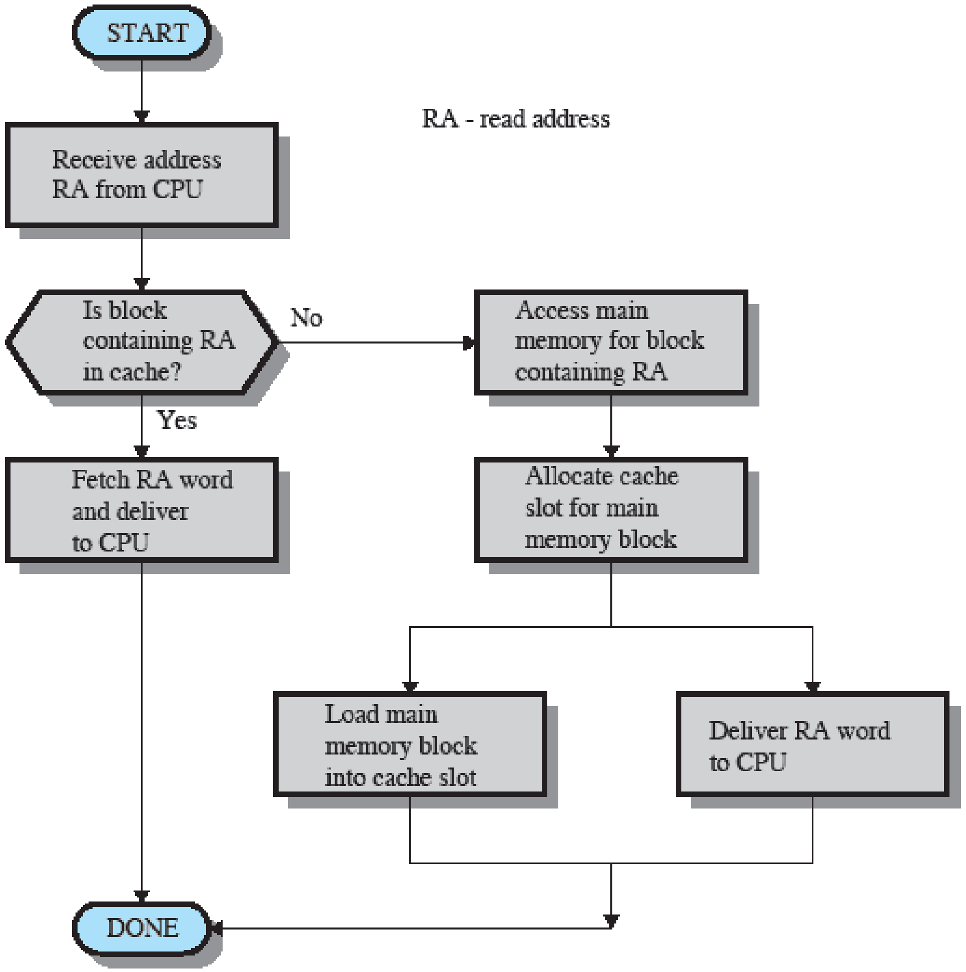
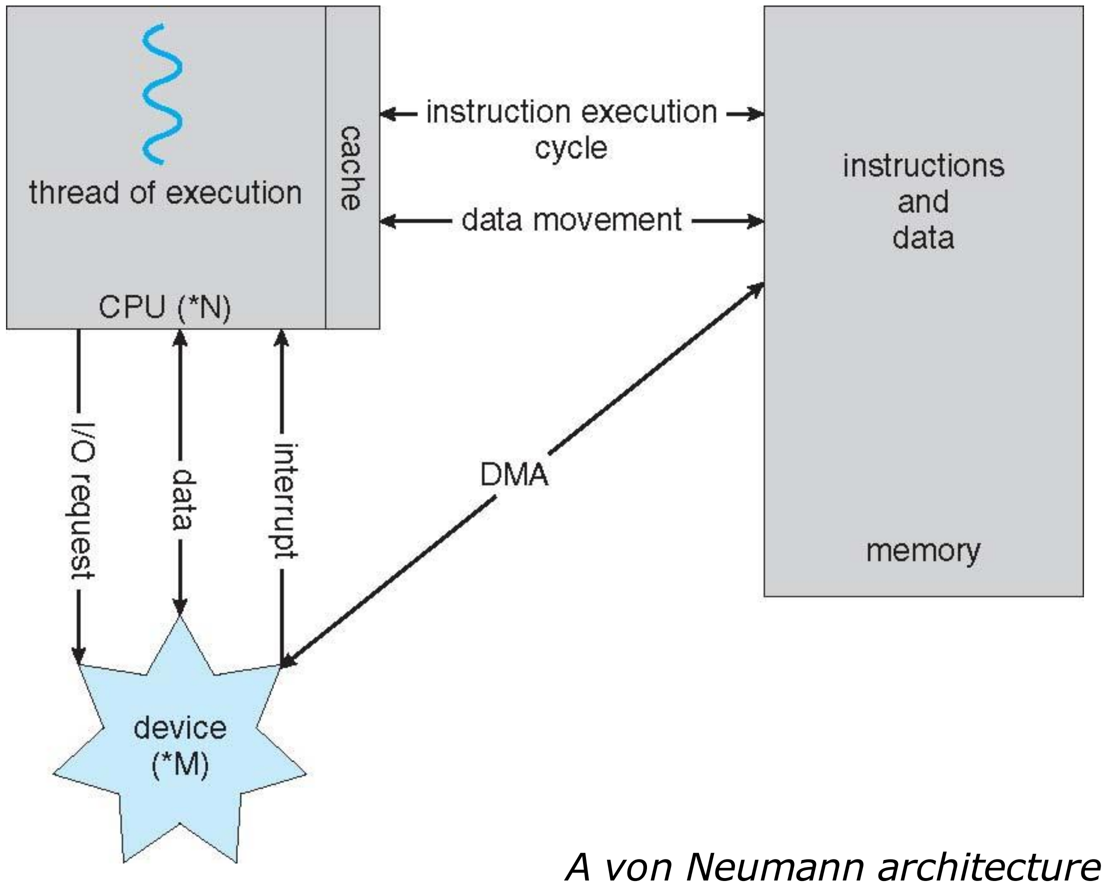


Figure 1.18 Cache Read Operation

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

How a Modern Computer Works



A von Neumann architecture

Thank You!

Chapter 1a Completed