Chapter 01

Introduction to Computers

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer
- Data processing
- Characteristic features of computers
- Computers' evolution to their present form
- Computer generations
- Characteristic features of each computer generation

Computer

- The word computer comes from the word "compute", which means, "to calculate"
- Thereby, a computer is an electronic device that can perform arithmetic operations at high speed
- A computer is also called a *data processor* because it can store, process, and retrieve data whenever desired

Data Processing

The activity of processing data using a computer is called *data processing*

Data

↓

Capture Data

↓

Manipulate Data

↓

Output Results

↓

Information

Data is raw material used as input and *information* is processed data obtained as output of data processing

Characteristics of Computers

1) **Automatic:** Given a job, computer can work on it automatically without human interventions

2) **Speed:** Computer can perform data processing jobs very fast, usually measured in microseconds (10^{-6}), nanoseconds (10^{-9}), and picoseconds (10^{-12})

3) **Accuracy:** Accuracy of a computer is consistently high and the degree of its accuracy depends upon its design. Computer errors caused due to incorrect input data or unreliable programs are often referred to as ***Garbage-In-Garbage-Out*** (GIGO)

(Continued on next slide)

Characteristics of Computers

(Continued from previous slide..)

4) **Diligence: Computer is free from monotony, tiredness, and lack of concentration. It can continuously work for hours without creating any error and without grumbling**

5) **Versatility: Computer is capable of performing almost any task, if the task can be reduced to a finite series of logical steps**

6) **Power of Remembering: Computer can store and recall any amount of information because of its secondary storage capability. It forgets or looses certain information only when it is asked to do so**

(Continued on next slide)

Characteristics of Computers

(Continued from previous slide..)

7) **No I.Q.: A computer does only what it is programmed to do. It cannot take its own *decision* in this regard**

8) **No Feelings: Computers are devoid of emotions. Their judgement is based on the instructions given to them in the form of programs that are written by us (human beings)**

(Continued on next slide)

Evolution of Computers

- Blaise Pascal invented the first *mechanical adding machine* in 1642
- Baron Gottfried Wilhelm von Leibniz invented the first *calculator for multiplication* in 1671
- *Keyboard machines* originated in the United States around 1880
- Around 1880, Herman Hollerith came up with the concept of *punched cards* that were extensively used as input media until late 1970s

Evolution of Computers

(Continued from previous slide..)

- ***Charles Babbage*** **is considered to be the father of modern digital computers**
  - **He designed "Difference Engine" in 1822**
  - **He designed a *fully automatic analytical engine* in 1842 for performing basic arithmetic functions**
  - **His efforts established a number of principles that are fundamental to the design of any digital computer**

(Continued on next slide)

Some Well Known Early Computers

- The Mark I Computer (1937-44)
- The Atanasoff-Berry Computer (1939-42)
- The ENIAC (1943-46)
- The EDVAC (1946-52)
- The EDSAC (1947-49)
- Manchester Mark I (1948)
- The UNIVAC I (1951)

Computer Generations

- "*Generation*" in computer talk is a step in technology. It provides a framework for the growth of computer industry

- Originally it was used to distinguish between various hardware technologies, but now it has been extended to include both hardware and software

- Till today, there are five computer generations

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some representative systems
First (1942-1955)	§ Vacuum tubes § Electromagnetic relay memory § Punched cards secondary storage	§ Machine and assembly languages § Stored program concept § Mostly scientific applications	§ Bulky in size § Highly unreliable § Limited commercial use and costly § Difficult commercial production § Difficult to use	§ ENIAC § EDVAC § EDSAC § UNIVAC I § IBM 701
Second (1955-1964)	§ Transistors § Magnetic cores memory § Magnetic tapes § Disks for secondary storage	§ Batch operating system § High-level programming languages § Scientific and commercial applications	§ Faster, smaller, more reliable and easier to program than previous generation systems § Commercial production was still difficult and costly	§ Honeywell 400 § IBM 7030 § CDC 1604 § UNIVAC LARC

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some rep. systems
Third (1964-1975)	§ ICs with SSI and MSI technologies § Larger magnetic cores memory § Larger capacity disks and magnetic tapes secondary storage § Minicomputers; upward compatible family of computers	§ Timesharing operating system § Standardization of high-level programming languages § Unbundling of software from hardware	§ Faster, smaller, more reliable, easier and cheaper to produce § Commercially, easier to use, and easier to upgrade than previous generation systems § Scientific, commercial and interactive on-line applications	§ IBM 360/370 § PDP-8 § PDP-11 § CDC 6600

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware Technologies	Key software technologies	Key characteristics	Some rep. systems
Fourth (1975-1989)	§ ICs with VLSI technology § Microprocessors; semiconductor memory § Larger capacity hard disks as in-built secondary storage § Magnetic tapes and floppy disks as portable storage media § Personal computers § Supercomputers based on parallel vector processing and symmetric multiprocessing technologies § Spread of high-speed computer networks	§ Operating systems for PCs with GUI and multiple windows on a single terminal screen § Multiprocessing OS with concurrent programming languages § UNIX operating system with C programming language § Object-oriented design and programming § PC, Network-based, and supercomputing applications	§ Small, affordable, reliable, and easy to use PCs § More powerful and reliable mainframe systems and supercomputers § Totally general purpose machines § Easier to produce commercially § Easier to upgrade § Rapid software development possible	§ IBM PC and its clones § Apple II § TRS-80 § VAX 9000 § CRAY-1 § CRAY-2 § CRAY-X/MP

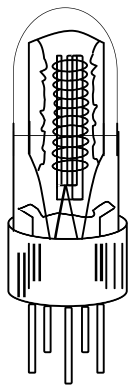
(Continued on next slide)

Computer Generations

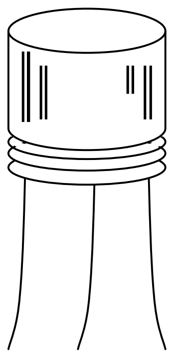
(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some rep. systems
Fifth (1989-Present)	§ ICs with ULSI technology § Larger capacity main memory, hard disks with RAID support § Optical disks as portable read-only storage media § Notebooks, powerful desktop PCs and workstations § Powerful servers, supercomputers § Internet § Cluster computing	§ Micro-kernel based, multithreading, distributed OS § Parallel programming libraries like MPI & PVM § JAVA § World Wide Web § Multimedia, Internet applications § More complex supercomputing applications	§ Portable computers § Powerful, cheaper, reliable, and easier to use desktop machines § Powerful supercomputers § High uptime due to hot-pluggable components § Totally general purpose machines § Easier to produce commercially, easier to upgrade § Rapid software development possible	§ IBM notebooks § Pentium PCs § SUN Workstations § IBM SP/2 § SGI Origin 2000 § PARAM 10000

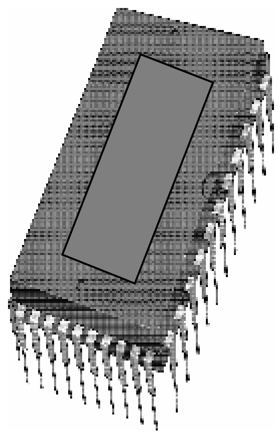
Electronic Devices Used in Computers of Different Generations



(a) A Vacuum Tube



(b) A Transistor



(c) An IC Chip

Key Words / Phrases

- Computer
- Computer generations
- Computer Supported Cooperative Working (CSCW)
- Data
- Data processing
- Data processor
- First-generation computers
- Fourth-generation computers
- Garbage-in-garbage-out (GIGO)
- Graphical User Interface (GUI)
- Groupware
- Information
- Integrated Circuit (IC)
- Large Scale Integration (VLSI)
- Medium Scale Integration (MSI)
- Microprocessor
- Personal Computer (PC)
- Second-generation computers
- Small Scale Integration (SSI)
- Stored program concept
- Third-generation computers
- Transistor
- Ultra Large Scale Integration (ULSI)
- Vacuum tubes

Chapter 01

Introduction to Computers

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer
- Data processing
- Characteristic features of computers
- Computers' evolution to their present form
- Computer generations
- Characteristic features of each computer generation

Computer

- The word computer comes from the word “compute”, which means, “to calculate”
- Thereby, a computer is an electronic device that can perform arithmetic operations at high speed
- A computer is also called a *data processor* because it can store, process, and retrieve data whenever desired

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Data Processing

The activity of processing data using a computer is called *data processing*

Data
↓
Capture Data
↓
Manipulate Data
↓
Output Results
↓
Information

Data is raw material used as input and *information* is processed data obtained as output of data processing

Characteristics of Computers

1) **Automatic:** Given a job, computer can work on it automatically without human interventions

2) **Speed:** Computer can perform data processing jobs very fast, usually measured in microseconds (10^{-6}), nanoseconds (10^{-9}), and picoseconds (10^{-12})

3) **Accuracy:** Accuracy of a computer is consistently high and the degree of its accuracy depends upon its design. Computer errors caused due to incorrect input data or unreliable programs are often referred to as *Garbage-In-Garbage-Out* (GIGO)

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Characteristics of Computers

(Continued from previous slide..)

4) **Diligence:** Computer is free from monotony, tiredness, and lack of concentration. It can continuously work for hours without creating any error and without grumbling

5) **Versatility:** Computer is capable of performing almost any task, if the task can be reduced to a finite series of logical steps

6) **Power of Remembering:** Computer can store and recall any amount of information because of its secondary storage capability. It forgets or looses certain information only when it is asked to do so

(Continued on next slide)

Characteristics of Computers

(Continued from previous slide..)

7) **No I.Q.:** **A computer does only what it is programmed to do. It cannot take its own *decision* in this regard**

8) **No Feelings: Computers are devoid of emotions. Their judgement is based on the instructions given to them in the form of programs that are written by us (human beings)**

(Continued on next slide)

Evolution of Computers

- **Blaise Pascal invented the first *mechanical adding machine* in 1642**
- **Baron Gottfried Wilhelm von Leibniz invented the first *calculator for multiplication* in 1671**
- ***Keyboard machines* originated in the United States around 1880**
- **Around 1880, Herman Hollerith came up with the concept of *punched cards* that were extensively used as input media until late 1970s**

Evolution of Computers

(Continued from previous slide..)

- *Charles Babbage* is considered to be the father of modern digital computers
  - He designed "Difference Engine" in 1822
  - He designed a *fully automatic analytical engine* in 1842 for performing basic arithmetic functions
  - His efforts established a number of principles that are fundamental to the design of any digital computer

(Continued on next slide)

Some Well Known Early Computers

- The Mark I Computer (1937-44)
- The Atanasoff-Berry Computer (1939-42)
- The ENIAC (1943-46)
- The EDVAC (1946-52)
- The EDSAC (1947-49)
- Manchester Mark I (1948)
- The UNIVAC I (1951)

Computer Generations

- "*Generation*" in computer talk is a step in technology. It provides a framework for the growth of computer industry
- Originally it was used to distinguish between various hardware technologies, but now it has been extended to include both hardware and software
- Till today, there are five computer generations

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some representative systems
First (1942-1955)	§ Vacuum tubes § Electromagnetic relay memory § Punched cards secondary storage	§ Machine and assembly languages § Stored program concept § Mostly scientific applications	§ Bulky in size § Highly unreliable § Limited commercial use and costly § Difficult commercial production § Difficult to use	§ ENIAC § EDVAC § EDSAC § UNIVAC I § IBM 701
Second (1955-1964)	§ Transistors § Magnetic cores memory § Magnetic tapes § Disks for secondary storage	§ Batch operating system § High-level programming languages § Scientific and commercial applications	§ Faster, smaller, more reliable and easier to program than previous generation systems § Commercial production was still difficult and costly	§ Honeywell 400 § IBM 7030 § CDC 1604 § UNIVAC LARC

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some rep. systems
Third (1964-1975)	§ ICs with SSI and MSI technologies § Larger magnetic cores memory § Larger capacity disks and magnetic tapes secondary storage § Minicomputers; upward compatible family of computers	§ Timesharing operating system § Standardization of high-level programming languages § Unbundling of software from hardware	§ Faster, smaller, more reliable, easier and cheaper to produce § Commercially, easier to use, and easier to upgrade than previous generation systems § Scientific, commercial and interactive on-line applications	§ IBM 360/370 § PDP-8 § PDP-11 § CDC 6600

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware Technologies	Key software technologies	Key characteristics	Some rep. systems
Fourth (1975-1989)	§ ICs with VLSI technology § Microprocessors; semiconductor memory § Larger capacity hard disks as in-built secondary storage § Magnetic tapes and floppy disks as portable storage media § Personal computers § Supercomputers based on parallel vector processing and symmetric multiprocessing technologies § Spread of high-speed computer networks	§ Operating systems for PCs with GUI and multiple windows on a single terminal screen § Multiprocessing OS with concurrent programming languages § UNIX operating system with C programming language § Object-oriented design and programming § PC, Network-based, and supercomputing applications	§ Small, affordable, reliable, and easy to use PCs § More powerful and reliable mainframe systems and supercomputers § Totally general purpose machines § Easier to produce commercially § Easier to upgrade § Rapid software development possible	§ IBM PC and its clones § Apple II § TRS-80 § VAX 9000 § CRAY-1 § CRAY-2 § CRAY-X/MP

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Computer Generations

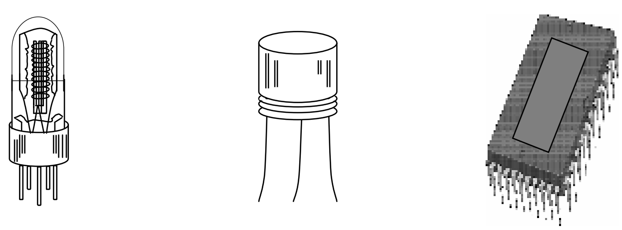
(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some rep. systems
Fifth (1989-Present)	§ ICs with ULSI technology § Larger capacity main memory, hard disks with RAID support § Optical disks as portable read-only storage media § Notebooks, powerful desktop PCs and workstations § Powerful servers, supercomputers § Internet § Cluster computing	§ Micro-kernel based, multithreading, distributed OS § Parallel programming libraries like MPI & PVM § JAVA § World Wide Web § Multimedia, Internet applications § More complex supercomputing applications	§ Portable computers § Powerful, cheaper, reliable, and easier to use desktop machines § Powerful supercomputers § High uptime due to hot-pluggable components § Totally general purpose machines § Easier to produce commercially, easier to upgrade § Rapid software development possible	§ IBM notebooks § Pentium PCs § SUN Workstations § IBM SP/2 § SGI Origin 2000 § PARAM 10000

Ref Page 13 Chapter 1: Introduction to Computers Slide 15/17

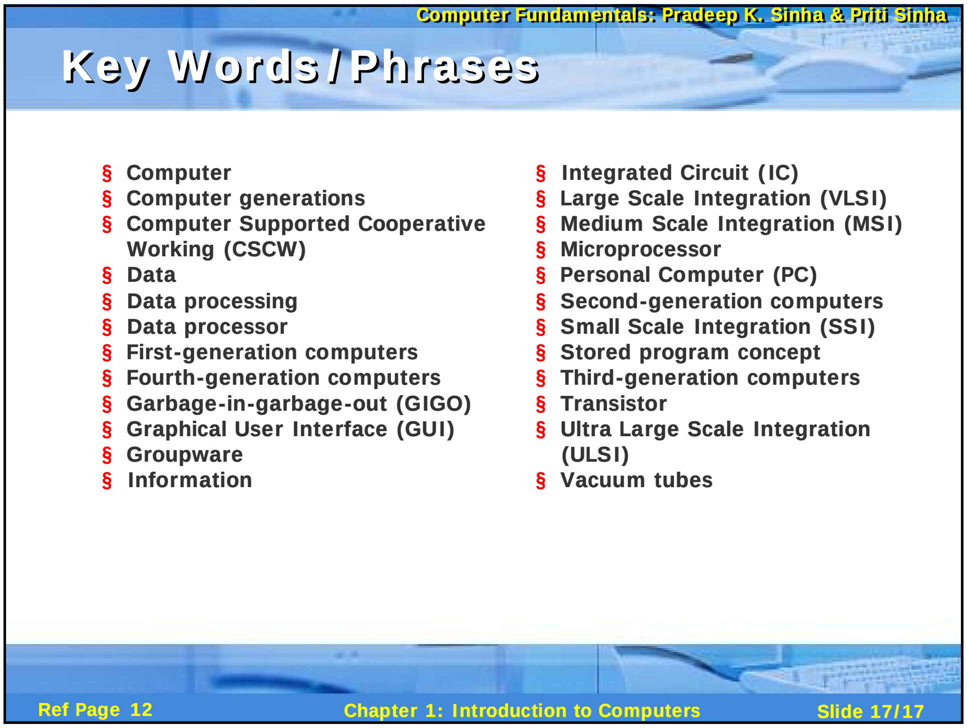
Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Electronic Devices Used in Computers of Different Generations



(a) A Vacuum Tube (b) A Transistor (c) An IC Chip

Ref Page 07 Chapter 1: Introduction to Computers Slide 16/17



Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Key Words / Phrases
§ Computer
§ Computer generations
§ Computer Supported Cooperative Working (CSCW)
§ Data
§ Data processing
§ Data processor
§ First-generation computers
§ Fourth-generation computers
§ Garbage-in-garbage-out (GIGO)
§ Graphical User Interface (GUI)
§ Groupware
§ Information
§ Integrated Circuit (IC)
§ Large Scale Integration (VLSI)
§ Medium Scale Integration (MSI)
§ Microprocessor
§ Personal Computer (PC)
§ Second-generation computers
§ Small Scale Integration (SSI)
§ Stored program concept
§ Third-generation computers
§ Transistor
§ Ultra Large Scale Integration (ULSI)
§ Vacuum tubes
Ref Page 12
Chapter 1: Introduction to Computers
Slide 17/17

Chapter 01

Introduction to Computers

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer
- Data processing
- Characteristic features of computers
- Computers' evolution to their present form
- Computer generations
- Characteristic features of each computer generation

Computer

- The word computer comes from the word "compute", which means, "to calculate"
- Thereby, a computer is an electronic device that can perform arithmetic operations at high speed
- A computer is also called a *data processor* because it can store, process, and retrieve data whenever desired

Data Processing

The activity of processing data using a computer is called *data processing*

Data
↓
Capture Data
↓
Manipulate Data
↓
Output Results
↓
Information

Data is raw material used as input and *information* is processed data obtained as output of data processing

Characteristics of Computers

1) **Automatic:** Given a job, computer can work on it automatically without human interventions

2) **Speed:** Computer can perform data processing jobs very fast, usually measured in microseconds (10^{-6}), nanoseconds (10^{-9}), and picoseconds (10^{-12})

3) **Accuracy:** Accuracy of a computer is consistently high and the degree of its accuracy depends upon its design. Computer errors caused due to incorrect input data or unreliable programs are often referred to as *Garbage-In-Garbage-Out* (GIGO)

(Continued on next slide)

Characteristics of Computers

(Continued from previous slide..)

4) **Diligence:** Computer is free from monotony, tiredness, and lack of concentration. It can continuously work for hours without creating any error and without grumbling

5) **Versatility:** Computer is capable of performing almost any task, if the task can be reduced to a finite series of logical steps

6) **Power of Remembering:** Computer can store and recall any amount of information because of its secondary storage capability. It forgets or looses certain information only when it is asked to do so

(Continued on next slide)

Characteristics of Computers

(Continued from previous slide..)

7) **No I.Q.: A computer does only what it is programmed to do. It cannot take its own *decision* in this regard**

8) **No Feelings: Computers are devoid of emotions. Their judgement is based on the instructions given to them in the form of programs that are written by us (human beings)**

(Continued on next slide)

Evolution of Computers

- **Blaise Pascal invented the first *mechanical adding machine* in 1642**
- **Baron Gottfried Wilhelm von Leibniz invented the first *calculator for multiplication* in 1671**
- ***Keyboard machines* originated in the United States around 1880**
- **Around 1880, Herman Hollerith came up with the concept of *punched cards* that were extensively used as input media until late 1970s**

Evolution of Computers

(Continued from previous slide..)

- ***Charles Babbage* is considered to be the father of modern digital computers**
  - **He designed "Difference Engine" in 1822**
  - **He designed a *fully automatic analytical engine* in 1842 for performing basic arithmetic functions**
  - **His efforts established a number of principles that are fundamental to the design of any digital computer**

(Continued on next slide)

Some Well Known Early Computers

- The Mark I Computer (1937-44)
- The Atanasoff-Berry Computer (1939-42)
- The ENIAC (1943-46)
- The EDVAC (1946-52)
- The EDSAC (1947-49)
- Manchester Mark I (1948)
- The UNIVAC I (1951)

Computer Generations

- "*Generation*" in computer talk is a step in technology. It provides a framework for the growth of computer industry
- Originally it was used to distinguish between various hardware technologies, but now it has been extended to include both hardware and software
- Till today, there are five computer generations

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some representative systems
First (1942-1955)	§ Vacuum tubes § Electromagnetic relay memory § Punched cards secondary storage	§ Machine and assembly languages § Stored program concept § Mostly scientific applications	§ Bulky in size § Highly unreliable § Limited commercial use and costly § Difficult commercial production § Difficult to use	§ ENIAC § EDVAC § EDSAC § UNIVAC I § IBM 701
Second (1955-1964)	§ Transistors § Magnetic cores memory § Magnetic tapes § Disks for secondary storage	§ Batch operating system § High-level programming languages § Scientific and commercial applications	§ Faster, smaller, more reliable and easier to program than previous generation systems § Commercial production was still difficult and costly	§ Honeywell 400 § IBM 7030 § CDC 1604 § UNIVAC LARC

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some rep. systems
Third (1964-1975)	§ ICs with SSI and MSI technologies § Larger magnetic cores memory § Larger capacity disks and magnetic tapes secondary storage § Minicomputers; upward compatible family of computers	§ Timesharing operating system § Standardization of high-level programming languages § Unbundling of software from hardware	§ Faster, smaller, more reliable, easier and cheaper to produce § Commercially, easier to use, and easier to upgrade than previous generation systems § Scientific, commercial and interactive on-line applications	§ IBM 360/370 § PDP-8 § PDP-11 § CDC 6600

(Continued on next slide)

Computer Generations

(Continued from previous slide..)

Generation (Period)	Key hardware Technologies	Key software technologies	Key characteristics	Some rep. systems
Fourth (1975-1989)	§ ICs with VLSI technology § Microprocessors; semiconductor memory § Larger capacity hard disks as in-built secondary storage § Magnetic tapes and floppy disks as portable storage media § Personal computers § Supercomputers based on parallel vector processing and symmetric multiprocessing technologies § Spread of high-speed computer networks	§ Operating systems for PCs with GUI and multiple windows on a single terminal screen § Multiprocessing OS with concurrent programming languages § UNIX operating system with C programming language § Object-oriented design and programming § PC, Network-based, and supercomputing applications	§ Small, affordable, reliable, and easy to use PCs § More powerful and reliable mainframe systems and supercomputers § Totally general purpose machines § Easier to produce commercially § Easier to upgrade § Rapid software development possible	§ IBM PC and its clones § Apple II § TRS-80 § VAX 9000 § CRAY-1 § CRAY-2 § CRAY-X/MP

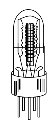
(Continued on next slide)

Computer Generations

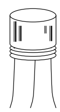
(Continued from previous slide..)

Generation (Period)	Key hardware technologies	Key software technologies	Key characteristics	Some rep. systems
Fifth (1989-Present)	§ ICs with ULSI technology § Larger capacity main memory, hard disks with RAID support § Optical disks as portable read-only storage media § Notebooks, powerful desktop PCs and workstations § Powerful servers, supercomputers § Internet § Cluster computing	§ Micro-kernel based, multithreading, distributed OS § Parallel programming libraries like MPI & PVM § JAVA § World Wide Web § Multimedia, Internet applications § More complex supercomputing applications	§ Portable computers § Powerful, cheaper, reliable, and easier to use desktop machines § Powerful supercomputers § High uptime due to hot-pluggable components § Totally general purpose machines § Easier to produce commercially, easier to upgrade § Rapid software development possible	§ IBM notebooks § Pentium PCs § SUN Workstations § IBM SP/2 § SGI Origin 2000 § PARAM 10000

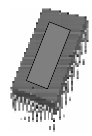
Electronic Devices Used in Computers of Different Generations



(a) A Vacuum Tube



(b) A Transistor



(c) An IC Chip

Key Words / Phrases

- Computer
- Computer generations
- Computer Supported Cooperative Working (CSCW)
- Data
- Data processing
- Data processor
- First-generation computers
- Fourth-generation computers
- Garbage-in-garbage-out (GIGO)
- Graphical User Interface (GUI)
- Groupware
- Information
- Integrated Circuit (IC)
- Large Scale Integration (VLSI)
- Medium Scale Integration (MSI)
- Microprocessor
- Personal Computer (PC)
- Second-generation computers
- Small Scale Integration (SSI)
- Stored program concept
- Third-generation computers
- Transistor
- Ultra Large Scale Integration (ULSI)
- Vacuum tubes

Chapter 02

Basic Computer Organization

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Basic operations performed by all types of computer systems
- Basic organization of a computer system
- Input unit and its functions
- Output unit and its functions
- Storage unit and its functions
- Types of storage used in a computer system

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Arithmetic Logic Unit (ALU)
- Control Unit (CU)
- Central Processing Unit (CPU)
- Computer as a system

The Five Basic Operations of a Computer System

- **Inputting.** The process of entering data and instructions into the computer system

- **Storing.** Saving data and instructions to make them readily available for initial or additional processing whenever required

- **Processing.** Performing arithmetic operations (add, subtract, multiply, divide, etc.) or logical operations (comparisons like equal to, less than, greater than, etc.) on data to convert them into useful information

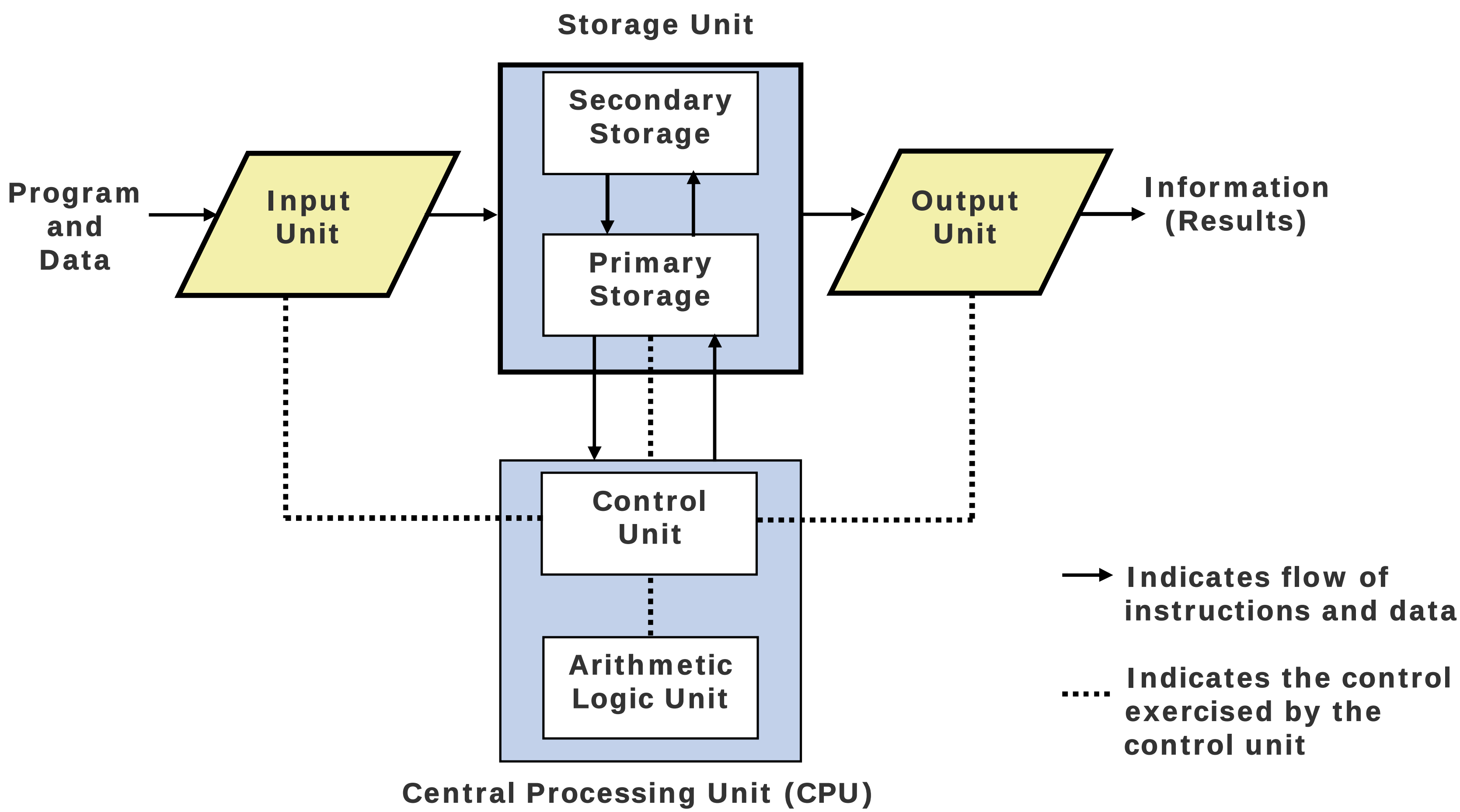
(Continued on next slide)

The Five Basic Operations of a Computer System

- **Outputting.** The process of producing useful information or results for the user such as a printed report or visual display

- **Controlling.** Directing the manner and sequence in which all of the above operations are performed

Basic Organization of a Computer System



Input Unit

An input unit of a computer system performs the following functions:

1. It accepts (or reads) instructions and data from outside world
2. It converts these instructions and data in computer acceptable form
3. It supplies the converted instructions and data to the computer system for further processing

Output Unit

An output unit of a computer system performs the following functions:

1. It accepts the results produced by the computer, which are in coded form and hence, cannot be easily understood by us
2. It converts these coded results to human acceptable (readable) form
3. It supplies the converted results to outside world

Storage Unit

The storage unit of a computer system holds (or stores) the following :

1. Data and instructions required for processing (received from input devices)
2. Intermediate results of processing
3. Final results of processing, before they are released to an output device

Two Types of Storage

- Primary storage
    - Used to hold running program instructions
    - Used to hold data, intermediate results, and results of ongoing processing of job(s)
    - Fast in operation
    - Small Capacity
    - Expensive
    - Volatile (looses data on power dissipation)

(Continued on next slide)

Two Types of Storage

(Continued from previous slide..)

- **Secondary storage**
  - Used to hold stored program instructions
  - Used to hold data and information of stored jobs
  - Slower than primary storage
  - Large Capacity
  - Lot cheaper that primary storage
  - Retains data even without power

Arithmetic Logic Unit (ALU)

Arithmetic Logic Unit of a computer system is the place where the actual executions of instructions takes place during processing operation

Control Unit (CU)

Control Unit of a computer system manages and coordinates the operations of all other components of the computer system

Central Processing Unit (CPU)

Arithmetic Logic Unit (ALU)	+	Control Unit (CU)	=	Central Processing Unit (CPU)

- It is the brain of a computer system

- It is responsible for controlling the operations of all other units of a computer system

The System Concept

A system has following three characteristics:

1. A system has more than one element
2. All elements of a system are logically related
3. All elements of a system are controlled in a manner to achieve the system goal

A computer is a system as it comprises of integrated components (input unit, output unit, storage unit, and CPU) that work together to perform the steps called for in the executing program

Key Words/Phrases

- Arithmetic Logic Unit (ALU)
- Auxiliary storage
- Central Processing Unit (CPU)
- Computer system
- Control Unit (CU)
- Controlling
- Input interface
- Input unit
- Inputting
- Main memory
- Output interface
- Output unit
- Outputting
- Primate storage
- Processing
- Secondary storage
- Storage unit
- Storing
- System

Chapter 02

Basic Computer Organization

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- **Basic operations performed by all types of computer systems**
- **Basic organization of a computer system**
- **Input unit and its functions**
- **Output unit and its functions**
- **Storage unit and its functions**
- **Types of storage used in a computer system**

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Arithmetic Logic Unit (ALU)
- Control Unit (CU)
- Central Processing Unit (CPU)
- Computer as a system

The Five Basic Operations of a Computer System

- **Inputting.** The process of entering data and instructions into the computer system
- **Storing.** Saving data and instructions to make them readily available for initial or additional processing whenever required
- **Processing.** Performing arithmetic operations (add, subtract, multiply, divide, etc.) or logical operations (comparisons like equal to, less than, greater than, etc.) on data to convert them into useful information

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

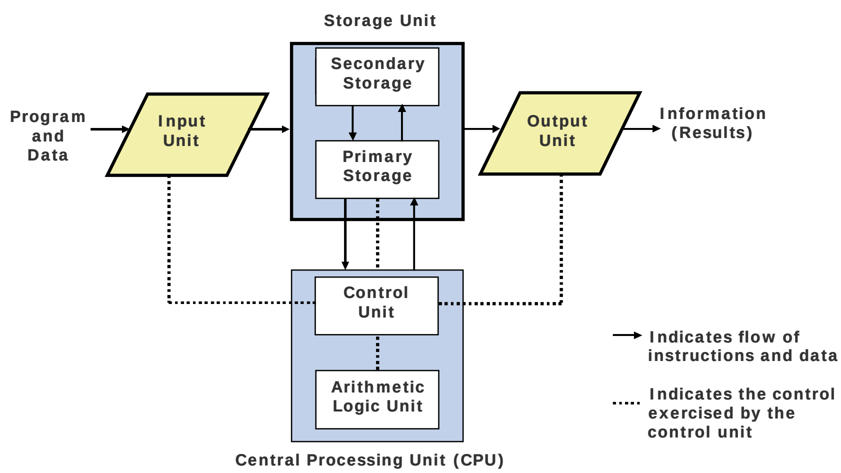
The Five Basic Operations of a Computer System

- **Outputting.** The process of producing useful information or results for the user such as a printed report or visual display
- **Controlling.** Directing the manner and sequence in which all of the above operations are performed

Ref. Page 15 Chapter 2: Basic Computer Organization Slide 5/16

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Basic Organization of a Computer System



Ref. Page 16 Chapter 2: Basic Computer Organization Slide 6/16

Input Unit

An input unit of a computer system performs the following functions:

1. It accepts (or reads) instructions and data from outside world
2. It converts these instructions and data in computer acceptable form
3. It supplies the converted instructions and data to the computer system for further processing

Output Unit

An output unit of a computer system performs the following functions:

1. It accepts the results produced by the computer, which are in coded form and hence, cannot be easily understood by us
2. It converts these coded results to human acceptable (readable) form
3. It supplies the converted results to outside world

Storage Unit

The storage unit of a computer system holds (or stores) the following :

1. Data and instructions required for processing (received from input devices)
2. Intermediate results of processing
3. Final results of processing, before they are released to an output device

Two Types of Storage

- Primary storage
  - Used to hold running program instructions
  - Used to hold data, intermediate results, and results of ongoing processing of job(s)
  - Fast in operation
  - Small Capacity
  - Expensive
  - Volatile (looses data on power dissipation)

(Continued on next slide)

Two Types of Storage

(Continued from previous slide..)

- Secondary storage
  - Used to hold stored program instructions
  - Used to hold data and information of stored jobs
  - Slower than primary storage
  - Large Capacity
  - Lot cheaper that primary storage
  - Retains data even without power

Arithmetic Logic Unit (ALU)

Arithmetic Logic Unit of a computer system is the place where the actual executions of instructions takes place during processing operation

Control Unit (CU)

Control Unit of a computer system manages and coordinates the operations of all other components of the computer system

Central Processing Unit (CPU)

Arithmetic Logic Unit (ALU) + Control Unit (CU) = Central Processing Unit (CPU)

§ It is the brain of a computer system

§ It is responsible for controlling the operations of all other units of a computer system

The System Concept

A system has following three characteristics:

1. A system has more than one element
2. All elements of a system are logically related
3. All elements of a system are controlled in a manner to achieve the system goal

A computer is a system as it comprises of integrated components (input unit, output unit, storage unit, and CPU) that work together to perform the steps called for in the executing program

Key Words / Phrases

- Arithmetic Logic Unit (ALU)
- Auxiliary storage
- Central Processing Unit (CPU)
- Computer system
- Control Unit (CU)
- Controlling
- Input interface
- Input unit
- Inputting
- Main memory
- Output interface
- Output unit
- Outputting
- Primate storage
- Processing
- Secondary storage
- Storage unit
- Storing
- System

Chapter 02

Basic Computer Organization

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Basic operations performed by all types of computer systems
- Basic organization of a computer system
- Input unit and its functions
- Output unit and its functions
- Storage unit and its functions
- Types of storage used in a computer system

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Arithmetic Logic Unit (ALU)
- Control Unit (CU)
- Central Processing Unit (CPU)
- Computer as a system

The Five Basic Operations of a Computer System

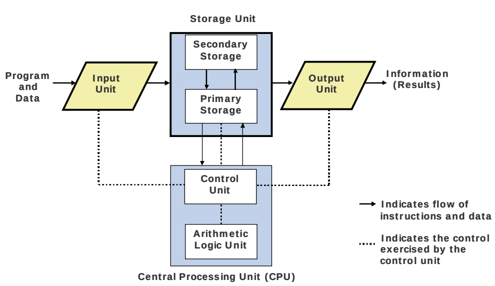
- **Inputting.** The process of entering data and instructions into the computer system
- **Storing.** Saving data and instructions to make them readily available for initial or additional processing whenever required
- **Processing.** Performing arithmetic operations (add, subtract, multiply, divide, etc.) or logical operations (comparisons like equal to, less than, greater than, etc.) on data to convert them into useful information

(Continued on next slide)

The Five Basic Operations of a Computer System

- **Outputting.** The process of producing useful information or results for the user such as a printed report or visual display
- **Controlling.** Directing the manner and sequence in which all of the above operations are performed

Basic Organization of a Computer System



Input Unit

An input unit of a computer system performs the following functions:

1. It accepts (or reads) instructions and data from outside world
2. It converts these instructions and data in computer acceptable form
3. It supplies the converted instructions and data to the computer system for further processing

Output Unit

An output unit of a computer system performs the following functions:

1. It accepts the results produced by the computer, which are in coded form and hence, cannot be easily understood by us
2. It converts these coded results to human acceptable (readable) form
3. It supplies the converted results to outside world

Storage Unit

The storage unit of a computer system holds (or stores) the following :

1. Data and instructions required for processing (received from input devices)
2. Intermediate results of processing
3. Final results of processing, before they are released to an output device

Two Types of Storage

- Primary storage
  - Used to hold running program instructions
  - Used to hold data, intermediate results, and results of ongoing processing of job(s)
  - Fast in operation
  - Small Capacity
  - Expensive
  - Volatile (looses data on power dissipation)

(Continued on next slide)

Two Types of Storage

(Continued from previous slide..)

- Secondary storage
  - Used to hold stored program instructions
  - Used to hold data and information of stored jobs
  - Slower than primary storage
  - Large Capacity
  - Lot cheaper that primary storage
  - Retains data even without power

Arithmetic Logic Unit (ALU)

Arithmetic Logic Unit of a computer system is the place where the actual executions of instructions takes place during processing operation

Control Unit (CU)

Control Unit of a computer system manages and coordinates the operations of all other components of the computer system

Central Processing Unit (CPU)

Arithmetic Logic Unit (ALU) + Control Unit (CU) = Central Processing Unit (CPU)

- It is the brain of a computer system
- It is responsible for controlling the operations of all other units of a computer system

The System Concept

A system has following three characteristics:

1. A system has more than one element
2. All elements of a system are logically related
3. All elements of a system are controlled in a manner to achieve the system goal

A computer is a system as it comprises of integrated components (input unit, output unit, storage unit, and CPU) that work together to perform the steps called for in the executing program

Key Words/Phrases

- Arithmetic Logic Unit (ALU)
- Auxiliary storage
- Central Processing Unit (CPU)
- Computer system
- Control Unit (CU)
- Controlling
- Input interface
- Input unit
- Inputting
- Main memory
- Output interface
- Output unit
- Outputting
- Primate storage
- Processing
- Secondary storage
- Storage unit
- Storing
- System

Chapter 03

Number Systems

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Non-positional number system
- Positional number system
- Decimal number system
- Binary number system
- Octal number system
- Hexadecimal number system

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Convert a number's base
  - Another base to decimal base
  - Decimal base to another base
  - Some base to another base
- Shortcut methods for converting
  - Binary to octal number
  - Octal to binary number
  - Binary to hexadecimal number
  - Hexadecimal to binary number
- Fractional numbers in binary number system

Number Systems

Two types of number systems are:

- Non-positional number systems
- Positional number systems

Non-positional Number Systems

- Characteristics
  - Use symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5, etc
  - Each symbol represents the same value regardless of its position in the number
  - The symbols are simply added to find out the value of a particular number
- Difficulty
  - It is difficult to perform arithmetic with such a number system

Positional Number Systems

- Characteristics
  - Use only a few symbols called digits
  - These symbols represent different values depending on the position they occupy in the number

(Continued on next slide)

Positional Number Systems

(Continued from previous slide..)

- The value of each digit is determined by:
  1. The digit itself
  2. The position of the digit in the number
  3. The base of the number system

(base = total number of digits in the number system)

- The maximum value of a single digit is always equal to one less than the value of the base

Decimal Number System

Characteristics

- A positional number system
- Has 10 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Hence, its base = 10
- The maximum value of a single digit is 9 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (10)
- We use this number system in our day-to-day life

(Continued on next slide)

Decimal Number System

(Continued from previous slide..)

Example

$$2586_{10} = (2 \text{ x } 10^3) + (5 \text{ x } 10^2) + (8 \text{ x } 10^1) + (6 \text{ x } 10^0)$$

$$= 2000 + 500 + 80 + 6$$

Binary Number System

Characteristics

- A positional number system
- Has only 2 symbols or digits (0 and 1). Hence its base = 2
- The maximum value of a single digit is 1 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (2)
- This number system is used in computers

(Continued on next slide)

Binary Number System

(Continued from previous slide..)

Example

$$10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \times (1 \times 2^0)$$

$$= 16 + 0 + 4 + 0 + 1$$

$$= 21_{10}$$

Representing Numbers in Different Number Systems

In order to be specific about which number system we are referring to, it is a common practice to indicate the base as a subscript. Thus, we write:

$$10101_2 = 21_{10}$$

Bit

- Bit stands for **bi**nary digi**t**
- A bit in computer terminology means either a 0 or a 1
- A binary number consisting of *n* bits is called an n-bit number

Octal Number System

Characteristics

- A positional number system
- Has total 8 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7). Hence, its base = 8
- The maximum value of a single digit is 7 (one less than the value of the base
- Each position of a digit represents a specific power of the base (8)

(Continued on next slide)

Octal Number System

(Continued from previous slide..)

- Since there are only 8 digits, 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary

Example

$$2057_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$$

$$= 1024 + 0 + 40 + 7$$

$$= 1071_{10}$$

Hexadecimal Number System

Characteristics

- A positional number system
- Has total 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Hence its base = 16
- The symbols A, B, C, D, E and F represent the decimal values 10, 11, 12, 13, 14 and 15 respectively
- The maximum value of a single digit is 15 (one less than the value of the base)

(Continued on next slide)

Hexadecimal Number System

(Continued from previous slide..)

- Each position of a digit represents a specific power of the base (16)
- Since there are only 16 digits, 4 bits ($2^4 = 16$) are sufficient to represent any hexadecimal number in binary

Example

$$1AF_{16} = (1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$$

$$= 1 \times 256 + 10 \times 16 + 15 \times 1$$

$$= 256 + 160 + 15$$

$$= 431_{10}$$

Converting a Number of Another Base to a Decimal Number

Method

Step 1: Determine the column (positional) value of each digit

Step 2: Multiply the obtained column values by the digits in the corresponding columns

Step 3: Calculate the sum of these products

(Continued on next slide)

Converting a Number of Another Base to a Decimal Number

(Continued from previous slide..)

Example

$4706_8 = ?_{10}$

Common values multiplied by the corresponding digits

$$\begin{aligned}4706_8 &= 4 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 6 \times 8^0 \\ &= 4 \times 512 + 7 \times 64 + 0 + 6 \times 1 \\ &= 2048 + 448 + 0 + 6 \\ &= 2502_{10}\end{aligned}$$

Sum of these products

Converting a Decimal Number to a Number of Another Base

Division-Remainder Method

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number

Step 3: Divide the quotient of the previous divide by the new base

(Continued on next slide)

Converting a Decimal Number to a Number of Another Base

(Continued from previous slide..)

Step 4: Record the remainder from Step 3 as the next digit (to the left) of the new base number

Repeat Steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in Step 3

Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

(Continued on next slide)

Converting a Decimal Number to a Number of Another Base

(Continued from previous slide..)

Example

$952_{10} = ?_8$

Solution:

		Remainders
8	952	
	119	0
	14	7
	1	6
	0	1

Hence, $952_{10} = 1670_8$

Converting a Number of Some Base to a Number of Another Base

Method

Step 1: Convert the original number to a decimal number (base 10)

Step 2: Convert the decimal number so obtained to the new base number

(Continued on next slide)

Converting a Number of Some Base to a Number of Another Base

(Continued from previous slide..)

Example

$545_6 = ?_4$

Solution:

Step 1: Convert from base 6 to base 10

$$\begin{aligned} 545_6 &= 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0 \\ &= 5 \times 36 + 4 \times 6 + 5 \times 1 \\ &= 180 + 24 + 5 \\ &= 209_{10} \end{aligned}$$

(Continued on next slide)

Converting a Number of Some Base to a Number of Another Base

(Continued from previous slide..)

Step 2: Convert 209_{10} to base 4

		Remainders
4	209	
	52	1
	13	0
	3	1
	0	3

Hence, $209_{10} = 3101_4$

So, $545_6 = 209_{10} = 3101_4$

Thus, $545_6 = 3101_4$

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

Method

Step 1: Divide the digits into groups of three starting from the right

Step 2: Convert each group of three binary digits to one octal digit using the method of binary to decimal conversion

(Continued on next slide)

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

(Continued from previous slide..)

Example

$1101010_2 = ?_8$

Step 1: Divide the binary digits into groups of 3 starting from right

<u>001</u> <u>101</u> <u>010</u>

Step 2: Convert each group into one octal digit

$001_2 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$

$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$

$010_2 = 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2$

Hence, $1101010_2 = 152_8$

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

Method

Step 1: Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion)

Step 2: Combine all the resulting binary groups (of 3 digits each) into a single binary number

(Continued on next slide)

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

(Continued from previous slide..)

Example

$562_8 = ?_2$

Step 1: Convert each octal digit to 3 binary digits

$5_8 = 101_2$, $6_8 = 110_2$, $2_8 = 010_2$

Step 2: Combine the binary groups

$$562_8 = \underset{5}{\underline{101}} \quad \underset{6}{\underline{110}} \quad \underset{2}{\underline{010}}$$

Hence, $562_8 = 101110010_2$

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

Method

Step 1: Divide the binary digits into groups of four starting from the right

Step 2: Combine each group of four binary digits to one hexadecimal digit

(Continued on next slide)

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

(Continued from previous slide..)

Example

$111101_2 = ?_{16}$

Step 1: Divide the binary digits into groups of four starting from the right

<u>0011</u> <u>1101</u>

Step 2: Convert each group into a hexadecimal digit

$0011_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10} = 3_{16}$

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 3_{10} = D_{16}$

Hence, $111101_2 = 3D_{16}$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

Method

Step 1: Convert the decimal equivalent of each hexadecimal digit to a 4 digit binary number

Step 2: Combine all the resulting binary groups (of 4 digits each) in a single binary number

(Continued on next slide)

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

(Continued from previous slide..)

Example

$2AB_{16} = ?_2$

Step 1: Convert each hexadecimal digit to a 4 digit binary number

$2_{16} = 2_{10} = 0010_2$
$A_{16} = 10_{10} = 1010_2$
$B_{16} = 11_{10} = 1011_2$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

(Continued from previous slide..)

Step 2: Combine the binary groups

$$2AB_{16} = \underset{2}{\underline{0010}} \quad \underset{A}{\underline{1010}} \quad \underset{B}{\underline{1011}}$$

Hence, $2AB_{16} = 001010101011_2$

Fractional Numbers

Fractional numbers are formed same way as decimal number system

In general, a number in a number system with base *b* would be written as:

$a_n \; a_{n-1} \ldots \; a_0 \; . \; a_{-1} \; a_{-2} \ldots \; a_{-m}$

And would be interpreted to mean:

$a_n \times b^n + a_{n-1} \times b^{n-1} + \ldots + a_0 \times b^0 + a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \ldots + a_{-m} \times b^{-m}$

The symbols a_n, a_{n-1}, …, a_{-m} in above representation should be one of the *b* symbols allowed in the number system

Formation of Fractional Numbers in Binary Number System (Example)

Binary Point

Position	4	3	2	1	0	.	-1	-2	-3	-4
Position Value	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
Quantity Represented	16	8	4	2	1		$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$

(Continued on next slide)

Formation of Fractional Numbers in Binary Number System (Example)

(Continued from previous slide..)

Example

$$110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$
$$= 4 + 2 + 0 + 0.5 + 0 + 0.125$$
$$= 6.625_{10}$$

Formation of Fractional Numbers in Octal Number System (Example)

Octal Point

Position	3	2	1	0	.	-1	-2	-3
Position Value	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
Quantity Represented	512	64	8	1		$^1/_8$	$^1/_{64}$	$^1/_{512}$

(Continued on next slide)

Formation of Fractional Numbers in Octal Number System (Example)

(Continued from previous slide..)

Example

$$127.54_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 5 \times 8^{-1} + 4 \times 8^{-2}$$

$$= 64 + 16 + 7 + {}^5/_8 + {}^4/_{64}$$

$$= 87 + 0.625 + 0.0625$$

$$= 87.6875_{10}$$

Key Words/Phrases

- Base
- Binary number system
- Binary point
- Bit
- Decimal number system
- Division-Remainder technique
- Fractional numbers
- Hexadecimal number system
- Least Significant Digit (LSD)
- Memory dump
- Most Significant Digit (MSD)
- Non-positional number system
- Number system
- Octal number system
- Positional number system

Chapter 03

Number Systems

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- **Non-positional number system**
- **Positional number system**
- **Decimal number system**
- **Binary number system**
- **Octal number system**
- **Hexadecimal number system**

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Convert a number's base
  - Another base to decimal base
  - Decimal base to another base
  - Some base to another base
- Shortcut methods for converting
  - Binary to octal number
  - Octal to binary number
  - Binary to hexadecimal number
  - Hexadecimal to binary number
- Fractional numbers in binary number system

Number Systems

Two types of number systems are:

- Non-positional number systems
- Positional number systems

Non-positional Number Systems

- Characteristics
  - Use symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5, etc
  - Each symbol represents the same value regardless of its position in the number
  - The symbols are simply added to find out the value of a particular number
- Difficulty
  - It is difficult to perform arithmetic with such a number system

Positional Number Systems

- Characteristics
  - Use only a few symbols called digits
  - These symbols represent different values depending on the position they occupy in the number

(Continued on next slide)

Positional Number Systems

(Continued from previous slide..)

- The value of each digit is determined by:
  1. The digit itself
  2. The position of the digit in the number
  3. The base of the number system

(base = total number of digits in the number system)

- The maximum value of a single digit is always equal to one less than the value of the base

Decimal Number System

Characteristics

- A positional number system
- Has 10 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Hence, its base = 10
- The maximum value of a single digit is 9 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (10)
- We use this number system in our day-to-day life

(Continued on next slide)

Decimal Number System

(Continued from previous slide..)

Example

$2586_{10} = (2 \times 10^3) + (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0)$

$= 2000 + 500 + 80 + 6$

Binary Number System

Characteristics

- A positional number system
- Has only 2 symbols or digits (0 and 1). Hence its base = 2
- The maximum value of a single digit is 1 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (2)
- This number system is used in computers

(Continued on next slide)

Binary Number System

(Continued from previous slide..)

Example

$10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \times (1 \times 2^0)$

$= 16 + 0 + 4 + 0 + 1$

$= 21_{10}$

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Representing Numbers in Different Number Systems

In order to be specific about which number system we are referring to, it is a common practice to indicate the base as a subscript. Thus, we write:

$10101_2 = 21_{10}$

Bit

- Bit stands for **bi**nary digi**t**
- A bit in computer terminology means either a 0 or a 1
- A binary number consisting of *n* bits is called an n-bit number

Octal Number System

Characteristics

- **A positional number system**
- **Has total 8 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7). Hence, its base = 8**
- **The maximum value of a single digit is 7 (one less than the value of the base**
- **Each position of a digit represents a specific power of the base (8)**

(Continued on next slide)

Octal Number System

(Continued from previous slide..)

- Since there are only 8 digits, 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary

Example

$$2057_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$$

$$= 1024 + 0 + 40 + 7$$

$$= 1071_{10}$$

Hexadecimal Number System

Characteristics

- A positional number system
- Has total 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Hence its base = 16
- The symbols A, B, C, D, E and F represent the decimal values 10, 11, 12, 13, 14 and 15 respectively
- The maximum value of a single digit is 15 (one less than the value of the base)

(Continued on next slide)

Hexadecimal Number System

(Continued from previous slide..)

- Each position of a digit represents a specific power of the base (16)
- Since there are only 16 digits, 4 bits (2^4 = 16) are sufficient to represent any hexadecimal number in binary

Example

$1AF_{16}$ = $(1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$

= $1 \times 256 + 10 \times 16 + 15 \times 1$

= $256 + 160 + 15$

= 431_{10}

Converting a Number of Another Base to a Decimal Number

Method

Step 1: Determine the column (positional) value of each digit

Step 2: Multiply the obtained column values by the digits in the corresponding columns

Step 3: Calculate the sum of these products

(Continued on next slide)

Converting a Number of Another Base to a Decimal Number

(Continued from previous slide..)

Example

$4706_8 = ?_{10}$

Common values multiplied by the corresponding digits

$4706_8 = 4 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 6 \times 8^0$

$= 4 \times 512 + 7 \times 64 + 0 + 6 \times 1$

$= 2048 + 448 + 0 + 6$ ← Sum of these products

$= 2502_{10}$

Converting a Decimal Number to a Number of Another Base

Division-Remainder Method

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number

Step 3: Divide the quotient of the previous divide by the new base

(Continued on next slide)

Converting a Decimal Number to a Number of Another Base

(Continued from previous slide..)

Step 4: Record the remainder from Step 3 as the next digit (to the left) of the new base number

Repeat Steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in Step 3

Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Converting a Decimal Number to a Number of Another Base

(Continued from previous slide..)

Example

$952_{10} = ?_8$

Solution:

		Remainders
8	952	
	119	0
	14	7
	1	6
	0	1

Hence, $952_{10} = 1670_8$

Converting a Number of Some Base to a Number of Another Base

Method

Step 1: Convert the original number to a decimal number (base 10)

Step 2: Convert the decimal number so obtained to the new base number

(Continued on next slide)

Converting a Number of Some Base to a Number of Another Base

(Continued from previous slide..)

Example

$545_6 = ?_4$

Solution:

Step 1: Convert from base 6 to base 10

$$\begin{aligned} 545_6 &= 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0 \\ &= 5 \times 36 + 4 \times 6 + 5 \times 1 \\ &= 180 + 24 + 5 \\ &= 209_{10} \end{aligned}$$

(Continued on next slide)

Converting a Number of Some Base to a Number of Another Base

(Continued from previous slide..)

Step 2: Convert 209_{10} to base 4

Divisor	Number	Remainders
4	209	
	52	1
	13	0
	3	1
	0	3

Hence, $209_{10} = 3101_4$

So, $545_6 = 209_{10} = 3101_4$

Thus, $545_6 = 3101_4$

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

Method

Step 1: Divide the digits into groups of three starting from the right

Step 2: Convert each group of three binary digits to one octal digit using the method of binary to decimal conversion

(Continued on next slide)

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

(Continued from previous slide..)

Example

$1101010_2 = ?_8$

Step 1: Divide the binary digits into groups of 3 starting from right

<u>001</u>   <u>101</u>   <u>010</u>

Step 2: Convert each group into one octal digit

$001_2 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$
$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$
$010_2 = 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2$

Hence, $1101010_2 = 152_8$

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

Method

Step 1: Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion)

Step 2: Combine all the resulting binary groups (of 3 digits each) into a single binary number

(Continued on next slide)

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

(Continued from previous slide..)

Example

$562_8 = ?_2$

Step 1: Convert each octal digit to 3 binary digits

$5_8 = 101_2$, $6_8 = 110_2$, $2_8 = 010_2$

Step 2: Combine the binary groups

$562_8 = \underbrace{101}_{5}\ \underbrace{110}_{6}\ \underbrace{010}_{2}$

Hence, $562_8 = 101110010_2$

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

Method

Step 1: Divide the binary digits into groups of four starting from the right

Step 2: Combine each group of four binary digits to one hexadecimal digit

(Continued on next slide)

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

(Continued from previous slide..)

Example

$111101_2 = ?_{16}$

Step 1: Divide the binary digits into groups of four starting from the right

<u>0011</u>   <u>1101</u>

Step 2: Convert each group into a hexadecimal digit

$0011_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10} = 3_{16}$

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 3_{10} = D_{16}$

Hence, $111101_2 = 3D_{16}$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

Method

Step 1: Convert the decimal equivalent of each hexadecimal digit to a 4 digit binary number

Step 2: Combine all the resulting binary groups (of 4 digits each) in a single binary number

(Continued on next slide)

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

(Continued from previous slide..)

Example

$2AB_{16} = ?_2$

Step 1: Convert each hexadecimal digit to a 4 digit binary number

$2_{16} = 2_{10} = 0010_2$
$A_{16} = 10_{10} = 1010_2$
$B_{16} = 11_{10} = 1011_2$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

(Continued from previous slide..)

Step 2: Combine the binary groups

$$2AB_{16} = \underbrace{0010}_{2} \quad \underbrace{1010}_{A} \quad \underbrace{1011}_{B}$$

Hence, $2AB_{16} = 001010101011_2$

Fractional Numbers

Fractional numbers are formed same way as decimal number system

In general, a number in a number system with base *b* would be written as:
$a_n\, a_{n-1} \ldots a_0 \,.\, a_{-1}\, a_{-2} \ldots a_{-m}$

And would be interpreted to mean:
$a_n \times b^n + a_{n-1} \times b^{n-1} + \ldots + a_0 \times b^0 + a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \ldots + a_{-m} \times b^{-m}$

The symbols a_n, a_{n-1}, …, a_{-m} in above representation should be one of the *b* symbols allowed in the number system

Formation of Fractional Numbers in Binary Number System (Example)

Binary Point ↓

Position	4	3	2	1	0	.	-1	-2	-3	-4
Position Value	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
Quantity Represented	16	8	4	2	1		$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$

(Continued on next slide)

Formation of Fractional Numbers in Binary Number System (Example)

(Continued from previous slide..)

Example

$$110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$
$$= 4 + 2 + 0 + 0.5 + 0 + 0.125$$
$$= 6.625_{10}$$

Formation of Fractional Numbers in Octal Number System (Example)

Octal Point ↓

Position	3	2	1	0	.	-1	-2	-3
Position Value	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
Quantity Represented	512	64	8	1		$^1/_8$	$^1/_{64}$	$^1/_{512}$

(Continued on next slide)

Formation of Fractional Numbers in Octal Number System (Example)

(Continued from previous slide..)

Example

$$
\begin{aligned}
127.54_8 &= 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 5 \times 8^{-1} + 4 \times 8^{-2} \\
&= 64 + 16 + 7 + {}^5/_8 + {}^4/_{64} \\
&= 87 + 0.625 + 0.0625 \\
&= 87.6875_{10}
\end{aligned}
$$

Key Words / Phrases

- Base
- Binary number system
- Binary point
- Bit
- Decimal number system
- Division-Remainder technique
- Fractional numbers
- Hexadecimal number system
- Least Significant Digit (LSD)
- Memory dump
- Most Significant Digit (MSD)
- Non-positional number system
- Number system
- Octal number system
- Positional number system

Chapter 03

Number Systems

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Non-positional number system
- Positional number system
- Decimal number system
- Binary number system
- Octal number system
- Hexadecimal number system

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Convert a number's base
  - Another base to decimal base
  - Decimal base to another base
  - Some base to another base
- Shortcut methods for converting
  - Binary to octal number
  - Octal to binary number
  - Binary to hexadecimal number
  - Hexadecimal to binary number
- Fractional numbers in binary number system

Number Systems

Two types of number systems are:

- Non-positional number systems
- Positional number systems

Non-positional Number Systems

- Characteristics
  - Use symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5, etc
  - Each symbol represents the same value regardless of its position in the number
  - The symbols are simply added to find out the value of a particular number
- Difficulty
  - It is difficult to perform arithmetic with such a number system

Positional Number Systems

- Characteristics
  - Use only a few symbols called digits
  - These symbols represent different values depending on the position they occupy in the number

(Continued on next slide)

Positional Number Systems

(Continued from previous slide..)

- The value of each digit is determined by:
  1. The digit itself
  2. The position of the digit in the number
  3. The base of the number system

(base = total number of digits in the number system)

- The maximum value of a single digit is always equal to one less than the value of the base

Decimal Number System

Characteristics

- A positional number system
- Has 10 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Hence, its base = 10
- The maximum value of a single digit is 9 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (10)
- We use this number system in our day-to-day life

(Continued on next slide)

Decimal Number System

(Continued from previous slide..)

Example

$2586_{10} = (2 \times 10^3) + (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0)$

$= 2000 + 500 + 80 + 6$

Binary Number System

Characteristics

- A positional number system
- Has only 2 symbols or digits (0 and 1). Hence its base = 2
- The maximum value of a single digit is 1 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (2)
- This number system is used in computers

(Continued on next slide)

Binary Number System

(Continued from previous slide..)

Example

$10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \times (1 \times 2^0)$

$= 16 + 0 + 4 + 0 + 1$

$= 21_{10}$

Representing Numbers in Different Number Systems

In order to be specific about which number system we are referring to, it is a common practice to indicate the base as a subscript. Thus, we write:

$10101_2 = 21_{10}$

Bit

- Bit stands for **bi**nary digi**t**
- A bit in computer terminology means either a 0 or a 1
- A binary number consisting of *n* bits is called an n-bit number

Octal Number System

Characteristics

- **A positional number system**
- **Has total 8 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7). Hence, its base = 8**
- **The maximum value of a single digit is 7 (one less than the value of the base**
- **Each position of a digit represents a specific power of the base (8)**

(Continued on next slide)

Octal Number System

(Continued from previous slide..)

- **Since there are only 8 digits, 3 bits (2^3 = 8) are sufficient to represent any octal number in binary**

Example

$2057_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$

$= 1024 + 0 + 40 + 7$

$= 1071_{10}$

Hexadecimal Number System

Characteristics

- A positional number system
- Has total 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Hence its base = 16
- The symbols A, B, C, D, E and F represent the decimal values 10, 11, 12, 13, 14 and 15 respectively
- The maximum value of a single digit is 15 (one less than the value of the base)

(Continued on next slide)

Hexadecimal Number System

(Continued from previous slide..)

- Each position of a digit represents a specific power of the base (16)
- Since there are only 16 digits, 4 bits (2^4 = 16) are sufficient to represent any hexadecimal number in binary

Example

$1AF_{16}$ = $(1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$

= $1 \times 256 + 10 \times 16 + 15 \times 1$

= $256 + 160 + 15$

= 431_{10}

Converting a Number of Another Base to a Decimal Number

Method

Step 1: Determine the column (positional) value of each digit

Step 2: Multiply the obtained column values by the digits in the corresponding columns

Step 3: Calculate the sum of these products

(Continued on next slide)

Converting a Number of Another Base to a Decimal Number

(Continued from previous slide..)

Example

$4706_8 = ?_{10}$

Common values multiplied by the corresponding digits

$4706_8 = 4 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 6 \times 8^0$

$= 4 \times 512 + 7 \times 64 + 0 + 6 \times 1$

$= 2048 + 448 + 0 + 6$ ← Sum of these products

$= 2502_{10}$

Converting a Decimal Number to a Number of Another Base

Division-Remainder Method

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number

Step 3: Divide the quotient of the previous divide by the new base

(Continued on next slide)

Converting a Decimal Number to a Number of Another Base

(Continued from previous slide..)

Step 4: Record the remainder from Step 3 as the next digit (to the left) of the new base number

Repeat Steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in Step 3

Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

(Continued on next slide)

Converting a Decimal Number to a Number of Another Base

(Continued from previous slide..)

Example

$952_{10} = ?_8$

Solution:

		Remainders
8	952	
	119	0
	14	7
	1	6
	0	1

Hence, $952_{10} = 1670_8$

Converting a Number of Some Base to a Number of Another Base

Method

Step 1: Convert the original number to a decimal number (base 10)

Step 2: Convert the decimal number so obtained to the new base number

(Continued on next slide)

Converting a Number of Some Base to a Number of Another Base

(Continued from previous slide..)

Example

$545_6 = ?_4$

Solution:

Step 1: Convert from base 6 to base 10

$$\begin{aligned} 545_6 &= 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0 \\ &= 5 \times 36 + 4 \times 6 + 5 \times 1 \\ &= 180 + 24 + 5 \\ &= 209_{10} \end{aligned}$$

(Continued on next slide)

Converting a Number of Some Base to a Number of Another Base

(Continued from previous slide..)

Step 2: Convert 209_{10} to base 4

4	209	Remainders
	52	1
	13	0
	3	1
	0	3

Hence, $209_{10} = 3101_4$

So, $545_6 = 209_{10} = 3101_4$

Thus, $545_6 = 3101_4$

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

Method

Step 1: Divide the digits into groups of three starting from the right

Step 2: Convert each group of three binary digits to one octal digit using the method of binary to decimal conversion

(Continued on next slide)

Shortcut Method for Converting a Binary Number to its Equivalent Octal Number

(Continued from previous slide..)

Example

$1101010_2 = ?_8$

Step 1: Divide the binary digits into groups of 3 starting from right

<u>001</u> <u>101</u> <u>010</u>

Step 2: Convert each group into one octal digit

$001_2 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$
$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$
$010_2 = 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2$

Hence, $1101010_2 = 152_8$

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

Method

Step 1: Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion)

Step 2: Combine all the resulting binary groups (of 3 digits each) into a single binary number

(Continued on next slide)

Shortcut Method for Converting an Octal Number to Its Equivalent Binary Number

(Continued from previous slide..)

Example

$562_8 = ?_2$

Step 1: Convert each octal digit to 3 binary digits

$5_8 = 101_2$, $6_8 = 110_2$, $2_8 = 010_2$

Step 2: Combine the binary groups

$$562_8 = \underset{5}{\underline{101}}\ \ \underset{6}{\underline{110}}\ \ \underset{2}{\underline{010}}$$

Hence, $562_8 = 101110010_2$

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

Method

Step 1: Divide the binary digits into groups of four starting from the right

Step 2: Combine each group of four binary digits to one hexadecimal digit

(Continued on next slide)

Shortcut Method for Converting a Binary Number to its Equivalent Hexadecimal Number

(Continued from previous slide..)

Example

$111101_2 = ?_{16}$

Step 1: Divide the binary digits into groups of four starting from the right

<u>0011</u> <u>1101</u>

Step 2: Convert each group into a hexadecimal digit

$0011_2 = 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3_{10} = 3_{16}$

$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 3_{10} = D_{16}$

Hence, $111101_2 = 3D_{16}$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

Method

Step 1: Convert the decimal equivalent of each hexadecimal digit to a 4 digit binary number

Step 2: Combine all the resulting binary groups (of 4 digits each) in a single binary number

(Continued on next slide)

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

(Continued from previous slide..)

Example

$2AB_{16} = ?_2$

Step 1: Convert each hexadecimal digit to a 4 digit binary number

$2_{16} = 2_{10} = 0010_2$

$A_{16} = 10_{10} = 1010_2$

$B_{16} = 11_{10} = 1011_2$

Shortcut Method for Converting a Hexadecimal Number to its Equivalent Binary Number

(Continued from previous slide..)

Step 2: Combine the binary groups

$2AB_{16}$ = <u>0010</u> <u>1010</u> <u>1011</u>

$2AB_{16}$ =	<u>0010</u>	<u>1010</u>	<u>1011</u>
	2	A	B

Hence, $2AB_{16} = 001010101011_2$

Fractional Numbers

- ***Fractional numbers*** are formed same way as decimal number system
- In general, a number in a number system with base *b* would be written as:
  $a_n\ a_{n-1} \ldots\ a_0\ .\ a_{-1}\ a_{-2} \ldots\ a_{-m}$

  And would be interpreted to mean:
  $a_n \times b^n + a_{n-1} \times b^{n-1} + \ldots + a_0 \times b^0 + a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \ldots + a_{-m} \times b^{-m}$
- The symbols a_n, a_{n-1}, …, a_{-m} in above representation should be one of the *b* symbols allowed in the number system

Formation of Fractional Numbers in Binary Number System (Example)

Binary Point ↓

Position	4	3	2	1	0 .	-1	-2	-3	-4
Position Value	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Quantity Represented	16	8	4	2	1	$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$

(Continued on next slide)

Formation of Fractional Numbers in Binary Number System (Example)

(Continued from previous slide..)

Example

$$110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$
$$= 4 + 2 + 0 + 0.5 + 0 + 0.125$$
$$= 6.625_{10}$$

Formation of Fractional Numbers in Octal Number System (Example)

Octal Point

Position	3	2	1	0	.	-1	-2	-3
Position Value	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
Quantity Represented	512	64	8	1		$^1/_8$	$^1/_{64}$	$^1/_{512}$

(Continued on next slide)

Formation of Fractional Numbers in Octal Number System (Example)

(Continued from previous slide..)

Example

$$127.54_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 5 \times 8^{-1} + 4 \times 8^{-2}$$
$$= 64 + 16 + 7 + {}^5/_8 + {}^4/_{64}$$
$$= 87 + 0.625 + 0.0625$$
$$= 87.6875_{10}$$

Key Words/Phrases

- Base
- Binary number system
- Binary point
- Bit
- Decimal number system
- Division-Remainder technique
- Fractional numbers
- Hexadecimal number system
- Least Significant Digit (LSD)
- Memory dump
- Most Significant Digit (MSD)
- Non-positional number system
- Number system
- Octal number system
- Positional number system

Chapter 04

Computer Codes

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer data
- Computer codes: representation of data in binary
- Most commonly used computer codes
- Collating sequence

Data Types

- **Numeric Data** consists of only numbers 0, 1, 2, …, 9
- **Alphabetic Data** consists of only the letters A, B, C, …, Z, in both uppercase and lowercase, and blank character
- **Alphanumeric Data** is a string of symbols where a symbol may be one of the letters A, B, C, …, Z, in either uppercase or lowercase, or one of the digits 0, 1, 2, …, 9, or a special character, such as + - * / , . ( ) = etc.

Computer Codes

- Computer codes are used for internal representation of data in computers
- As computers use binary numbers for internal data representation, computer codes use binary coding schemes
- In binary coding, every symbol that appears in the data is represented by a group of bits
- The group of bits used to represent a symbol is called a byte

(Continued on next slide)

Computer Codes

(Continued from previous slide..)

- As most modern coding schemes use 8 bits to represent a symbol, the term byte is often used to mean a group of 8 bits
- Commonly used computer codes are BCD, EBCDIC, and ASCII

BCD

- BCD stands for Binary Coded Decimal
- It is one of the early computer codes
- It uses 6 bits to represent a symbol
- It can represent 64 (2^6) different characters

Coding of Alphabetic and Numeric Characters in BCD

Char	BCD Code		Octal
	Zone	Digit	
A	11	0001	61
B	11	0010	62
C	11	0011	63
D	11	0100	64
E	11	0101	65
F	11	0110	66
G	11	0111	67
H	11	1000	70
I	11	1001	71
J	10	0001	41
K	10	0010	42
L	10	0011	43
M	10	0100	44

Char	BCD Code		Octal
	Zone	Digit	
N	10	0101	45
O	10	0110	46
P	10	0111	47
Q	10	1000	50
R	10	1001	51
S	01	0010	22
T	01	0011	23
U	01	0100	24
V	01	0101	25
W	01	0110	26
X	01	0111	27
Y	01	1000	30
Z	01	1001	31

(Continued on next slide)

Coding of Alphabetic and Numeric Characters in BCD

(Continued from previous slide..)

Character	BCD Code		Octal Equivalent
	Zone	Digit	
1	00	0001	01
2	00	0010	02
3	00	0011	03
4	00	0100	04
5	00	0101	05
6	00	0110	06
7	00	0111	07
8	00	1000	10
9	00	1001	11
0	00	1010	12

BCD Coding Scheme (Example 1)

Example

Show the binary digits used to record the word BASE in BCD

Solution:

B = 110010 in BCD binary notation
A = 110001 in BCD binary notation
S = 010010 in BCD binary notation
E = 110101 in BCD binary notation

So the binary digits

110010	110001	010010	110101
B	A	S	E

will record the word BASE in BCD

BCD Coding Scheme (Example 2)

Example

Using octal notation, show BCD coding for the word DIGIT

Solution:

D = 64 in BCD octal notation
I = 71 in BCD octal notation
G = 67 in BCD octal notation
I = 71 in BCD octal notation
T = 23 in BCD octal notation

Hence, BCD coding for the word DIGIT in octal notation will be

64	71	67	71	23
D	I	G	I	T

EBCDIC

- EBCDIC stands for Extended Binary Coded Decimal Interchange Code
- It uses 8 bits to represent a symbol
- It can represent 256 (2^8) different characters

Coding of Alphabetic and Numeric Characters in EBCDIC

Char	EBCDIC Code		Hex
	Digit	Zone	
A	1100	0001	C1
B	1100	0010	C2
C	1100	0011	C3
D	1100	0100	C4
E	1100	0101	C5
F	1100	0110	C6
G	1100	0111	C7
H	1100	1000	C8
I	1100	1001	C9
J	1101	0001	D1
K	1101	0010	D2
L	1101	0011	D3
M	1101	0100	D4

Char	EBCDIC Code		Hex
	Digit	Zone	
N	1101	0101	D5
O	1101	0110	D6
P	1101	0111	D7
Q	1101	1000	D8
R	1101	1001	D9
S	1110	0010	E2
T	1110	0011	E3
U	1110	0100	E4
V	1110	0101	E5
W	1110	0110	E6
X	1110	0111	E7
Y	1110	1000	E8
Z	1110	1001	E9

(Continued on next slide)

Coding of Alphabetic and Numeric Characters in EBCDIC

(Continued from previous slide..)

Character	EBCDIC Code		Hexadecimal Equivalent
	Digit	Zone	
0	1111	0000	F0
1	1111	0001	F1
2	1111	0010	F2
3	1111	0011	F3
4	1111	0100	F4
5	1111	0101	F5
6	1111	0110	F6
7	1111	0111	F7
8	1111	1000	F8
9	1111	1001	F9

Zoned Decimal Numbers

- Zoned decimal numbers are used to represent numeric values (positive, negative, or unsigned) in EBCDIC
- A sign indicator (C for plus, D for minus, and F for unsigned) is used in the zone position of the rightmost digit
- Zones for all other digits remain as F, the zone value for numeric characters in EBCDIC
- In zoned format, there is only one digit per byte

Examples Zoned Decimal Numbers

Numeric Value	EBCDIC	Sign Indicator
345	F3F4F5	F for unsigned
+345	F3F4C5	C for positive
-345	F3F4D5	D for negative

Packed Decimal Numbers

- Packed decimal numbers are formed from zoned decimal numbers in the following manner:

  Step 1: The zone half and the digit half of the rightmost byte are reversed

  Step 2: All remaining zones are dropped out

- Packed decimal format requires fewer number of bytes than zoned decimal format for representing a number
- Numbers represented in packed decimal format can be used for arithmetic operations

Examples of Conversion of Zoned Decimal Numbers to Packed Decimal Format

Numeric Value	EBCDIC	Sign Indicator
345	F3F4F5	345F
+345	F3F4C5	345C
-345	F3F4D5	345D
3456	F3F4F5F6	03456F

EBCDIC Coding Scheme

Example

Using binary notation, write EBCDIC coding for the word BIT. How many bytes are required for this representation?

Solution:

B = 1100 0010 in EBCDIC binary notation
I = 1100 1001 in EBCDIC binary notation
T = 1110 0011 in EBCDIC binary notation

Hence, EBCDIC coding for the word BIT in binary notation will be

11000010	**11001001**	**11100011**
B	**I**	**T**

3 bytes will be required for this representation because each letter requires 1 byte (or 8 bits)

ASCII

- ASCII stands for American Standard Code for Information Interchange.
- ASCII is of two types – ASCII-7 and ASCII-8
- ASCII-7 uses 7 bits to represent a symbol and can represent 128 (2^7) different characters
- ASCII-8 uses 8 bits to represent a symbol and can represent 256 (2^8) different characters
- First 128 characters in ASCII-7 and ASCII-8 are same

Coding of Numeric and Alphabetic Characters in ASCII

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
0	0011	0000	30
1	0011	0001	31
2	0011	0010	32
3	0011	0011	33
4	0011	0100	34
5	0011	0101	35
6	0011	0110	36
7	0011	0111	37
8	0011	1000	38
9	0011	1001	39

(Continued on next slide)

Coding of Numeric and Alphabetic Characters in ASCII

(Continued from previous slide..)

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
A	0100	0001	41
B	0100	0010	42
C	0100	0011	43
D	0100	0100	44
E	0100	0101	45
F	0100	0110	46
G	0100	0111	47
H	0100	1000	48
I	0100	1001	49
J	0100	1010	4A
K	0100	1011	4B
L	0100	1100	4C
M	0100	1101	4D

(Continued on next slide)

Coding of Numeric and Alphabetic Characters in ASCII

(Continued from previous slide..)

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
N	0100	1110	4E
O	0100	1111	4F
P	0101	0000	50
Q	0101	0001	51
R	0101	0010	52
S	0101	0011	53
T	0101	0100	54
U	0101	0101	55
V	0101	0110	56
W	0101	0111	57
X	0101	1000	58
Y	0101	1001	59
Z	0101	1010	5A

ASCII-7 Coding Scheme

Example

Write binary coding for the word BOY in ASCII-7. How many bytes are required for this representation?

Solution:

B = 1000010 in ASCII-7 binary notation
O = 1001111 in ASCII-7 binary notation
Y = 1011001 in ASCII-7 binary notation

Hence, binary coding for the word BOY in ASCII-7 will be

1000010	1001111	1011001
B	O	Y

Since each character in ASCII-7 requires one byte for its representation and there are 3 characters in the word BOY, 3 bytes will be required for this representation

ASCII-8 Coding Scheme

Example

Write binary coding for the word SKY in ASCII-8. How many bytes are required for this representation?

Solution:

S = 01010011 in ASCII-8 binary notation
K = 01001011 in ASCII-8 binary notation
Y = 01011001 in ASCII-8 binary notation

Hence, binary coding for the word SKY in ASCII-8 will be

01010011	01001011	01011001
S	K	Y

Since each character in ASCII-8 requires one byte for its representation and there are 3 characters in the word SKY, 3 bytes will be required for this representation

Unicode

- Why Unicode:
  - No single encoding system supports all languages
  - Different encoding systems conflict
- Unicode features:
  - Provides a consistent way of encoding multilingual plain text
  - Defines codes for characters used in all major languages of the world
  - Defines codes for special characters, mathematical symbols, technical symbols, and diacritics

Unicode

- Unicode features (continued):
  - Capacity to encode as many as a million characters
  - Assigns each character a unique numeric value and name
  - Reserves a part of the code space for private use
  - Affords simplicity and consistency of ASCII, even corresponding characters have same code
  - Specifies an algorithm for the presentation of text with bi-directional behavior
- Encoding Forms
  - UTF-8, UTF-16, UTF-32

Collating Sequence

- Collating sequence defines the assigned ordering among the characters used by a computer

- Collating sequence may vary, depending on the type of computer code used by a particular computer

- In most computers, collating sequences follow the following rules:

  1. Letters are considered in alphabetic order
     (A < B < C … < Z)

  2. Digits are considered in numeric order
     (0 < 1 < 2 … < 9)

Sorting in EBCDIC

Example

Suppose a computer uses EBCDIC as its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A?

Solution:

In EBCDIC, numeric characters are treated to be greater than alphabetic characters. Hence, in the said computer, numeric characters will be placed after alphabetic characters and the given string will be treated as:

A1 < 1A < 23

Therefore, the sorted sequence will be: A1, 1A, 23.

Sorting in ASCII

Example

Suppose a computer uses ASCII for its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A, a2, 2a, aA, and Aa?

Solution:

In ASCII, numeric characters are treated to be less than alphabetic characters. Hence, in the said computer, numeric characters will be placed before alphabetic characters and the given string will be treated as:

1A < 23 < 2a < A1 < Aa < a2 < aA

Therefore, the sorted sequence will be: 1A, 23, 2a, A1, Aa, a2, and aA

Key Words/Phrases

- Alphabetic data
- Alphanumeric data
- American Standard Code for Information Interchange (ASCII)
- Binary Coded Decimal (BCD) code
- Byte
- Collating sequence
- Computer codes
- Control characters
- Extended Binary-Coded Decimal Interchange Code (EBCDIC)
- Hexadecimal equivalent
- Numeric data
- Octal equivalent
- Packed decimal numbers
- Unicode
- Zoned decimal numbers

Chapter 04

Computer Codes

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer data
- Computer codes: representation of data in binary
- Most commonly used computer codes
- Collating sequence

Data Types

- **Numeric Data** consists of only numbers 0, 1, 2, …, 9
- **Alphabetic Data** consists of only the letters A, B, C, …, Z, in both uppercase and lowercase, and blank character
- **Alphanumeric Data** is a string of symbols where a symbol may be one of the letters A, B, C, …, Z, in either uppercase or lowercase, or one of the digits 0, 1, 2, …, 9, or a special character, such as + - * / , . ( ) = etc.

Computer Codes

- Computer codes are used for internal representation of data in computers
- As computers use binary numbers for internal data representation, computer codes use binary coding schemes
- In binary coding, every symbol that appears in the data is represented by a group of bits
- The group of bits used to represent a symbol is called a byte

(Continued on next slide)

Computer Codes

(Continued from previous slide..)

- As most modern coding schemes use 8 bits to represent a symbol, the term byte is often used to mean a group of 8 bits
- Commonly used computer codes are BCD, EBCDIC, and ASCII

BCD

- BCD stands for Binary Coded Decimal
- It is one of the early computer codes
- It uses 6 bits to represent a symbol
- It can represent 64 (2^6) different characters

Coding of Alphabetic and Numeric Characters in BCD

Char	BCD Code		Octal
	Zone	Digit	
A	11	0001	61
B	11	0010	62
C	11	0011	63
D	11	0100	64
E	11	0101	65
F	11	0110	66
G	11	0111	67
H	11	1000	70
I	11	1001	71
J	10	0001	41
K	10	0010	42
L	10	0011	43
M	10	0100	44

Char	BCD Code		Octal
	Zone	Digit	
N	10	0101	45
O	10	0110	46
P	10	0111	47
Q	10	1000	50
R	10	1001	51
S	01	0010	22
T	01	0011	23
U	01	0100	24
V	01	0101	25
W	01	0110	26
X	01	0111	27
Y	01	1000	30
Z	01	1001	31

(Continued on next slide)

Coding of Alphabetic and Numeric Characters in BCD

(Continued from previous slide..)

Character	BCD Code		Octal Equivalent
	Zone	Digit	
1	00	0001	01
2	00	0010	02
3	00	0011	03
4	00	0100	04
5	00	0101	05
6	00	0110	06
7	00	0111	07
8	00	1000	10
9	00	1001	11
0	00	1010	12

BCD Coding Scheme (Example 1)

Example

Show the binary digits used to record the word BASE in BCD

Solution:

B = 110010 in BCD binary notation
A = 110001 in BCD binary notation
S = 010010 in BCD binary notation
E = 110101 in BCD binary notation

So the binary digits

110010	110001	010010	110101
B	A	S	E

will record the word BASE in BCD

BCD Coding Scheme (Example 2)

Example

Using octal notation, show BCD coding for the word DIGIT

Solution:

D = 64 in BCD octal notation
I = 71 in BCD octal notation
G = 67 in BCD octal notation
I = 71 in BCD octal notation
T = 23 in BCD octal notation

Hence, BCD coding for the word DIGIT in octal notation will be

64	71	67	71	23
D	I	G	I	T

EBCDIC

- EBCDIC stands for Extended Binary Coded Decimal Interchange Code
- It uses 8 bits to represent a symbol
- It can represent 256 (2^8) different characters

Coding of Alphabetic and Numeric Characters in EBCDIC

Char	EBCDIC Code		Hex
	Digit	Zone	
A	1100	0001	C1
B	1100	0010	C2
C	1100	0011	C3
D	1100	0100	C4
E	1100	0101	C5
F	1100	0110	C6
G	1100	0111	C7
H	1100	1000	C8
I	1100	1001	C9
J	1101	0001	D1
K	1101	0010	D2
L	1101	0011	D3
M	1101	0100	D4

Char	EBCDIC Code		Hex
	Digit	Zone	
N	1101	0101	D5
O	1101	0110	D6
P	1101	0111	D7
Q	1101	1000	D8
R	1101	1001	D9
S	1110	0010	E2
T	1110	0011	E3
U	1110	0100	E4
V	1110	0101	E5
W	1110	0110	E6
X	1110	0111	E7
Y	1110	1000	E8
Z	1110	1001	E9

(Continued on next slide)

Coding of Alphabetic and Numeric Characters in EBCDIC

(Continued from previous slide..)

Character	EBCDIC Code		Hexadecimal Equivalent
	Digit	Zone	
0	1111	0000	F0
1	1111	0001	F1
2	1111	0010	F2
3	1111	0011	F3
4	1111	0100	F4
5	1111	0101	F5
6	1111	0110	F6
7	1111	0111	F7
8	1111	1000	F8
9	1111	1001	F9

Zoned Decimal Numbers

- Zoned decimal numbers are used to represent numeric values (positive, negative, or unsigned) in EBCDIC
- A sign indicator (C for plus, D for minus, and F for unsigned) is used in the zone position of the rightmost digit
- Zones for all other digits remain as F, the zone value for numeric characters in EBCDIC
- In zoned format, there is only one digit per byte

Examples Zoned Decimal Numbers

Numeric Value	EBCDIC	Sign Indicator
345	F3F4F5	F for unsigned
+345	F3F4C5	C for positive
-345	F3F4D5	D for negative

Packed Decimal Numbers

- Packed decimal numbers are formed from zoned decimal numbers in the following manner:

  Step 1: The zone half and the digit half of the rightmost byte are reversed

  Step 2: All remaining zones are dropped out

- Packed decimal format requires fewer number of bytes than zoned decimal format for representing a number
- Numbers represented in packed decimal format can be used for arithmetic operations

Examples of Conversion of Zoned Decimal Numbers to Packed Decimal Format

Numeric Value	EBCDIC	Sign Indicator
345	F3F4F5	345F
+345	F3F4C5	345C
-345	F3F4D5	345D
3456	F3F4F5F6	03456F

EBCDIC Coding Scheme

Example

Using binary notation, write EBCDIC coding for the word BIT. How many bytes are required for this representation?

Solution:

B = 1100 0010 in EBCDIC binary notation
I = 1100 1001 in EBCDIC binary notation
T = 1110 0011 in EBCDIC binary notation

Hence, EBCDIC coding for the word BIT in binary notation will be

11000010	11001001	11100011
B	I	T

3 bytes will be required for this representation because each letter requires 1 byte (or 8 bits)

ASCII

- ASCII stands for American Standard Code for Information Interchange.
- ASCII is of two types – ASCII-7 and ASCII-8
- ASCII-7 uses 7 bits to represent a symbol and can represent 128 (2^7) different characters
- ASCII-8 uses 8 bits to represent a symbol and can represent 256 (2^8) different characters
- First 128 characters in ASCII-7 and ASCII-8 are same

Coding of Numeric and Alphabetic Characters in ASCII

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
0	0011	0000	30
1	0011	0001	31
2	0011	0010	32
3	0011	0011	33
4	0011	0100	34
5	0011	0101	35
6	0011	0110	36
7	0011	0111	37
8	0011	1000	38
9	0011	1001	39

(Continued on next slide)

Coding of Numeric and Alphabetic Characters in ASCII

(Continued from previous slide..)

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
A	0100	0001	41
B	0100	0010	42
C	0100	0011	43
D	0100	0100	44
E	0100	0101	45
F	0100	0110	46
G	0100	0111	47
H	0100	1000	48
I	0100	1001	49
J	0100	1010	4A
K	0100	1011	4B
L	0100	1100	4C
M	0100	1101	4D

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Coding of Numeric and Alphabetic Characters in ASCII

(Continued from previous slide..)

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
N	0100	1110	4E
O	0100	1111	4F
P	0101	0000	50
Q	0101	0001	51
R	0101	0010	52
S	0101	0011	53
T	0101	0100	54
U	0101	0101	55
V	0101	0110	56
W	0101	0111	57
X	0101	1000	58
Y	0101	1001	59
Z	0101	1010	5A

ASCII-7 Coding Scheme

Example

Write binary coding for the word BOY in ASCII-7. How many bytes are required for this representation?

Solution:

B = 1000010 in ASCII-7 binary notation
O = 1001111 in ASCII-7 binary notation
Y = 1011001 in ASCII-7 binary notation

Hence, binary coding for the word BOY in ASCII-7 will be

1000010	1001111	1011001
B	O	Y

Since each character in ASCII-7 requires one byte for its representation and there are 3 characters in the word BOY, 3 bytes will be required for this representation

ASCII-8 Coding Scheme

Example

Write binary coding for the word SKY in ASCII-8. How many bytes are required for this representation?

Solution:

S = 01010011 in ASCII-8 binary notation
K = 01001011 in ASCII-8 binary notation
Y = 01011001 in ASCII-8 binary notation

Hence, binary coding for the word SKY in ASCII-8 will be

01010011	01001011	01011001
S	K	Y

Since each character in ASCII-8 requires one byte for its representation and there are 3 characters in the word SKY, 3 bytes will be required for this representation

Unicode

- **Why Unicode:**
  - No single encoding system supports all languages
  - Different encoding systems conflict
- **Unicode features:**
  - Provides a consistent way of encoding multilingual plain text
  - Defines codes for characters used in all major languages of the world
  - Defines codes for special characters, mathematical symbols, technical symbols, and diacritics

Unicode

- **Unicode features (continued):**
  - Capacity to encode as many as a million characters
  - Assigns each character a unique numeric value and name
  - Reserves a part of the code space for private use
  - Affords simplicity and consistency of ASCII, even corresponding characters have same code
  - Specifies an algorithm for the presentation of text with bi-directional behavior
- **Encoding Forms**
  - UTF-8, UTF-16, UTF-32

Collating Sequence

- Collating sequence defines the assigned ordering among the characters used by a computer
- Collating sequence may vary, depending on the type of computer code used by a particular computer
- In most computers, collating sequences follow the following rules:
  1. Letters are considered in alphabetic order (A < B < C … < Z)
  2. Digits are considered in numeric order (0 < 1 < 2 … < 9)

Sorting in EBCDIC

Example

Suppose a computer uses EBCDIC as its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A?

Solution:

In EBCDIC, numeric characters are treated to be greater than alphabetic characters. Hence, in the said computer, numeric characters will be placed after alphabetic characters and the given string will be treated as:

A1 < 1A < 23

Therefore, the sorted sequence will be: A1, 1A, 23.

Sorting in ASCII

Example

Suppose a computer uses ASCII for its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A, a2, 2a, aA, and Aa?

Solution:

In ASCII, numeric characters are treated to be less than alphabetic characters. Hence, in the said computer, numeric characters will be placed before alphabetic characters and the given string will be treated as:

1A < 23 < 2a < A1 < Aa < a2 < aA

Therefore, the sorted sequence will be: 1A, 23, 2a, A1, Aa, a2, and aA

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Key Words / Phrases

- Alphabetic data
- Alphanumeric data
- American Standard Code for Information Interchange (ASCII)
- Binary Coded Decimal (BCD) code
- Byte
- Collating sequence
- Computer codes
- Control characters
- Extended Binary-Coded Decimal Interchange Code (EBCDIC)
- Hexadecimal equivalent
- Numeric data
- Octal equivalent
- Packed decimal numbers
- Unicode
- Zoned decimal numbers

Chapter 04

Computer Codes

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer data
- Computer codes: representation of data in binary
- Most commonly used computer codes
- Collating sequence

Data Types

- **Numeric Data** consists of only numbers 0, 1, 2, …, 9
- **Alphabetic Data** consists of only the letters A, B, C, …, Z, in both uppercase and lowercase, and blank character
- **Alphanumeric Data** is a string of symbols where a symbol may be one of the letters A, B, C, …, Z, in either uppercase or lowercase, or one of the digits 0, 1, 2, …, 9, or a special character, such as + - * / , . ( ) = etc.

Computer Codes

- Computer codes are used for internal representation of data in computers
- As computers use binary numbers for internal data representation, computer codes use binary coding schemes
- In binary coding, every symbol that appears in the data is represented by a group of bits
- The group of bits used to represent a symbol is called a byte

(Continued on next slide)

Computer Codes

(Continued from previous slide..)

- As most modern coding schemes use 8 bits to represent a symbol, the term byte is often used to mean a group of 8 bits
- Commonly used computer codes are BCD, EBCDIC, and ASCII

BCD

- BCD stands for Binary Coded Decimal
- It is one of the early computer codes
- It uses 6 bits to represent a symbol
- It can represent 64 (2^6) different characters

Coding of Alphabetic and Numeric Characters in BCD

Char	BCD Code Zone	BCD Code Digit	Octal
A	11	0001	61
B	11	0010	62
C	11	0011	63
D	11	0100	64
E	11	0101	65
F	11	0110	66
G	11	0111	67
H	11	1000	70
I	11	1001	71
J	10	0001	41
K	10	0010	42
L	10	0011	43
M	10	0100	44

Char	BCD Code Zone	BCD Code Digit	Octal
N	10	0101	45
O	10	0110	46
P	10	0111	47
Q	10	1000	50
R	10	1001	51
S	01	0010	22
T	01	0011	23
U	01	0100	24
V	01	0101	25
W	01	0110	26
X	01	0111	27
Y	01	1000	30
Z	01	1001	31

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Coding of Alphabetic and Numeric Characters in BCD

(Continued from previous slide..)

Character	BCD Code Zone	BCD Code Digit	Octal Equivalent
1	00	0001	01
2	00	0010	02
3	00	0011	03
4	00	0100	04
5	00	0101	05
6	00	0110	06
7	00	0111	07
8	00	1000	10
9	00	1001	11
0	00	1010	12

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

BCD Coding Scheme (Example 1)

Example

Show the binary digits used to record the word BASE in BCD

Solution:

B = 110010 in BCD binary notation
A = 110001 in BCD binary notation
S = 010010 in BCD binary notation
E = 110101 in BCD binary notation

So the binary digits

$\underset{B}{\underline{110010}}\ \underset{A}{\underline{110001}}\ \underset{S}{\underline{010010}}\ \underset{E}{\underline{110101}}$

will record the word BASE in BCD

BCD Coding Scheme (Example 2)

Example

Using octal notation, show BCD coding for the word DIGIT

Solution:

D = 64 in BCD octal notation
I = 71 in BCD octal notation
G = 67 in BCD octal notation
I = 71 in BCD octal notation
T = 23 in BCD octal notation

Hence, BCD coding for the word DIGIT in octal notation will be

64	71	67	71	23
D	I	G	I	T

EBCDIC

- EBCDIC stands for Extended Binary Coded Decimal Interchange Code
- It uses 8 bits to represent a symbol
- It can represent 256 (2^8) different characters

Coding of Alphabetic and Numeric Characters in EBCDIC

Char	EBCDIC Code		Hex
	Digit	Zone	
A	1100	0001	C1
B	1100	0010	C2
C	1100	0011	C3
D	1100	0100	C4
E	1100	0101	C5
F	1100	0110	C6
G	1100	0111	C7
H	1100	1000	C8
I	1100	1001	C9
J	1101	0001	D1
K	1101	0010	D2
L	1101	0011	D3
M	1101	0100	D4

Char	EBCDIC Code		Hex
	Digit	Zone	
N	1101	0101	D5
O	1101	0110	D6
P	1101	0111	D7
Q	1101	1000	D8
R	1101	1001	D9
S	1110	0010	E2
T	1110	0011	E3
U	1110	0100	E4
V	1110	0101	E5
W	1110	0110	E6
X	1110	0111	E7
Y	1110	1000	E8
Z	1110	1001	E9

(Continued on next slide)

Coding of Alphabetic and Numeric Characters in EBCDIC

(Continued from previous slide..)

Character	EBCDIC Code		Hexadecimal Equivalent
	Digit	Zone	
0	1111	0000	F0
1	1111	0001	F1
2	1111	0010	F2
3	1111	0011	F3
4	1111	0100	F4
5	1111	0101	F5
6	1111	0110	F6
7	1111	0111	F7
8	1111	1000	F8
9	1111	1001	F9

Zoned Decimal Numbers

- Zoned decimal numbers are used to represent numeric values (positive, negative, or unsigned) in EBCDIC
- A sign indicator (C for plus, D for minus, and F for unsigned) is used in the zone position of the rightmost digit
- Zones for all other digits remain as F, the zone value for numeric characters in EBCDIC
- In zoned format, there is only one digit per byte

Examples Zoned Decimal Numbers

Numeric Value	EBCDIC	Sign Indicator
345	F3F4F5	F for unsigned
+345	F3F4C5	C for positive
-345	F3F4D5	D for negative

Packed Decimal Numbers

- Packed decimal numbers are formed from zoned decimal numbers in the following manner:

  Step 1: The zone half and the digit half of the rightmost byte are reversed

  Step 2: All remaining zones are dropped out

- Packed decimal format requires fewer number of bytes than zoned decimal format for representing a number
- Numbers represented in packed decimal format can be used for arithmetic operations

Examples of Conversion of Zoned Decimal Numbers to Packed Decimal Format

Numeric Value	EBCDIC	Sign Indicator
345	F3F4F5	345F
+345	F3F4C5	345C
-345	F3F4D5	345D
3456	F3F4F5F6	03456F

EBCDIC Coding Scheme

Example

Using binary notation, write EBCDIC coding for the word BIT. How many bytes are required for this representation?

Solution:

B = 1100 0010 in EBCDIC binary notation
I = 1100 1001 in EBCDIC binary notation
T = 1110 0011 in EBCDIC binary notation

Hence, EBCDIC coding for the word BIT in binary notation will be

$\underbrace{11000010}_{B}\ \underbrace{11001001}_{I}\ \underbrace{11100011}_{T}$

3 bytes will be required for this representation because each letter requires 1 byte (or 8 bits)

ASCII

- ASCII stands for American Standard Code for Information Interchange.
- ASCII is of two types – ASCII-7 and ASCII-8
- ASCII-7 uses 7 bits to represent a symbol and can represent 128 (2^7) different characters
- ASCII-8 uses 8 bits to represent a symbol and can represent 256 (2^8) different characters
- First 128 characters in ASCII-7 and ASCII-8 are same

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Coding of Numeric and Alphabetic Characters in ASCII

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
0	0011	0000	30
1	0011	0001	31
2	0011	0010	32
3	0011	0011	33
4	0011	0100	34
5	0011	0101	35
6	0011	0110	36
7	0011	0111	37
8	0011	1000	38
9	0011	1001	39

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Coding of Numeric and Alphabetic Characters in ASCII

(Continued from previous slide..)

Character	ASCII-7 / ASCII-8		Hexadecimal Equivalent
	Zone	Digit	
A	0100	0001	41
B	0100	0010	42
C	0100	0011	43
D	0100	0100	44
E	0100	0101	45
F	0100	0110	46
G	0100	0111	47
H	0100	1000	48
I	0100	1001	49
J	0100	1010	4A
K	0100	1011	4B
L	0100	1100	4C
M	0100	1101	4D

(Continued on next slide)

Coding of Numeric and Alphabetic Characters in ASCII

(Continued from previous slide..)

Character	ASCII-7 / ASCII-8 Zone	ASCII-7 / ASCII-8 Digit	Hexadecimal Equivalent
N	0100	1110	4E
O	0100	1111	4F
P	0101	0000	50
Q	0101	0001	51
R	0101	0010	52
S	0101	0011	53
T	0101	0100	54
U	0101	0101	55
V	0101	0110	56
W	0101	0111	57
X	0101	1000	58
Y	0101	1001	59
Z	0101	1010	5A

ASCII-7 Coding Scheme

Example

Write binary coding for the word BOY in ASCII-7. How many bytes are required for this representation?

Solution:

B = 1000010 in ASCII-7 binary notation
O = 1001111 in ASCII-7 binary notation
Y = 1011001 in ASCII-7 binary notation

Hence, binary coding for the word BOY in ASCII-7 will be

$\underbrace{1000010}_{B}\ \underbrace{1001111}_{O}\ \underbrace{1011001}_{Y}$

Since each character in ASCII-7 requires one byte for its representation and there are 3 characters in the word BOY, 3 bytes will be required for this representation

ASCII-8 Coding Scheme

Example

Write binary coding for the word SKY in ASCII-8. How many bytes are required for this representation?

Solution:

S = 01010011 in ASCII-8 binary notation
K = 01001011 in ASCII-8 binary notation
Y = 01011001 in ASCII-8 binary notation

Hence, binary coding for the word SKY in ASCII-8 will be

$\underbrace{01010011}_{S}\ \underbrace{01001011}_{K}\ \underbrace{01011001}_{Y}$

Since each character in ASCII-8 requires one byte for its representation and there are 3 characters in the word SKY, 3 bytes will be required for this representation

Unicode

- Why Unicode:
  - No single encoding system supports all languages
  - Different encoding systems conflict
- Unicode features:
  - Provides a consistent way of encoding multilingual plain text
  - Defines codes for characters used in all major languages of the world
  - Defines codes for special characters, mathematical symbols, technical symbols, and diacritics

Unicode

- Unicode features (continued):
  - Capacity to encode as many as a million characters
  - Assigns each character a unique numeric value and name
  - Reserves a part of the code space for private use
  - Affords simplicity and consistency of ASCII, even corresponding characters have same code
  - Specifies an algorithm for the presentation of text with bi-directional behavior
- Encoding Forms
  - UTF-8, UTF-16, UTF-32

Collating Sequence

- Collating sequence defines the assigned ordering among the characters used by a computer
- Collating sequence may vary, depending on the type of computer code used by a particular computer
- In most computers, collating sequences follow the following rules:

  1. Letters are considered in alphabetic order (A < B < C … < Z)
  2. Digits are considered in numeric order (0 < 1 < 2 … < 9)

Sorting in EBCDIC

Example

Suppose a computer uses EBCDIC as its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A?

Solution:

In EBCDIC, numeric characters are treated to be greater than alphabetic characters. Hence, in the said computer, numeric characters will be placed after alphabetic characters and the given string will be treated as:

A1 < 1A < 23

Therefore, the sorted sequence will be: A1, 1A, 23.

Sorting in ASCII

Example

Suppose a computer uses ASCII for its internal representation of characters. In which order will this computer sort the strings 23, A1, 1A, a2, 2a, aA, and Aa?

Solution:

In ASCII, numeric characters are treated to be less than alphabetic characters. Hence, in the said computer, numeric characters will be placed before alphabetic characters and the given string will be treated as:

1A < 23 < 2a < A1 < Aa < a2 < aA

Therefore, the sorted sequence will be: 1A, 23, 2a, A1, Aa, a2, and aA

Key Words / Phrases

- Alphabetic data
- Alphanumeric data
- American Standard Code for Information Interchange (ASCII)
- Binary Coded Decimal (BCD) code
- Byte
- Collating sequence
- Computer codes
- Control characters
- Extended Binary-Coded Decimal Interchange Code (EBCDIC)
- Hexadecimal equivalent
- Numeric data
- Octal equivalent
- Packed decimal numbers
- Unicode
- Zoned decimal numbers

Chapter 05

Computer Arithmetic

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

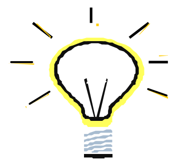
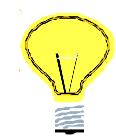
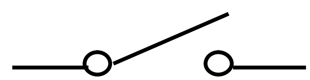
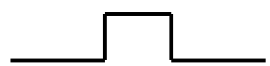
In this chapter you will learn about:

- Reasons for using binary instead of decimal numbers
- Basic arithmetic operations using binary numbers
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)

Binary over Decimal

- Information is handled in a computer by electronic/ electrical components
- Electronic components operate in binary mode (can only indicate two states – on (1) or off (0)
- Binary number system has only two digits (0 and 1), and is suitable for expressing two possible states
- In binary system, computer circuits only have to handle two binary digits rather than ten decimal digits causing:
  - Simpler internal circuit design
  - Less expensive
  - More reliable circuits
- Arithmetic rules/processes possible with binary numbers

Examples of a Few Devices that work in Binary Mode

Binary State	On (1)	Off (0)
Bulb		
Switch		
Circuit Pulse		

Binary Arithmetic

- Binary arithmetic is simple to learn as binary number system has only two digits – 0 and 1

- Following slides show rules and example for the four basic arithmetic operations using binary numbers

Binary Addition

Rule for binary addition is as follows:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 plus a carry of 1 to next higher column

Binary Addition (Example 1)

Example

Add binary numbers 10011 and 1001 in both decimal and binary form

Solution

Binary	Decimal
carry 11	carry 1
10011	19
+1001	+9
11100	28

In this example, carry are generated for first and second columns

Binary Addition (Example 2)

Example

Add binary numbers 100111 and 11011 in both decimal and binary form

Solution

	Binary	Decimal
carry	11111	carry 1
	100111	39
	+11011	+27
	1000010	66

The addition of three 1s can be broken up into two steps. First, we add only two 1s giving 10 (1 + 1 = 10). The third 1 is now added to this result to obtain 11 (a 1 sum with a 1 carry). Hence, 1 + 1 + 1 = 1, plus a carry of 1 to next higher column.

Binary Subtraction

Rule for binary subtraction is as follows:

0 - 0 = 0
0 - 1 = 1 with a borrow from the next column
1 - 0 = 1
1 - 1 = 0

Binary Subtraction (Example)

Example

Subtract 01110_2 from 10101_2

Solution

```
  12
 0202
 10101
-01110
------
 00111
------
```

Note: Go through explanation given in the book

Complement of a Number

Number of digits
in the number

$$C = B^n - 1 - N$$

Complement
of the number

Base of the
number

The number

Complement of a Number (Example 1)

Example

Find the complement of 37_{10}

Solution

Since the number has 2 digits and the value of base is 10,

$(\text{Base})^n - 1 = 10^2 - 1 = 99$

Now $99 - 37 = 62$

Hence, complement of $37_{10} = 62_{10}$

Complement of a Number (Example 2)

Example

Find the complement of 6_8

Solution

Since the number has 1 digit and the value of base is 8,

$$(\text{Base})^n - 1 = 8^1 - 1 = 7_{10} = 7_8$$

$$\text{Now } 7_8 - 6_8 = 1_8$$

Hence, complement of $6_8 = 1_8$

Complement of a Binary Number

Complement of a binary number can be obtained by transforming all its 0's to 1's and all its 1's to 0's

Example

Complement of 1 0 1 1 0 1 0 is

↓ ↓ ↓ ↓ ↓ ↓ ↓

0 1 0 0 1 0 1

Note: Verify by conventional complement

Complementary Method of Subtraction

Involves following 3 steps:

Step 1: Find the complement of the number you are subtracting (subtrahend)

Step 2: Add this to the number from which you are taking away (minuend)

Step 3: If there is a carry of 1, add it to obtain the result; if there is no carry, recomplement the sum and attach a negative sign

Complementary subtraction is an additive approach of subtraction

Complementary Subtraction (Example 1)

Example:

Subtract 56_{10} from 92_{10} using complementary method.

Solution

Step 1: Complement of 56_{10}

$= 10^2 - 1 - 56 = 99 – 56 = 43_{10}$

Step 2: 92 + 43 (complement of 56)

= 135 (note 1 as carry)

Step 3: 35 + 1 (add 1 carry to sum)

Result = 36

The result may be verified using the method of normal subtraction:

92 - 56 = 36

Complementary Subtraction (Example 2)

Example

Subtract 35_{10} from 18_{10} using complementary method.

Solution

Step 1: Complement of 35_{10}

$= 10^2 - 1 - 35$

$= 99 - 35$

$= 64_{10}$

Step 2:

$$\begin{array}{r} 18 \\ +\ 64 \\ \hline 82 \\ \hline \end{array}$$

(complement of 35)

Step 3: Since there is no carry, re-complement the sum and attach a negative sign to obtain the result.

Result $= -(99 - 82)$

$= -17$

The result may be verified using normal subtraction:

$18 - 35 = -17$

Binary Subtraction Using Complementary Method (Example 1)

Example

Subtract 0111000_2 (56_{10}) from 1011100_2 (92_{10}) using complementary method.

Solution

```
  1011100
 +1000111   (complement of 0111000)
 --------
 10100011
 |
 +-------->  1   (add the carry of 1)
 --------
  0100100
 --------
```

Result = 0100100_2 = 36_{10}

Binary Subtraction Using Complementary Method (Example 2)

Example

Subtract 100011_2 (35_{10}) from 010010_2 (18_{10}) using complementary method.

Solution

$$\begin{array}{rl} 010010 & \\ +011100 & \text{(complement of 100011)} \\ \hline 101110 & \\ \hline \end{array}$$

Since there is no carry, we have to complement the sum and attach a negative sign to it. Hence,

$$\begin{aligned} \text{Result} &= -010001_2 \text{ (complement of } 101110_2\text{)} \\ &= -17_{10} \end{aligned}$$

Binary Multiplication

Table for binary multiplication is as follows:

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

Binary Multiplication (Example 1)

Example

Multiply the binary numbers 1010 and 1001

Solution

```
       1010     Multiplicand
      x1001     Multiplier
    ---------
       1010     Partial Product
      0000      Partial Product
     0000       Partial Product
    1010        Partial Product
    ---------
    1011010     Final Product
    ---------
```

(Continued on next slide)

Binary Multiplication (Example 2)

(Continued from previous slide..)

Whenever a 0 appears in the multiplier, a separate partial product consisting of a string of zeros need not be generated (only a shift will do). Hence,

```
     1010
    x1001
  -------
     1010
   1010SS   (S = left shift)
  -------
  1011010
  -------
```

Binary Division

Table for binary division is as follows:

0 ÷ 0 = Divide by zero error
0 ÷ 1 = 0
1 ÷ 0 = Divide by zero error
1 ÷ 1 = 1

As in the decimal number system (or in any other number system), division by zero is meaningless

The computer deals with this problem by raising an error condition called ‘Divide by zero’ error

Rules for Binary Division

1. Start from the left of the dividend
2. Perform a series of subtractions in which the divisor is subtracted from the dividend
3. If subtraction is possible, put a 1 in the quotient and subtract the divisor from the corresponding digits of dividend
4. If subtraction is not possible (divisor greater than remainder), record a 0 in the quotient
5. Bring down the next digit to add to the remainder digits. Proceed as before in a manner similar to long division

Binary Division (Example 1)

Example

Divide 100001_2 by 110_2

Solution

```
          0101    (Quotient)
      ________
110 ) 100001   (Dividend)
      110       1 <------   Divisor greater than 100, so put 0 in quotient
      ______
      1000      2 <------   Add digit from dividend to group used above
       110      3 <------   Subtraction possible, so put 1 in quotient
      ______
        100     4 <------   Remainder from subtraction plus digit from dividend
        110     5 <------   Divisor greater, so put 0 in quotient
      ______
        1001    6 <------   Add digit from dividend to group
         110    7 <------   Subtraction possible, so put 1 in quotient
      ______
          11      Remainder
      ______
```

Additive Method of Multiplication and Division

Most computers use the additive method for performing multiplication and division operations because it simplifies the internal circuit design of computer systems

Example

$4 \times 8 = 8 + 8 + 8 + 8 = 32$

Rules for Additive Method of Division

- Subtract the divisor repeatedly from the dividend until the result of subtraction becomes less than or equal to zero
- If result of subtraction is zero, then:
  - quotient = total number of times subtraction was performed
  - remainder = 0
- If result of subtraction is less than zero, then:
  - quotient = total number of times subtraction was performed minus 1
  - remainder = result of the subtraction previous to the last subtraction

Additive Method of Division (Example)

Example

Divide 33_{10} by 6_{10} using the method of addition

Solution:

33 - 6 = 27
27 - 6 = 21
21 - 6 = 15
15 - 6 = 9
9 - 6 = 3
3 - 6 = -3

Total subtractions = 6

Since the result of the last subtraction is less than zero,

Quotient = 6 - 1 (ignore last subtraction) = 5

Remainder = 3 (result of previous subtraction)

Key Words/Phrases

- Additive method of division
- Additive method of multiplication
- Additive method of subtraction
- Binary addition
- Binary arithmetic
- Binary division
- Binary multiplication
- Binary subtraction
- Complement
- Complementary subtraction
- Computer arithmetic

Chapter 05

Computer Arithmetic

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

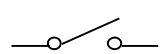
In this chapter you will learn about:

- Reasons for using binary instead of decimal numbers
- Basic arithmetic operations using binary numbers
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)

Binary over Decimal

- Information is handled in a computer by electronic/ electrical components
- Electronic components operate in binary mode (can only indicate two states – on (1) or off (0)
- Binary number system has only two digits (0 and 1), and is suitable for expressing two possible states
- In binary system, computer circuits only have to handle two binary digits rather than ten decimal digits causing:
  - Simpler internal circuit design
  - Less expensive
  - More reliable circuits
- Arithmetic rules/processes possible with binary numbers

Examples of a Few Devices that work in Binary Mode

Binary State	On (1)	Off (0)
Bulb		
Switch		
Circuit Pulse		

Binary Arithmetic

- Binary arithmetic is simple to learn as binary number system has only two digits – 0 and 1
- Following slides show rules and example for the four basic arithmetic operations using binary numbers

Binary Addition

Rule for binary addition is as follows:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 plus a carry of 1 to next higher column

Binary Addition (Example 1)

Example

Add binary numbers 10011 and 1001 in both decimal and binary form

Solution

Binary	Decimal
carry 11	carry 1
10011	19
+1001	+9
11100	28

In this example, carry are generated for first and second columns

Binary Addition (Example 2)

Example

Add binary numbers 100111 and 11011 in both decimal and binary form

Solution

Binary	Decimal
carry 11111	carry 1
100111	39
+11011	+27
1000010	66

The addition of three 1s can be broken up into two steps. First, we add only two 1s giving 10 (1 + 1 = 10). The third 1 is now added to this result to obtain 11 (a 1 sum with a 1 carry). Hence, 1 + 1 + 1 = 1, plus a carry of 1 to next higher column.

Binary Subtraction

Rule for binary subtraction is as follows:

0 - 0 = 0
0 - 1 = 1 with a borrow from the next column
1 - 0 = 1
1 - 1 = 0

Binary Subtraction (Example)

Example

Subtract 01110_2 from 10101_2

Solution

```
  12
 0202
 10101
-01110
------
 00111
------
```

Note: Go through explanation given in the book

Complement of a Number

Number of digits
in the number

$$C = B^n - 1 - N$$

Complement
of the number

Base of the
number

The number

Complement of a Number (Example 1)

Example

Find the complement of 37_{10}

Solution

Since the number has 2 digits and the value of base is 10,

$(\text{Base})^n - 1 = 10^2 - 1 = 99$

Now $99 - 37 = 62$

Hence, complement of $37_{10} = 62_{10}$

Complement of a Number (Example 2)

Example

Find the complement of 6_8

Solution

Since the number has 1 digit and the value of base is 8,

$(\text{Base})^n - 1 = 8^1 - 1 = 7_{10} = 7_8$

Now $7_8 - 6_8 = 1_8$

Hence, complement of $6_8 = 1_8$

Complement of a Binary Number

Complement of a binary number can be obtained by transforming all its 0's to 1's and all its 1's to 0's

Example

Complement of 1 0 1 1 0 1 0 is

↓ ↓ ↓ ↓ ↓ ↓ ↓

0 1 0 0 1 0 1

Note: Verify by conventional complement

Complementary Method of Subtraction

Involves following 3 steps:

Step 1: Find the complement of the number you are subtracting (subtrahend)

Step 2: Add this to the number from which you are taking away (minuend)

Step 3: If there is a carry of 1, add it to obtain the result; if there is no carry, recomplement the sum and attach a negative sign

Complementary subtraction is an additive approach of subtraction

Complementary Subtraction (Example 1)

Example:

Subtract 56_{10} from 92_{10} using complementary method.

Solution

Step 1: Complement of 56_{10}
$= 10^2 - 1 - 56 = 99 - 56 = 43_{10}$

Step 2: 92 + 43 (complement of 56)
= 135 (note 1 as carry)

Step 3: 35 + 1 (add 1 carry to sum)

Result = 36

The result may be verified using the method of normal subtraction:

92 - 56 = 36

Complementary Subtraction (Example 2)

Example

Subtract 35_{10} from 18_{10} using complementary method.

Solution

Step 1: Complement of 35_{10}
= 10^2 - 1 - 35
= 99 - 35
= 64_{10}

Step 2:
```
   18
 + 64   (complement
 ----          of 35)
   82
```

Step 3: Since there is no carry, re-complement the sum and attach a negative sign to obtain the result.

Result = -(99 - 82)
= -17

The result may be verified using normal subtraction:

18 - 35 = -17

Binary Subtraction Using Complementary Method (Example 1)

Example

Subtract 0111000_2 (56_{10}) from 1011100_2 (92_{10}) using complementary method.

Solution

```
  1011100
 +1000111   (complement of 0111000)
 --------
 10100011
 |
 └──────→ 1   (add the carry of 1)
 --------
  0100100
 --------
```

Result = 0100100_2 = 36_{10}

Binary Subtraction Using Complementary Method (Example 2)

Example

Subtract 100011_2 (35_{10}) from 010010_2 (18_{10}) using complementary method.

Solution

```
 010010
+011100  (complement of 100011)
-------
 101110
-------
```

Since there is no carry, we have to complement the sum and attach a negative sign to it. Hence,

Result = -010001_2 (complement of 101110_2)
= -17_{10}

Binary Multiplication

Table for binary multiplication is as follows:

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

Binary Multiplication (Example 1)

Example

Multiply the binary numbers 1010 and 1001

Solution

```
      1010    Multiplicand
     x1001    Multiplier
   -------
      1010    Partial Product
     0000     Partial Product
    0000      Partial Product
   1010       Partial Product
   -------
   1011010    Final Product
   -------
```

(Continued on next slide)

Binary Multiplication (Example 2)

(Continued from previous slide..)

Whenever a 0 appears in the multiplier, a separate partial product consisting of a string of zeros need not be generated (only a shift will do). Hence,

```
    1010
   x1001
  ------
    1010
  1010SS   (S = left shift)
  ------
  1011010
  ------
```

Binary Division

Table for binary division is as follows:

0 ÷ 0 = Divide by zero error
0 ÷ 1 = 0
1 ÷ 0 = Divide by zero error
1 ÷ 1 = 1

As in the decimal number system (or in any other number system), division by zero is meaningless

The computer deals with this problem by raising an error condition called 'Divide by zero' error

Rules for Binary Division

1. Start from the left of the dividend
2. Perform a series of subtractions in which the divisor is subtracted from the dividend
3. If subtraction is possible, put a 1 in the quotient and subtract the divisor from the corresponding digits of dividend
4. If subtraction is not possible (divisor greater than remainder), record a 0 in the quotient
5. Bring down the next digit to add to the remainder digits. Proceed as before in a manner similar to long division

Binary Division (Example 1)

Example

Divide 100001_2 by 110_2

Solution

```
          0101     (Quotient)
    110 ) 100001   (Dividend)
          110      1 <---   Divisor greater than 100, so put 0 in quotient
          ----
          1000     2 <---   Add digit from dividend to group used above
           110     3 <---   Subtraction possible, so put 1 in quotient
          ----
            100    4 <---   Remainder from subtraction plus digit from dividend
            110    5 <---   Divisor greater, so put 0 in quotient
          ----
            1001   6 <---   Add digit from dividend to group
             110   7 <---   Subtraction possible, so put 1 in quotient
          ----
              11      Remainder
          ----
```


Additive Method of Multiplication and Division

Most computers use the additive method for performing multiplication and division operations because it simplifies the internal circuit design of computer systems

Example

4 x 8 = 8 + 8 + 8 + 8 = 32

Rules for Additive Method of Division

- Subtract the divisor repeatedly from the dividend until the result of subtraction becomes less than or equal to zero
- If result of subtraction is zero, then:
  - quotient = total number of times subtraction was performed
  - remainder = 0
- If result of subtraction is less than zero, then:
  - quotient = total number of times subtraction was performed minus 1
  - remainder = result of the subtraction previous to the last subtraction

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Additive Method of Division (Example)

Example

Divide 33_{10} by 6_{10} using the method of addition

Solution:

33 - 6 = 27
27 - 6 = 21
21 - 6 = 15
15 - 6 = 9
9 - 6 = 3
3 - 6 = -3

Total subtractions = 6

Since the result of the last subtraction is less than zero,

Quotient = 6 - 1 (ignore last subtraction) = 5

Remainder = 3 (result of previous subtraction)

Key Words/Phrases

- Additive method of division
- Additive method of multiplication
- Additive method of subtraction
- Binary addition
- Binary arithmetic
- Binary division
- Binary multiplication
- Binary subtraction
- Complement
- Complementary subtraction
- Computer arithmetic

Chapter 05

Computer Arithmetic

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

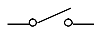
In this chapter you will learn about:

- **Reasons for using binary instead of decimal numbers**
- **Basic arithmetic operations using binary numbers**
  - **Addition (+)**
  - **Subtraction (-)**
  - **Multiplication (*)**
  - **Division (/)**

Binary over Decimal

- **Information is handled in a computer by electronic/ electrical components**
- **Electronic components operate in binary mode (can only indicate two states – on (1) or off (0)**
- **Binary number system has only two digits (0 and 1), and is suitable for expressing two possible states**
- **In binary system, computer circuits only have to handle two binary digits rather than ten decimal digits causing:**
  - **Simpler internal circuit design**
  - **Less expensive**
  - **More reliable circuits**
- **Arithmetic rules/processes possible with binary numbers**

Examples of a Few Devices that work in Binary Mode

Binary State	On (1)	Off (0)
Bulb		
Switch		
Circuit Pulse		

Binary Arithmetic

- Binary arithmetic is simple to learn as binary number system has only two digits – 0 and 1
- Following slides show rules and example for the four basic arithmetic operations using binary numbers

Binary Addition

Rule for binary addition is as follows:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 plus a carry of 1 to next higher column

Binary Addition (Example 1)

Example

Add binary numbers 10011 and 1001 in both decimal and binary form

Solution

Binary	Decimal
carry 11	carry 1
10011	19
+1001	+9
11100	28

In this example, carry are generated for first and second columns

Binary Addition (Example 2)

Example

Add binary numbers 100111 and 11011 in both decimal and binary form

Solution

Binary	Decimal
carry 11111	carry 1
100111	39
+11011	+27
1000010	66

The addition of three 1s can be broken up into two steps. First, we add only two 1s giving 10 (1 + 1 = 10). The third 1 is now added to this result to obtain 11 (a 1 sum with a 1 carry). Hence, 1 + 1 + 1 = 1, plus a carry of 1 to next higher column.

Binary Subtraction

Rule for binary subtraction is as follows:

0 - 0 = 0
0 - 1 = 1 with a borrow from the next column
1 - 0 = 1
1 - 1 = 0

Binary Subtraction (Example)

Example

Subtract 01110_2 from 10101_2

Solution

```
   12
  0202
  10101
 -01110
 ------
  00111
```

Note: Go through explanation given in the book

Complement of a Number

$$C = B^n - 1 - N$$

- C: Complement of the number
- n: Number of digits in the number
- B: Base of the number
- N: The number

Complement of a Number (Example 1)

Example

Find the complement of 37_{10}

Solution

Since the number has 2 digits and the value of base is 10,

$(Base)^n - 1 = 10^2 - 1 = 99$

Now $99 - 37 = 62$

Hence, complement of $37_{10} = 62_{10}$

Complement of a Number (Example 2)

Example

Find the complement of 6_8

Solution

Since the number has 1 digit and the value of base is 8,

$(Base)^n - 1 = 8^1 - 1 = 7_{10} = 7_8$

Now $7_8 - 6_8 = 1_8$

Hence, complement of $6_8 = 1_8$

Complement of a Binary Number

Complement of a binary number can be obtained by transforming all its 0's to 1's and all its 1's to 0's

Example

Complement of 1 0 1 1 0 1 0 is

↓ ↓ ↓ ↓ ↓ ↓ ↓

0 1 0 0 1 0 1

Note: Verify by conventional complement

Complementary Method of Subtraction

Involves following 3 steps:

Step 1: Find the complement of the number you are subtracting (subtrahend)

Step 2: Add this to the number from which you are taking away (minuend)

Step 3: If there is a carry of 1, add it to obtain the result; if there is no carry, recomplement the sum and attach a negative sign

Complementary subtraction is an additive approach of subtraction

Complementary Subtraction (Example 1)

Example:

Subtract 56_{10} from 92_{10} using complementary method.

Solution

Step 1: Complement of 56_{10}
$= 10^2 - 1 - 56 = 99 – 56 = 43_{10}$

Step 2: 92 + 43 (complement of 56)
= 135 (note 1 as carry)

Step 3: 35 + 1 (add 1 carry to sum)

Result = 36

The result may be verified using the method of normal subtraction:

92 - 56 = 36

Complementary Subtraction (Example 2)

Example

Subtract 35_{10} from 18_{10} using complementary method.

Solution

Step 1: Complement of 35_{10}
$= 10^2 - 1 - 35$
$= 99 - 35$
$= 64_{10}$

Step 2:
```
   18
 + 64  (complement
 ----       of 35)
   82
```

Step 3: Since there is no carry, re-complement the sum and attach a negative sign to obtain the result.

Result = -(99 - 82)
= -17

The result may be verified using normal subtraction:

18 - 35 = -17

Binary Subtraction Using Complementary Method (Example 1)

Example

Subtract 0111000_2 (56_{10}) from 1011100_2 (92_{10}) using complementary method.

Solution

```
 1011100
+1000111  (complement of 0111000)
--------
10100011
└──────→ 1  (add the carry of 1)
--------
 0100100
```

Result = $0100100_2 = 36_{10}$

Binary Subtraction Using Complementary Method (Example 2)

Example

Subtract 100011_2 (35_{10}) from 010010_2 (18_{10}) using complementary method.

Solution

```
 010010
+011100  (complement of 100011)
 ------
 101110
```

Since there is no carry, we have to complement the sum and attach a negative sign to it. Hence,

Result = -010001_2 (complement of 101110_2)
= -17_{10}

Binary Multiplication

Table for binary multiplication is as follows:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Binary Multiplication (Example 1)

Example

Multiply the binary numbers 1010 and 1001

Solution

```
    1010   Multiplicand
   x1001   Multiplier
   -----
    1010   Partial Product
   0000    Partial Product
  0000     Partial Product
 1010      Partial Product
 -------
 1011010   Final Product
```

(Continued on next slide)

Binary Multiplication (Example 2)

(Continued from previous slide..)

Whenever a 0 appears in the multiplier, a separate partial product consisting of a string of zeros need not be generated (only a shift will do). Hence,

```
     1010
    x1001
  -------
     1010
   1010SS   (S = left shift)
  -------
  1011010
  -------
```


Binary Division

Table for binary division is as follows:

0 ÷ 0 = Divide by zero error
0 ÷ 1 = 0
1 ÷ 0 = Divide by zero error
1 ÷ 1 = 1

As in the decimal number system (or in any other number system), division by zero is meaningless

The computer deals with this problem by raising an error condition called 'Divide by zero' error

Rules for Binary Division

1. Start from the left of the dividend
2. Perform a series of subtractions in which the divisor is subtracted from the dividend
3. If subtraction is possible, put a 1 in the quotient and subtract the divisor from the corresponding digits of dividend
4. If subtraction is not possible (divisor greater than remainder), record a 0 in the quotient
5. Bring down the next digit to add to the remainder digits. Proceed as before in a manner similar to long division

Binary Division (Example 1)

Example

Divide 100001_2 by 110_2

Solution

```
        0101     (Quotient)
  110)  100001   (Dividend)
        110        1 <---  Divisor greater than 100, so put 0 in quotient
        1000       2 <---  Add digit from dividend to group used above
         110       3 <---  Subtraction possible, so put 1 in quotient
          100      4 <---  Remainder from subtraction plus digit from dividend
          110      5 <---  Divisor greater, so put 0 in quotient
          1001     6 <---  Add digit from dividend to group
           110     7 <---  Subtraction possible, so put 1 in quotient
            11     Remainder
```


Additive Method of Multiplication and Division

Most computers use the additive method for performing multiplication and division operations because it simplifies the internal circuit design of computer systems

Example

4 x 8 = 8 + 8 + 8 + 8 = 32

Rules for Additive Method of Division

- Subtract the divisor repeatedly from the dividend until the result of subtraction becomes less than or equal to zero
- If result of subtraction is zero, then:
  - quotient = total number of times subtraction was performed
  - remainder = 0
- If result of subtraction is less than zero, then:
  - quotient = total number of times subtraction was performed minus 1
  - remainder = result of the subtraction previous to the last subtraction

Additive Method of Division (Example)

Example

Divide 33_{10} by 6_{10} using the method of addition

Solution:

33 - 6 = 27
27 - 6 = 21
21 - 6 = 15
15 - 6 = 9
9 - 6 = 3
3 - 6 = -3

Since the result of the last subtraction is less than zero,

Quotient = 6 - 1 (ignore last subtraction) = 5

Total subtractions = 6

Remainder = 3 (result of previous subtraction)

Key Words / Phrases

- Additive method of division
- Additive method of multiplication
- Additive method of subtraction
- Binary addition
- Binary arithmetic
- Binary division
- Binary multiplication
- Binary subtraction
- Complement
- Complementary subtraction
- Computer arithmetic

Chapter 06

Boolean Algebra and Logic Circuits

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Boolean algebra
- Fundamental concepts and basic laws of Boolean algebra
- Boolean function and minimization
- Logic gates
- Logic circuits and Boolean expressions
- Combinational circuits and design

Boolean Algebra

- An algebra that deals with binary number system
- George Boole (1815-1864), an English mathematician, developed it for:
  - Simplifying representation
  - Manipulation of propositional logic
- In 1938, Claude E. Shannon proposed using Boolean algebra in design of relay switching circuits
- Provides economical and straightforward approach
- Used extensively in designing electronic circuits used in computers

Fundamental Concepts of Boolean Algebra

- Use of Binary Digit
  - Boolean equations can have either of two possible values, 0 and 1
- Logical Addition
  - Symbol '+', also known as 'OR' operator, used for logical addition. Follows law of binary addition
- Logical Multiplication
  - Symbol '.', also known as 'AND' operator, used for logical multiplication. Follows law of binary multiplication
- Complementation
  - Symbol '-', also known as 'NOT' operator, used for complementation. Follows law of binary compliment

Operator Precedence

- Each operator has a precedence level
- Higher the operator's precedence level, earlier it is evaluated
- Expression is scanned from left to right
- First, expressions enclosed within parentheses are evaluated
- Then, all complement (NOT) operations are performed
- Then, all '·' (AND) operations are performed
- Finally, all '+' (OR) operations are performed

(Continued on next slide)

Operator Precedence

(Continued from previous slide..)

$$\boxed{X + \boxed{\boxed{\overline{Y}} \cdot Z}}$$

1^{st} 2^{nd} 3^{rd}

Postulates of Boolean Algebra

Postulate 1:

(a) A = 0, if and only if, A is not equal to 1

(b) A = 1, if and only if, A is not equal to 0

Postulate 2:

(a) $x + 0 = x$

(b) $x \cdot 1 = x$

Postulate 3: Commutative Law

(a) $x + y = y + x$

(b) $x \cdot y = y \cdot x$

(Continued on next slide)

Postulates of Boolean Algebra

(Continued from previous slide..)

Postulate 4: Associative Law

(a) $x + (y + z) = (x + y) + z$

(b) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Postulate 5: Distributive Law

(a) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

(b) $x + (y \cdot z) = (x + y) \cdot (x + z)$

Postulate 6:

(a) $x + \overline{x} = 1$

(b) $x \cdot \overline{x} = 0$

The Principle of Duality

There is a precise duality between the operators . (AND) and + (OR), and the digits 0 and 1.

For example, in the table below, the second row is obtained from the first row and vice versa simply by interchanging '+' with '.' and '0' with '1'

	Column 1	Column 2	Column 3
Row 1	$1 + 1 = 1$	$1 + 0 = 0 + 1 = 1$	$0 + 0 = 0$
Row 2	$0 \cdot 0 = 0$	$0 \cdot 1 = 1 \cdot 0 = 0$	$1 \cdot 1 = 1$

Therefore, if a particular theorem is proved, its dual theorem automatically holds and need not be proved separately

Some Important Theorems of Boolean Algebra

Sr. No.	Theorems/ Identities	Dual Theorems/ Identities	Name (if any)
1	$x + x = x$	$x \cdot x = x$	Idempotent Law
2	$x + 1 = 1$	$x \cdot 0 = 0$	
3	$x + x \cdot y = x$	$x \cdot x + y = x$	Absorption Law
4	$\bar{\bar{x}} = x$		Involution Law
5	$x \cdot \bar{x} + y = x \cdot y$	$x + \bar{x} \cdot y = x + y$	
6	$\overline{x+y} = \bar{x}\ \bar{y}\ \cdot$	$\overline{x \cdot y} = \bar{x}\ \bar{y} +$	De Morgan's Law

Methods of Proving Theorems

The theorems of Boolean algebra may be proved by using one of the following methods:

1. By using postulates to show that L.H.S. = R.H.S
2. By *Perfect Induction* or *Exhaustive Enumeration* method where all possible combinations of variables involved in L.H.S. and R.H.S. are checked to yield identical results
3. By the *Principle of Duality* where the dual of an already proved theorem is derived from the proof of its corresponding pair

Proving a Theorem by Using Postulates (Example)

Theorem:

$$x + x \cdot y = x$$

Proof:

L.H.S.

$= x + x \cdot y$	
$= x \cdot 1 + x \cdot y$	by postulate 2(b)
$= x \cdot (1 + y)$	by postulate 5(a)
$= x \cdot (y + 1)$	by postulate 3(a)
$= x \cdot 1$	by theorem 2(a)
$= x$	by postulate 2(b)
= R.H.S.	

Proving a Theorem by Perfect Induction (Example)

Theorem:

$$x + x \cdot y = x$$

x	y	$x \cdot y$	$x + x \cdot y$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Proving a Theorem by the Principle of Duality (Example)

Theorem:

$x + x = x$

Proof:

L.H.S.	
$= x + x$	
$= (x + x) \cdot 1$	by postulate 2(b)
$= (x + x) \cdot (x + \bar{x})$	by postulate 6(a)
$= x + x \cdot \bar{x}$	by postulate 5(b)
$= x + 0$	by postulate 6(b)
$= x$	by postulate 2(a)
$=$ R.H.S.	

(Continued on next slide)

Proving a Theorem by the Principle of Duality (Example)

(Continued from previous slide..)

Dual Theorem:

$$x \cdot x = x$$

Proof:

L.H.S.		
$= x \cdot x$		
$= x \cdot x + 0$	by postulate 2(a)	
$= x \cdot x + x \cdot \overline{X}$	by postulate 6(b)	
$= x \cdot (x + \overline{X})$	by postulate 5(a)	
$= x \cdot 1$	by postulate 6(a)	
$= x$	by postulate 2(b)	
$=$ R.H.S.		

Notice that each step of the proof of the dual theorem is derived from the proof of its corresponding pair in the original theorem

Boolean Functions

- A Boolean function is an expression formed with:
  - Binary variables
  - Operators (OR, AND, and NOT)
  - Parentheses, and equal sign
- The value of a Boolean function can be either 0 or 1
- A Boolean function may be represented as:
  - An algebraic expression, or
  - A truth table

Representation as an Algebraic Expression

$$W = X + \bar{Y} \cdot Z$$

- Variable W is a function of X, Y, and Z, can also be written as W = f (X, Y, Z)
- The RHS of the equation is called an ***expression***
- The symbols X, Y, Z are the ***literals*** of the function
- For a given Boolean function, there may be more than one algebraic expressions

Representation as a Truth Table

X	Y	Z	W
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(Continued on next slide)

Representation as a Truth Table

(Continued from previous slide..)

- **The number of rows in the table is equal to 2^n, where *n* is the number of literals in the function**
- **The combinations of 0s and 1s for rows of this table are obtained from the binary numbers by counting from 0 to $2^n - 1$**

Minimization of Boolean Functions

- Minimization of Boolean functions deals with
  - Reduction in number of literals
  - Reduction in number of terms

- Minimization is achieved through manipulating expression to obtain equal and simpler expression(s) (having fewer literals and/or terms)

(Continued on next slide)

Minimization of Boolean Functions

(Continued from previous slide..)

$$F_1 = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y}$$

F_1 has 3 literals (x, y, z) and 3 terms

$$F_2 = x \cdot \overline{y} + \overline{x} \cdot z$$

F_2 has 3 literals (x, y, z) and 2 terms

F_2 can be realized with fewer electronic components, resulting in a cheaper circuit

(Continued on next slide)

Minimization of Boolean Functions

(Continued from previous slide..)

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Both F_1 and F_2 produce the same result

Try out some Boolean Function Minimization

(a) $x + \overline{x} \cdot y$

(b) $x \cdot \left(\overline{x} + y\right)$

(c) $\overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y}$

(d) $x \cdot y + \overline{x} \cdot z + y \cdot z$

(e) $\left(x + y\right) \cdot \left(\overline{x} + z\right) \cdot \left(y + z\right)$

Complement of a Boolean Function

- The complement of a Boolean function is obtained by interchanging:
  - Operators OR and AND
  - Complementing each literal
- This is based on *De Morgan's theorems*, whose general form is:

$$\overline{A_1 + A_2 + A_3 + \ldots + A_n} = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 \cdot \ldots \cdot \bar{A}_n$$

$$\overline{A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n} = \bar{A}_1 + \bar{A}_2 + \bar{A}_3 + \ldots + \bar{A}_n$$

Complementing a Boolean Function (Example)

$$F_1 = \overline{x} \cdot y \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot z$$

To obtain $\overline{F_1}$, we first interchange the OR and the AND operators giving

$$\left(\overline{x} + y + \overline{z}\right) \cdot \left(\overline{x} + \overline{y} + z\right)$$

Now we complement each literal giving

$$\overline{F_1} = \left(x + \overline{y} + z\right) \cdot \left(x + y + \overline{z}\right)$$

Canonical Forms of Boolean Functions

Minterms : ***n*** **variables forming an AND term, with each variable being primed or unprimed, provide 2^n possible combinations called *minterms* or *standard products***

Maxterms : ***n*** **variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations called *maxterms* or *standard sums***

Minterms and Maxterms for three Variables

Variables			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$\overline{x} \cdot \overline{y} \cdot \overline{z}$	m_0	$x + y + z$	M_0
0	0	1	$\overline{x} \cdot \overline{y} \cdot z$	m_1	$x + y + \overline{z}$	M_1
0	1	0	$\overline{x} \cdot y \cdot \overline{z}$	m_2	$x + \overline{y} + z$	M_2
0	1	1	$\overline{x} \cdot y \cdot z$	m_3	$x + \overline{y} + \overline{z}$	M_3
1	0	0	$x \cdot \overline{y} \cdot \overline{z}$	m_4	$\overline{x} + y + z$	M_4
1	0	1	$x \cdot \overline{y} \cdot z$	m_5	$\overline{x} + y + \overline{z}$	M_5
1	1	0	$x \cdot y \cdot \overline{z}$	m_6	$\overline{x} + \overline{y} + z$	M_6
1	1	1	$x \cdot y \cdot z$	m_7	$\overline{x} + \overline{y} + \overline{z}$	M_7

Note that each minterm is the complement of its corresponding maxterm and vice-versa

Sum-of-Products (SOP) Expression

A sum-of-products (SOP) expression is a product term (minterm) or several product terms (minterms) logically added (ORed) together. Examples are:

$$x \qquad x + y$$

$$x + y \cdot z \qquad x \cdot y + z$$

$$x \cdot \overline{y} + \overline{x} \cdot y \qquad \overline{x} \cdot \overline{y} + x \cdot \overline{y} \cdot z$$

Steps to Express a Boolean Function in its Sum-of-Products Form

1. Construct a truth table for the given Boolean function
2. Form a minterm for each combination of the variables, which produces a 1 in the function
3. The desired expression is the sum (OR) of all the minterms obtained in Step 2

Expressing a Function in its Sum-of-Products Form (Example)

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The following 3 combinations of the variables produce a 1:
001, 100, and 111

(Continued on next slide)

Expressing a Function in its Sum-of-Products Form (Example)

(Continued from previous slide..)

- **Their corresponding minterms are:**

$$\bar{x} \cdot \bar{y} \cdot z, \quad x \cdot \bar{y} \cdot \bar{z}, \quad \text{and} \quad x \cdot y \cdot z$$

- **Taking the OR of these minterms, we get**

$$F_1 = \bar{x} \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot z = m_1 + m_4 + m_7$$
$$F_1 \left(x \cdot y \cdot z \right) = \sum \left(1, 4, 7 \right)$$

Product-of Sums (POS) Expression

A product-of-sums (POS) expression is a sum term (maxterm) or several sum terms (maxterms) logically multiplied (ANDed) together. Examples are:

$$x \qquad (x+\overline{y})\cdot(\overline{x}+y)\cdot(\overline{x}+\overline{y})$$

$$\overline{x}+y \qquad (x+y)\cdot(\overline{x}+y+z)$$

$$(\overline{x}+\overline{y})\cdot z \qquad (\overline{x}+y)\cdot(x+\overline{y})$$

Steps to Express a Boolean Function in its Product-of-Sums Form

1. Construct a truth table for the given Boolean function
2. Form a maxterm for each combination of the variables, which produces a 0 in the function
3. The desired expression is the product (AND) of all the maxterms obtained in Step 2

Expressing a Function in its Product-of-Sums Form

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- **The following 5 combinations of variables produce a 0:**

  000, 010, 011, 101, and 110

(Continued on next slide)

Expressing a Function in its Product-of-Sums Form

(Continued from previous slide..)

- **Their corresponding maxterms are:**

$$\left(x+y+z\right),\left(x+\bar{y}+z\right),\left(x+\bar{y}+\bar{z}\right),$$

$$\left(\bar{x}+y+\bar{z}\right)\text{ and }\left(\bar{x}+\bar{y}+z\right)$$

- **Taking the AND of these maxterms, we get:**

$$F_1=\left(x+y+z\right)\cdot\left(x+\bar{y}+z\right)\cdot\left(x+\bar{y}+\bar{z}\right)\cdot\left(\bar{x}+y+\bar{z}\right)\cdot$$

$$\left(\bar{x}+\bar{y}+z\right)=M_0\cdot M_2\cdot M_3\cdot M_5\cdot M_6$$

$$F_1\left(x,y,z\right)=\Pi\left(0,2,3,5,6\right)$$

Conversion Between Canonical Forms (Sum-of-Products and Product-of-Sums)

To convert from one canonical form to another, interchange the symbol and list those numbers missing from the original form.

Example:

$$F(x,y,z) = \Pi(0,2,4,5) = \Sigma(1,3,6,7)$$

$$F(x,y,z) = \Pi(1,4,7) = \Sigma(0,2,3,5,6)$$

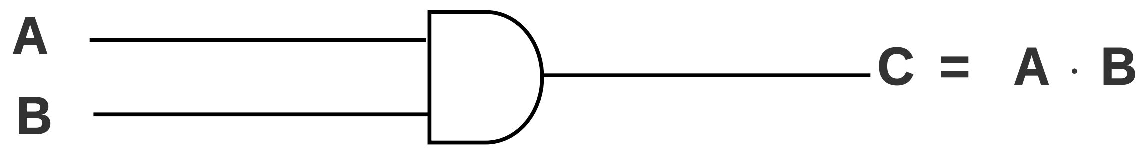
Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce standard output signal
- Are the building blocks of all the circuits in a computer
- Some of the most basic and useful logic gates are AND, OR, NOT, NAND and NOR gates

AND Gate

- Physical realization of logical multiplication (AND) operation
- Generates an output signal of 1 only if all input signals are also 1

AND Gate (Block Diagram Symbol and Truth Table)

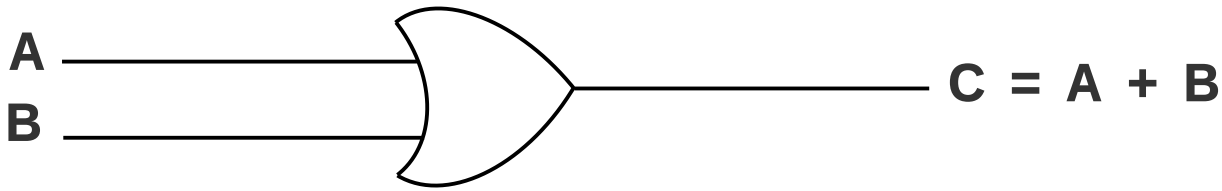


Inputs		Output
A	B	$C = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

- Physical realization of logical addition (OR) operation
- Generates an output signal of 1 if at least one of the input signals is also 1

OR Gate (Block Diagram Symbol and Truth Table)

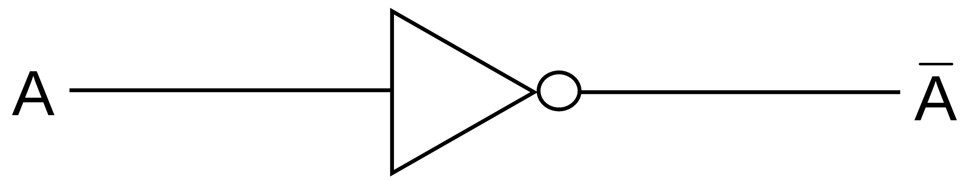


Inputs		Output
A	B	C = A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

- Physical realization of complementation operation
- Generates an output signal, which is the reverse of the input signal

NOT Gate (Block Diagram Symbol and Truth Table)

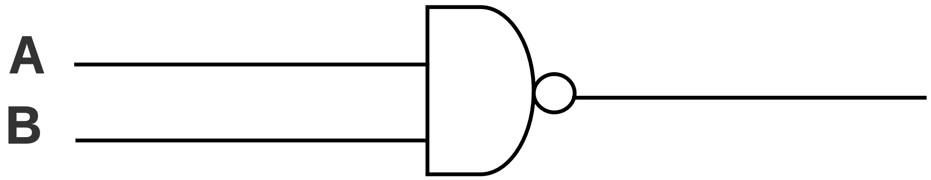


Input	Output
A	$\overline{A}$
0	1
1	0

NAND Gate

- Complemented AND gate
- Generates an output signal of:
  - 1 if any one of the inputs is a 0
  - 0 when all the inputs are 1

NAND Gate (Block Diagram Symbol and Truth Table)



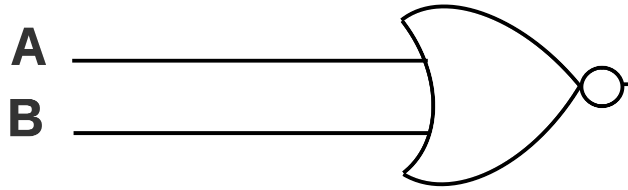
$C = A \uparrow B = \overline{A \cdot B} = \overline{A} + \overline{B}$

Inputs		Output
A	B	$C = \overline{A} + \overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

- Complemented OR gate
- Generates an output signal of:
  - 1 only when all inputs are 0
  - 0 if any one of inputs is a 1

NOR Gate (Block Diagram Symbol and Truth Table)



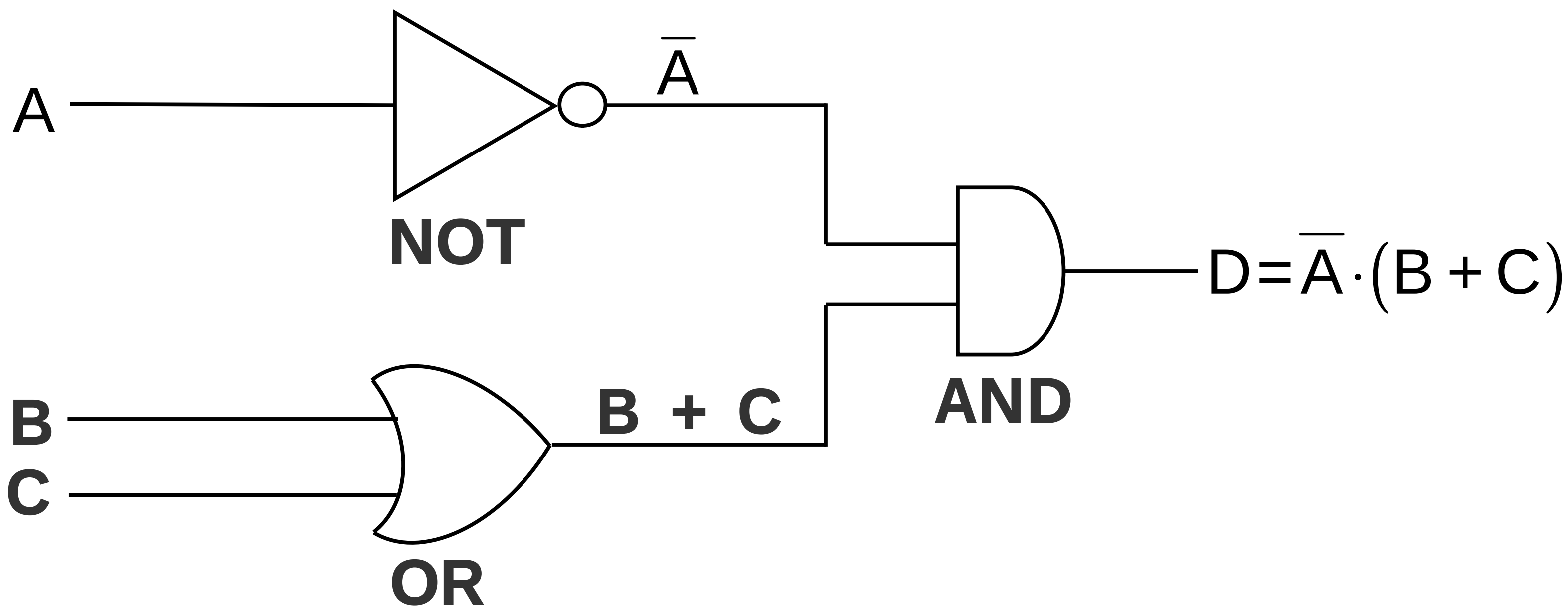
$$C = A \downarrow B = \overline{A + B} = \overline{A} \cdot \overline{B}$$

Inputs		Output
A	B	$C = \overline{A} \cdot \overline{B}$
0	0	1
0	1	0
1	0	0
1	1	0

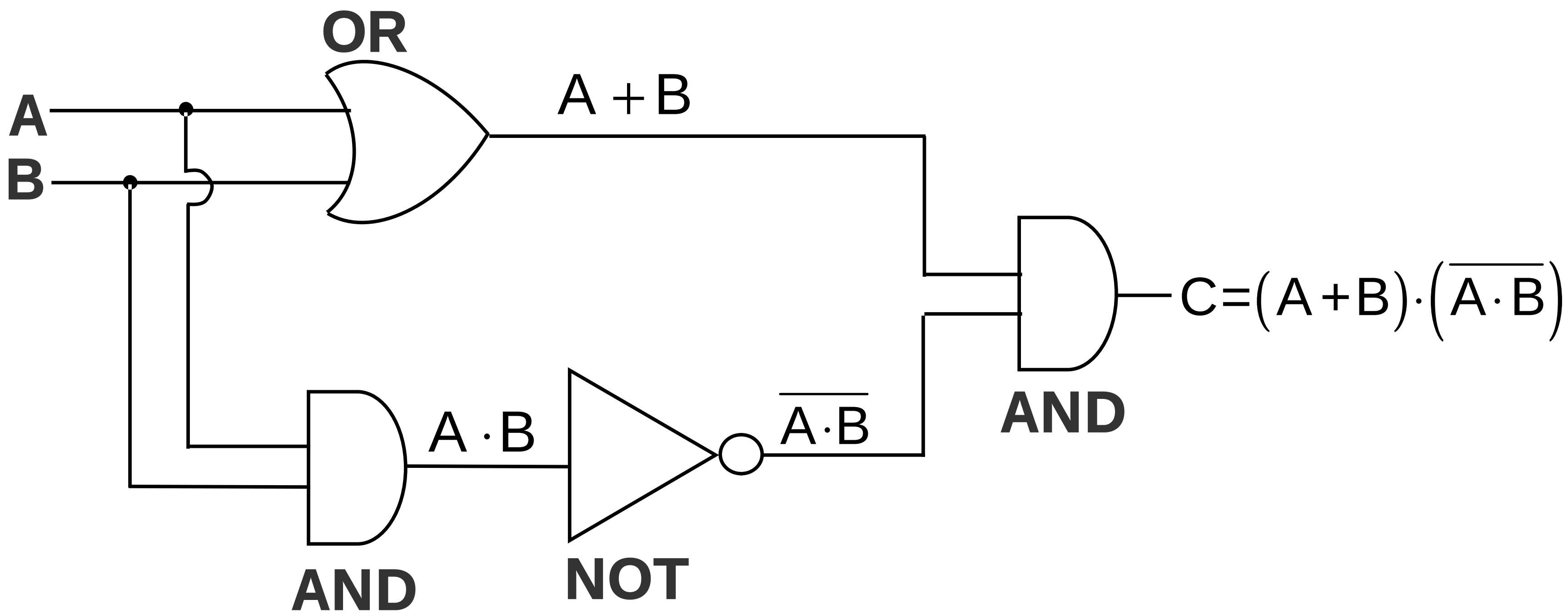
Logic Circuits

- When logic gates are interconnected to form a gating / logic network, it is known as a *combinational logic circuit*
- The Boolean algebra expression for a given logic circuit can be derived by systematically progressing from input to output on the gates
- The three logic gates (AND, OR, and NOT) are logically complete because any Boolean expression can be realized as a logic circuit using only these three gates

Finding Boolean Expression of a Logic Circuit (Example 1)

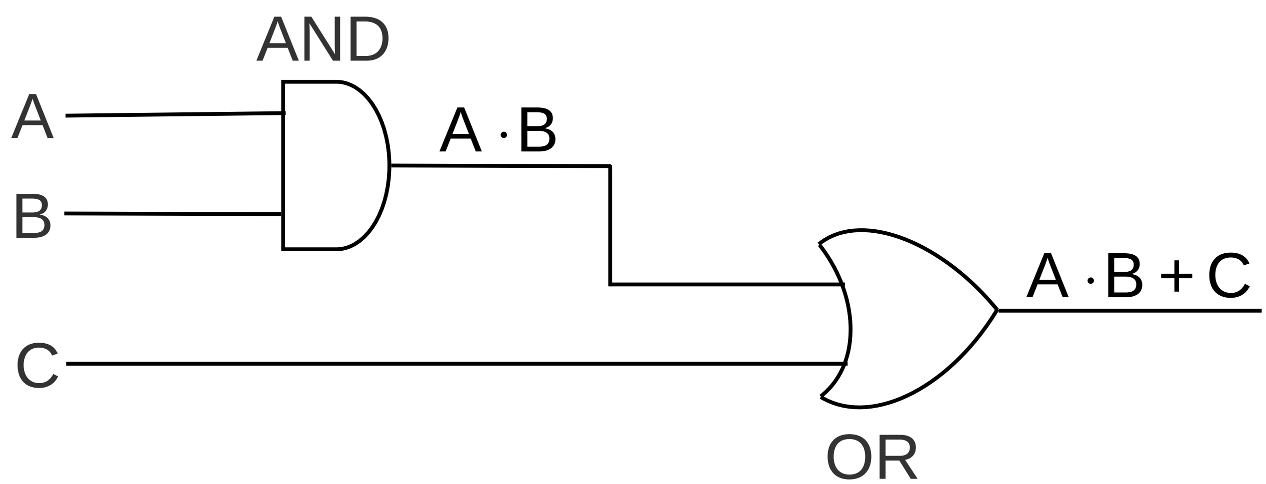


Finding Boolean Expression of a Logic Circuit (Example 2)



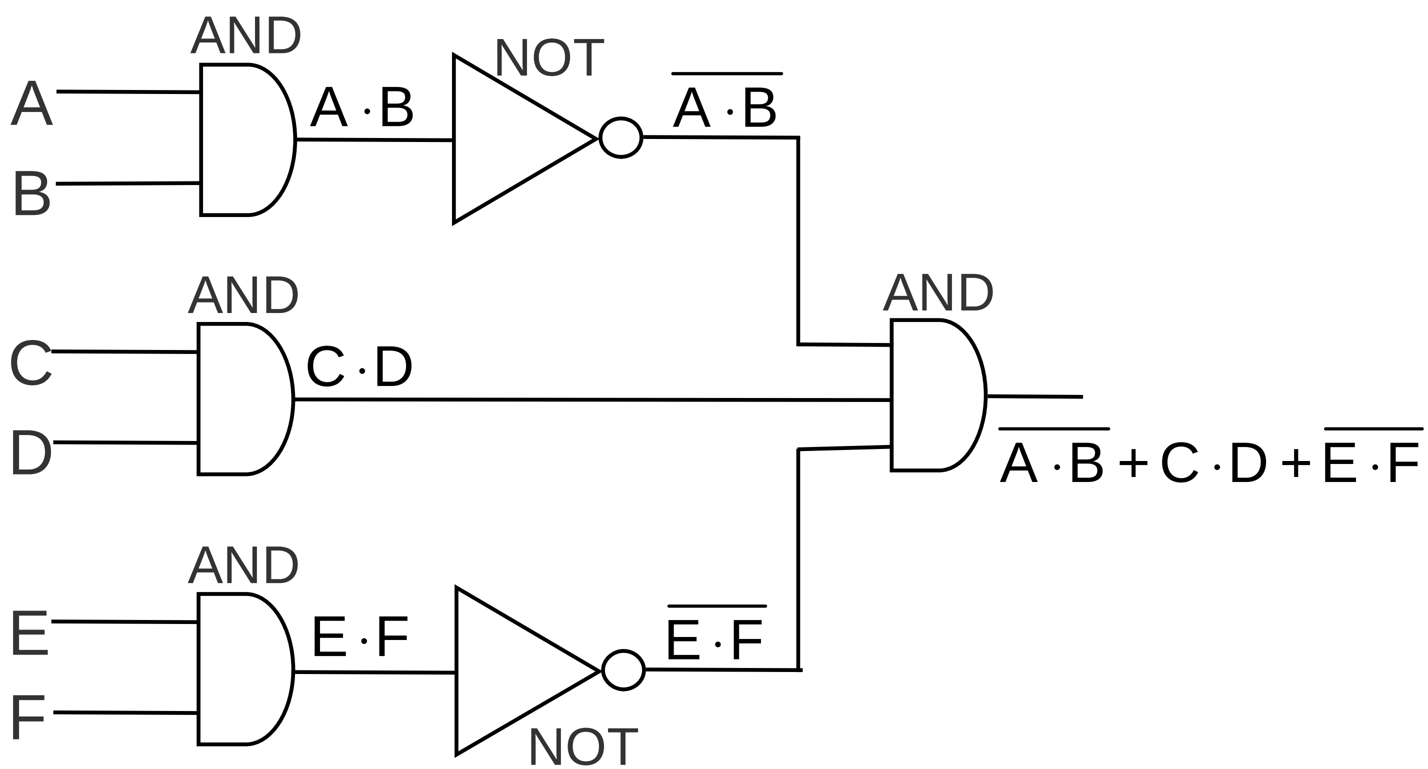
Constructing a Logic Circuit from a Boolean Expression (Example 1)

Boolean Expression = $A \cdot B + C$



Constructing a Logic Circuit from a Boolean Expression (Example 2)

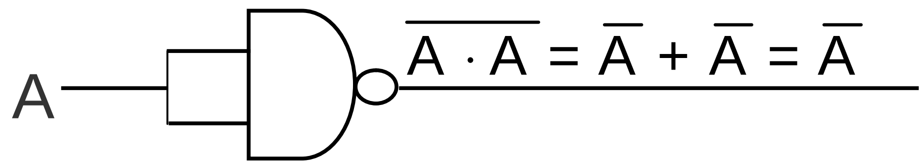
Boolean Expression = $\overline{A \cdot B} + C \cdot D + \overline{E \cdot F}$



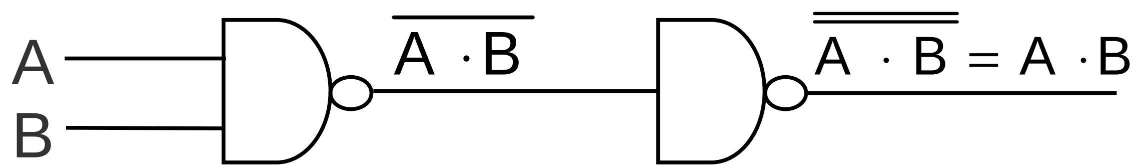
Universal NAND Gate

- NAND gate is an universal gate, it is alone sufficient to implement any Boolean expression
- To understand this, consider:
  - Basic logic gates (AND, OR, and NOT) are logically complete
  - Sufficient to show that AND, OR, and NOT gates can be implemented with NAND gates

Implementation of NOT, AND and OR Gates by NAND Gates



(a) NOT gate implementation.

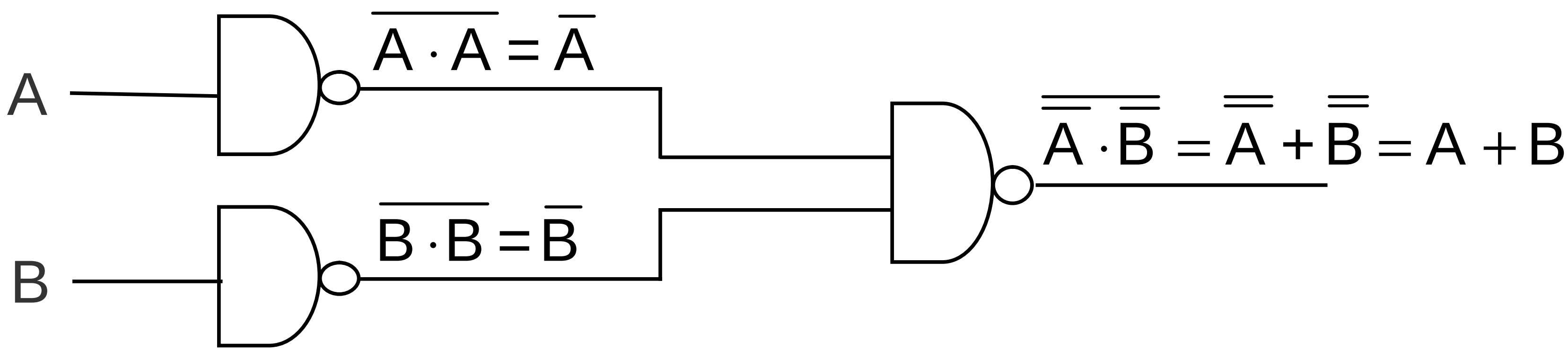


(b) AND gate implementation.

(Continued on next slide)

Implementation of NOT, AND and OR Gates by NAND Gates

(Continued from previous slide..)



(c) OR gate implementation.

Method of Implementing a Boolean Expression with Only NAND Gates

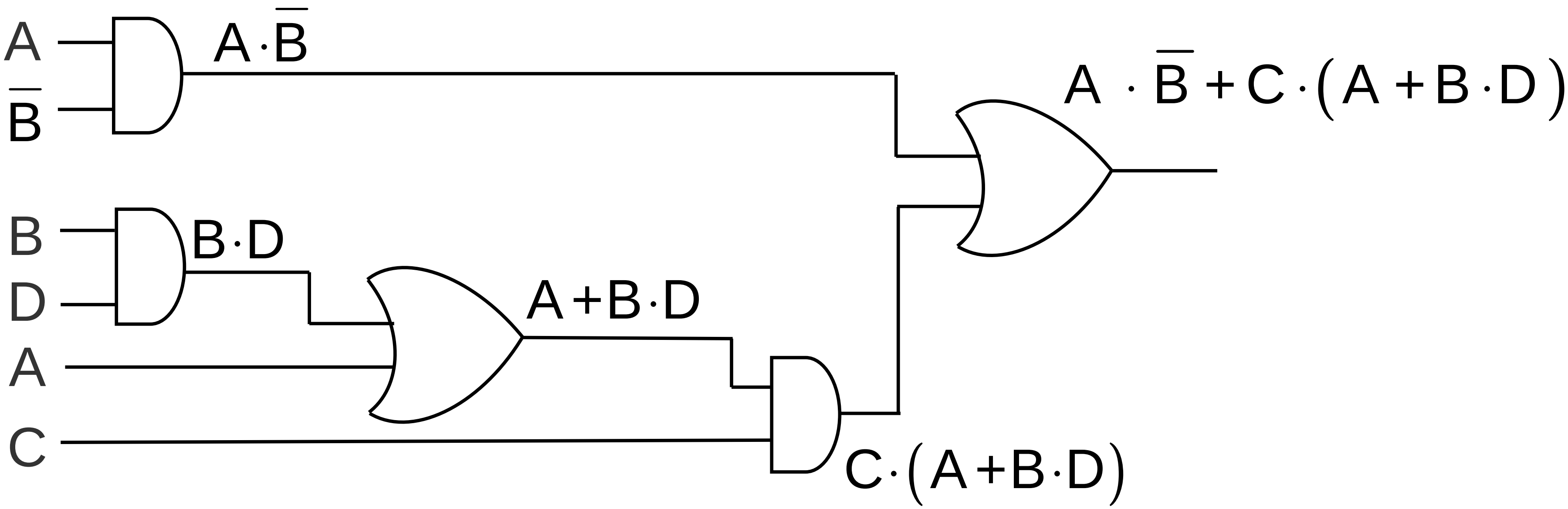
Step 1: From the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement ($\overline{A}$) inputs are available

Step 2: Draw a second logic diagram with the equivalent NAND logic substituted for each AND, OR, and NOT gate

Step 3: Remove all pairs of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable

Implementing a Boolean Expression with Only NAND Gates (Example)

Boolean Expression = $A \cdot \overline{B} + C \cdot (A + B \cdot D)$

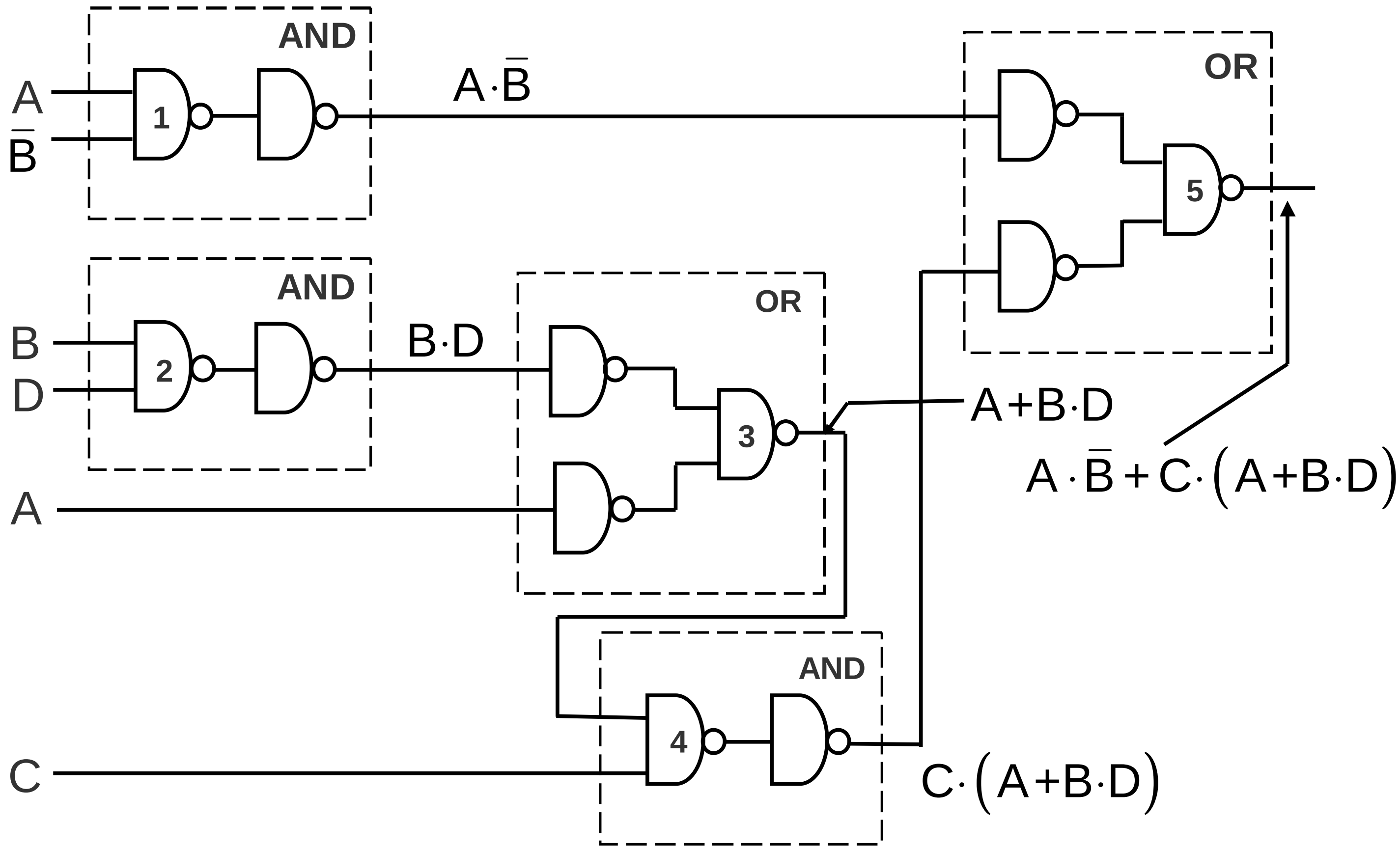


(a) Step 1: AND/OR implementation

(Continued on next slide)

Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

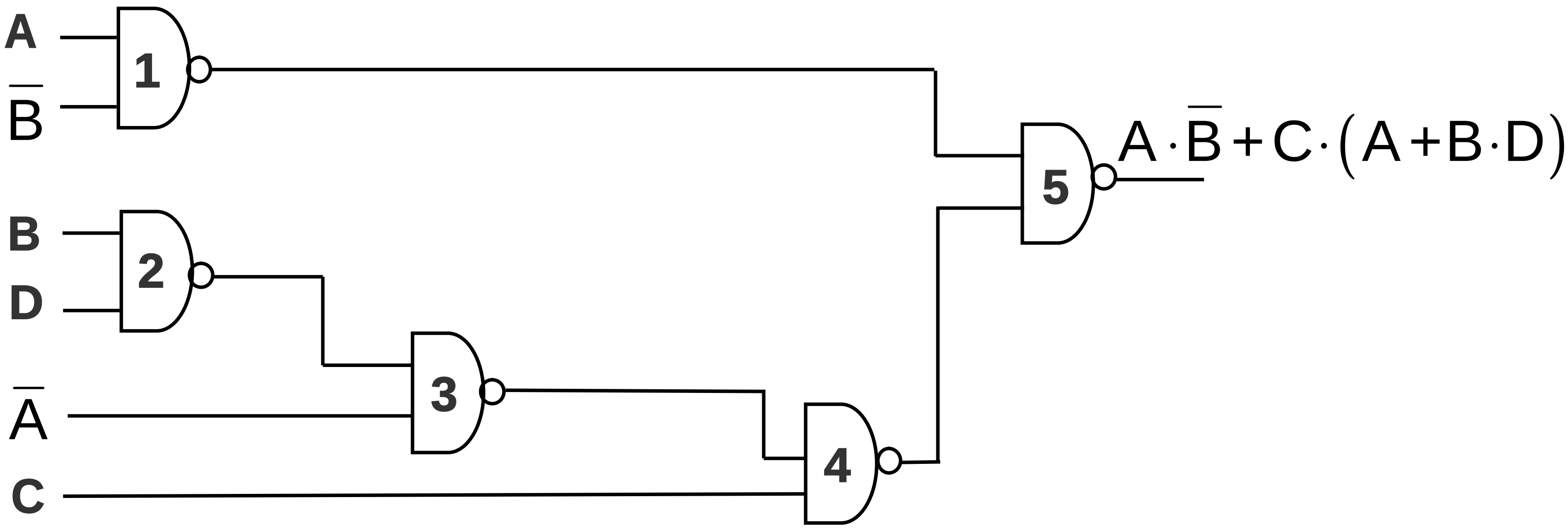


(b) Step 2: Substituting equivalent NAND functions

(Continued on next slide)

Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

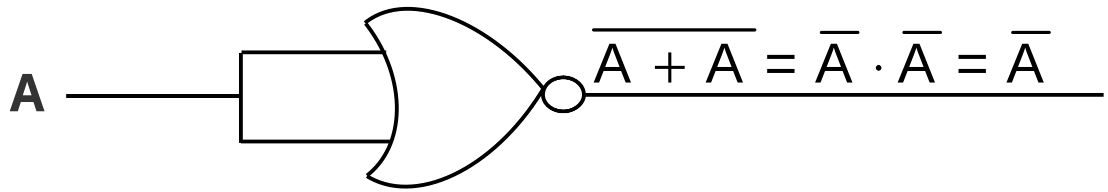


(c) Step 3: NAND implementation.

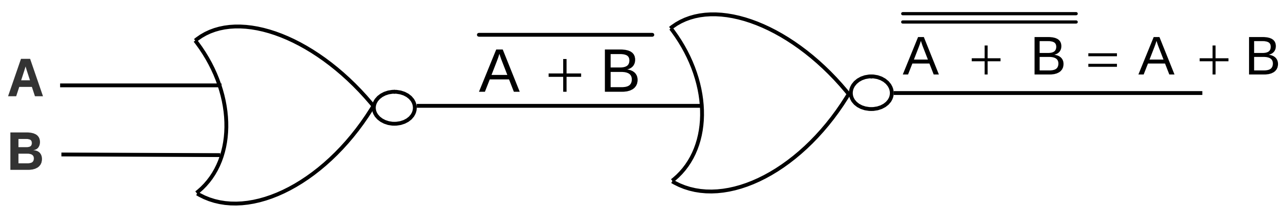
Universal NOR Gate

- NOR gate is an universal gate, it is alone sufficient to implement any Boolean expression
- To understand this, consider:
  - Basic logic gates (AND, OR, and NOT) are logically complete
  - Sufficient to show that AND, OR, and NOT gates can be implemented with NOR gates

Implementation of NOT, OR and AND Gates by NOR Gates



(a) NOT gate implementation.

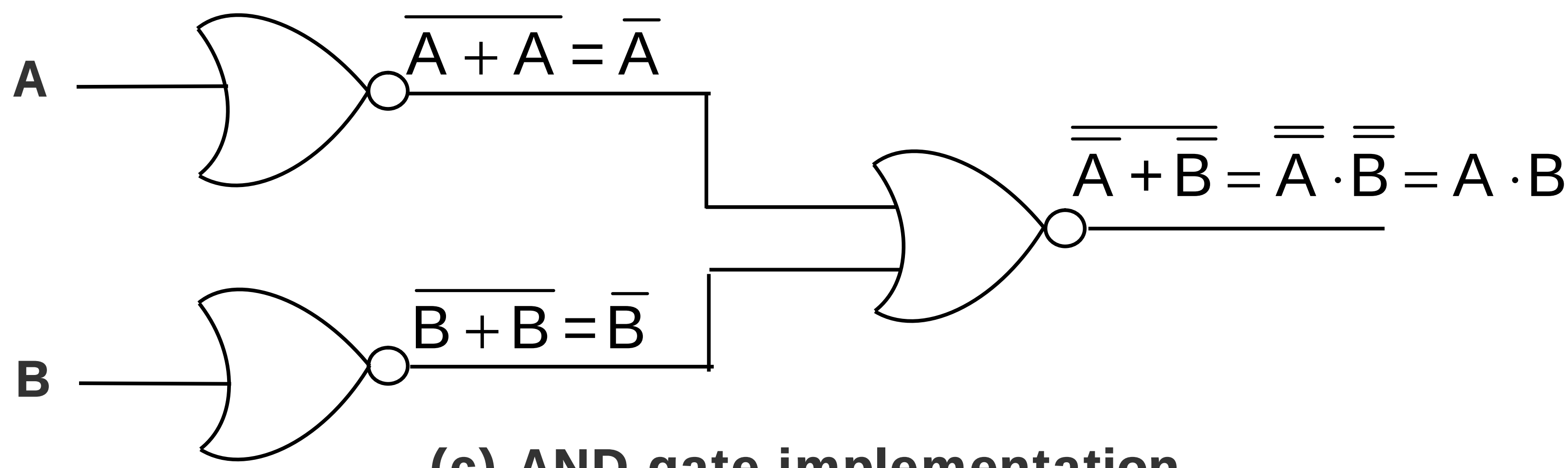


(b) OR gate implementation.

(Continued on next slide)

Implementation of NOT, OR and AND Gates by NOR Gates

(Continued from previous slide..)



(c) AND gate implementation.

Method of Implementing a Boolean Expression with Only NOR Gates

Step 1: For the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement $(\overline{A})$ inputs are available

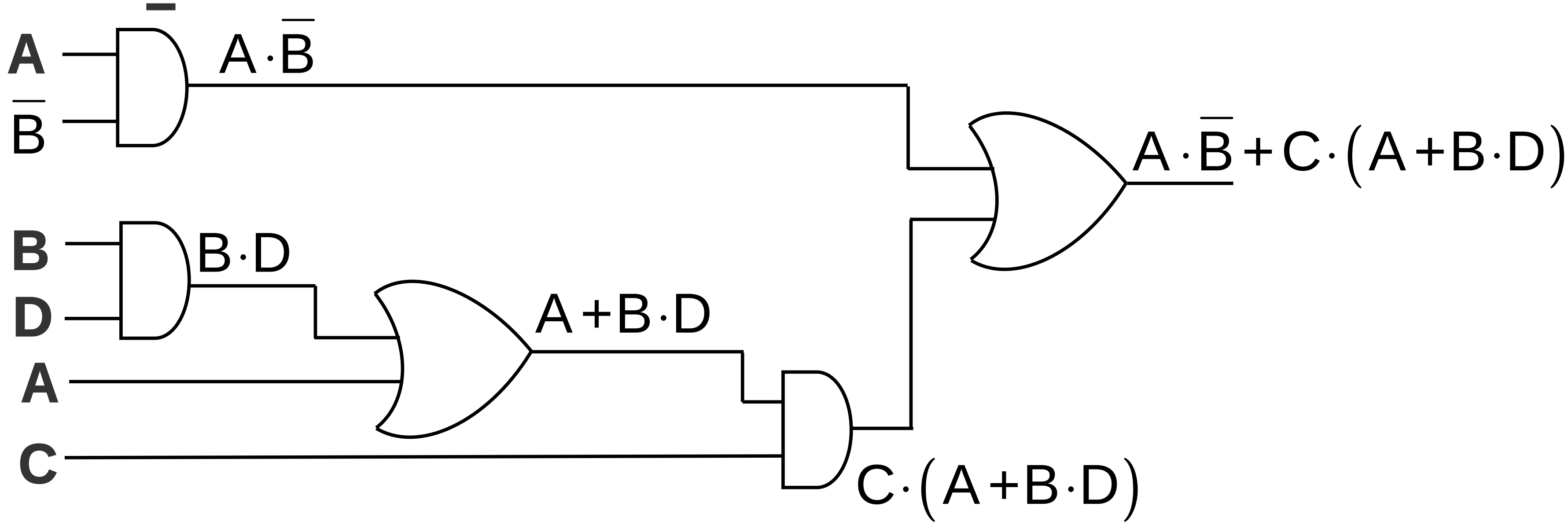
Step 2: Draw a second logic diagram with equivalent NOR logic substituted for each AND, OR, and NOT gate

Step 3: Remove all parts of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable

Implementing a Boolean Expression with Only NOR Gates (Examples)

(Continued from previous slide..)

Boolean Expression = $A \cdot \overline{B} + C \cdot (A + B \cdot D)$

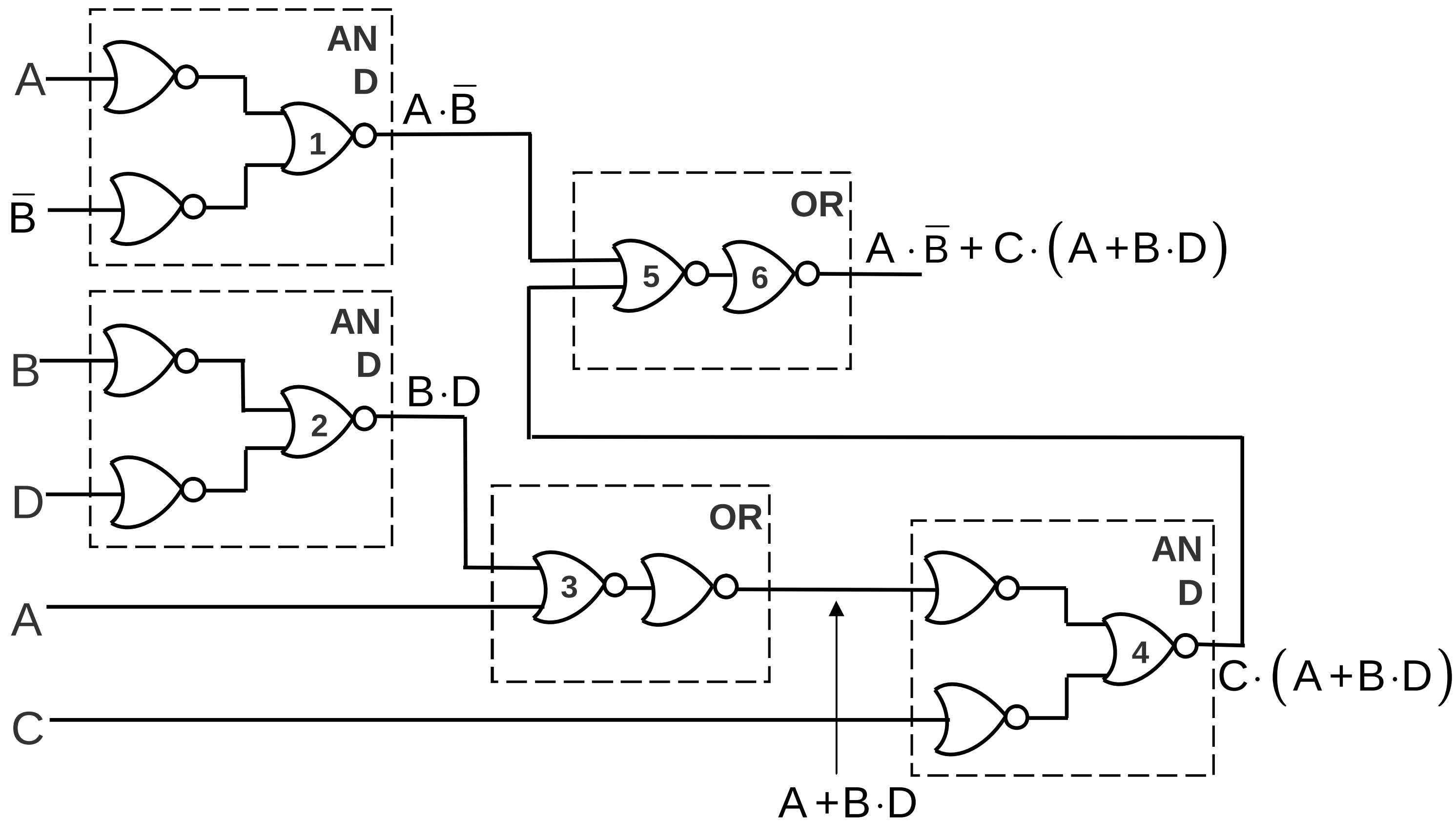


(a) Step 1: AND/OR implementation.

(Continued on next slide)

Implementing a Boolean Expression with Only NOR Gates (Examples)

(Continued from previous slide..)

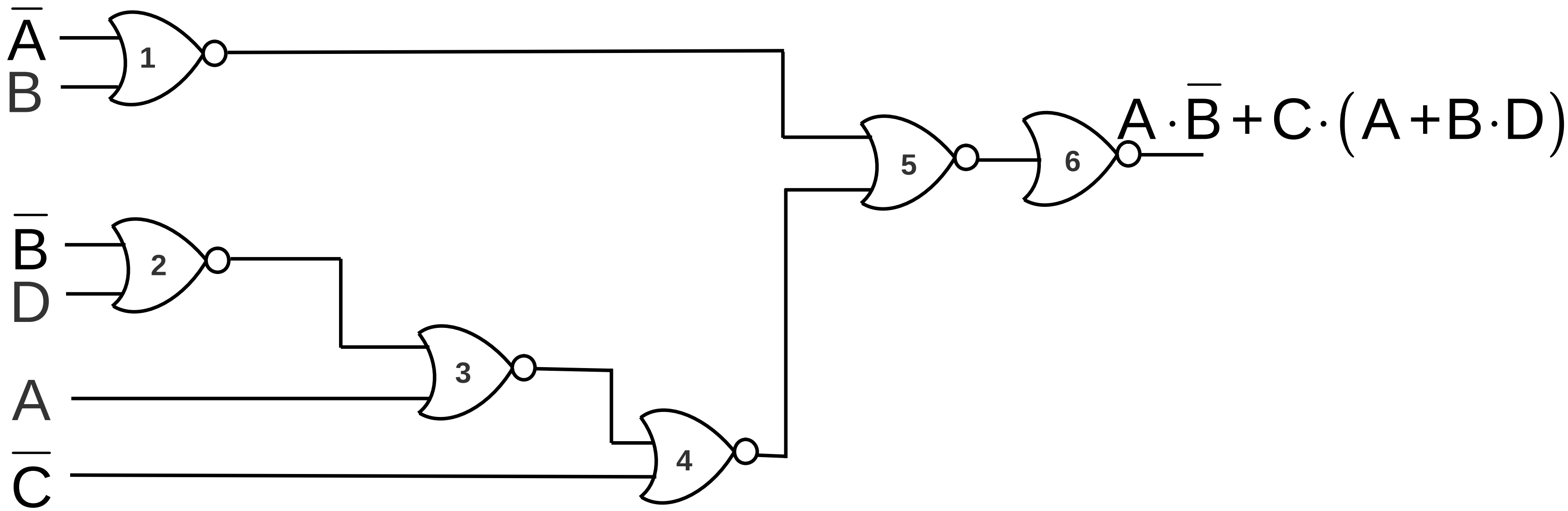


(b) Step 2: Substituting equivalent NOR functions.

(Continued on next slide)

Implementing a Boolean Expression with Only NOR Gates (Examples)

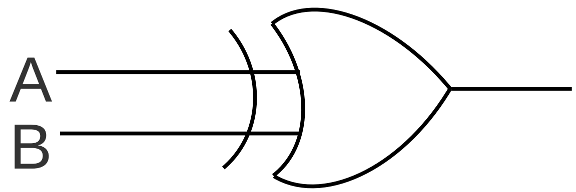
(Continued from previous slide..)



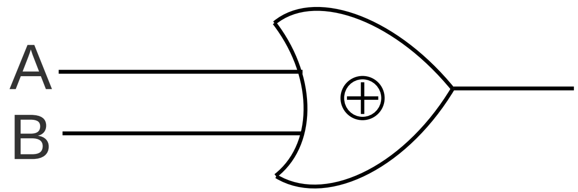
(c) Step 3: NOR implementation.

Exclusive-OR Function

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$



$$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$



$$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$

Also, $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

(Continued on next slide)

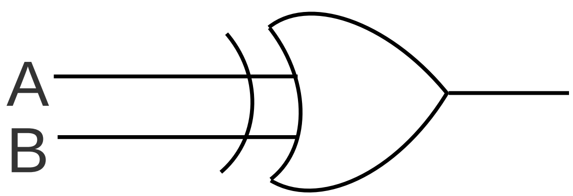
Exclusive-OR Function (Truth Table)

(Continued from previous slide..)

Inputs		Output
A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Equivalence Function with Block Diagram Symbol

$$A \text{ € } B = A \cdot B + \overline{A} \cdot \overline{B}$$



$$C = A \text{ € } B = A \cdot B + \overline{A} \cdot \overline{B}$$

$$\text{Also, } (A \text{ € } B) \text{ € } = A \text{ € } (B \text{ € } C) = A \text{ € } B \text{ € } C$$

(Continued on next slide)

Equivalence Function (Truth Table)

Inputs		Output
A	B	C = A € B
0	0	1
0	1	0
1	0	0
1	1	1

Steps in Designing Combinational Circuits

1. State the given problem completely and exactly
2. Interpret the problem and determine the available input variables and required output variables
3. Assign a letter symbol to each input and output variables
4. Design the truth table that defines the required relations between inputs and outputs
5. Obtain the simplified Boolean function for each output
6. Draw the logic circuit diagram to implement the Boolean function

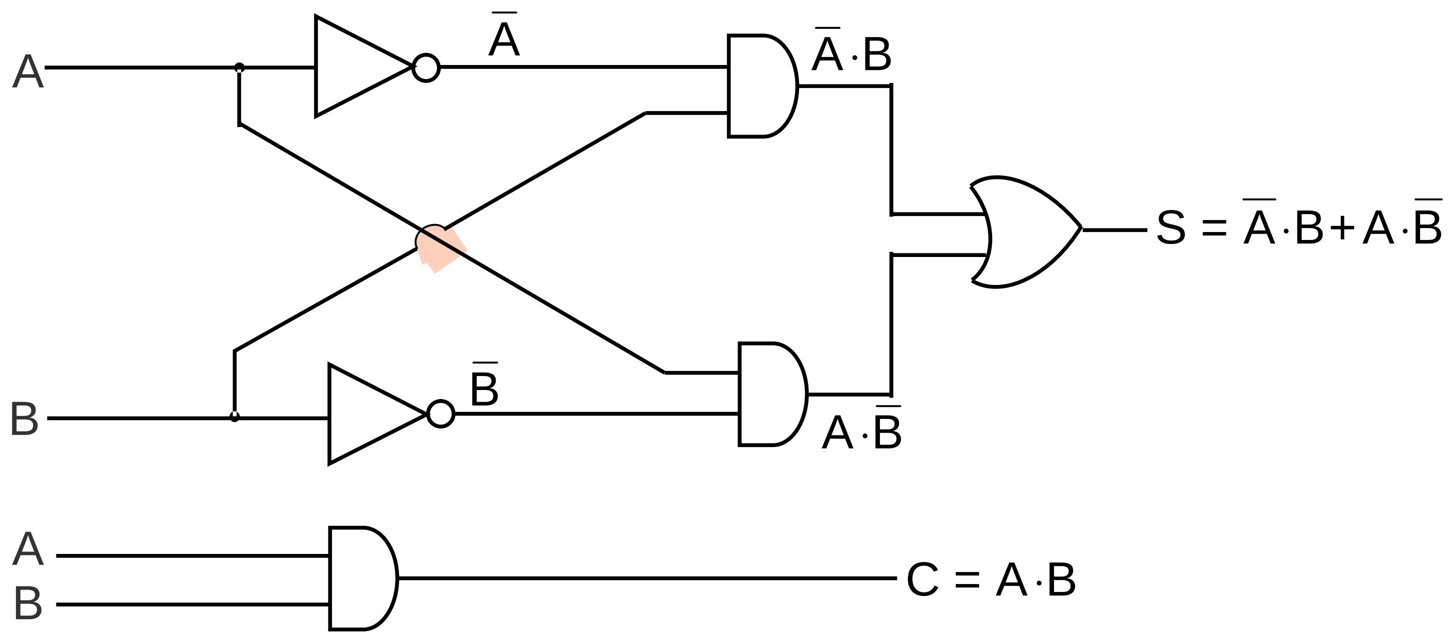
Designing a Combinational Circuit
Example 1 – Half-Adder Design

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\left.\begin{aligned} S &= \overline{A}\cdot B + A\cdot\overline{B} \\ C &= A\cdot B \end{aligned}\right\}$$ **Boolean functions for the two outputs.**

Designing a Combinational Circuit Example 1 – Half-Adder Design

(Continued from previous slide..)



Logic circuit diagram to implement the Boolean functions

Designing a Combinational Circuit Example 2 – Full-Adder Design

Inputs			Outputs	
A	B	D	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table for a full adder

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)

Boolean functions for the two outputs:

$$S = \overline{A} \cdot \overline{B} \cdot D + \overline{A} \cdot B \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{D} + A \cdot B \cdot D$$

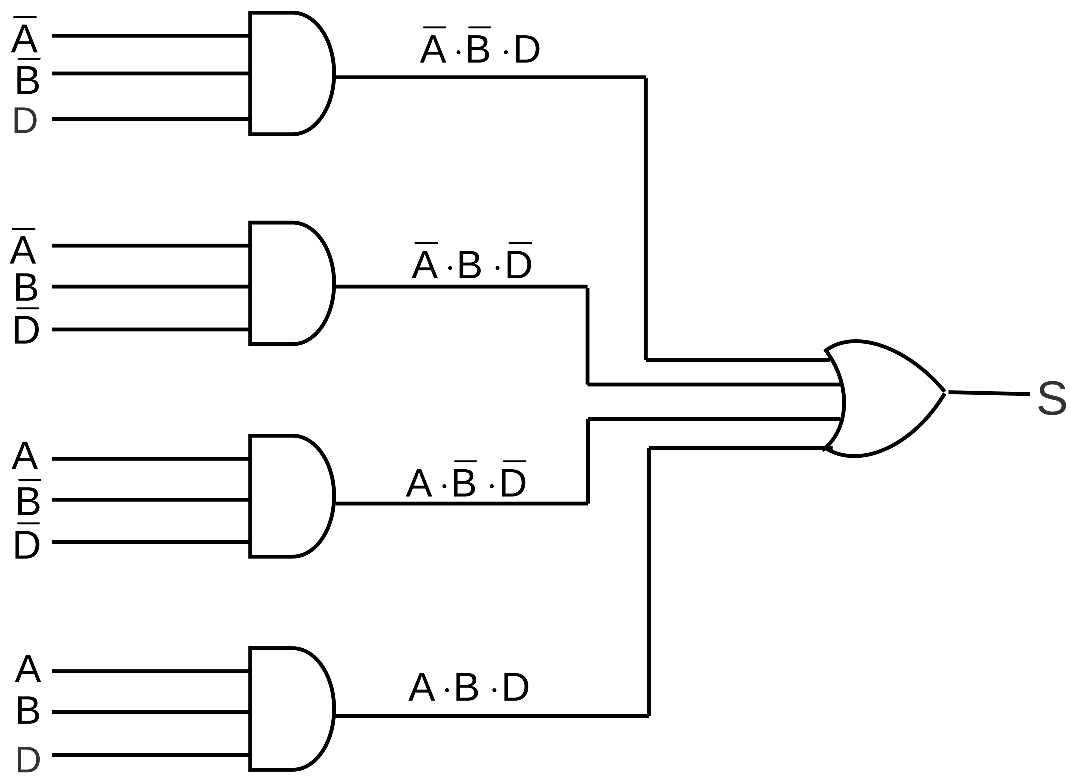
$$C = \overline{A} \cdot B \cdot D + A \cdot \overline{B} \cdot D + A \cdot B \cdot \overline{D} + A \cdot B \cdot D$$

$$= A \cdot B + A \cdot D + B \cdot D$$ **(when simplified)**

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)

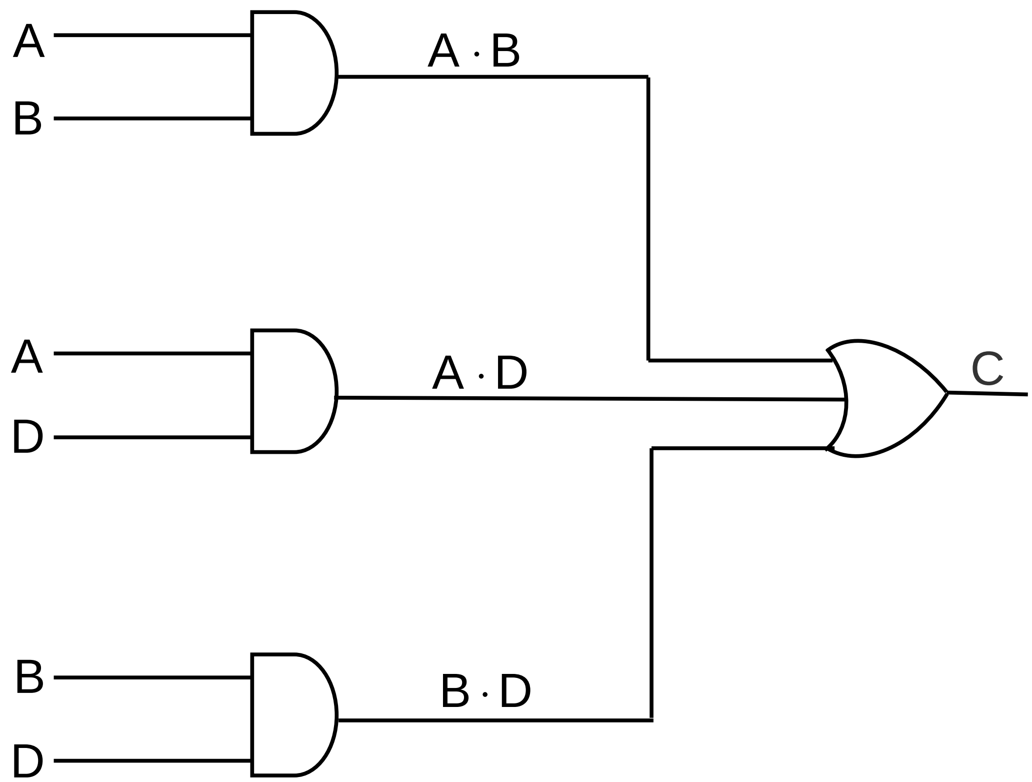


(a) Logic circuit diagram for sums

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)



(b) Logic circuit diagram for carry

Key Words / Phrases

- Absorption law
- AND gate
- Associative law
- Boolean algebra
- Boolean expression
- Boolean functions
- Boolean identities
- Canonical forms for Boolean functions
- Combination logic circuits
- Cumulative law
- Complement of a function
- Complementation
- De Morgan's law
- Distributive law
- Dual identities
- Equivalence function
- Exclusive-OR function
- Exhaustive enumeration method
- Half-adder
- Idempotent law
- Involution law
- Literal
- Logic circuits
- Logic gates
- Logical addition
- Logical multiplication
- Maxterms
- Minimization of Boolean functions
- Minterms
- NAND gate
- NOT gate
- Operator precedence
- OR gate
- Parallel Binary Adder
- Perfect induction method
- Postulates of Boolean algebra
- Principle of duality
- Product-of-Sums expression
- Standard forms
- Sum-of Products expression
- Truth table
- Universal NAND gate
- Universal NOR gate

Chapter 06

Boolean Algebra and Logic Circuits

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Boolean algebra
- Fundamental concepts and basic laws of Boolean algebra
- Boolean function and minimization
- Logic gates
- Logic circuits and Boolean expressions
- Combinational circuits and design

Boolean Algebra

- An algebra that deals with binary number system
- George Boole (1815-1864), an English mathematician, developed it for:
  - Simplifying representation
  - Manipulation of propositional logic
- In 1938, Claude E. Shannon proposed using Boolean algebra in design of relay switching circuits
- Provides economical and straightforward approach
- Used extensively in designing electronic circuits used in computers

Fundamental Concepts of Boolean Algebra

- **Use of Binary Digit**
  - **Boolean equations can have either of two possible values, 0 and 1**
- **Logical Addition**
  - **Symbol '+', also known as 'OR' operator, used for logical addition. Follows law of binary addition**
- **Logical Multiplication**
  - **Symbol '.', also known as 'AND' operator, used for logical multiplication. Follows law of binary multiplication**
- **Complementation**
  - **Symbol '-', also known as 'NOT' operator, used for complementation. Follows law of binary compliment**

Operator Precedence

- Each operator has a precedence level
- Higher the operator’s precedence level, earlier it is evaluated
- Expression is scanned from left to right
- First, expressions enclosed within parentheses are evaluated
- Then, all complement (NOT) operations are performed
- Then, all ‘·’ (AND) operations are performed
- Finally, all ‘+’ (OR) operations are performed

(Continued on next slide)

Operator Precedence

(Continued from previous slide..)

$$\boxed{X + \boxed{\boxed{\overline{Y}} \cdot Z}}$$

1^{st} 2^{nd} 3^{rd}

Postulates of Boolean Algebra

Postulate 1:

(a) A = 0, if and only if, A is not equal to 1

(b) A = 1, if and only if, A is not equal to 0

Postulate 2:

(a) $x + 0 = x$

(b) $x \cdot 1 = x$

Postulate 3: Commutative Law

(a) $x + y = y + x$

(b) $x \cdot y = y \cdot x$

(Continued on next slide)

Postulates of Boolean Algebra

(Continued from previous slide..)

Postulate 4: Associative Law

(a) $x + (y + z) = (x + y) + z$

(b) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Postulate 5: Distributive Law

(a) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

(b) $x + (y \cdot z) = (x + y) \cdot (x + z)$

Postulate 6:

(a) $x + \bar{x} = 1$

(b) $x \cdot \bar{x} = 0$

The Principle of Duality

There is a precise duality between the operators . (AND) and + (OR), and the digits 0 and 1.

For example, in the table below, the second row is obtained from the first row and vice versa simply by interchanging '+' with '.' and '0' with '1'

	Column 1	Column 2	Column 3
Row 1	$1 + 1 = 1$	$1 + 0 = 0 + 1 = 1$	$0 + 0 = 0$
Row 2	$0 \cdot 0 = 0$	$0 \cdot 1 = 1 \cdot 0 = 0$	$1 \cdot 1 = 1$

Therefore, if a particular theorem is proved, its dual theorem automatically holds and need not be proved separately

Some Important Theorems of Boolean Algebra

Sr. No.	Theorems/ Identities	Dual Theorems/ Identities	Name (if any)
1	$x + x = x$	$x \cdot x = x$	Idempotent Law
2	$x + 1 = 1$	$x \cdot 0 = 0$	
3	$x + x \cdot y = x$	$x \cdot x + y = x$	Absorption Law
4	$\bar{\bar{x}} = x$		Involution Law
5	$x \cdot \bar{x} + y = x \cdot y$	$x + \bar{x} \cdot y = x + y$	
6	$\overline{x+y} = \bar{x} \cdot \bar{y}$	$\overline{x \cdot y} = \bar{x} + \bar{y}$	De Morgan's Law

Methods of Proving Theorems

The theorems of Boolean algebra may be proved by using one of the following methods:

1. By using postulates to show that L.H.S. = R.H.S
2. By *Perfect Induction* or *Exhaustive Enumeration* method where all possible combinations of variables involved in L.H.S. and R.H.S. are checked to yield identical results
3. By the *Principle of Duality* where the dual of an already proved theorem is derived from the proof of its corresponding pair

Proving a Theorem by Using Postulates (Example)

Theorem:

$x + x \cdot y = x$

Proof:

L.H.S.

$= x + x \cdot y$	
$= x \cdot 1 + x \cdot y$	by postulate 2(b)
$= x \cdot (1 + y)$	by postulate 5(a)
$= x \cdot (y + 1)$	by postulate 3(a)
$= x \cdot 1$	by theorem 2(a)
$= x$	by postulate 2(b)
= R.H.S.	

Proving a Theorem by Perfect Induction (Example)

Theorem:

$$x + x \cdot y = x$$

x	y	$x \cdot y$	$x + x \cdot y$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Proving a Theorem by the Principle of Duality (Example)

Theorem:

$x + x = x$

Proof:

L.H.S.	
$= x + x$	
$= (x + x) \cdot 1$	by postulate 2(b)
$= (x + x) \cdot (x + \bar{x})$	by postulate 6(a)
$= x + x \cdot \bar{x}$	by postulate 5(b)
$= x + 0$	by postulate 6(b)
$= x$	by postulate 2(a)
= R.H.S.	

(Continued on next slide)

Proving a Theorem by the Principle of Duality (Example)

(Continued from previous slide..)

Dual Theorem:

$x \cdot x = x$

Proof:

L.H.S.		
$= x \cdot x$		
$= x \cdot x + 0$	by postulate 2(a)	
$= x \cdot x + x \cdot \bar{x}$	by postulate 6(b)	
$= x \cdot (x + \bar{x})$	by postulate 5(a)	
$= x \cdot 1$	by postulate 6(a)	
$= x$	by postulate 2(b)	
$=$ R.H.S.		

Notice that each step of the proof of the dual theorem is derived from the proof of its corresponding pair in the original theorem

Boolean Functions

- A Boolean function is an expression formed with:
  - Binary variables
  - Operators (OR, AND, and NOT)
  - Parentheses, and equal sign
- The value of a Boolean function can be either 0 or 1
- A Boolean function may be represented as:
  - An algebraic expression, or
  - A truth table

Representation as an Algebraic Expression

$$W = X + \overline{Y} \cdot Z$$

- Variable W is a function of X, Y, and Z, can also be written as W = f (X, Y, Z)
- The RHS of the equation is called an *expression*
- The symbols X, Y, Z are the *literals* of the function
- For a given Boolean function, there may be more than one algebraic expressions

Representation as a Truth Table

X	Y	Z	W
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(Continued on next slide)

Representation as a Truth Table

(Continued from previous slide..)

- The number of rows in the table is equal to 2^n, where *n* is the number of literals in the function
- The combinations of 0s and 1s for rows of this table are obtained from the binary numbers by counting from 0 to $2^n - 1$

Minimization of Boolean Functions

- Minimization of Boolean functions deals with
  - Reduction in number of literals
  - Reduction in number of terms
- Minimization is achieved through manipulating expression to obtain equal and simpler expression(s) (having fewer literals and/or terms)

(Continued on next slide)

Minimization of Boolean Functions

(Continued from previous slide..)

$$F_1 = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y}$$

F_1 has 3 literals (x, y, z) and 3 terms

$$F_2 = x \cdot \overline{y} + \overline{x} \cdot z$$

F_2 has 3 literals (x, y, z) and 2 terms

F_2 can be realized with fewer electronic components, resulting in a cheaper circuit

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Minimization of Boolean Functions

(Continued from previous slide..)

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Both F_1 and F_2 produce the same result

Try out some Boolean Function Minimization

(a) $x + \overline{x} \cdot y$

(b) $x \cdot \left(\overline{x} + y\right)$

(c) $\overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y}$

(d) $x \cdot y + \overline{x} \cdot z + y \cdot z$

(e) $(x + y) \cdot \left(\overline{x} + z\right) \cdot (y + z)$

Complement of a Boolean Function

- **The complement of a Boolean function is obtained by interchanging:**
  - **Operators OR and AND**
  - **Complementing each literal**
- **This is based on *De Morgan's theorems*, whose general form is:**

$$\overline{A_1 + A_2 + A_3 + \ldots + A_n} = \overline{A}_1 \cdot \overline{A}_2 \cdot \overline{A}_3 \cdot \ldots \cdot \overline{A}_n$$
$$\overline{A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n} = \overline{A}_1 + \overline{A}_2 + \overline{A}_3 + \ldots + \overline{A}_n$$

Complementing a Boolean Function (Example)

$$F_1 = \bar{x} \cdot y \cdot \bar{z} + \bar{x} \cdot \bar{y} \cdot z$$

To obtain $\overline{F_1}$, we first interchange the OR and the AND operators giving

$$\left(\bar{x} + y + \bar{z}\right) \cdot \left(\bar{x} + \bar{y} + z\right)$$

Now we complement each literal giving

$$\overline{F_1} = \left(x + \bar{y} + z\right) \cdot \left(x + y + \bar{z}\right)$$

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Canonical Forms of Boolean Functions

Minterms : ***n*** **variables forming an AND term, with each variable being primed or unprimed, provide 2^n possible combinations called *minterms* or *standard products***

Maxterms : ***n*** **variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations called *maxterms* or *standard sums***

Minterms and Maxterms for three Variables

Variables			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$\overline{x} \cdot \overline{y} \cdot \overline{z}$	m_0	$x + y + z$	M_0
0	0	1	$\overline{x} \cdot \overline{y} \cdot z$	m_1	$x + y + \overline{z}$	M_1
0	1	0	$\overline{x} \cdot y \cdot \overline{z}$	m_2	$x + \overline{y} + z$	M_2
0	1	1	$\overline{x} \cdot y \cdot z$	m_3	$x + \overline{y} + \overline{z}$	M_3
1	0	0	$x \cdot \overline{y} \cdot \overline{z}$	m_4	$\overline{x} + y + z$	M_4
1	0	1	$x \cdot \overline{y} \cdot z$	m_5	$\overline{x} + y + \overline{z}$	M_5
1	1	0	$x \cdot y \cdot \overline{z}$	m_6	$\overline{x} + \overline{y} + z$	M_6
1	1	1	$x \cdot y \cdot z$	m_7	$\overline{x} + \overline{y} + \overline{z}$	M_7

Note that each minterm is the complement of its corresponding maxterm and vice-versa

Sum-of-Products (SOP) Expression

A sum-of-products (SOP) expression is a product term (minterm) or several product terms (minterms) logically added (ORed) together. Examples are:

$$x \qquad x + y$$

$$x + y \cdot z \qquad x \cdot y + z$$

$$x \cdot \overline{y} + \overline{x} \cdot y \qquad \overline{x} \cdot \overline{y} + x \cdot \overline{y} \cdot z$$

Steps to Express a Boolean Function in its Sum-of-Products Form

1. Construct a truth table for the given Boolean function
2. Form a minterm for each combination of the variables, which produces a 1 in the function
3. The desired expression is the sum (OR) of all the minterms obtained in Step 2

Expressing a Function in its Sum-of-Products Form (Example)

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The following 3 combinations of the variables produce a 1:
001, 100, and 111

(Continued on next slide)

Expressing a Function in its Sum-of-Products Form (Example)

(Continued from previous slide..)

- Their corresponding minterms are:

$\bar{x} \cdot \bar{y} \cdot z, \quad x \cdot \bar{y} \cdot \bar{z}, \quad \text{and} \quad x \cdot y \cdot z$

- Taking the OR of these minterms, we get

$$F_1 = \bar{x} \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot z = m_1 + m_4 + m_7$$
$$F_1(x \cdot y \cdot z) = \sum(1,4,7)$$

Product-of Sums (POS) Expression

A product-of-sums (POS) expression is a sum term (maxterm) or several sum terms (maxterms) logically multiplied (ANDed) together. Examples are:

x	$(x+\bar{y}) \cdot (\bar{x}+y) \cdot (\bar{x}+\bar{y})$
$\bar{x}+y$	$(x+y) \cdot (\bar{x}+y+z)$
$(\bar{x}+\bar{y}) \cdot z$	$(\bar{x}+y) \cdot (x+\bar{y})$

Steps to Express a Boolean Function in its Product-of-Sums Form

1. Construct a truth table for the given Boolean function
2. Form a maxterm for each combination of the variables, which produces a 0 in the function
3. The desired expression is the product (AND) of all the maxterms obtained in Step 2

Expressing a Function in its Product-of-Sums Form

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

§ The following 5 combinations of variables produce a 0:
000, 010, 011, 101, and 110

(Continued on next slide)

Expressing a Function in its Product-of-Sums Form

(Continued from previous slide..)

- Their corresponding maxterms are:

$$\left(x+y+z\right),\left(x+\bar{y}+z\right),\left(x+\bar{y}+\bar{z}\right),$$
$$\left(\bar{x}+y+\bar{z}\right) \text{ and } \left(\bar{x}+\bar{y}+z\right)$$

- Taking the AND of these maxterms, we get:

$$F_1=\left(x+y+z\right)\cdot\left(x+\bar{y}+z\right)\cdot\left(x+\bar{y}+\bar{z}\right)\cdot\left(\bar{x}+y+\bar{z}\right)\cdot$$
$$\left(\bar{x}+\bar{y}+z\right)=M_0\cdot M_2\cdot M_3\cdot M_5\cdot M_6$$
$$F_1\left(x,y,z\right)=\Pi\left(0,2,3,5,6\right)$$

Conversion Between Canonical Forms (Sum-of-Products and Product-of-Sums)

To convert from one canonical form to another, interchange the symbol and list those numbers missing from the original form.

Example:

$$F\left(x,y,z\right)=\Pi\left(0,2,4,5\right)=\Sigma\left(1,3,6,7\right)$$
$$F\left(x,y,z\right)=\Pi\left(1,4,7\right)=\Sigma\left(0,2,3,5,6\right)$$

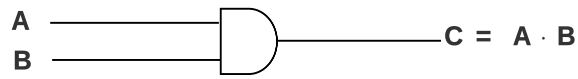
Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce standard output signal
- Are the building blocks of all the circuits in a computer
- Some of the most basic and useful logic gates are AND, OR, NOT, NAND and NOR gates

AND Gate

- Physical realization of logical multiplication (AND) operation
- Generates an output signal of 1 only if all input signals are also 1

AND Gate (Block Diagram Symbol and Truth Table)

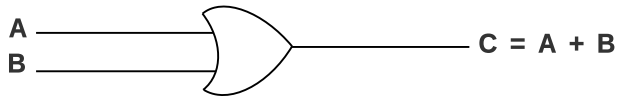


Inputs		Output
A	B	C = A · B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

- Physical realization of logical addition (OR) operation
- Generates an output signal of 1 if at least one of the input signals is also 1

OR Gate (Block Diagram Symbol and Truth Table)



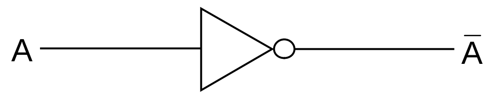
Inputs		Output
A	B	C = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

NOT Gate

- Physical realization of complementation operation
- Generates an output signal, which is the reverse of the input signal

NOT Gate (Block Diagram Symbol and Truth Table)

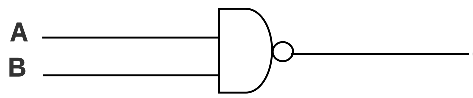


Input	Output
A	$\bar{A}$
0	1
1	0

NAND Gate

- Complemented AND gate
- Generates an output signal of:
  - 1 if any one of the inputs is a 0
  - 0 when all the inputs are 1

NAND Gate (Block Diagram Symbol and Truth Table)



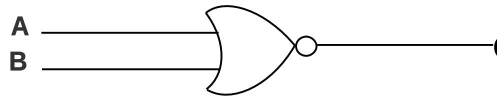
$C = A \uparrow B = \overline{A \cdot B} = \overline{A} + \overline{B}$

Inputs		Output
A	B	$C = \overline{A} + \overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

- Complemented OR gate
- Generates an output signal of:
  - 1 only when all inputs are 0
  - 0 if any one of inputs is a 1

NOR Gate (Block Diagram Symbol and Truth Table)



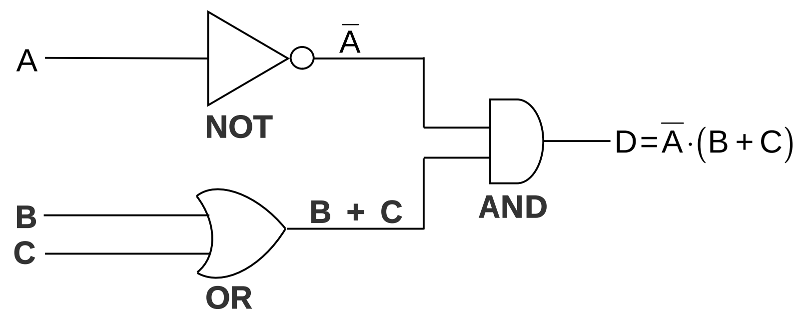
$C = A \downarrow B = \overline{A + B} = \overline{A} \cdot \overline{B}$

Inputs		Output
A	B	$C = \overline{A} \cdot \overline{B}$
0	0	1
0	1	0
1	0	0
1	1	0

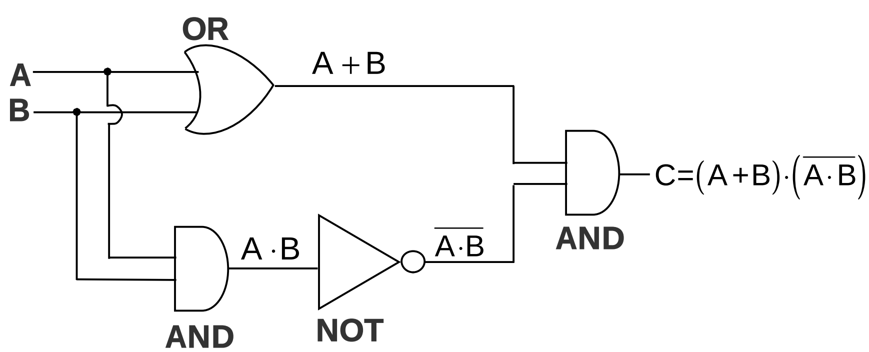
Logic Circuits

- When logic gates are interconnected to form a gating / logic network, it is known as a *combinational logic circuit*
- The Boolean algebra expression for a given logic circuit can be derived by systematically progressing from input to output on the gates
- The three logic gates (AND, OR, and NOT) are logically complete because any Boolean expression can be realized as a logic circuit using only these three gates

Finding Boolean Expression of a Logic Circuit (Example 1)

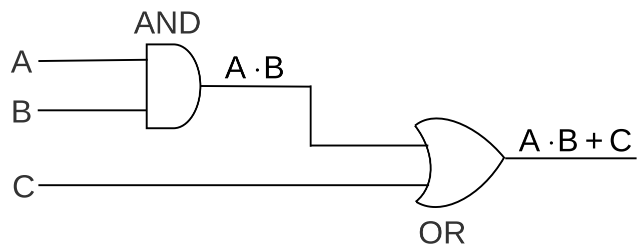


Finding Boolean Expression of a Logic Circuit (Example 2)



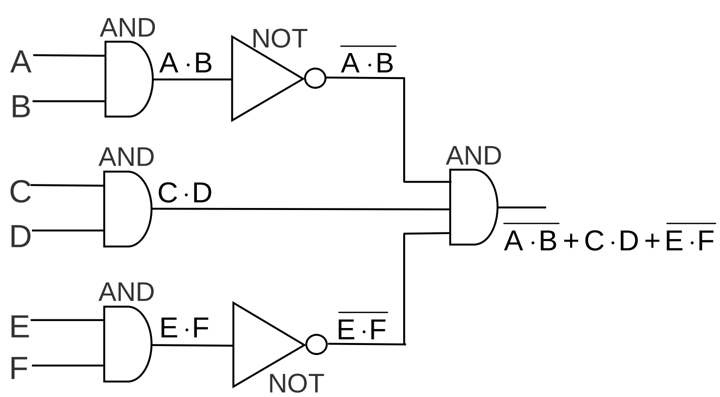
Constructing a Logic Circuit from a Boolean Expression (Example 1)

Boolean Expression = $A \cdot B + C$



Constructing a Logic Circuit from a Boolean Expression (Example 2)

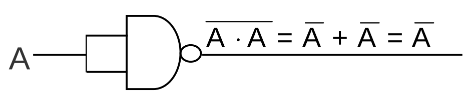
Boolean Expression = $\overline{A \cdot B} + C \cdot D + \overline{E \cdot F}$



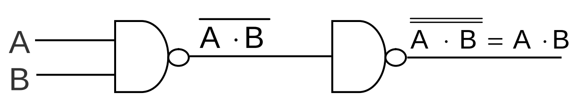
Universal NAND Gate

- NAND gate is an universal gate, it is alone sufficient to implement any Boolean expression
- To understand this, consider:
  - Basic logic gates (AND, OR, and NOT) are logically complete
  - Sufficient to show that AND, OR, and NOT gates can be implemented with NAND gates

Implementation of NOT, AND and OR Gates by NAND Gates



(a) NOT gate implementation.

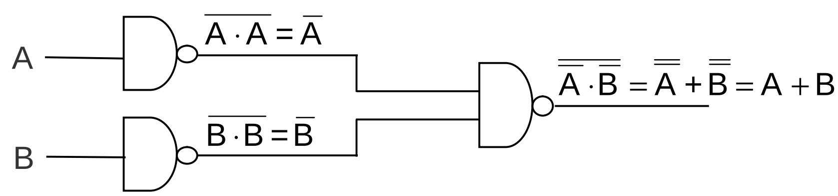


(b) AND gate implementation.

(Continued on next slide)

Implementation of NOT, AND and OR Gates by NAND Gates

(Continued from previous slide..)



(c) OR gate implementation.

Method of Implementing a Boolean Expression with Only NAND Gates

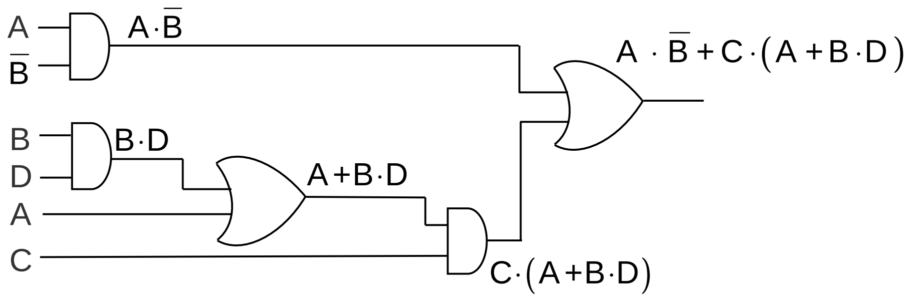
Step 1: From the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement $(\overline{A})$ inputs are available

Step 2: Draw a second logic diagram with the equivalent NAND logic substituted for each AND, OR, and NOT gate

Step 3: Remove all pairs of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable

Implementing a Boolean Expression with Only NAND Gates (Example)

Boolean Expression = $A \cdot \overline{B} + C \cdot (A + B \cdot D)$



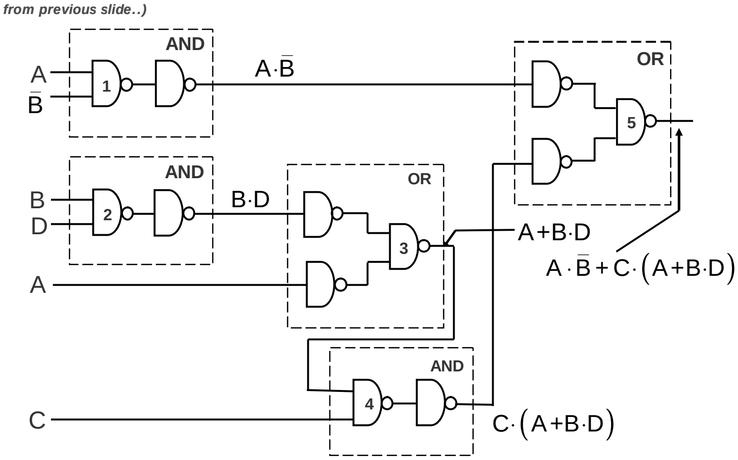
(a) Step 1: AND/OR implementation

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

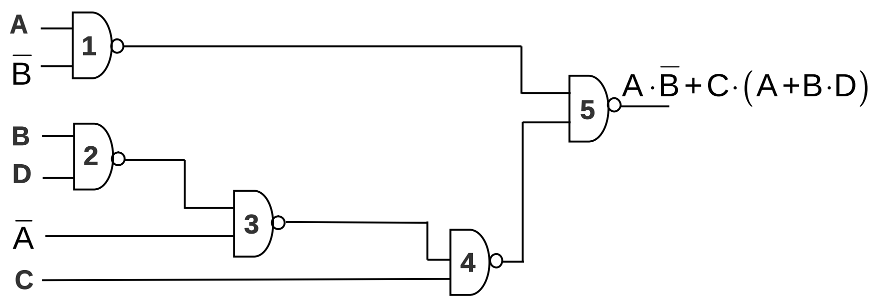


(b) Step 2: Substituting equivalent NAND functions

(Continued on next slide)

Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

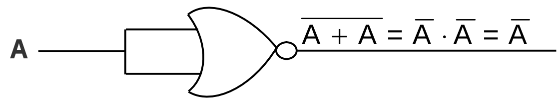


(c) Step 3: NAND implementation.

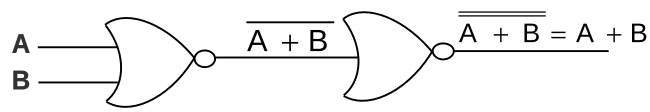
Universal NOR Gate

- NOR gate is an universal gate, it is alone sufficient to implement any Boolean expression
- To understand this, consider:
    - Basic logic gates (AND, OR, and NOT) are logically complete
    - Sufficient to show that AND, OR, and NOT gates can be implemented with NOR gates

Implementation of NOT, OR and AND Gates by NOR Gates



(a) NOT gate implementation.

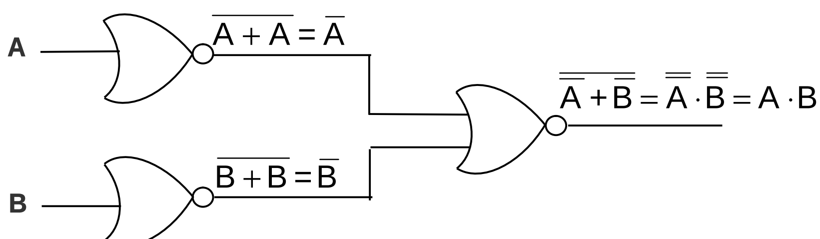


(b) OR gate implementation.

(Continued on next slide)

Implementation of NOT, OR and AND Gates by NOR Gates

(Continued from previous slide..)



(c) AND gate implementation.

Method of Implementing a Boolean Expression with Only NOR Gates

Step 1: For the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement $(\overline{A})$ inputs are available

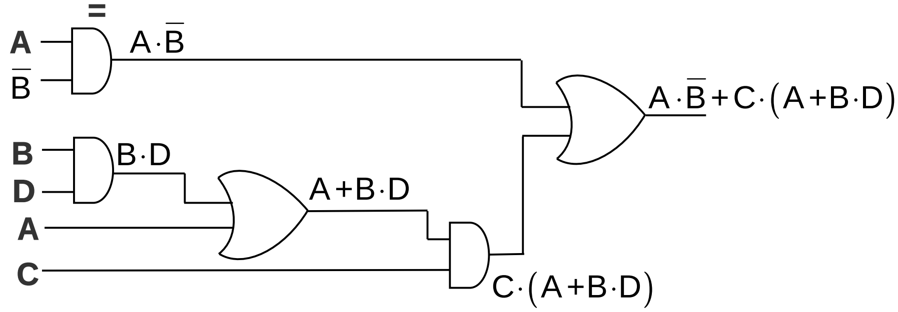
Step 2: Draw a second logic diagram with equivalent NOR logic substituted for each AND, OR, and NOT gate

Step 3: Remove all parts of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable

Implementing a Boolean Expression with Only NOR Gates (Examples)

(Continued from previous slide..)

Boolean Expression = $A \cdot \overline{B} + C \cdot (A + B \cdot D)$

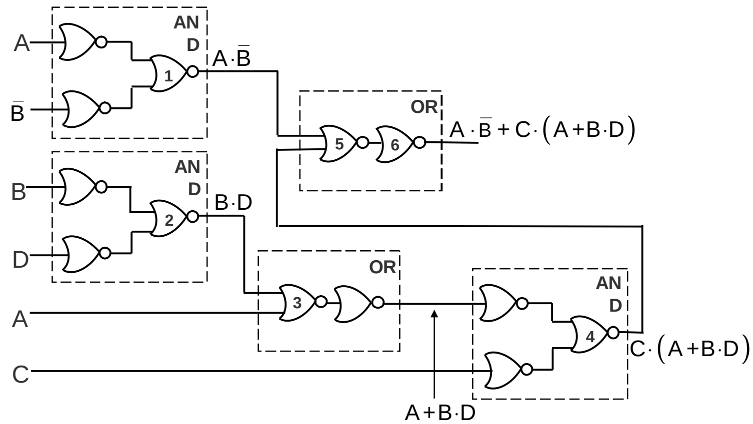


(a) Step 1: AND/OR implementation.

(Continued on next slide)

Implementing a Boolean Expression with Only NOR Gates (Examples)

(Continued from previous slide..)



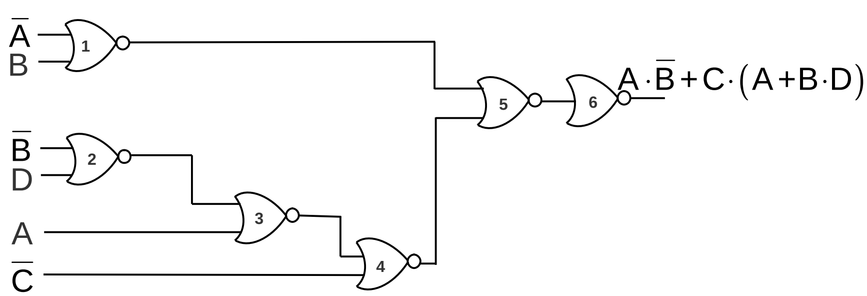
(b) Step 2: Substituting equivalent NOR functions.

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Implementing a Boolean Expression with Only NOR Gates (Examples)

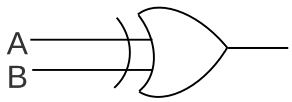
(Continued from previous slide..)



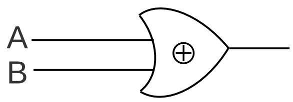
(c) Step 3: NOR implementation.

Exclusive-OR Function

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$



$$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$



$$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$

Also, $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

(Continued on next slide)

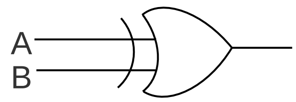
Exclusive-OR Function (Truth Table)

(Continued from previous slide..)

Inputs		Output
A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Equivalence Function with Block Diagram Symbol

A € B = $A \cdot B + \overline{A} \cdot \overline{B}$



C = A € B = $A \cdot B + \overline{A} \cdot \overline{B}$

Also, (A € B) € = A € (B € C) = A € B € C

(Continued on next slide)

Equivalence Function (Truth Table)

Inputs		Output
A	**B**	**C = A € B**
0	0	1
0	1	0
1	0	0
1	1	1

Steps in Designing Combinational Circuits

1. State the given problem completely and exactly
2. Interpret the problem and determine the available input variables and required output variables
3. Assign a letter symbol to each input and output variables
4. Design the truth table that defines the required relations between inputs and outputs
5. Obtain the simplified Boolean function for each output
6. Draw the logic circuit diagram to implement the Boolean function

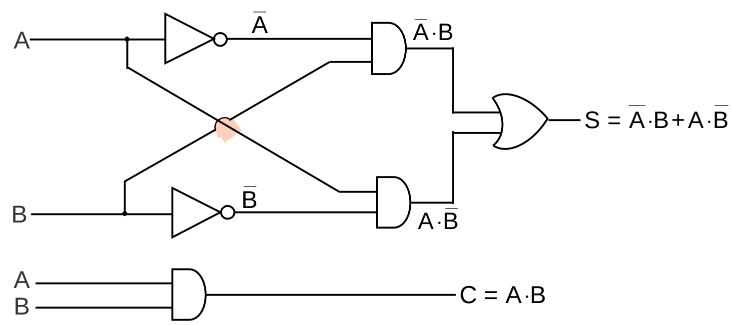
Designing a Combinational Circuit Example 1 – Half-Adder Design

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\left.\begin{aligned} S &= \overline{A}\cdot B + A\cdot\overline{B} \\ C &= A\cdot B \end{aligned}\right\}$$ Boolean functions for the two outputs.

Designing a Combinational Circuit Example 1 – Half-Adder Design

(Continued from previous slide..)



Logic circuit diagram to implement the Boolean functions

Designing a Combinational Circuit Example 2 – Full-Adder Design

Inputs			Outputs	
A	B	D	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table for a full adder

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)

Boolean functions for the two outputs:

$$S = \bar{A}\cdot\bar{B}\cdot D + \bar{A}\cdot B\cdot\bar{D} + A\cdot\bar{B}\cdot\bar{D} + A\cdot B\cdot D$$

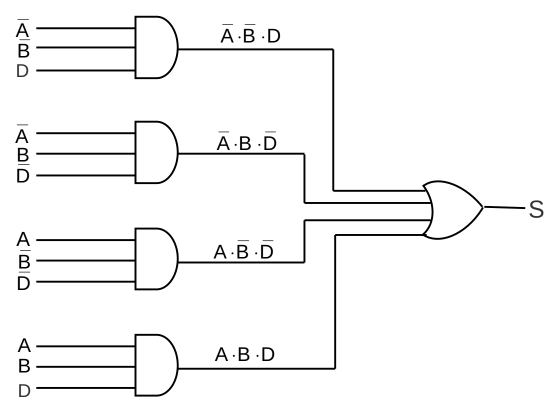
$$C = \bar{A}\cdot B\cdot D + A\cdot\bar{B}\cdot D + A\cdot B\cdot\bar{D} + A\cdot B\cdot D$$

$$= A\cdot B + A\cdot D + B\cdot D \quad \textbf{(when simplified)}$$

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)

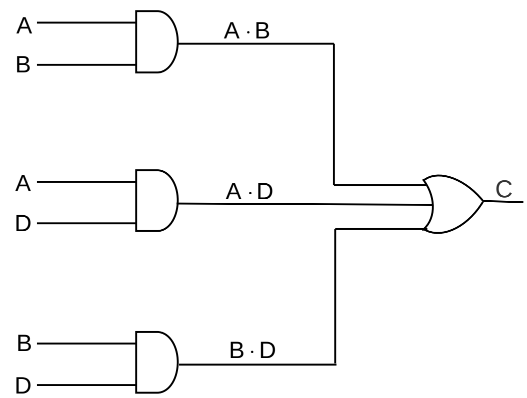


(a) Logic circuit diagram for sums

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)



(b) Logic circuit diagram for carry

Key Words / Phrases

- Absorption law
- AND gate
- Associative law
- Boolean algebra
- Boolean expression
- Boolean functions
- Boolean identities
- Canonical forms for Boolean functions
- Combination logic circuits
- Cumulative law
- Complement of a function
- Complementation
- De Morgan's law
- Distributive law
- Dual identities
- Equivalence function
- Exclusive-OR function
- Exhaustive enumeration method
- Half-adder
- Idempotent law
- Involution law
- Literal
- Logic circuits
- Logic gates
- Logical addition
- Logical multiplication
- Maxterms
- Minimization of Boolean functions
- Minterms
- NAND gate
- NOT gate
- Operator precedence
- OR gate
- Parallel Binary Adder
- Perfect induction method
- Postulates of Boolean algebra
- Principle of duality
- Product-of-Sums expression
- Standard forms
- Sum-of Products expression
- Truth table
- Universal NAND gate
- Universal NOR gate

Chapter 06

Boolean Algebra and Logic Circuits

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- **Boolean algebra**
- **Fundamental concepts and basic laws of Boolean algebra**
- **Boolean function and minimization**
- **Logic gates**
- **Logic circuits and Boolean expressions**
- **Combinational circuits and design**

Boolean Algebra

- An algebra that deals with binary number system
- George Boole (1815-1864), an English mathematician, developed it for:
  - Simplifying representation
  - Manipulation of propositional logic
- In 1938, Claude E. Shannon proposed using Boolean algebra in design of relay switching circuits
- Provides economical and straightforward approach
- Used extensively in designing electronic circuits used in computers

Fundamental Concepts of Boolean Algebra

- Use of Binary Digit
  - Boolean equations can have either of two possible values, 0 and 1
- Logical Addition
  - Symbol '+', also known as 'OR' operator, used for logical addition. Follows law of binary addition
- Logical Multiplication
  - Symbol '.', also known as 'AND' operator, used for logical multiplication. Follows law of binary multiplication
- Complementation
  - Symbol '-', also known as 'NOT' operator, used for complementation. Follows law of binary compliment

Operator Precedence

- Each operator has a precedence level
- Higher the operator's precedence level, earlier it is evaluated
- Expression is scanned from left to right
- First, expressions enclosed within parentheses are evaluated
- Then, all complement (NOT) operations are performed
- Then, all '·' (AND) operations are performed
- Finally, all '+' (OR) operations are performed

(Continued on next slide)

Operator Precedence

(Continued from previous slide..)

$$\boxed{X + \boxed{\boxed{\overline{Y}}_{1^{st}} \cdot Z}_{2^{nd}}}_{3^{rd}}$$

Postulates of Boolean Algebra

Postulate 1:

(a) A = 0, if and only if, A is not equal to 1
(b) A = 1, if and only if, A is not equal to 0

Postulate 2:

(a) $x + 0 = x$
(b) $x \cdot 1 = x$

Postulate 3: Commutative Law

(a) $x + y = y + x$
(b) $x \cdot y = y \cdot x$

(Continued on next slide)

Postulates of Boolean Algebra

(Continued from previous slide..)

Postulate 4: Associative Law

(a) $x + (y + z) = (x + y) + z$
(b) $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Postulate 5: Distributive Law

(a) $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
(b) $x + (y \cdot z) = (x + y) \cdot (x + z)$

Postulate 6:

(a) $x + \overline{x} = 1$
(b) $x \cdot \overline{x} = 0$

The Principle of Duality

There is a precise duality between the operators . (AND) and + (OR), and the digits 0 and 1.

For example, in the table below, the second row is obtained from the first row and vice versa simply by interchanging '+' with '.' and '0' with '1'

	Column 1	Column 2	Column 3
Row 1	$1 + 1 = 1$	$1 + 0 = 0 + 1 = 1$	$0 + 0 = 0$
Row 2	$0 \cdot 0 = 0$	$0 \cdot 1 = 1 \cdot 0 = 0$	$1 \cdot 1 = 1$

Therefore, if a particular theorem is proved, its dual theorem automatically holds and need not be proved separately

Some Important Theorems of Boolean Algebra

Sr. No.	Theorems/ Identities	Dual Theorems/ Identities	Name (if any)
1	$x + x = x$	$x \cdot x = x$	Idempotent Law
2	$x + 1 = 1$	$x \cdot 0 = 0$	
3	$x + x \cdot y = x$	$x \cdot x + y = x$	Absorption Law
4	$\bar{\bar{x}} = x$		Involution Law
5	$x \cdot \bar{x} + y = x \cdot y$	$x + \bar{x} \cdot y = x + y$	
6	$\overline{x+y} = \bar{x}\,\bar{y} \cdot$	$\overline{x \cdot y} = \bar{x}\ \bar{y} +$	De Morgan's Law

Methods of Proving Theorems

The theorems of Boolean algebra may be proved by using one of the following methods:

1. By using postulates to show that L.H.S. = R.H.S
2. By *Perfect Induction* or *Exhaustive Enumeration* method where all possible combinations of variables involved in L.H.S. and R.H.S. are checked to yield identical results
3. By the *Principle of Duality* where the dual of an already proved theorem is derived from the proof of its corresponding pair

Proving a Theorem by Using Postulates (Example)

Theorem:

$x + x \cdot y = x$

Proof:

L.H.S.

$= x + x \cdot y$	
$= x \cdot 1 + x \cdot y$	by postulate 2(b)
$= x \cdot (1 + y)$	by postulate 5(a)
$= x \cdot (y + 1)$	by postulate 3(a)
$= x \cdot 1$	by theorem 2(a)
$= x$	by postulate 2(b)
= R.H.S.	

Proving a Theorem by Perfect Induction (Example)

Theorem:

$x + x \cdot y = x$

x	y	$x \cdot y$	$x + x \cdot y$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Proving a Theorem by the Principle of Duality (Example)

Theorem:

$x + x = x$

Proof:

L.H.S.

$= x + x$	
$= (x + x) \cdot 1$	by postulate 2(b)
$= (x + x) \cdot (x + \bar{x})$	by postulate 6(a)
$= x + x \cdot \bar{x}$	by postulate 5(b)
$= x + 0$	by postulate 6(b)
$= x$	by postulate 2(a)
= R.H.S.	

(Continued on next slide)

Proving a Theorem by the Principle of Duality (Example)

(Continued from previous slide..)

Dual Theorem:

$x \cdot x = x$

Proof:

L.H.S.

$= x \cdot x$	
$= x \cdot x + 0$	by postulate 2(a)
$= x \cdot x + x \cdot \bar{x}$	by postulate 6(b)
$= x \cdot (x + \bar{x})$	by postulate 5(a)
$= x \cdot 1$	by postulate 6(a)
$= x$	by postulate 2(b)
= R.H.S.	

Notice that each step of the proof of the dual theorem is derived from the proof of its corresponding pair in the original theorem

Boolean Functions

- A Boolean function is an expression formed with:
  - Binary variables
  - Operators (OR, AND, and NOT)
  - Parentheses, and equal sign
- The value of a Boolean function can be either 0 or 1
- A Boolean function may be represented as:
  - An algebraic expression, or
  - A truth table

Representation as an Algebraic Expression

$$W = X + \overline{Y} \cdot Z$$

- Variable W is a function of X, Y, and Z, can also be written as W = f (X, Y, Z)
- The RHS of the equation is called an ***expression***
- The symbols X, Y, Z are the ***literals*** of the function
- For a given Boolean function, there may be more than one algebraic expressions

Representation as a Truth Table

X	Y	Z	W
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(Continued on next slide)

Representation as a Truth Table

(Continued from previous slide..)

- The number of rows in the table is equal to 2^n, where *n* is the number of literals in the function
- The combinations of 0s and 1s for rows of this table are obtained from the binary numbers by counting from 0 to 2^n - 1

Minimization of Boolean Functions

- Minimization of Boolean functions deals with
  - Reduction in number of literals
  - Reduction in number of terms
- Minimization is achieved through manipulating expression to obtain equal and simpler expression(s) (having fewer literals and/or terms)

(Continued on next slide)

Minimization of Boolean Functions

(Continued from previous slide..)

$$F_1 = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot z + x \cdot \overline{y}$$

F_1 has 3 literals (x, y, z) and 3 terms

$$F_2 = x \cdot \overline{y} + \overline{x} \cdot z$$

F_2 has 3 literals (x, y, z) and 2 terms

F_2 can be realized with fewer electronic components, resulting in a cheaper circuit

(Continued on next slide)

Minimization of Boolean Functions

(Continued from previous slide..)

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Both F_1 and F_2 produce the same result

Try out some Boolean Function Minimization

(a) $x + \bar{x} \cdot y$

(b) $x \cdot \left(\bar{x} + y\right)$

(c) $\bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot z + x \cdot \bar{y}$

(d) $x \cdot y + \bar{x} \cdot z + y \cdot z$

(e) $\left(x + y\right) \cdot \left(\bar{x} + z\right) \cdot \left(y + z\right)$

Complement of a Boolean Function

- **The complement of a Boolean function is obtained by interchanging:**
  - **Operators OR and AND**
  - **Complementing each literal**
- **This is based on *De Morgan's theorems*, whose general form is:**

$$\overline{A_1 + A_2 + A_3 + \ldots + A_n} = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 \cdot \ldots \cdot \bar{A}_n$$
$$\overline{A_1 \cdot A_2 \cdot A_3 \cdot \ldots \cdot A_n} = \bar{A}_1 + \bar{A}_2 + \bar{A}_3 + \ldots + \bar{A}_n$$

Complementing a Boolean Function (Example)

$$F_1 = \bar{x} \cdot y \cdot \bar{z} + \bar{x} \cdot \bar{y} \cdot z$$

To obtain $\bar{F_1}$, we first interchange the OR and the AND operators giving

$$(\bar{x} + y + \bar{z}) \cdot (\bar{x} + \bar{y} + z)$$

Now we complement each literal giving

$$\bar{F_1} = (x + \bar{y} + z) \cdot (x + y + \bar{z})$$

Canonical Forms of Boolean Functions

Minterms : ***n*** **variables forming an AND term, with each variable being primed or unprimed, provide 2^n possible combinations called** ***minterms*** **or** ***standard products***

Maxterms : ***n*** **variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations called** ***maxterms*** **or** ***standard sums***

Minterms and Maxterms for three Variables

Variables			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$\bar{x} \cdot \bar{y} \cdot \bar{z}$	m_0	$x + y + z$	M_0
0	0	1	$\bar{x} \cdot \bar{y} \cdot z$	m_1	$x + y + \bar{z}$	M_1
0	1	0	$\bar{x} \cdot y \cdot \bar{z}$	m_2	$x + \bar{y} + z$	M_2
0	1	1	$\bar{x} \cdot y \cdot z$	m_3	$x + \bar{y} + \bar{z}$	M_3
1	0	0	$x \cdot \bar{y} \cdot \bar{z}$	m_4	$\bar{x} + y + z$	M_4
1	0	1	$x \cdot \bar{y} \cdot z$	m_5	$\bar{x} + y + \bar{z}$	M_5
1	1	0	$x \cdot y \cdot \bar{z}$	m_6	$\bar{x} + \bar{y} + z$	M_6
1	1	1	$x \cdot y \cdot z$	m_7	$\bar{x} + \bar{y} + \bar{z}$	M_7

Note that each minterm is the complement of its corresponding maxterm and vice-versa

Sum-of-Products (SOP) Expression

A sum-of-products (SOP) expression is a product term (minterm) or several product terms (minterms) logically added (ORed) together. Examples are:

x	$x+y$
$x+y\cdot z$	$x\cdot y+z$
$x\cdot\overline{y}+\overline{x}\cdot y$	$\overline{x}\cdot\overline{y}+x\cdot\overline{y}\cdot z$

Steps to Express a Boolean Function in its Sum-of-Products Form

1. Construct a truth table for the given Boolean function
2. Form a minterm for each combination of the variables, which produces a 1 in the function
3. The desired expression is the sum (OR) of all the minterms obtained in Step 2

Expressing a Function in its Sum-of-Products Form (Example)

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The following 3 combinations of the variables produce a 1:
001, 100, and 111

(Continued on next slide)

Expressing a Function in its Sum-of-Products Form (Example)

(Continued from previous slide..)

- Their corresponding minterms are:

  $\bar{x} \cdot \bar{y} \cdot z, \quad x \cdot \bar{y} \cdot \bar{z}, \quad \text{and} \quad x \cdot y \cdot z$

- Taking the OR of these minterms, we get

$$F_1 = \bar{x} \cdot \bar{y} \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot z = m_1 + m_4 + m_7$$
$$F_1(x \cdot y \cdot z) = \Sigma(1, 4, 7)$$

Product-of Sums (POS) Expression

A product-of-sums (POS) expression is a sum term (maxterm) or several sum terms (maxterms) logically multiplied (ANDed) together. Examples are:

x	$(x+\bar{y}) \cdot (\bar{x}+y) \cdot (\bar{x}+\bar{y})$
$\bar{x}+y$	$(x+y) \cdot (\bar{x}+y+z)$
$(\bar{x}+\bar{y}) \cdot z$	$(\bar{x}+y) \cdot (x+\bar{y})$

Steps to Express a Boolean Function in its Product-of-Sums Form

1. Construct a truth table for the given Boolean function
2. Form a maxterm for each combination of the variables, which produces a 0 in the function
3. The desired expression is the product (AND) of all the maxterms obtained in Step 2

Expressing a Function in its Product-of-Sums Form

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- The following 5 combinations of variables produce a 0:
  000, 010, 011, 101, and 110

(Continued on next slide)

Expressing a Function in its Product-of-Sums Form

(Continued from previous slide..)

- Their corresponding maxterms are:

$$\left(x+y+z\right),\left(x+\bar{y}+z\right),\left(x+\bar{y}+\bar{z}\right),$$
$$\left(\bar{x}+y+\bar{z}\right) \text{ and } \left(\bar{x}+\bar{y}+z\right)$$

- Taking the AND of these maxterms, we get:

$$F_1=\left(x+y+z\right)\cdot\left(x+\bar{y}+z\right)\cdot\left(x+\bar{y}+\bar{z}\right)\cdot\left(\bar{x}+y+\bar{z}\right)\cdot$$
$$\left(\bar{x}+\bar{y}+z\right)=M_0\cdot M_2\cdot M_3\cdot M_5\cdot M_6$$
$$F_1\left(x,y,z\right)=\Pi\left(0,2,3,5,6\right)$$

Conversion Between Canonical Forms (Sum-of-Products and Product-of-Sums)

To convert from one canonical form to another, interchange the symbol and list those numbers missing from the original form.

Example:

$$F\left(x,y,z\right)=\Pi\left(0,2,4,5\right)=\Sigma\left(1,3,6,7\right)$$
$$F\left(x,y,z\right)=\Pi\left(1,4,7\right)=\Sigma\left(0,2,3,5,6\right)$$

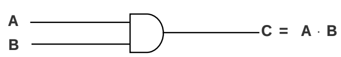
Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce standard output signal
- Are the building blocks of all the circuits in a computer
- Some of the most basic and useful logic gates are AND, OR, NOT, NAND and NOR gates

AND Gate

- Physical realization of logical multiplication (AND) operation
- Generates an output signal of 1 only if all input signals are also 1

AND Gate (Block Diagram Symbol and Truth Table)

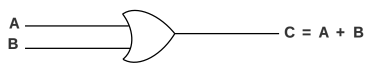


Inputs		Output
A	B	$C = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

- Physical realization of logical addition (OR) operation
- Generates an output signal of 1 if at least one of the input signals is also 1

OR Gate (Block Diagram Symbol and Truth Table)

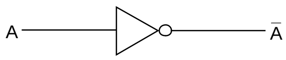


Inputs		Output
A	B	C = A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

- Physical realization of complementation operation
- Generates an output signal, which is the reverse of the input signal

NOT Gate (Block Diagram Symbol and Truth Table)

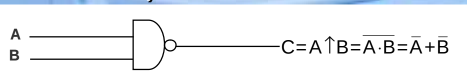


Input	Output
A	$\bar{A}$
0	1
1	0

NAND Gate

- Complemented AND gate
- Generates an output signal of:
    - 1 if any one of the inputs is a 0
    - 0 when all the inputs are 1

NAND Gate (Block Diagram Symbol and Truth Table)

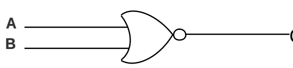


Inputs		Output
A	B	$C = \bar{A} + \bar{B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

- Complemented OR gate
- Generates an output signal of:
  - 1 only when all inputs are 0
  - 0 if any one of inputs is a 1

NOR Gate (Block Diagram Symbol and Truth Table)

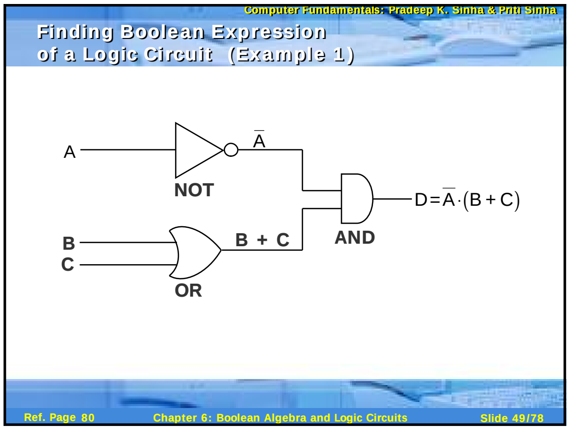


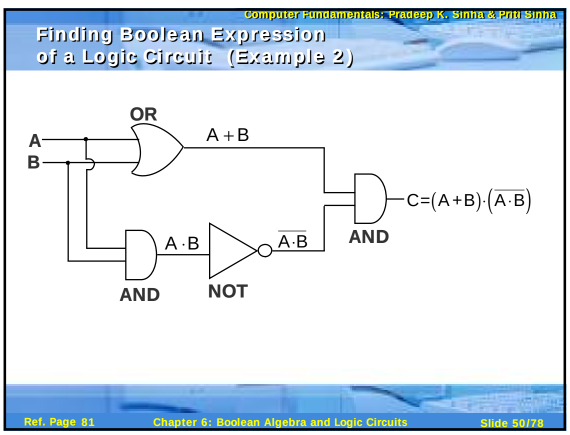
$C = A \downarrow B = \overline{A+B} = \bar{A} \cdot \bar{B}$

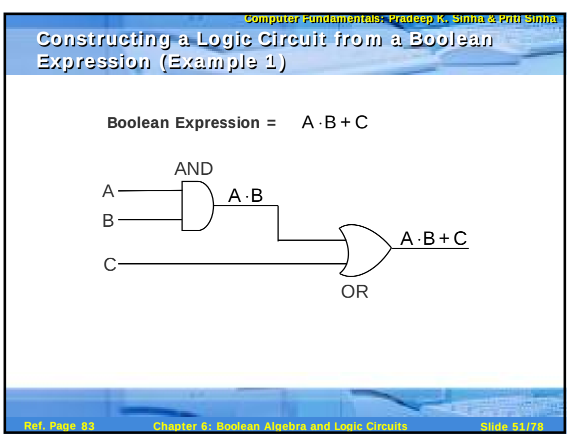
Inputs		Output
A	B	$C = \bar{A} \cdot \bar{B}$
0	0	1
0	1	0
1	0	0
1	1	0

Logic Circuits

- When logic gates are interconnected to form a gating / logic network, it is known as a *combinational logic circuit*
- The Boolean algebra expression for a given logic circuit can be derived by systematically progressing from input to output on the gates
- The three logic gates (AND, OR, and NOT) are logically complete because any Boolean expression can be realized as a logic circuit using only these three gates

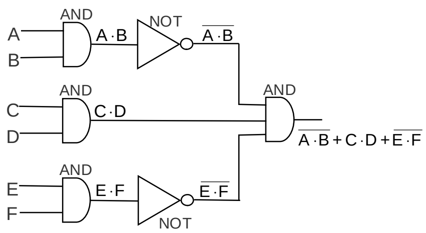

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Finding Boolean Expression of a Logic Circuit (Example 1)
A
NOT
$\overline{A}$
B
C
OR
B + C
AND
$D = \overline{A} \cdot (B + C)$
Ref. Page 80
Chapter 6: Boolean Algebra and Logic Circuits
Slide 49/78


Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Finding Boolean Expression of a Logic Circuit (Example 2)
OR
A
B
$A + B$
$C = (A + B) \cdot \left(\overline{A \cdot B}\right)$
AND
$A \cdot B$
$\overline{A \cdot B}$
AND
NOT
Ref. Page 81
Chapter 6: Boolean Algebra and Logic Circuits
Slide 50/78


Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Constructing a Logic Circuit from a Boolean Expression (Example 1)
Boolean Expression = $A \cdot B + C$
AND
A
B
$A \cdot B$
C
$A \cdot B + C$
OR
Ref. Page 83
Chapter 6: Boolean Algebra and Logic Circuits
Slide 51/78

Constructing a Logic Circuit from a Boolean Expression (Example 2)

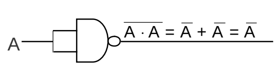
Boolean Expression = $\overline{A \cdot B} + C \cdot D + \overline{E \cdot F}$



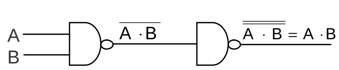
Universal NAND Gate

- NAND gate is an universal gate, it is alone sufficient to implement any Boolean expression
- To understand this, consider:
  - Basic logic gates (AND, OR, and NOT) are logically complete
  - Sufficient to show that AND, OR, and NOT gates can be implemented with NAND gates

Implementation of NOT, AND and OR Gates by NAND Gates



(a) NOT gate implementation.

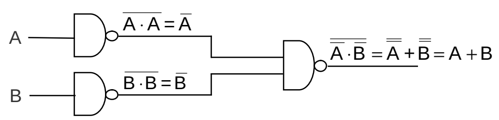


(b) AND gate implementation.

(Continued on next slide)

Implementation of NOT, AND and OR Gates by NAND Gates

(Continued from previous slide..)



(c) OR gate implementation.

Method of Implementing a Boolean Expression with Only NAND Gates

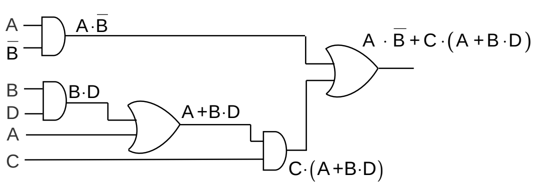
Step 1: From the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement $(\overline{A})$ inputs are available

Step 2: Draw a second logic diagram with the equivalent NAND logic substituted for each AND, OR, and NOT gate

Step 3: Remove all pairs of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable

Implementing a Boolean Expression with Only NAND Gates (Example)

Boolean Expression = $A\cdot\overline{B}+C\cdot(A+B\cdot D)$

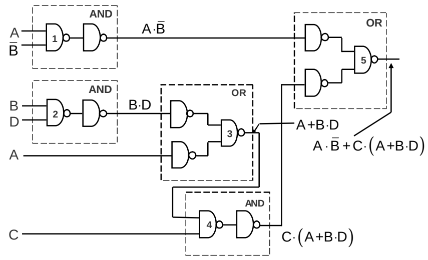


(a) Step 1: AND/OR implementation

(Continued on next slide)

Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

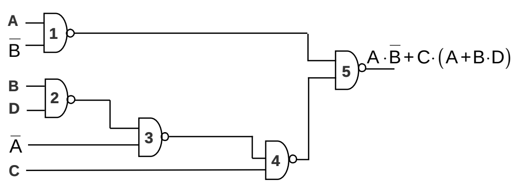


(b) Step 2: Substituting equivalent NAND functions

(Continued on next slide)

Implementing a Boolean Expression with Only NAND Gates (Example)

(Continued from previous slide..)

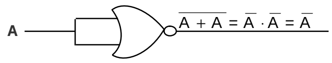


(c) Step 3: NAND implementation.

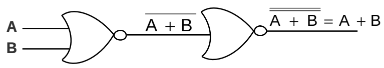
Universal NOR Gate

- NOR gate is an universal gate, it is alone sufficient to implement any Boolean expression
- To understand this, consider:
  - Basic logic gates (AND, OR, and NOT) are logically complete
  - Sufficient to show that AND, OR, and NOT gates can be implemented with NOR gates

Implementation of NOT, OR and AND Gates by NOR Gates



(a) NOT gate implementation.



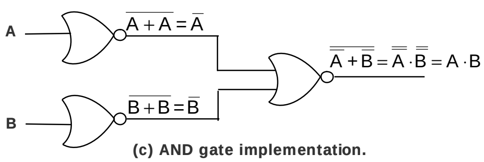
(b) OR gate implementation.

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Implementation of NOT, OR and AND Gates by NOR Gates

(Continued from previous slide..)



(c) AND gate implementation.

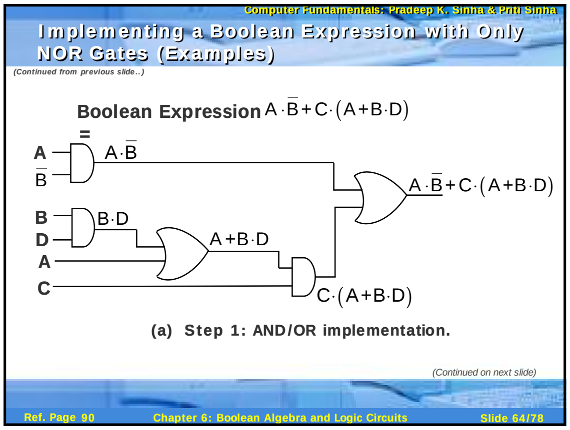
Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

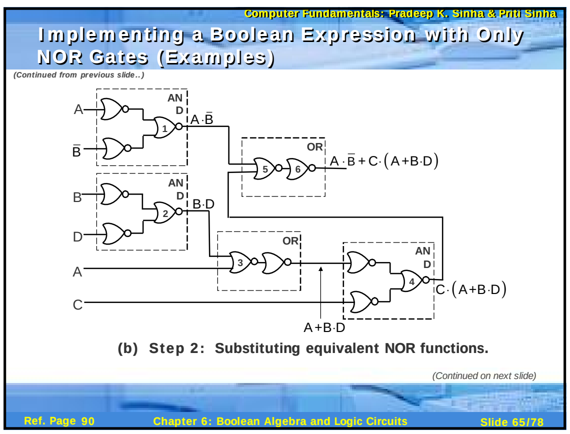
Method of Implementing a Boolean Expression with Only NOR Gates

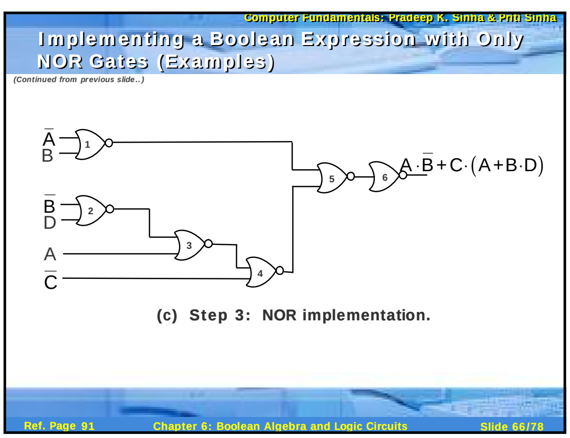
Step 1: For the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement $(\overline{A})$ inputs are available

Step 2: Draw a second logic diagram with equivalent NOR logic substituted for each AND, OR, and NOT gate

Step 3: Remove all parts of cascaded inverters from the diagram as double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable

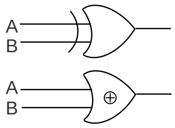

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Implementing a Boolean Expression with Only NOR Gates (Examples)
(Continued from previous slide..)
Boolean Expression = $A \cdot \overline{B} + C \cdot (A + B \cdot D)$
$A \cdot \overline{B}$
$B \cdot D$
$A + B \cdot D$
$C \cdot (A + B \cdot D)$
$A \cdot \overline{B} + C \cdot (A + B \cdot D)$
(a) Step 1: AND/OR implementation.
(Continued on next slide)
Ref. Page 90
Chapter 6: Boolean Algebra and Logic Circuits
Slide 64/78


Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Implementing a Boolean Expression with Only NOR Gates (Examples)
(Continued from previous slide..)
AND
OR
$A \cdot \overline{B}$
$B \cdot D$
$A + B \cdot D$
$C \cdot (A + B \cdot D)$
$A \cdot \overline{B} + C \cdot (A + B \cdot D)$
(b) Step 2: Substituting equivalent NOR functions.
(Continued on next slide)
Ref. Page 90
Chapter 6: Boolean Algebra and Logic Circuits
Slide 65/78


Computer Fundamentals: Pradeep K. Sinha & Priti Sinha
Implementing a Boolean Expression with Only NOR Gates (Examples)
(Continued from previous slide..)
$A \cdot \overline{B} + C \cdot (A + B \cdot D)$
(c) Step 3: NOR implementation.
Ref. Page 91
Chapter 6: Boolean Algebra and Logic Circuits
Slide 66/78

Exclusive-OR Function

$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$



$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$

$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$

Also, $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

(Continued on next slide)

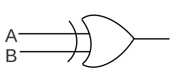
Exclusive-OR Function (Truth Table)

(Continued from previous slide..)

Inputs		Output
A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Equivalence Function with Block Diagram Symbol

A € B = $A \cdot B + \overline{A} \cdot \overline{B}$



C = A € B = $A \cdot B + \overline{A} \cdot \overline{B}$

Also, (A € B) € = A € (B € C) = A € B € C

(Continued on next slide)

Equivalence Function (Truth Table)

Inputs		Output
A	B	C = A € B
0	0	1
0	1	0
1	0	0
1	1	1

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Steps in Designing Combinational Circuits

1. State the given problem completely and exactly
2. Interpret the problem and determine the available input variables and required output variables
3. Assign a letter symbol to each input and output variables
4. Design the truth table that defines the required relations between inputs and outputs
5. Obtain the simplified Boolean function for each output
6. Draw the logic circuit diagram to implement the Boolean function

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

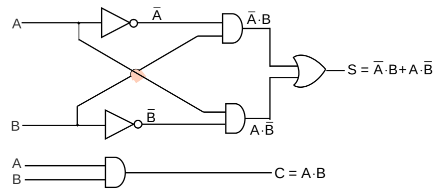
Designing a Combinational Circuit Example 1 – Half-Adder Design

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\left.\begin{array}{l} S = \overline{A} \cdot B + A \cdot \overline{B} \\ C = A \cdot B \end{array}\right\}$$ Boolean functions for the two outputs.

Designing a Combinational Circuit
Example 1 – Half-Adder Design

(Continued from previous slide..)



Logic circuit diagram to implement the Boolean functions

Designing a Combinational Circuit
Example 2 – Full-Adder Design

Inputs			Outputs	
A	B	D	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table for a full adder

(Continued on next slide)

Designing a Combinational Circuit
Example 2 – Full-Adder Design

(Continued from previous slide..)

Boolean functions for the two outputs:

$$S = \bar{A} \cdot \bar{B} \cdot D + \bar{A} \cdot B \cdot \bar{D} + A \cdot \bar{B} \cdot \bar{D} + A \cdot B \cdot D$$

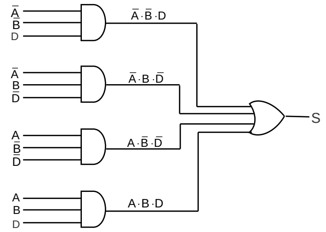
$$C = \bar{A} \cdot B \cdot D + A \cdot \bar{B} \cdot D + A \cdot B \cdot \bar{D} + A \cdot B \cdot D$$

$$= A \cdot B + A \cdot D + B \cdot D$$ **(when simplified)**

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)

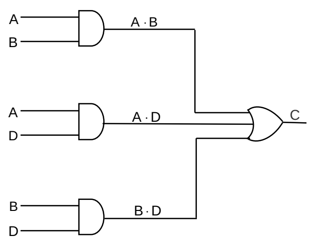


(a) Logic circuit diagram for sums

(Continued on next slide)

Designing a Combinational Circuit Example 2 – Full-Adder Design

(Continued from previous slide..)



(b) Logic circuit diagram for carry

Key Words/Phrases

- Absorption law
- AND gate
- Associative law
- Boolean algebra
- Boolean expression
- Boolean functions
- Boolean identities
- Canonical forms for Boolean functions
- Combination logic circuits
- Cumulative law
- Complement of a function
- Complementation
- De Morgan's law
- Distributive law
- Dual identities
- Equivalence function
- Exclusive-OR function
- Exhaustive enumeration method
- Half-adder
- Idempotent law
- Involution law
- Literal
- Logic circuits
- Logic gates
- Logical addition
- Logical multiplication
- Maxterms
- Minimization of Boolean functions
- Minterms
- NAND gate
- NOT gate
- Operator precedence
- OR gate
- Parallel Binary Adder
- Perfect induction method
- Postulates of Boolean algebra
- Principle of duality
- Product-of-Sums expression
- Standard forms
- Sum-of Products expression
- Truth table
- Universal NAND gate
- Universal NOR gate

Chapter 07

Processor and Memory

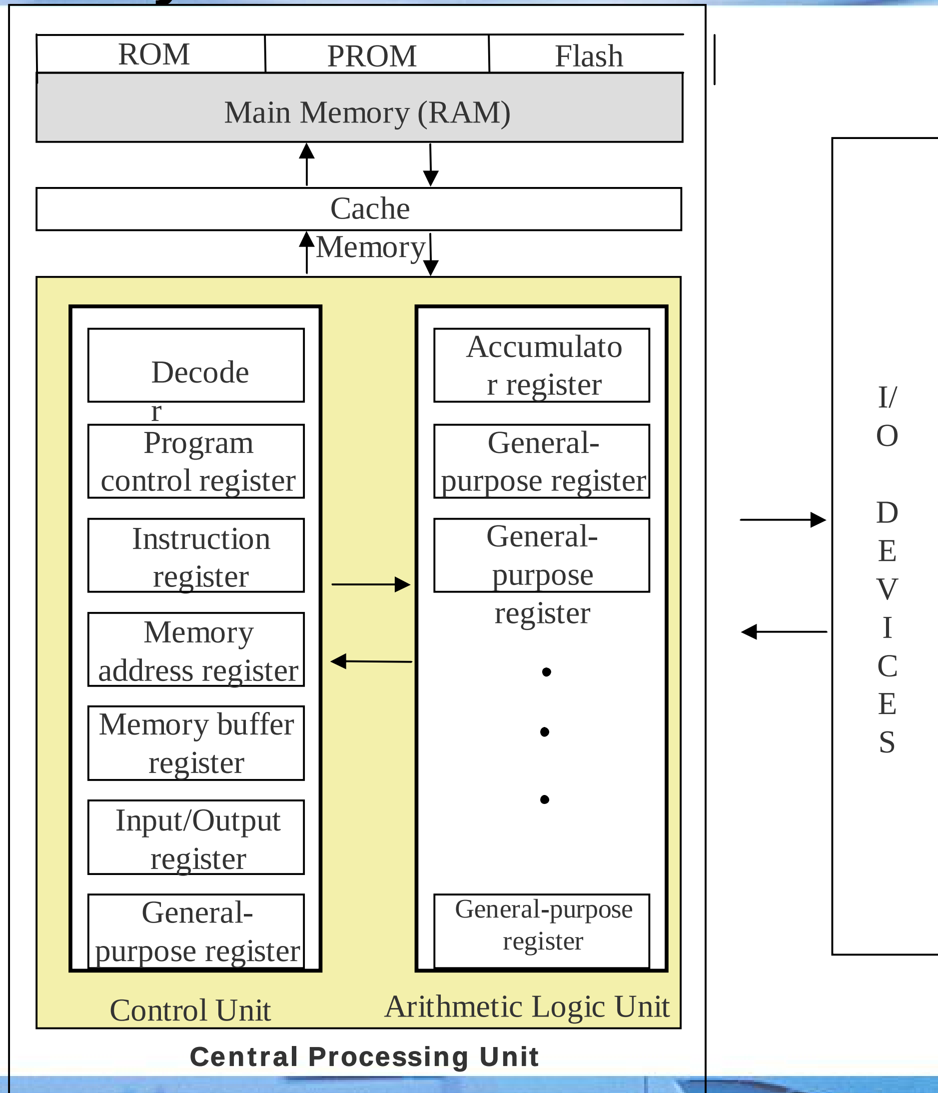
Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Internal structure of processor
- Memory structure
- Determining the speed of a processor
- Different types of processors available
- Determining the capacity of a memory
- Different types of memory available
- Several other terms related to the processor and main memory of a computer system

Basic Processor & Memory Architecture of a Computer System



Central Processing Unit (CPU)

- The *brain* of a computer system
- Performs all major calculations and comparisons
- Activates and controls the operations of other units of a computer system
- Two basic components are
  - Control Unit (CU)
  - Arithmetic Logic Unit (ALU)
- No other single component of a computer determines its overall performance as much as the CPU

Control Unit (CU)

- One of the two basic components of CPU
- Acts as the central nervous system of a computer system
- Selects and interprets program instructions, and coordinates execution
- Has some special purpose registers and a decoder to perform these activities

Arithmetic Logic Unit (ALU)

- One of the two basic components of CPU.
- Actual execution of instructions takes place in ALU
- Has some special purpose registers
- Has necessary circuitry to carry out all the arithmetic and logic operations included in the CPU instruction set

Instruction Set

- CPU has built-in ability to execute a particular set of machine instructions, called its *instruction set*
- Most CPUs have 200 or more instructions (such as add, subtract, compare, etc.) in their instruction set
- CPUs made by different manufacturers have different instruction sets
- Manufacturers tend to group their CPUs into “families” having similar instruction sets
- New CPU whose instruction set includes instruction set of its predecessor CPU is said to be *backward compatible* with its predecessor

Registers

- Special memory units, called registers, are used to hold information on a temporary basis as the instructions are interpreted and executed by the CPU
- Registers are part of the CPU (not main memory) of a computer
- The length of a register, sometimes called its *word size*, equals the number of bits it can store
- With all other parameters being the same, a CPU with 32-bit registers can process data twice larger than one with 16-bit registers

Functions of Commonly Used Registers

Sr. No.	Name of Register	Function
1	Memory Address (MAR)	Holds address of the active memory location
2	Memory Buffer (MBR)	Holds contents of the accessed (read/written) memory word
3	Program Control (PC)	Holds address of the next instruction to be executed
4	Accumulator (A)	Holds data to be operated upon, intermediate results, and the results
5	Instruction (I)	Holds an instruction while it is being executed
6	Input/Output (I/O)	Used to communicate with the I/O devices

Processor Speed

- Computer has a built-in *system clock* that emits millions of regularly spaced electric pulses per second (known as *clock cycles)*
- It takes one cycle to perform a basic operation, such as moving a byte of data from one memory location to another
- Normally, several clock cycles are required to fetch, decode, and execute a single program instruction
- Hence, shorter the clock cycle, faster the processor
- Clock speed (number of clock cycles per second) is measured in Megahertz (10^6 cycles/sec) or Gigahertz (10^9 cycles/sec)

Types of Processor

Type of Architecture	Features	Usage
CISC (Complex Instruction Set Computer)	§ Large instruction set § Variable-length instructions § Variety of addressing modes § Complex & expensive to produce	Mostly used in personal computers
RISC (Reduced Instruction Set Computer)	§ Small instruction set § Fixed-length instructions § Reduced references to memory to retrieve operands	Mostly used in workstations

(Continued on next slide)

Types of Processor

(Continued from previous slide..)

Type of Architecture	Features	Usage
EPIC (Explicitly Parallel Instruction Computing)	§ Allows software to communicate explicitly to the processor when operations are parallel § Uses tighter coupling between the compiler and the processor § Enables compiler to extract maximum parallelism in the original code, and explicitly describe it to the processor	Mostly used in high-end servers and workstations

(Continued on next slide)

Types of Processor

(Continued from previous slide..)

Type of Architecture	Features	Usage
Multi-Core Processor	§ Processor chip has multiple cooler-running, more energy-efficient processing cores § Improve overall performance by handling more work in parallel § can share architectural components, such as memory elements and memory management	Mostly used in high-end servers and workstations

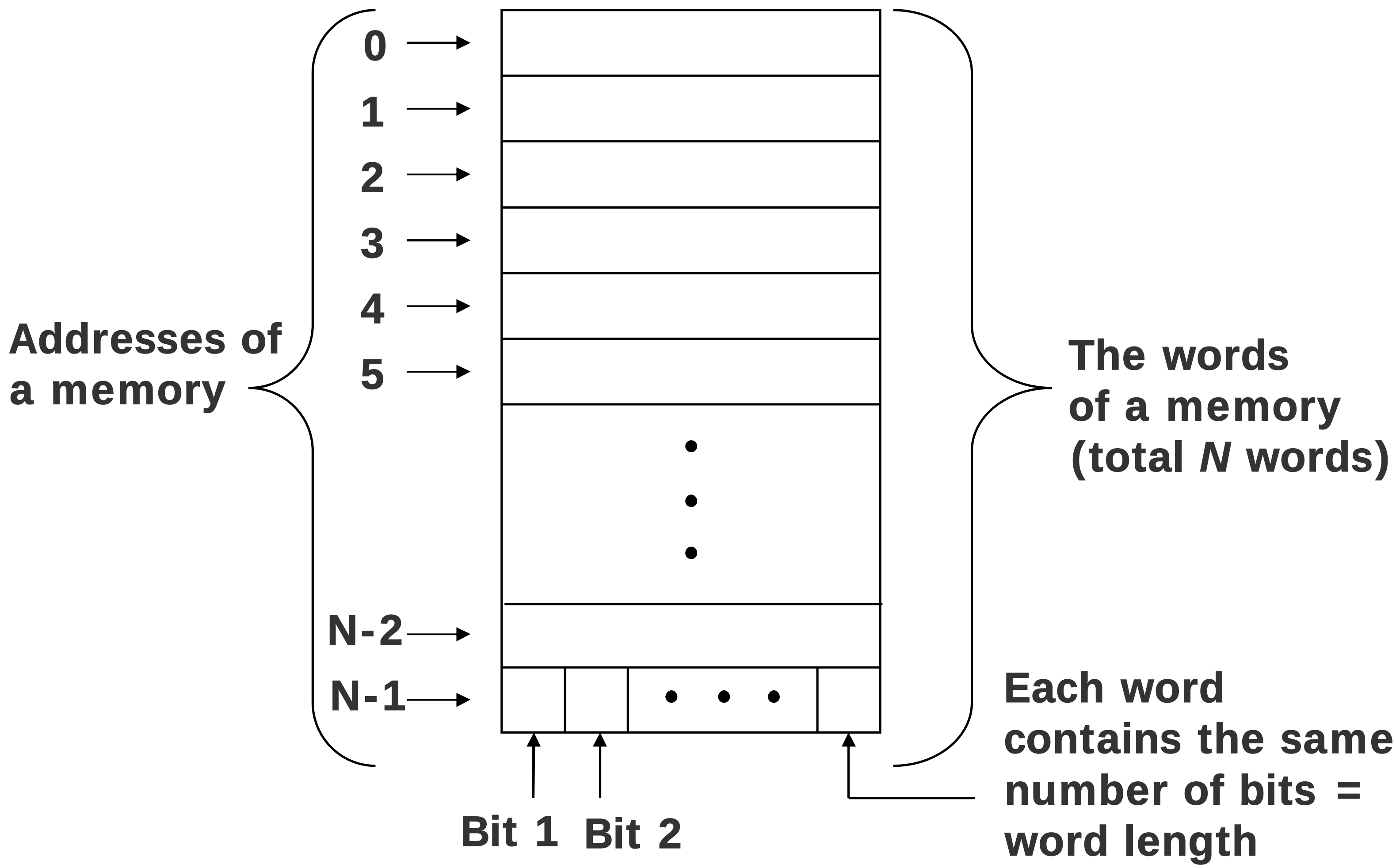
Main Memory

- Every computer has a temporary storage built into the computer hardware
- It stores instructions and data of a program mainly when the program is being executed by the CPU.
- This temporary storage is known as main memory, primary storage, or simply *memory*.
- Physically, it consists of some chips either on the motherboard or on a small circuit board attached to the motherboard of a computer
- It has random access property.
- It is volatile.

Storage Evaluation Criteria

Property	Desirable	Primary storage	Secondary storage
Storage capacity	Large storage capacity	Small	Large
Access Time	Fast access time	Fast	Slow
Cost per bit of storage	Lower cost per bit	High	Low
Volatility	Non-volatile	Volatile	Non-volatile
Access	Random access	Random access	Pseudo-random access or sequential access

Main Memory Organization



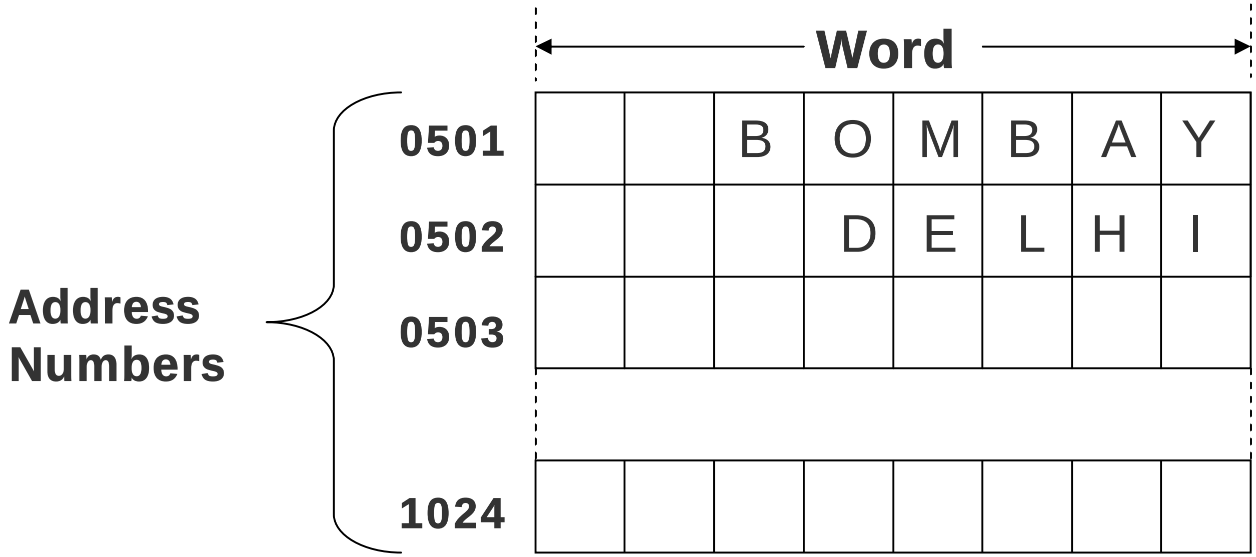
(Continued on next slide)

Main Memory Organization

(Continued from previous slide..)

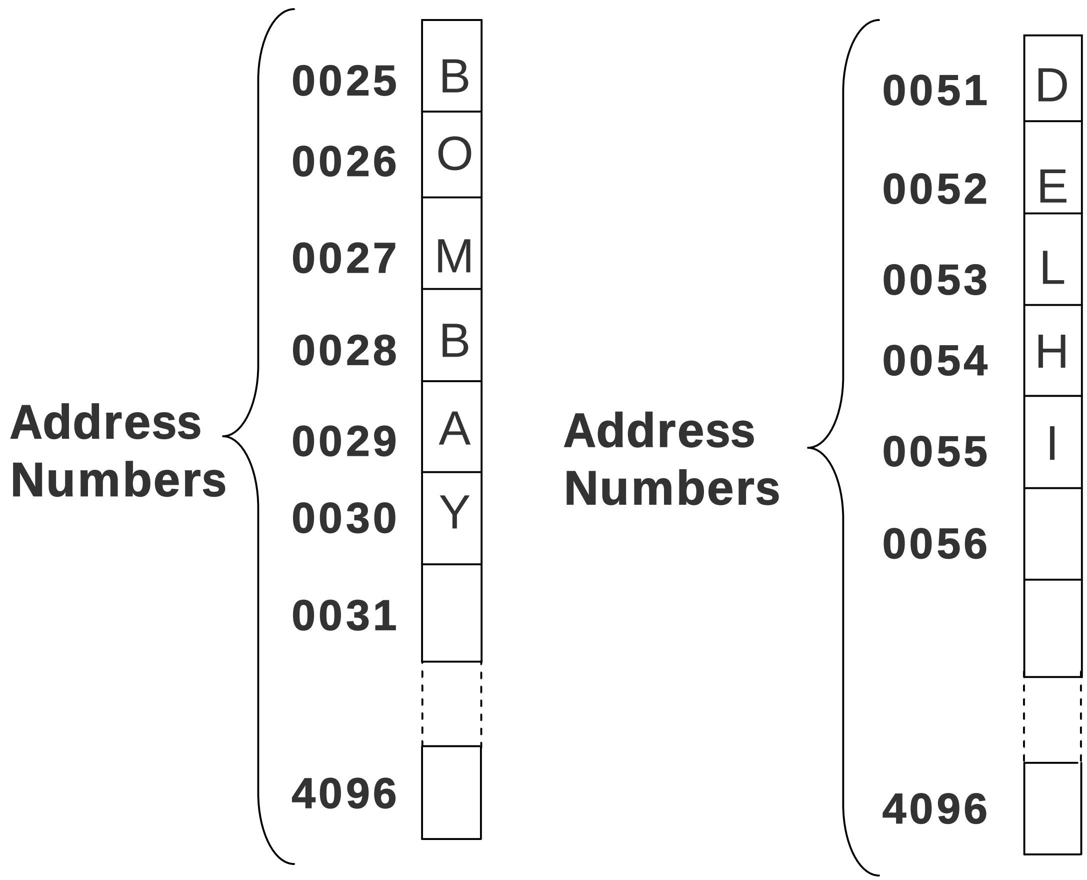
- **Machines having smaller word-length are slower in operation than machines having larger word-length**
- **A *write* to a memory location is destructive to its previous contents**
- **A *read* from a memory location is non-destructive to its previous contents**

Fixed Word-length Memory



- Storage space is always allocated in multiples of word-length
- Faster in speed of calculation than variable word-length memory
- Normally used in large scientific computers for gaining speed of calculation

Variable Word-length Memory



- Each memory location can store only a single character
- Slower in speed of calculation than fixed world-length memory
- Used in small business computers for optimizing the use of storage space

Note: With memory becoming cheaper and larger day-by-day, most modern computers employ fixed-word-length memory organization

Memory Capacity

- Memory capacity of a computer is equal to the number of bytes that can be stored in its primary storage
- Its units are:

Kilobytes (KB)	:	1024 (2^{10}) bytes
Megabytes (MB)	:	1,048,576 (2^{20}) bytes
Gigabytes (GB)	:	1,073,741824 (2^{30}) bytes

Random Access Memory (RAM)

- Primary storage of a computer is often referred to as RAM because of its random access capability
- RAM chips are volatile memory
- A computer's motherboard is designed in a manner that the memory capacity can be enhanced by adding more memory chips
- The additional RAM chips, which plug into special sockets on the motherboard, are known as *single-in-line memory modules (SIMMs)*

Read Only Memory (ROM)

- ROM a non-volatile memory chip
- Data stored in a ROM can only be read and used – they cannot be changed
- ROMs are mainly used to store programs and data, which do not change and are frequently used. For example, system boot program

Types of ROMs

Type	Usage
Manufacturer-programmed ROM	Data is burnt by the manufacturer of the electronic equipment in which it is used.
User-programmed ROM or Programmable ROM (PROM)	The user can load and store "read-only" programs and data in it
Erasable PROM (EPROM)	The user can erase information stored in it and the chip can be reprogrammed to store new information

(Continued on next slide)

Types of ROMs

(Continued from previous slide..)

Type	Usage
Ultra Violet EPROM (UVEPROM)	A type of EPROM chip in which the stored information is erased by exposing the chip for some time to ultra-violet light
Electrically EPROM (EEPROM) or Flash memory	A type of EPROM chip in which the stored information is erased by using high voltage electric pulses

Cache Memory

- It is commonly used for minimizing the memory-processor speed mismatch.
- It is an extremely fast, small memory between CPU and main memory whose access time is closer to the processing speed of the CPU.
- It is used to temporarily store very active data and instructions during processing.

Cache is pronounced as "cash"

Key Words / Phrases

- Accumulator Register (AR)
- Address
- Arithmetic Logic Unit (ALU)
- Branch Instruction
- Cache Memory
- Central Processing Unit (CPU)
- CISC (Complex Instruction Set Computer) architecture
- Clock cycles
- Clock speed
- Control Unit
- Electrically EPROM (EEPROM)
- Erasable Programmable Read-Only Memory (EPROM)
- Explicitly Parallel Instruction Computing (EPIC)
- Fixed-word-length memory
- Flash Memory
- Input/Output Register (I/O)
- Instruction Register (I)
- Instruction set
- Kilobytes (KB)
- Main Memory
- Manufacturer-Programmed ROM
- Megabytes (MB)
- Memory
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Microprogram
- Multi-core processor
- Non-Volatile storage Processor
- Program Control Register (PC)
- Programmable Read-Only Memory (PROM)
- Random Access Memory (RAM)

(Continued on next slide)

Key Words / Phrases

(Continued from previous slide..)

- Read-Only Memory (ROM)
- Register
- RISC (Reduced Instruction Set Computer) architecture
- Single In-line Memory Module (SIMM)
- Ultra Violet EPROM (UVEPROM)
- Upward compatible
- User-Programmed ROM
- Variable-word-length memory
- Volatile Storage
- Word length
- Word size

Chapter 07

Processor and Memory

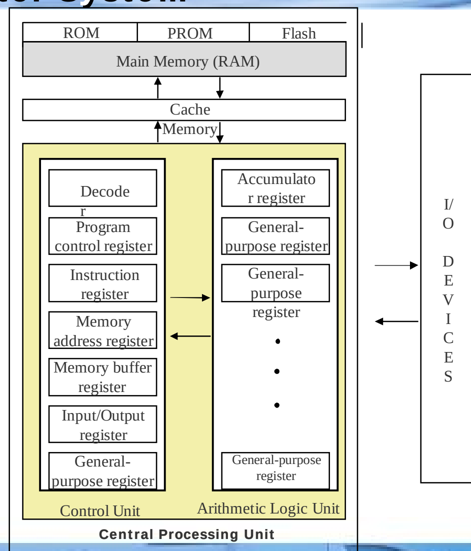
Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Internal structure of processor
- Memory structure
- Determining the speed of a processor
- Different types of processors available
- Determining the capacity of a memory
- Different types of memory available
- Several other terms related to the processor and main memory of a computer system

Basic Processor & Memory Architecture of a Computer System



Central Processing Unit (CPU)

- The *brain* of a computer system
- Performs all major calculations and comparisons
- Activates and controls the operations of other units of a computer system
- Two basic components are
  - Control Unit (CU)
  - Arithmetic Logic Unit (ALU)
- No other single component of a computer determines its overall performance as much as the CPU

Control Unit (CU)

- One of the two basic components of CPU
- Acts as the central nervous system of a computer system
- Selects and interprets program instructions, and coordinates execution
- Has some special purpose registers and a decoder to perform these activities

Arithmetic Logic Unit (ALU)

- One of the two basic components of CPU.
- Actual execution of instructions takes place in ALU
- Has some special purpose registers
- Has necessary circuitry to carry out all the arithmetic and logic operations included in the CPU instruction set

Instruction Set

- CPU has built-in ability to execute a particular set of machine instructions, called its *instruction set*
- Most CPUs have 200 or more instructions (such as add, subtract, compare, etc.) in their instruction set
- CPUs made by different manufacturers have different instruction sets
- Manufacturers tend to group their CPUs into “families” having similar instruction sets
- New CPU whose instruction set includes instruction set of its predecessor CPU is said to be *backward compatible* with its predecessor

Registers

- Special memory units, called registers, are used to hold information on a temporary basis as the instructions are interpreted and executed by the CPU
- Registers are part of the CPU (not main memory) of a computer
- The length of a register, sometimes called its *word size*, equals the number of bits it can store
- With all other parameters being the same, a CPU with 32-bit registers can process data twice larger than one with 16-bit registers

Functions of Commonly Used Registers

Sr. No.	Name of Register	Function
1	Memory Address (MAR)	Holds address of the active memory location
2	Memory Buffer (MBR)	Holds contents of the accessed (read/written) memory word
3	Program Control (PC)	Holds address of the next instruction to be executed
4	Accumulator (A)	Holds data to be operated upon, intermediate results, and the results
5	Instruction (I)	Holds an instruction while it is being executed
6	Input/Output (I/O)	Used to communicate with the I/O devices

Processor Speed

- Computer has a built-in *system clock* that emits millions of regularly spaced electric pulses per second (known as *clock cycles)*
- It takes one cycle to perform a basic operation, such as moving a byte of data from one memory location to another
- Normally, several clock cycles are required to fetch, decode, and execute a single program instruction
- Hence, shorter the clock cycle, faster the processor
- Clock speed (number of clock cycles per second) is measured in Megahertz (10^6 cycles/sec) or Gigahertz (10^9 cycles/sec)

Types of Processor

Type of Architecture	Features	Usage
CISC (Complex Instruction Set Computer)	§ Large instruction set § Variable-length instructions § Variety of addressing modes § Complex & expensive to produce	Mostly used in personal computers
RISC (Reduced Instruction Set Computer)	§ Small instruction set § Fixed-length instructions § Reduced references to memory to retrieve operands	Mostly used in workstations

(Continued on next slide)

Types of Processor

(Continued from previous slide..)

Type of Architecture	Features	Usage
EPIC (Explicitly Parallel Instruction Computing)	§ Allows software to communicate explicitly to the processor when operations are parallel § Uses tighter coupling between the compiler and the processor § Enables compiler to extract maximum parallelism in the original code, and explicitly describe it to the processor	Mostly used in high-end servers and workstations

(Continued on next slide)

Types of Processor

(Continued from previous slide..)

Type of Architecture	Features	Usage
Multi-Core Processor	§ Processor chip has multiple cooler-running, more energy-efficient processing cores § Improve overall performance by handling more work in parallel § can share architectural components, such as memory elements and memory management	Mostly used in high-end servers and workstations

Main Memory

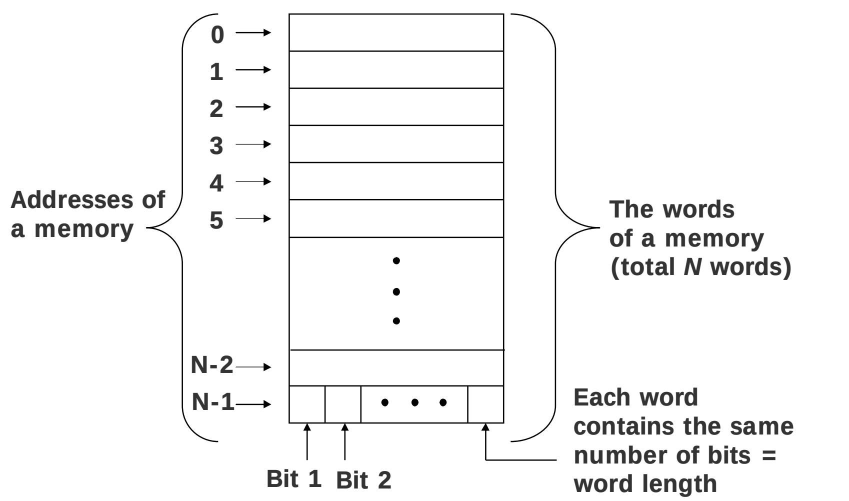
- Every computer has a temporary storage built into the computer hardware
- It stores instructions and data of a program mainly when the program is being executed by the CPU.
- This temporary storage is known as main memory, primary storage, or simply *memory*.
- Physically, it consists of some chips either on the motherboard or on a small circuit board attached to the motherboard of a computer
- It has random access property.
- It is volatile.

Storage Evaluation Criteria

Property	Desirable	Primary storage	Secondary storage
Storage capacity	Large storage capacity	Small	Large
Access Time	Fast access time	Fast	Slow
Cost per bit of storage	Lower cost per bit	High	Low
Volatility	Non-volatile	Volatile	Non-volatile
Access	Random access	Random access	Pseudo-random access or sequential access

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Main Memory Organization



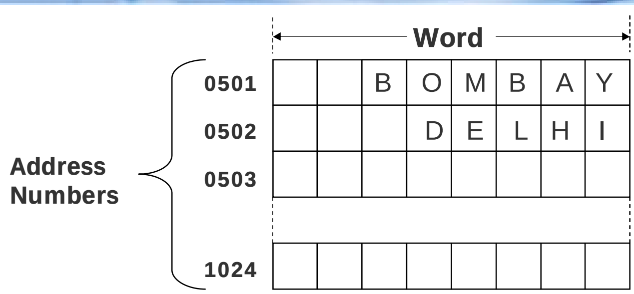
(Continued on next slide)

Main Memory Organization

(Continued from previous slide..)

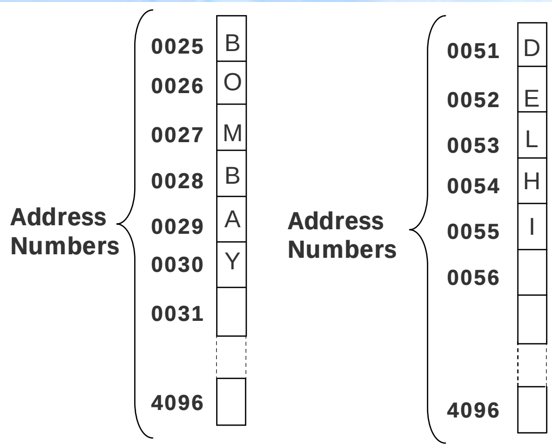
- Machines having smaller word-length are slower in operation than machines having larger word-length
- A *write* to a memory location is destructive to its previous contents
- A *read* from a memory location is non-destructive to its previous contents

Fixed Word-length Memory



- Storage space is always allocated in multiples of word-length
- Faster in speed of calculation than variable word-length memory
- Normally used in large scientific computers for gaining speed of calculation

Variable Word-length Memory



- § Each memory location can store only a single character
- § Slower in speed of calculation than fixed world-length memory
- § Used in small business computers for optimizing the use of storage space

Note: With memory becoming cheaper and larger day-by-day, most modern computers employ fixed-word-length memory organization

Memory Capacity

- § Memory capacity of a computer is equal to the number of bytes that can be stored in its primary storage
- § Its units are:

Kilobytes (KB)	:	1024 (2^{10}) bytes
Megabytes (MB)	:	1,048,576 (2^{20}) bytes
Gigabytes (GB)	:	1,073,741824 (2^{30}) bytes

Random Access Memory (RAM)

- Primary storage of a computer is often referred to as RAM because of its random access capability
- RAM chips are volatile memory
- A computer's motherboard is designed in a manner that the memory capacity can be enhanced by adding more memory chips
- The additional RAM chips, which plug into special sockets on the motherboard, are known as *single-in-line memory modules (SIMMs)*

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Read Only Memory (ROM)

- ROM a non-volatile memory chip
- Data stored in a ROM can only be read and used – they cannot be changed
- ROMs are mainly used to store programs and data, which do not change and are frequently used. For example, system boot program

Types of ROMs

Type	Usage
Manufacturer-programmed ROM	Data is burnt by the manufacturer of the electronic equipment in which it is used.
User-programmed ROM or Programmable ROM (PROM)	The user can load and store “read-only” programs and data in it
Erasable PROM (EPROM)	The user can erase information stored in it and the chip can be reprogrammed to store new information

(Continued on next slide)

Computer Fundamentals: Pradeep K. Sinha & Priti Sinha

Types of ROMs

(Continued from previous slide..)

Type	Usage
Ultra Violet EPROM (UVEPROM)	A type of EPROM chip in which the stored information is erased by exposing the chip for some time to ultra-violet light
Electrically EPROM (EEPROM) or Flash memory	A type of EPROM chip in which the stored information is erased by using high voltage electric pulses

Cache Memory

- It is commonly used for minimizing the memory-processor speed mismatch.
- It is an extremely fast, small memory between CPU and main memory whose access time is closer to the processing speed of the CPU.
- It is used to temporarily store very active data and instructions during processing.

Cache is pronounced as "cash"

Key Words / Phrases

- Accumulator Register (AR)
- Address
- Arithmetic Logic Unit (ALU)
- Branch Instruction
- Cache Memory
- Central Processing Unit (CPU)
- CISC (Complex Instruction Set Computer) architecture
- Clock cycles
- Clock speed
- Control Unit
- Electrically EPROM (EEPROM)
- Erasable Programmable Read-Only Memory (EPROM)
- Explicitly Parallel Instruction Computing (EPIC)
- Fixed-word-length memory
- Flash Memory
- Input/Output Register (I/O)
- Instruction Register (I)
- Instruction set
- Kilobytes (KB)
- Main Memory
- Manufacturer-Programmed ROM
- Megabytes (MB)
- Memory
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Microprogram
- Multi-core processor
- Non-Volatile storage Processor
- Program Control Register (PC)
- Programmable Read-Only Memory (PROM)
- Random Access Memory (RAM)

(Continued on next slide)

Key Words / Phrases

(Continued from previous slide..)

- Read-Only Memory (ROM)
- Register
- RISC (Reduced Instruction Set Computer) architecture
- Single In-line Memory Module (SIMM)
- Ultra Violet EPROM (UVEPROM)
- Upward compatible
- User-Programmed ROM
- Variable-word-length memory
- Volatile Storage
- Word length
- Word size

Chapter 07

Processor and Memory

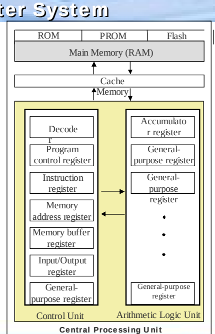
Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- **Internal structure of processor**
- **Memory structure**
- **Determining the speed of a processor**
- **Different types of processors available**
- **Determining the capacity of a memory**
- **Different types of memory available**
- **Several other terms related to the processor and main memory of a computer system**

Basic Processor & Memory Architecture of a Computer System



Central Processing Unit (CPU)

- The *brain* of a computer system
- Performs all major calculations and comparisons
- Activates and controls the operations of other units of a computer system
- Two basic components are
  - Control Unit (CU)
  - Arithmetic Logic Unit (ALU)
- No other single component of a computer determines its overall performance as much as the CPU

Control Unit (CU)

- One of the two basic components of CPU
- Acts as the central nervous system of a computer system
- Selects and interprets program instructions, and coordinates execution
- Has some special purpose registers and a decoder to perform these activities

Arithmetic Logic Unit (ALU)

- One of the two basic components of CPU.
- Actual execution of instructions takes place in ALU
- Has some special purpose registers
- Has necessary circuitry to carry out all the arithmetic and logic operations included in the CPU instruction set

Instruction Set

- CPU has built-in ability to execute a particular set of machine instructions, called its *instruction set*
- Most CPUs have 200 or more instructions (such as add, subtract, compare, etc.) in their instruction set
- CPUs made by different manufacturers have different instruction sets
- Manufacturers tend to group their CPUs into “families” having similar instruction sets
- New CPU whose instruction set includes instruction set of its predecessor CPU is said to be *backward compatible* with its predecessor

Registers

- Special memory units, called registers, are used to hold information on a temporary basis as the instructions are interpreted and executed by the CPU
- Registers are part of the CPU (not main memory) of a computer
- The length of a register, sometimes called its *word size*, equals the number of bits it can store
- With all other parameters being the same, a CPU with 32-bit registers can process data twice larger than one with 16-bit registers

Functions of Commonly Used Registers

Sr. No.	Name of Register	Function
1	Memory Address (MAR)	Holds address of the active memory location
2	Memory Buffer (MBR)	Holds contents of the accessed (read/written) memory word
3	Program Control (PC)	Holds address of the next instruction to be executed
4	Accumulator (A)	Holds data to be operated upon, intermediate results, and the results
5	Instruction (I)	Holds an instruction while it is being executed
6	Input/Output (I/O)	Used to communicate with the I/O devices

Processor Speed

- Computer has a built-in *system clock* that emits millions of regularly spaced electric pulses per second (known as *clock cycles)*
- It takes one cycle to perform a basic operation, such as moving a byte of data from one memory location to another
- Normally, several clock cycles are required to fetch, decode, and execute a single program instruction
- Hence, shorter the clock cycle, faster the processor
- Clock speed (number of clock cycles per second) is measured in Megahertz (10^6 cycles/sec) or Gigahertz (10^9 cycles/sec)

Types of Processor

Type of Architecture	Features	Usage
CISC (Complex Instruction Set Computer)	• Large instruction set • Variable-length instructions • Variety of addressing modes • Complex & expensive to produce	Mostly used in personal computers
RISC (Reduced Instruction Set Computer)	• Small instruction set • Fixed-length instructions • Reduced references to memory to retrieve operands	Mostly used in workstations

(Continued on next slide)

Types of Processor

(Continued from previous slide..)

Type of Architecture	Features	Usage
EPIC (Explicitly Parallel Instruction Computing)	• Allows software to communicate explicitly to the processor when operations are parallel • Uses tighter coupling between the compiler and the processor • Enables compiler to extract maximum parallelism in the original code, and explicitly describe it to the processor	Mostly used in high-end servers and workstations

(Continued on next slide)

Types of Processor

(Continued from previous slide..)

Type of Architecture	Features	Usage
Multi-Core Processor	§ Processor chip has multiple cooler-running, more energy-efficient processing cores § Improve overall performance by handling more work in parallel § can share architectural components, such as memory elements and memory management	Mostly used in high-end servers and workstations

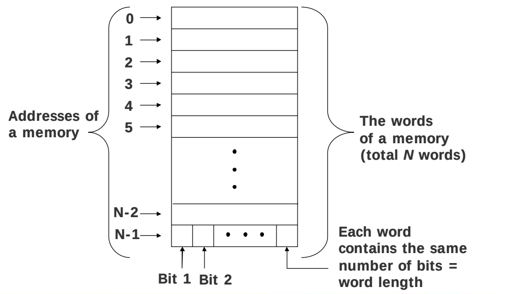
Main Memory

- Every computer has a temporary storage built into the computer hardware
- It stores instructions and data of a program mainly when the program is being executed by the CPU.
- This temporary storage is known as main memory, primary storage, or simply *memory*.
- Physically, it consists of some chips either on the motherboard or on a small circuit board attached to the motherboard of a computer
- It has random access property.
- It is volatile.

Storage Evaluation Criteria

Property	Desirable	Primary storage	Secondary storage
Storage capacity	Large storage capacity	Small	Large
Access Time	Fast access time	Fast	Slow
Cost per bit of storage	Lower cost per bit	High	Low
Volatility	Non-volatile	Volatile	Non-volatile
Access	Random access	Random access	Pseudo-random access or sequential access

Main Memory Organization



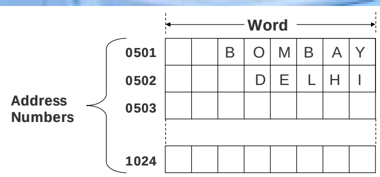
(Continued on next slide)

Main Memory Organization

(Continued from previous slide..)

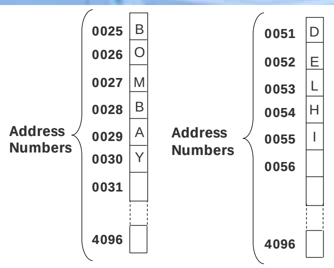
- Machines having smaller word-length are slower in operation than machines having larger word-length
- A *write* to a memory location is destructive to its previous contents
- A *read* from a memory location is non-destructive to its previous contents

Fixed Word-length Memory



- Storage space is always allocated in multiples of word-length
- Faster in speed of calculation than variable word-length memory
- Normally used in large scientific computers for gaining speed of calculation

Variable Word-length Memory



- Each memory location can store only a single character
- Slower in speed of calculation than fixed world-length memory
- Used in small business computers for optimizing the use of storage space

Note: With memory becoming cheaper and larger day-by-day, most modern computers employ fixed-word-length memory organization

Memory Capacity

- Memory capacity of a computer is equal to the number of bytes that can be stored in its primary storage
- Its units are:

Kilobytes (KB)	:	1024 (2^{10}) bytes
Megabytes (MB)	:	1,048,576 (2^{20}) bytes
Gigabytes (GB)	:	1,073,741824 (2^{30}) bytes

Random Access Memory (RAM)

- Primary storage of a computer is often referred to as RAM because of its random access capability
- RAM chips are volatile memory
- A computer's motherboard is designed in a manner that the memory capacity can be enhanced by adding more memory chips
- The additional RAM chips, which plug into special sockets on the motherboard, are known as *single-in-line memory modules (SIMMs)*

Read Only Memory (ROM)

- ROM a non-volatile memory chip
- Data stored in a ROM can only be read and used – they cannot be changed
- ROMs are mainly used to store programs and data, which do not change and are frequently used. For example, system boot program

Types of ROMs

Type	Usage
Manufacturer-programmed ROM	Data is burnt by the manufacturer of the electronic equipment in which it is used.
User-programmed ROM or Programmable ROM (PROM)	The user can load and store "read-only" programs and data in it
Erasable PROM (EPROM)	The user can erase information stored in it and the chip can be reprogrammed to store new information

(Continued on next slide)

Types of ROMs

(Continued from previous slide..)

Type	Usage
Ultra Violet EPROM (UVEPROM)	A type of EPROM chip in which the stored information is erased by exposing the chip for some time to ultra-violet light
Electrically EPROM (EEPROM) or Flash memory	A type of EPROM chip in which the stored information is erased by using high voltage electric pulses

Cache Memory

- It is commonly used for minimizing the memory-processor speed mismatch.
- It is an extremely fast, small memory between CPU and main memory whose access time is closer to the processing speed of the CPU.
- It is used to temporarily store very active data and instructions during processing.

Cache is pronounced as "cash"

Key Words/Phrases

- Accumulator Register (AR)
- Address
- Arithmetic Logic Unit (ALU)
- Branch Instruction
- Cache Memory
- Central Processing Unit (CPU)
- CISC (Complex Instruction Set Computer) architecture
- Clock cycles
- Clock speed
- Control Unit
- Electrically EPROM (EEPROM)
- Erasable Programmable Read-Only Memory (EPROM)
- Explicitly Parallel Instruction Computing (EPIC)
- Fixed-word-length memory
- Flash Memory
- Input/Output Register (I/O)
- Instruction Register (I)
- Instruction set
- Kilobytes (KB)
- Main Memory
- Manufacturer-Programmed ROM
- Megabytes (MB)
- Memory
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)
- Microprogram
- Multi-core processor
- Non-Volatile storage Processor
- Program Control Register (PC)
- Programmable Read-Only Memory (PROM)
- Random Access Memory (RAM)

(Continued on next slide)

Key Words/Phrases

(Continued from previous slide..)

- Read-Only Memory (ROM)
- Register
- RISC (Reduced Instruction Set Computer) architecture
- Single In-line Memory Module (SIMM)
- Ultra Violet EPROM (UVEPROM)
- Upward compatible
- User-Programmed ROM
- Variable-word-length memory
- Volatile Storage
- Word length
- Word size

Chapter 08

Secondary Storage Devices

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Secondary storage devices and their need
- Classification of commonly used secondary storage devices
- Difference between sequential and direct access storage devices
- Basic principles of operation, types, and uses of popular secondary storage devices such as magnetic tape, magnetic disk, and optical disk

(Continued on next slide)

Learning Objectives

(Continued from previous slide..)

- Commonly used mass storage devices
- Introduction to other related concepts such as RAID, Jukebox, storage hierarchy, etc.

Limitations of Primary Storage

- Limited capacity because the cost per bit of storage is high
- Volatile - data stored in it is lost when the electric power is turned off or interrupted

Secondary Storage

- Used in a computer system to overcome the limitations of primary storage
- Has virtually unlimited capacity because the cost per bit of storage is very low
- Has an operating speed far slower than that of the primary storage
- Used to store large volumes of data on a permanent basis
- Also known as *auxiliary memory*

Classification of Commonly Used Secondary Storage Devices

- Secondary Storage Devices
  - Sequential Access Device
    - Magnetic Tape
  - Direct Access Devices
    - Magnetic Disks
      - Floppy Disk
      - Hard Disks
        - Zip Disk
        - Disk Pack
        - Winchester Disk
    - Optical Disks
      - CD-ROM
      - WORM (CD-R)
      - CD-RW
      - DVD
    - Memory Storage Devices
      - Flash Drive
      - Memory Card

Sequential-access Storage Devices

- Arrival at the desired storage location may be preceded by sequencing through other locations
- Data can only be retrieved in the same sequence in which it is stored
- Access time varies according to the storage location of the information being accessed
- Suitable for sequential processing applications where most, if not all, of the data records need to be processed one after another
- Magnetic tape is a typical example of such a storage device

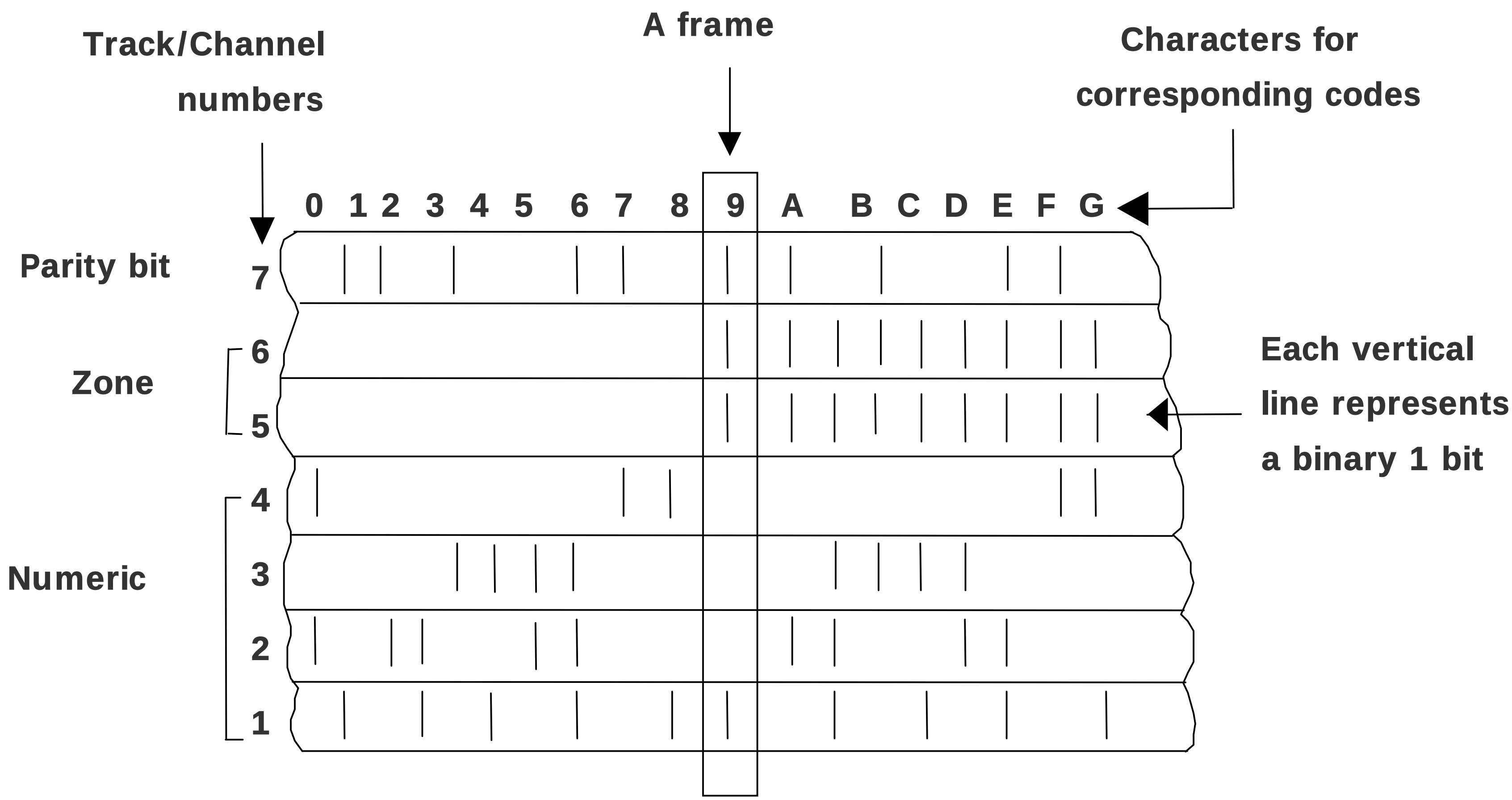
Direct-access Storage Devices

- Devices where any storage location may be selected and accessed at random
- Permits access to individual information in a more direct or immediate manner
- Approximately equal access time is required for accessing information from any storage location
- Suitable for direct processing applications such as on-line ticket booking systems, on-line banking systems
- Magnetic, optical, and magneto-optical disks are typical examples of such a storage device

Magnetic Tape Basics

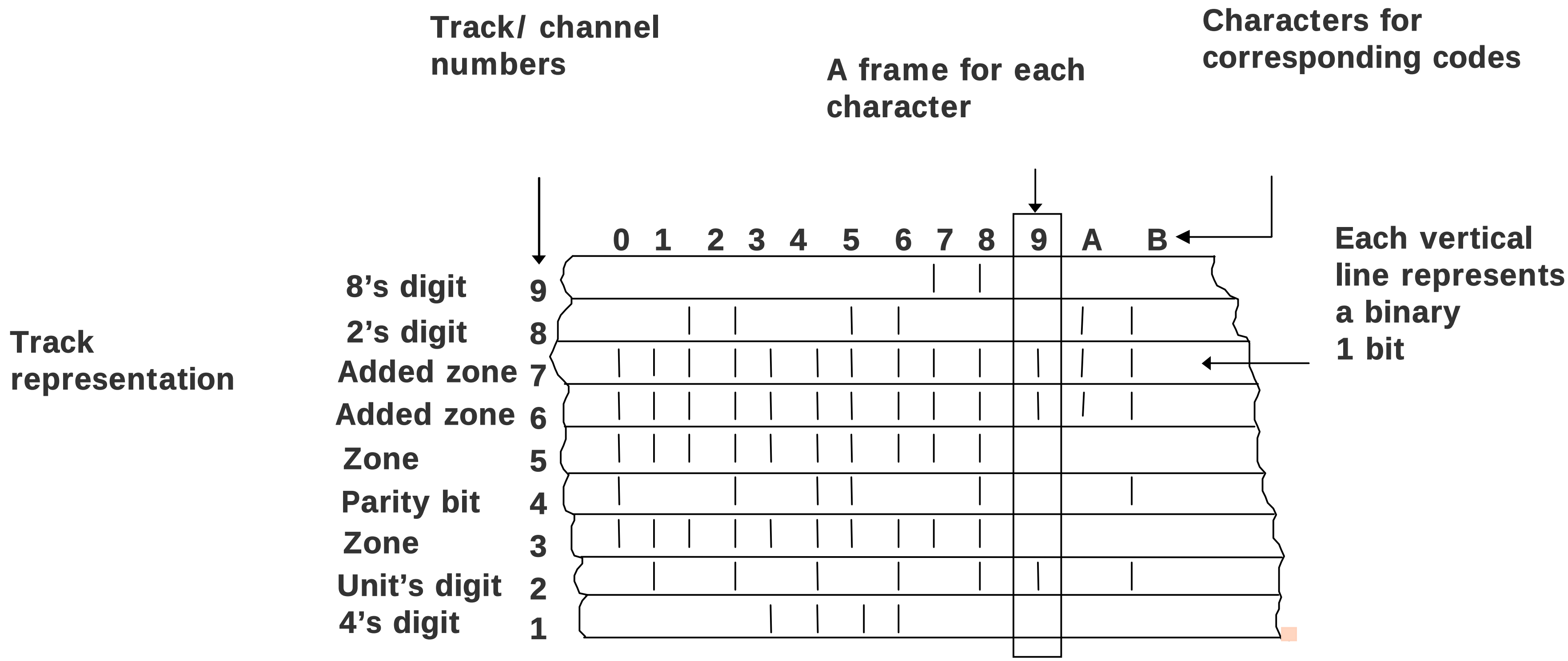
- Commonly used sequential-access secondary storage device
- Physically, the tape medium is a plastic ribbon, which is usually ½ inch or ¼ inch wide and 50 to 2400 feet long
- Plastic ribbon is coated with a magnetizable recording material such as iron-oxide or chromium dioxide
- Data are recorded on the tape in the form of tiny invisible magnetized and non-magnetized spots (representing 1s and 0s) on its coated surface
- Tape ribbon is stored in reels or a small cartridge or cassette

Magnetic Tape - Storage Organization (Example 1)



Illustrates the concepts of frames, tracks, parity bit, and character-by-character data storage

Magnetic Tape - Storage Organization (Example 2)



Illustrates the concepts of frames, tracks, parity bit, and character-by-character data storage

Magnetic Tape - Storage Organization (Example 3)

Tape motion

IBG	R1	IBG	R2	IBG	R3	IBG	R4	IBG	R5	IBG	R6

(a) An unblocked tape. There is an IBG after each record.

Tape motion

IBG	R1	R2	IBG	R3	R4	IBG	R5	R6	IBG	R7	R8	IBG

(b) A tape which uses a blocking factor of two. There is an IBG after every two records.

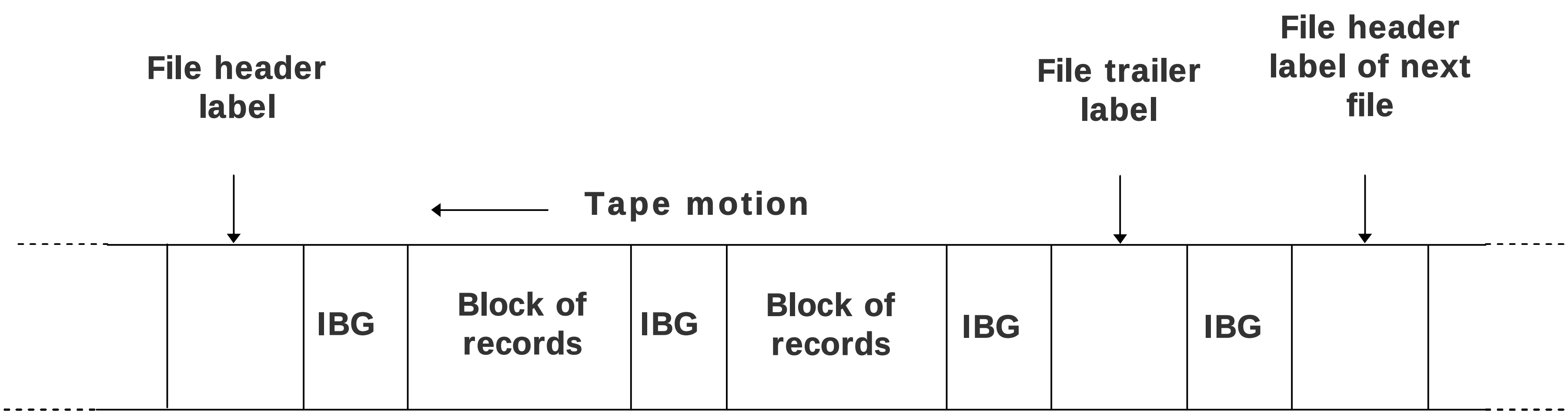
Tape motion

IBG	R1	R2	R3	IBG	R4	R5	R6	IBG	R7	R8	R9	IBG

(c) A tape which uses a blocking factor of three. There is an IBG after every three records.

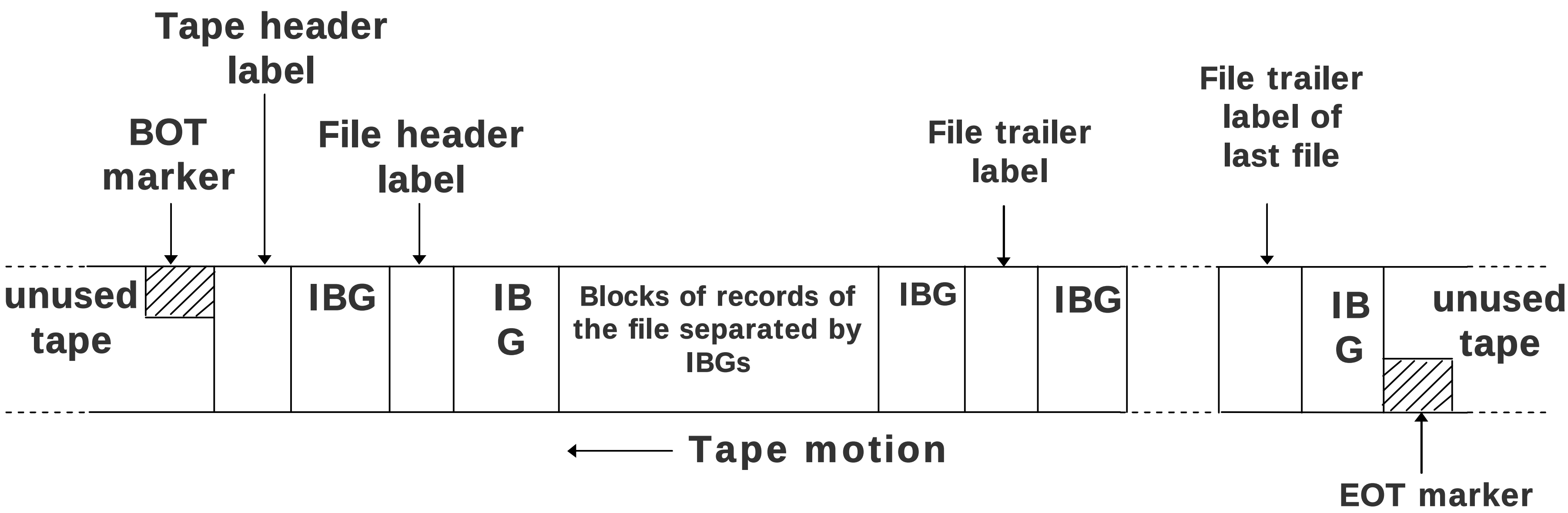
Illustrates the concepts of blocking of records, inter-block gap (IBG), and blocking factor

Magnetic Tape - Storage Organization (Example 4)



Illustrates the concepts of multiple blocks of records forming a file that is separated from other files by a file header label in the beginning and a file trailer label at the end of the file

Magnetic Tape-Storage Organization (Example 5)



Illustrates the concepts of Beginning of Tape (BoT) and End of Tape (EoT) markers, and tape header label

Magnetic Tape Storage Capacity

- Storage capacity of a tape =
  Data recording density x Length
- Data recording density is the amount of data that can be stored on a given length of tape. It is measured in bytes per inch (bpi)
- Tape density varies from 800 bpi in older systems to 77,000 bpi in some of the modern systems
- Actual storage capacity of a tape may be anywhere from 35% to 70% of its total storage capacity, depending on the storage organization used

Magnetic Tape – Data Transfer Rate

- Refers to characters/second that can be transmitted to the memory from the tape
- Transfer rate measurement unit is bytes/second (bps)
- Value depends on the data recording density and the speed with which the tape travels under the read/write head
- A typical value of data transfer rate is 7.7 MB/second

Magnetic Tape – Tape Drive

- Used for writing/reading of data to/from a magnetic tape ribbon
- Different for tape reels, cartridges, and cassettes
- Has read/write heads for reading/writing of data on tape
- A magnetic tape reel/cartridge/cassette has to be first loaded on a tape drive for reading/writing of data on it
- When processing is complete, the tape is removed from the tape drive for off-line storage

Magnetic Tape – Tape Controller

- Tape drive is connected to and controlled by a tape controller that interprets the commands for operating the tape drive

- A typical set of commands supported by a tape controller are:

Read	reads one block of data
Write	writes one block of data
Write tape header label	used to update the contents of tape header label
Erase tape	erases the data recorded on a tape
Back space one block	rewinds the tape to the beginning of previous block

(Continued on next slide)

Magnetic Tape – Tape Controller

(Continued from previous slide..)

Forward space one block	forwards the tape to the beginning of next block
Forward space one file	forwards the tape to the beginning of next file
Rewind	fully rewinds the tape
Unload	releases the tape drive's grip so that the tape spool can be unmountedfrom the tape drive

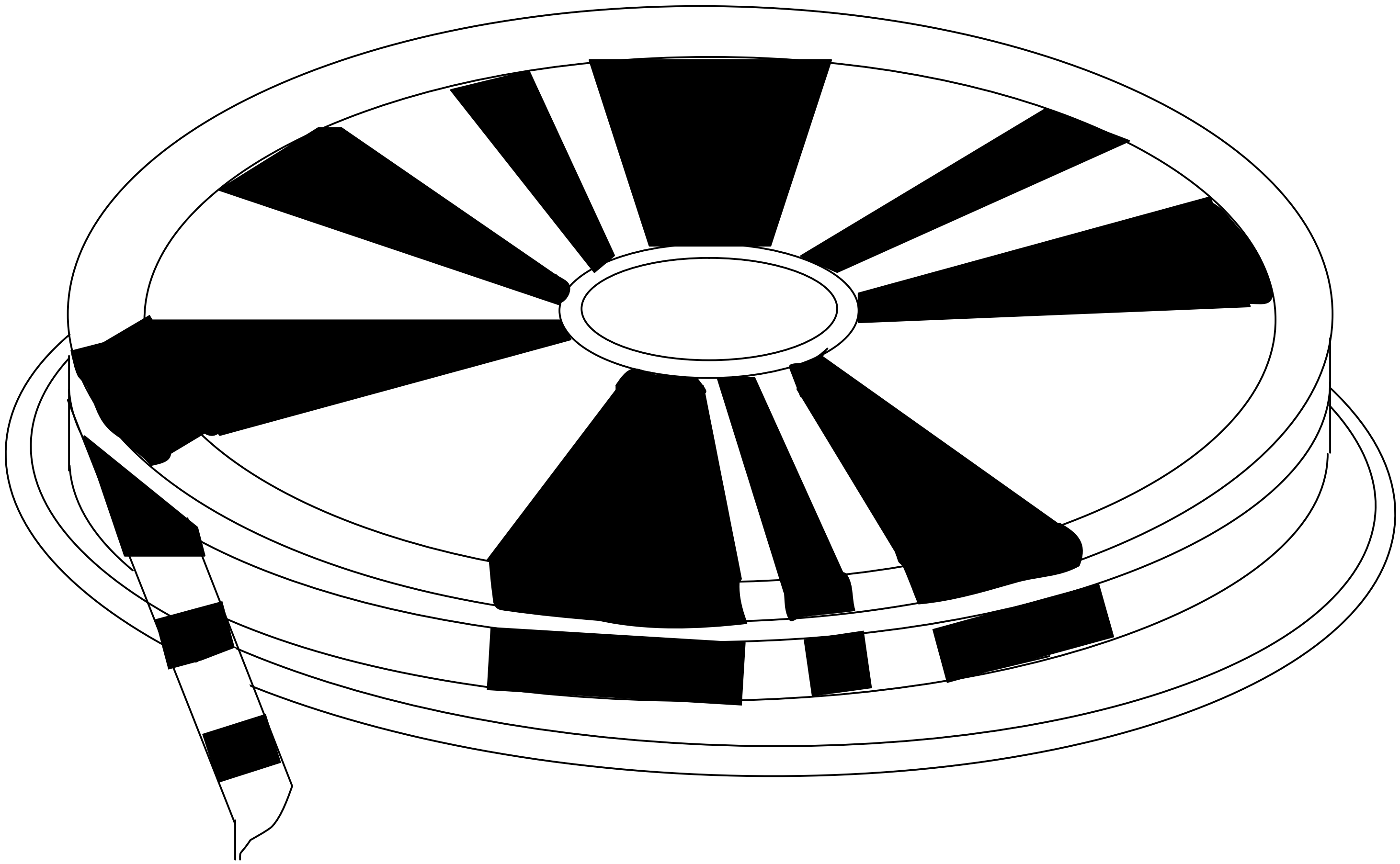
Types of Magnetic Tape

- ½-inch tape reel
- ½-inch tape cartridge
- ¼-inch streamer tape
- 4-mm digital audio tape (DAT)

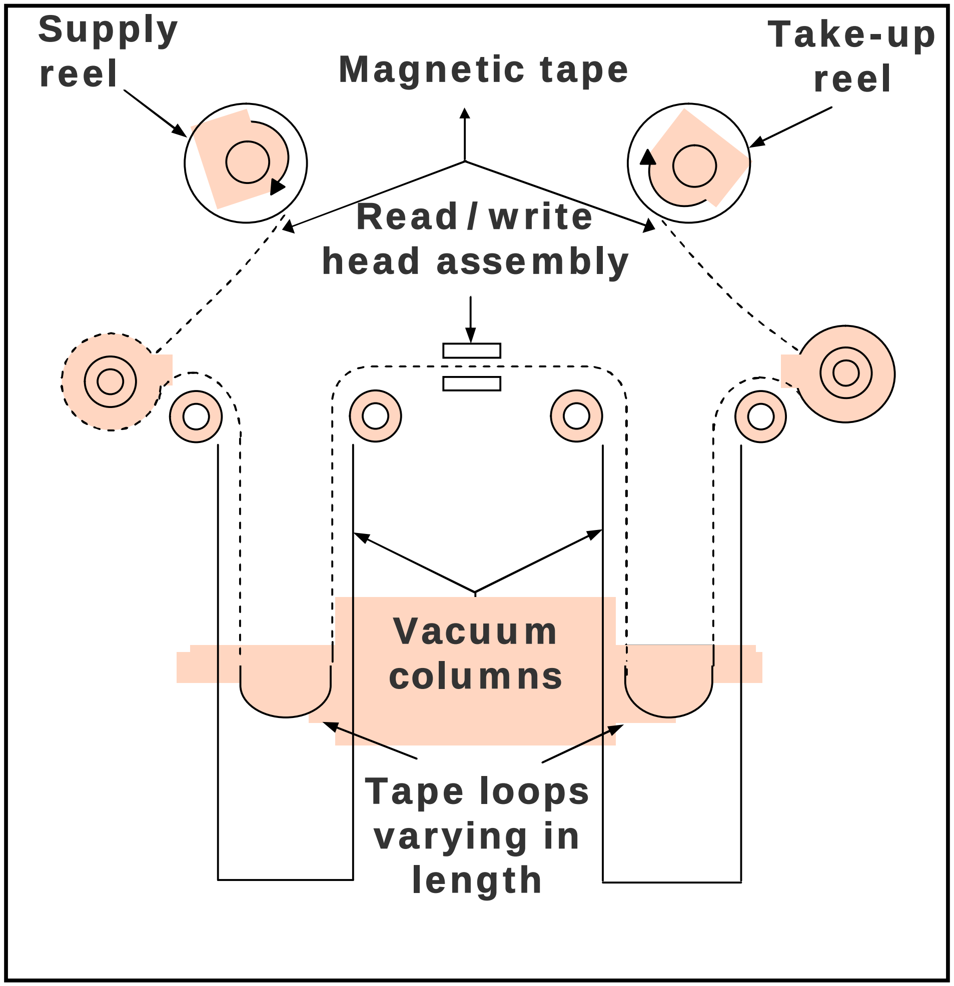
Half-inch Tape Reel

- Uses ½ inch wide tape ribbon stored on a tape reel
- Uses parallel representation method of storing data, in which data are read/written a byte at a time
- Uses a read/write head assembly that has one read/write head for each track
- Commonly used as archival storage for off-line storage of data and for exchange of data and programs between organizations
- Fast getting replaced by tape cartridge, streamer tape, and digital audio tape they are more compact, cheaper and easier to handle

Half-inch Tape Reel



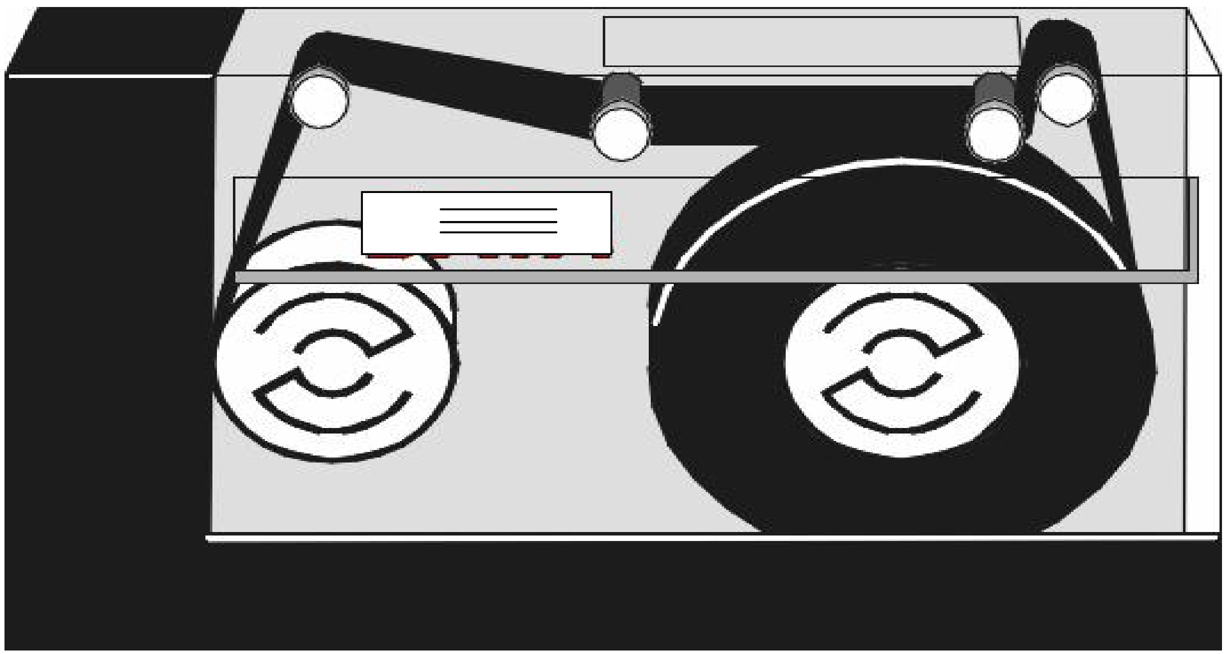
Tape Drive of Half-inch Tape Reel



Half-inch Tape Cartridge

- Uses ½ inch wide tape ribbon sealed in a cartridge
- Has 36 tracks, as opposed to 9 tracks for most half-inch tape reels
- Stores data using parallel representation. Hence, 4 bytes of data are stored across the width of the tape. This enables more bytes of data to be stored on the same length of tape
- Tape drive reads/writes on the top half of the tape in one direction and on the bottom half in the other direction

Half-inch Tape Cartridge



Quarter-inch Streamer Tape

- Uses ¼ inch wide tape ribbon sealed in a cartridge
- Uses serial representation of data recording (data bits are aligned in a row one after another in tracks)
- Can have from 4 to 30 tracks, depending on the tape drive
- Depending on the tape drive, the read/write head reads/writes data on one/two/four tracks at a time
- Eliminates the need for the start/stop operation of traditional tape drives

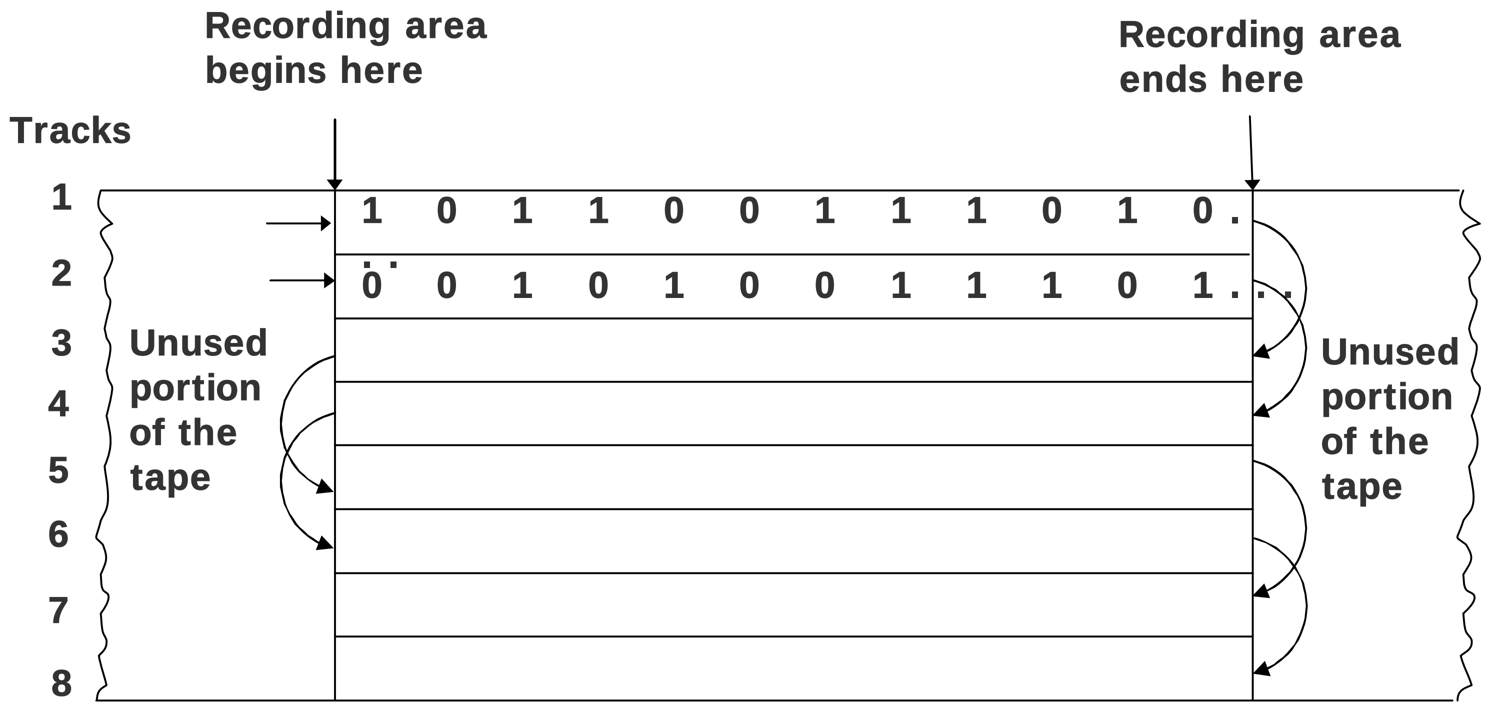
(Continued on next slide)

Quarter-inch Streamer Tape

(Continued from previous slide..)

- Can read/write data more efficiently than the traditional tape drives because there is no start/stop mechanism
- Make more efficient utilization of tape storage area than traditional tape drives because IBGs are not needed
- The standard data formats used in these tapes is known as the QIC standard

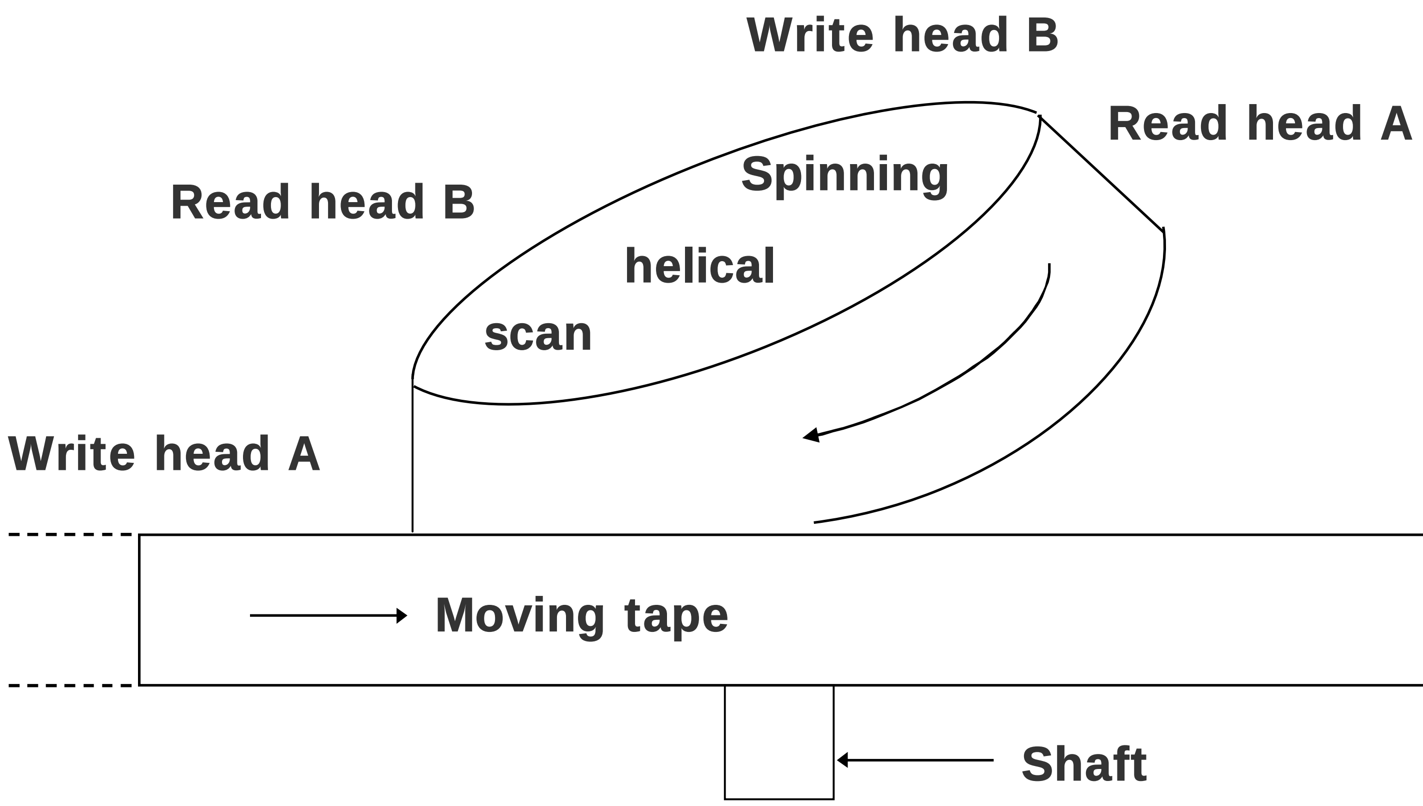
Quarter-inch Streamer Tape (Example)



4mm Digital Audio Tape (DAT)

- Uses 4mm wide tape ribbon sealed in a cartridge
- Has very high data recording density
- Uses a tape drive that uses helical scan technique for data recording, in which two read heads and two write heads are built into a small wheel
- DAT drives use a data recording format called Digital Data Storage (DDS), which provides three levels of error-correcting code
- Typical capacity of DAT cartridges varies from 4 GB to 14 GB

The Helical Scan Techniques Used in DAT Drives



Advantages of Magnetic Tapes

- Storage capacity is virtually unlimited because as many tapes as required can be used for storing very large data sets
- Cost per bit of storage is very low for magnetic tapes.
- Tapes can be erased and reused many times
- Tape reels and cartridges are compact and light in weight
- Easy to handle and store.
- Very large amount of data can be stored in a small storage space

(Continued on next slide)

Advantages of Magnetic Tapes

(Continued from previous slide..)

- Compact size and light weight
- Magnetic tape reels and cartridges are also easily portable from one place to another
- Often used for transferring data and programs from one computer to another that are not linked together

Limitations of Magnetic Tapes

- Due to their sequential access nature, they are not suitable for storage of those data that frequently require to be accessed randomly
- Must be stored in a dust-free environment because specks of dust can cause tape-reading errors
- Must be stored in an environment with properly controlled temperature and humidity levels
- Tape ribbon may get twisted due to warping, resulting in loss of stored data
- Should be properly labeled so that some useful data stored on a particular tape is not erased by mistake

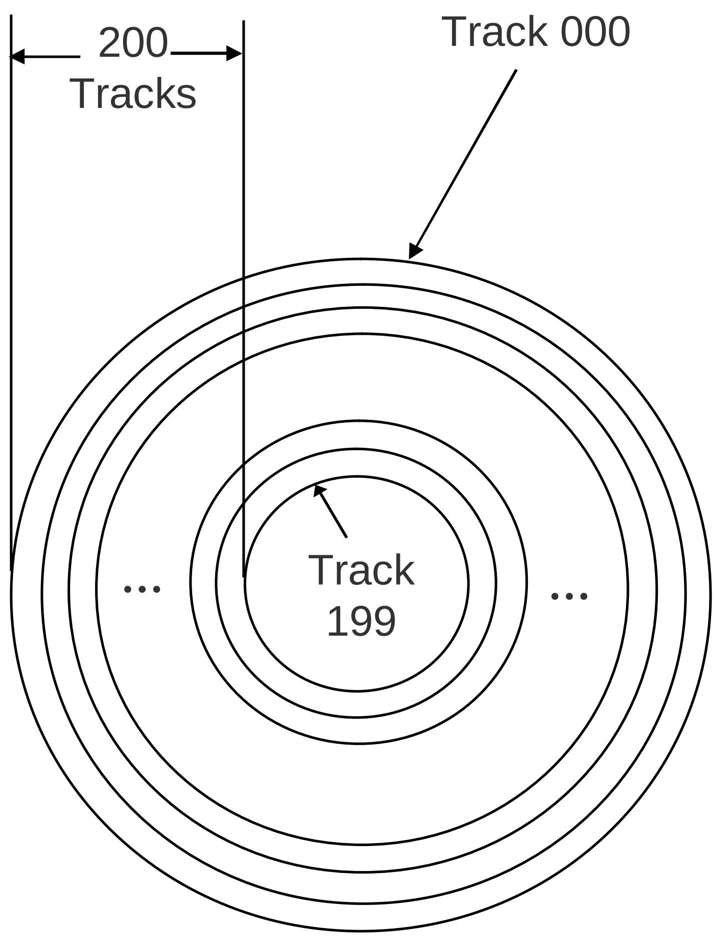
Uses of Magnetic Tapes

- For applications that are based on sequential data processing
- Backing up of data for off-line storage
- Archiving of infrequently used data
- Transferring of data from one computer to another that are not linked together
- As a distribution media for software by vendors

Magnetic Disk - Basics

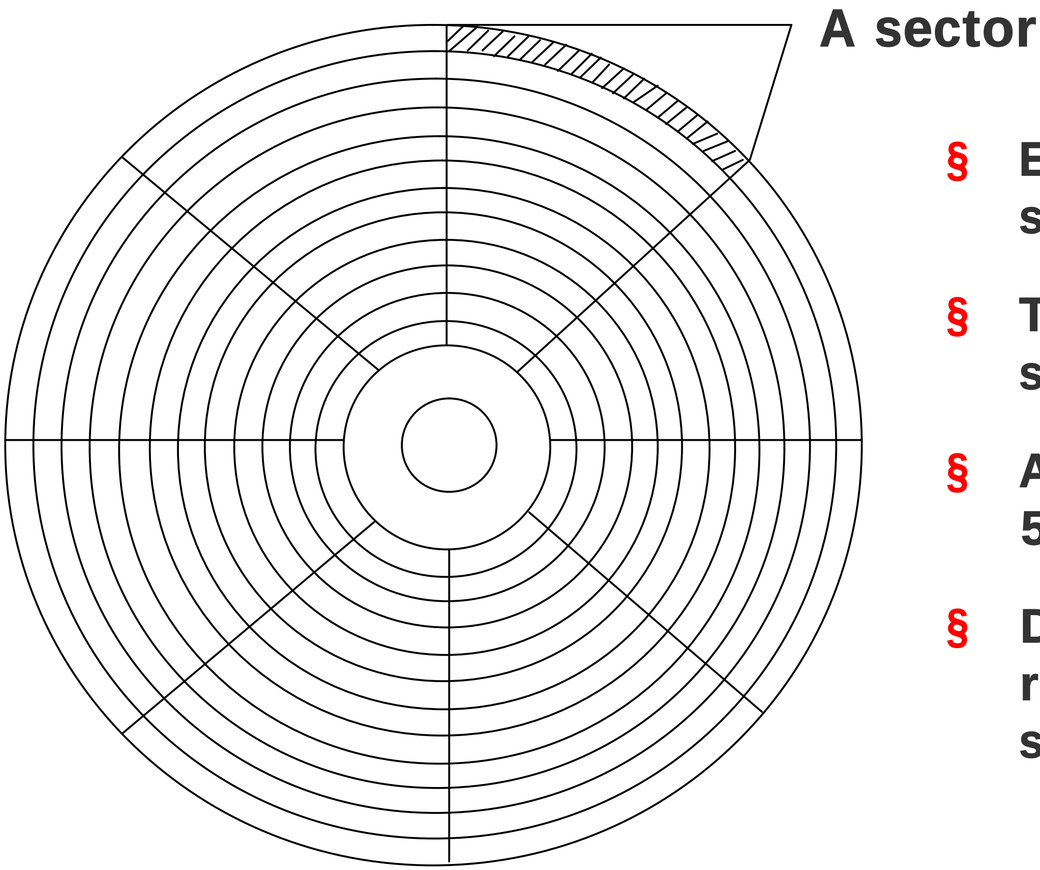
- Commonly used direct-access secondary storage device.
- Physically, a magnetic disk is a thin, circular plate/platter made of metal or plastic that is usually coated on both sides with a magnetizable recording material such as iron-oxide
- Data are recorded on the disk in the form of tiny invisible magnetized and non-magnetized spots (representing 1s and 0s) on the coated surfaces of the disk
- The disk is stored in a specially designed protective envelope or cartridge, or several of them are stacked together in a sealed, contamination-free container

Magnetic Disk – Storage Organization Illustrates the Concept of Tracks



- A disk's surface is divided into a number of invisible concentric circles called tracks
- The tracks are numbered consecutively from outermost to innermost starting from zero
- The number of tracks on a disk may be as few as 40 on small, low-capacity disks, to several thousand on large, high-capacity disks

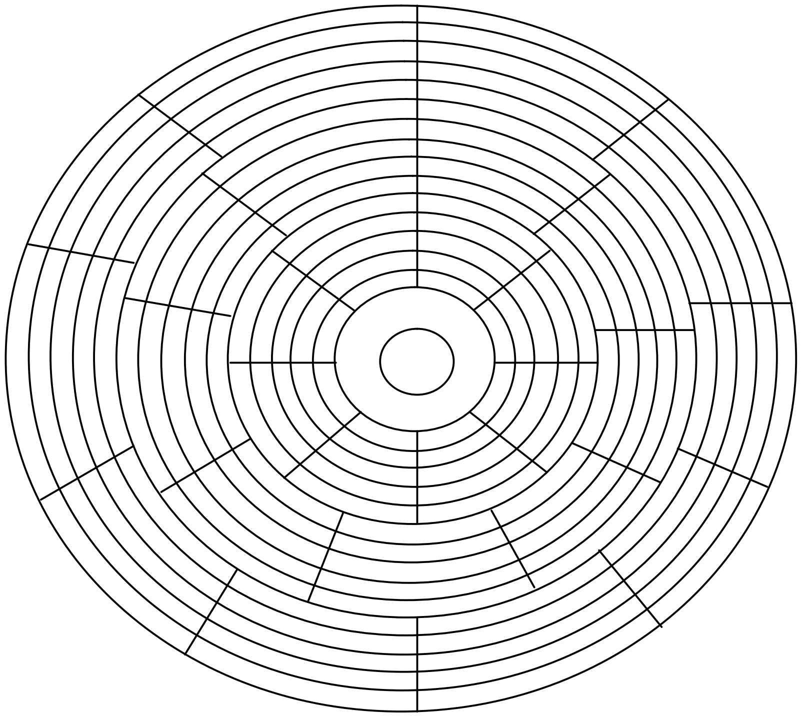
Magnetic Disk – Storage Organization Illustrates the Concept of Sectors



- Each track of a disk is subdivided into sectors
- There are 8 or more sectors per track
- A sector typically contains 512 bytes
- Disk drives are designed to read/write only whole sectors at a time

Magnetic Disk – Storage Organization

Illustrates Grouping of Tracks and Use of Different Number of Sectors in Tracks of Different Groups for Increased Storage Capacity

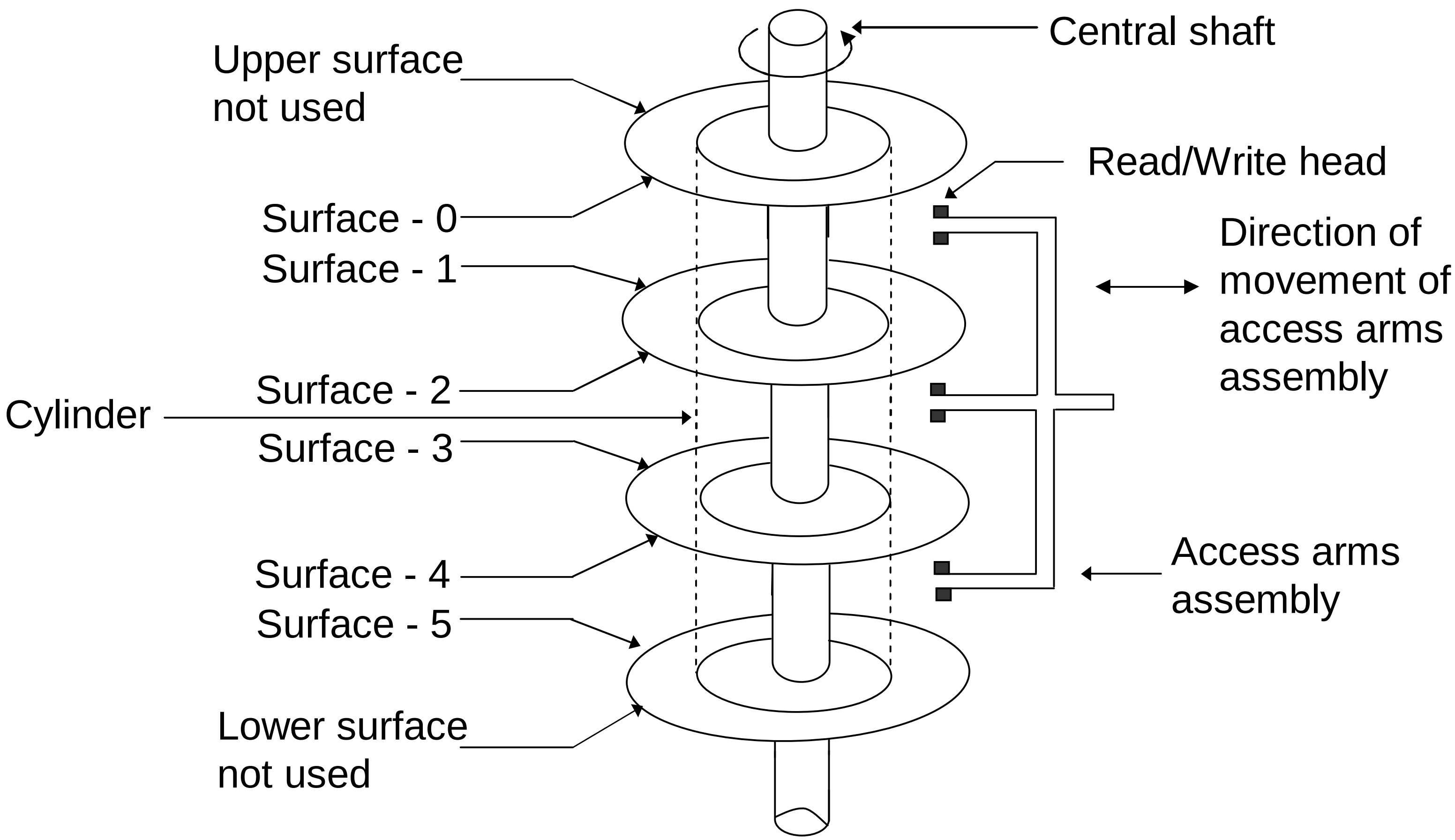


- **Innermost group of tracks has 8 sectors/track**
- **Next groups of tracks has 9 sectors/track**
- **Outermost group of tracks has 10 sectors/track**

Magnetic Disk – Disk Address or Address of a Record on a Disk

- Disk address represents the physical location of the record on the disk
- It is comprised of the sector number, track number, and surface number (when double-sided disks are used)
- This scheme is called the *CHS addressing* or *Cylinder-Head-Sector* addressing. The same is also referred to as *disk geometry*

Magnetic Disk – Storage Organization (Illustrates the Concept of Cylinder)

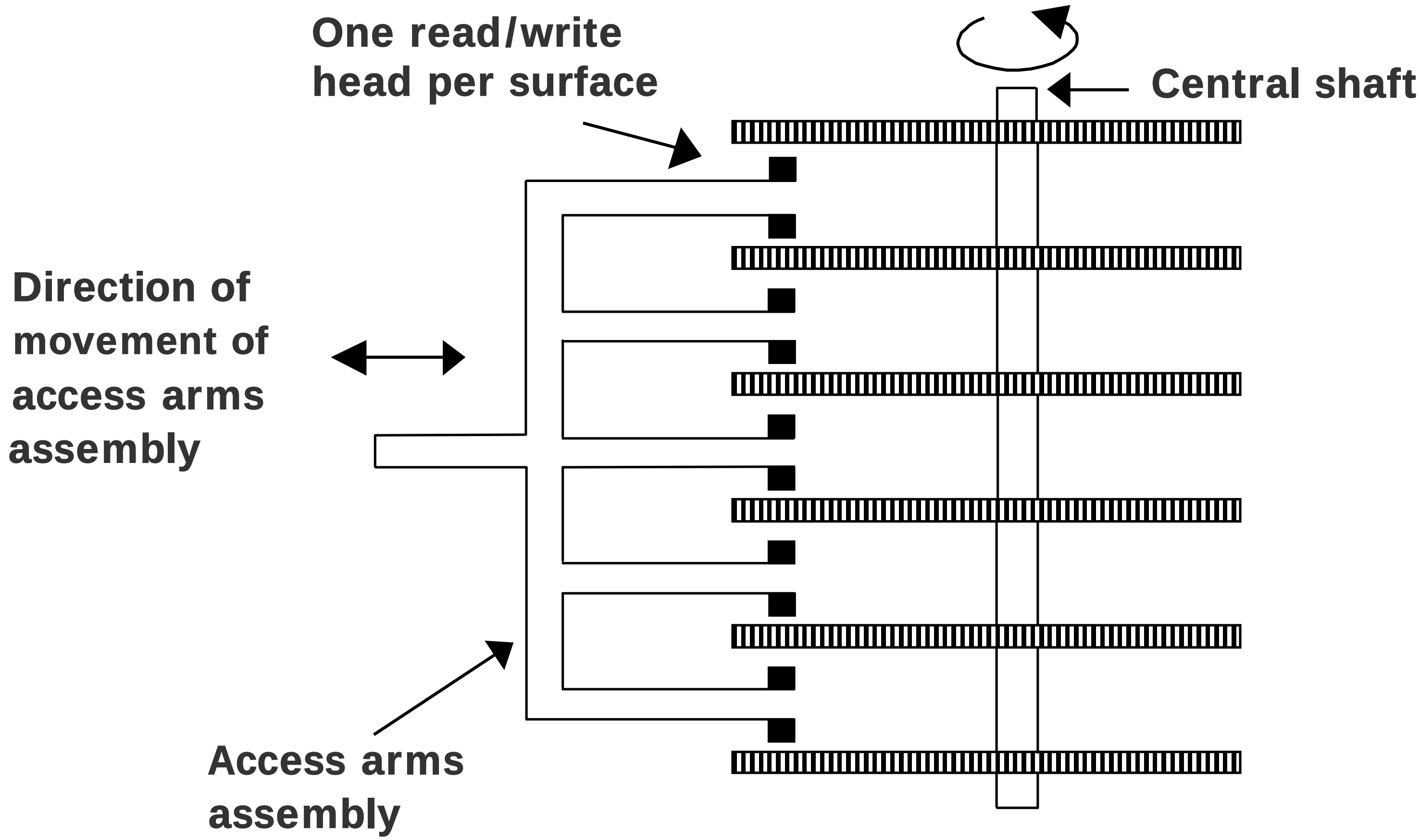


No. of disk platters = 4, No. of usable surfaces = 6. A set of corresponding tracks on all the 6 surfaces is called a cylinder.

Magnetic Disk – Storage Capacity

Storage capacity of a disk system = Number of recording surfaces
× Number of tracks per surface
× Number of sectors per track
× Number of bytes per sector

Magnetic Disk Pack – Access Mechanism



Vertical cross section of a disk system. There is one read/write head per recording surface

Magnetic Disk – Access Time

- *Disk access time* is the interval between the instant a computer makes a request for transfer of data from a disk system to the primary storage and the instant this operation is completed

- Disk access time depends on the following three parameters:

  – *Seek Time*: It is the time required to position the read/write head over the desired track, as soon as a read/write command is received by the disk unit

  – *Latency*: It is the time required to spin the desired sector under the read/write head, once the read/write head is positioned on the desired track

Magnetic Disk – Access Time

– *Transfer Rate*: It is the rate at which data are read/written to the disk, once the read/write head is positioned over the desired sector

§ As the transfer rate is negligible as compared to seek time and latency,

Average access time

= Average seek time + Average latency

Disk Formatting

- Process of preparing a new disk by the computer system in which the disk is to be used.
- For this, a new (unformatted) disk is inserted in the disk drive of the computer system and the disk formatting command is initiated
- Low-level disk formatting
  - Disk drive's read/write head lays down a magnetic pattern on the disk's surface
  - Enables the disk drive to organize and store the data in the data organization defined for the disk drive of the computer

(Continued on next slide)

Disk Formatting

(Continued from previous slide..)

- OS-level disk formatting
  - Creates the File Allocation Table (FAT) that is a table with the sector and track locations of data
  - Leaves sufficient space for FAT to grow
  - Scans and marks bad sectors
- One of the basic tasks handled by the computer's operating system
- Enables the use of disks manufactured by third party vendors into one's own computer system

Magnetic Disk – Disk Drive

- Unit used for reading/writing of data on/from a magnetic disk
- Contains all the mechanical, electrical and electronic components for holding one or more disks and for reading or writing of information on to it

(Continued on next slide)

Magnetic Disk – Disk Drive

(Continued from previous slide..)

- Although disk drives vary greatly in their shape, size and disk formatting pattern, they can be broadly classified into two types:
  - *Those with interchangeable magnetic disks*, which allow the loading and unloading of magnetic disks as and when they are needed for reading/writing of data on to them
  - *Those with fixed magnetic disks*, which come along with a set of permanently fixed disks. The disks are not removable from their disk drives

Magnetic Disk – Disk Controller

- Disk drive is connected to and controlled by a disk controller, which interprets the commands for operating the disk drive
- Typically supports only *read* and *write* commands, which need disk address (surface number, cylinder/track number, and sector number) as parameters
- Connected to and controls more than one disk drive, in which case the disk drive number is also needed as a parameters of *read* and *write* commands

Types of Magnetic Disks

- Magnetic Disks
  - Floppy Disks
  - Hard Disks
    - Zip/Bernoulli Disks
    - Disk Packs
    - Winchester Disks

Floppy Disks

- Round, flat piece of flexible plastic disks coated with magnetic oxide
- So called because they are made of flexible plastic plates which can bend
- Also known as floppies or diskettes
- Plastic disk is encased in a square plastic or vinyl jacket cover that gives handling protection to the disk surface

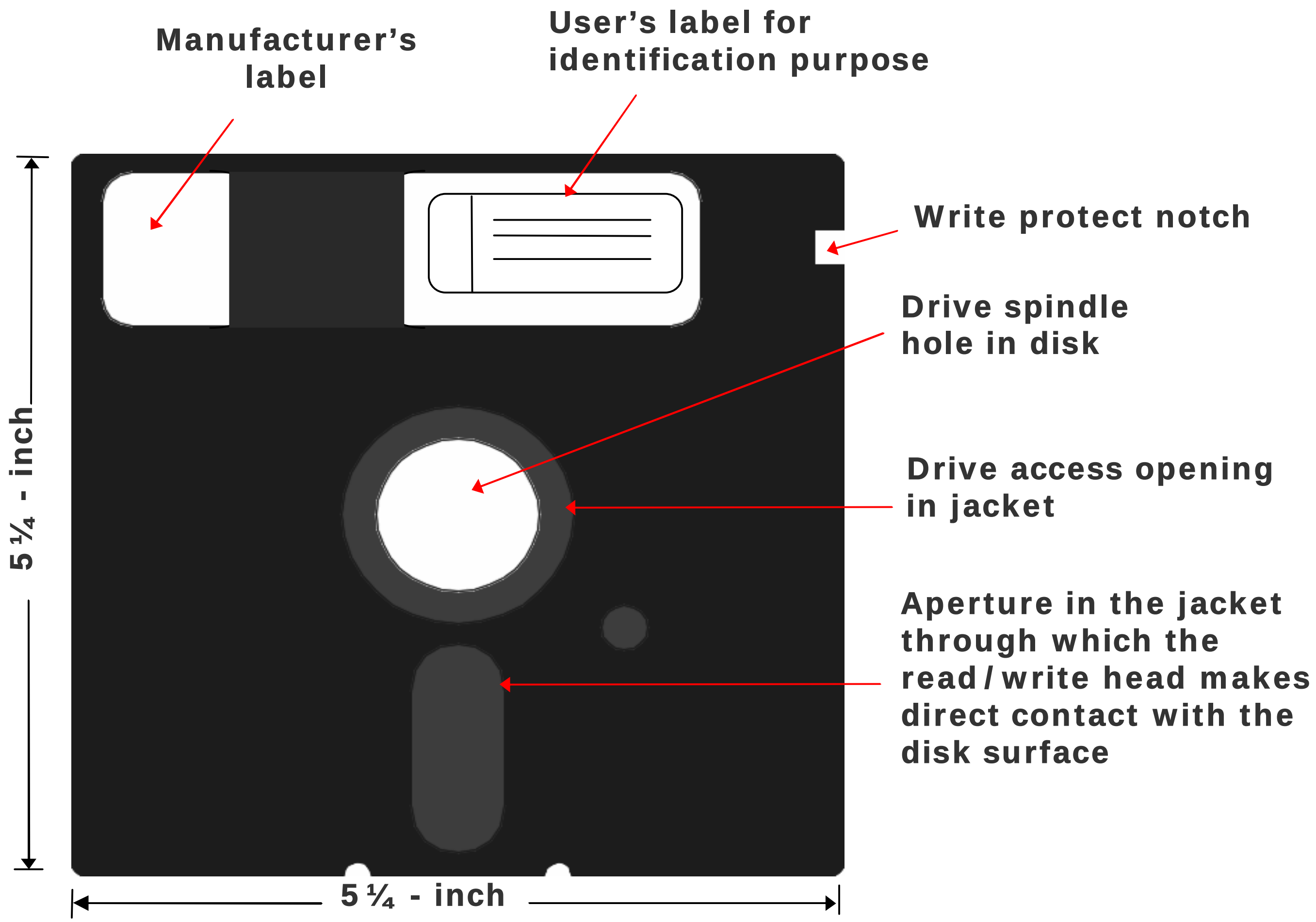
(Continued on next slide)

Floppy Disks

(Continued from previous slide..)

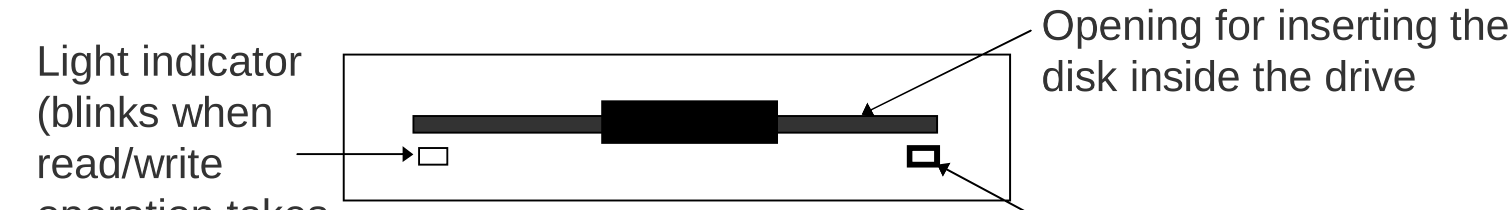
- The two types of floppy disks in use today are:
  - 5¼-inch diskette, whose diameter is 5¼-inch. It is encased in a square, flexible vinyl jacket
  - 3½-inch diskette, whose diameter is 3½-inch. It is encased in a square, hard plastic jacket
- Most popular and inexpensive secondary storage medium used in small computers

A 5 ¼-inch Floppy Disk

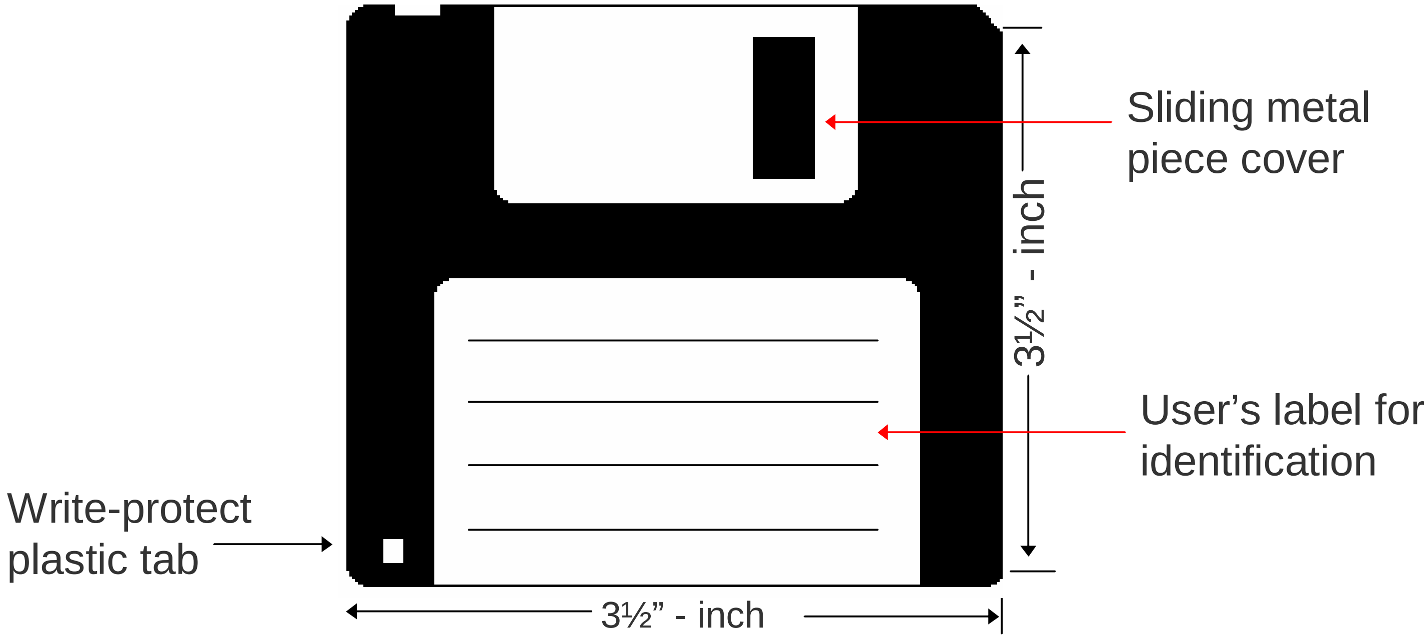


A 5 ¼-inch floppy disk enclosed within jacket. The drive mechanism clamps on to a portion of the disk exposed by the drive access opening in the jacket

A 3 ½ -inch Floppy Disk



(a) Front view of a floppy disk drive.



(b) A 3½ - inch floppy disk.

Storage Capacities of Various Types of Floppy Disks

Size (Diameter in inches)	No. of surfaces	No. of tracks	No. of sectors/track	No. of bytes/sector	Capacity in bytes	Approximate capacity
5¼	2	40	9	512	3,68,640	360 KB
5¼	2	80	15	512	12,28,800	1.2 MB
3½	2	40	18	512	7,37,280	720 KB
3½	2	80	18	512	14,74,560	1.4 MB
3½	2	80	36	512	29,49,120	2.88 MB

Hard Disks

- Round, flat piece of rigid metal (frequently aluminium) disks coated with magnetic oxide
- Come in many sizes, ranging from 1 to 14-inch diameter.
- Depending on how they are packaged, hard disks are of three types:
    - Zip/Bernoulli disks
    - Disk packs
    - Winchester disks
- Primary on-line secondary storage device for most computer systems today

Zip / Bernoulli Disks

- Uses a single hard disk platter encased in a plastic cartridge
- Disk drives may be portable or fixed type
- Fixed type is part of the computer system, permanently connected to it
- Portable type can be carried to a computer system, connected to it for the duration of use, and then can be disconnected and taken away when the work is done
- Zip disks can be easily inserted/removed from a zip drive just as we insert/remove floppy disks in a floppy disk drive

Disk Packs

- Uses multiple (two or more) hard disk platters mounted on a single central shaft
- Disk drives have a separate read/write head for each usable disk surface (the upper surface of the top-most disk and the lower surface of the bottom most disk is not used)
- Disks are of removable/interchangeable type in the sense that they have to be mounted on the disk drive before they can be used, and can be removed and kept off-line when not in use

Winchester Disks

- Uses multiple (two or more) hard disk platters mounted on a single central shaft
- Hard disk platters and the disk drive are sealed together in a contamination-free container and cannot be separated from each other

(Continued on next slide)

Winchester Disks

(Continued from previous slide..)

- For the same number of disks, Winchester disks have larger storage capacity than disk packs because:
  - All the surfaces of all disks are used for data recording

    They employ much greater precision of data recording, resulting in greater data recording density
- Named after the .30-30 Winchester rifle because the early Winchester disk systems had two 30-MB disks sealed together with the disk drive

Advantages of Magnetic Disks

- More suitable than magnetic tapes for a wider range of applications because they support direct access of data
- Random access property enables them to be used simultaneously by multiple users as a shared device. A tape is not suitable for such type of usage due to its sequential-access property
- Suitable for both on-line and off-line storage of data

(Continued on next slide)

Advantages of Magnetic Disks

(Continued from previous slide..)

- **Except for the fixed type Winchester disks, the storage capacity of other magnetic disks is virtually unlimited as many disks can be used for storing very large data sets**
- **Due to their low cost and high data recording densities, the cost per bit of storage is low for magnetic disks.**
- **An additional cost benefit is that magnetic disks can be erased and reused many times**
- **Floppy disks and zip disks are compact and light in weight. Hence they are easy to handle and store.**
- **Very large amount of data can be stored in a small storage space**

(Continued on next slide)

Advantages of Magnetic Disks

- Due to their compact size and light weight, floppy disks and zip disks are also easily portable from one place to another
- They are often used for transferring data and programs from one computer to another, which are not linked together
- Any information desired from a disk storage can be accessed in a few milliseconds because it is a direct access storage device

(Continued on next slide)

Advantages of Magnetic Disks

(Continued from previous slide..)

- Data transfer rate for a magnetic disk system is normally higher than a tape system
- Magnetic disks are less vulnerable to data corruption due to careless handling or unfavorable temperature and humidity conditions than magnetic tapes

Limitations of Magnetic Disks

- Although used for both random processing and sequential processing of data, for applications of the latter type, it may be less efficient than magnetic tapes
- More difficult to maintain the security of information stored on shared, on-line secondary storage devices, as compared to magnetic tapes or other types of magnetic disks

(Continued on next slide)

Limitations of Magnetic Disks

(Continued from previous slide..)

- For Winchester disks, a disk crash or drive failure often results in loss of entire stored data. It is not easy to recover the lost data. Suitable backup procedures are suggested for data stored on Winchester disks
- Some types of magnetic disks, such as disk packs and Winchester disks, are not so easily portable like magnetic tapes
- On a cost-per-bit basis, the cost of magnetic disks is low, but the cost of magnetic tapes is even lower

(Continued on next slide)

Limitations of Magnetic Disks

(Continued from previous slide..)

- Must be stored in a dust-free environment
- Floppy disks, zip disks and disk packs should be labeled properly to prevent erasure of useful data by mistake

Uses of Magnetic Disks

- For applications that are based on random data processing
- As a shared on-line secondary storage device. Winchester disks and disk packs are often used for this purpose
- As a backup device for off-line storage of data. Floppy disks, zip disks, and disk packs are often used for this purpose

(Continued on next slide)

Uses of Magnetic Disks

(Continued from previous slide..)

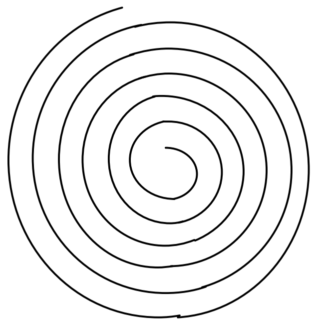
- Archiving of data not used frequently, but may be used once in a while. Floppy disks, zip disks, and disk packs are often used for this purpose
- Transferring of data and programs from one computer to another that are not linked together. Floppy disks and zip disks are often used for this purpose
- Distribution of software by vendors. Originally sold software or software updates are often distributed by vendors on floppy disks and zip disks

Optical Disk – Basics

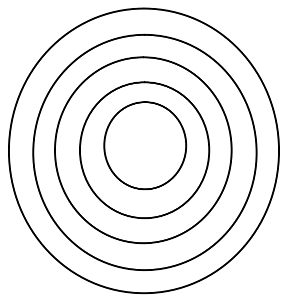
- Consists of a circular disk, which is coated with a thin metal or some other material that is highly reflective
- Laser beam technology is used for recording/reading of data on the disk
- Also known as laser disk / optical laser disk, due to the use of laser beam technology
- Proved to be a promising random access medium for high capacity secondary storage because it can store extremely large amounts of data in a limited space

Optical Disk – Storage Organization

- Has one long spiral track, which starts at the outer edge and spirals inward to the center
- Track is divided into equal size sectors



(a) Track pattern on an optical disk



(b) Track pattern on a magnetic disk

Difference in track patterns on optical and magnetic disks.

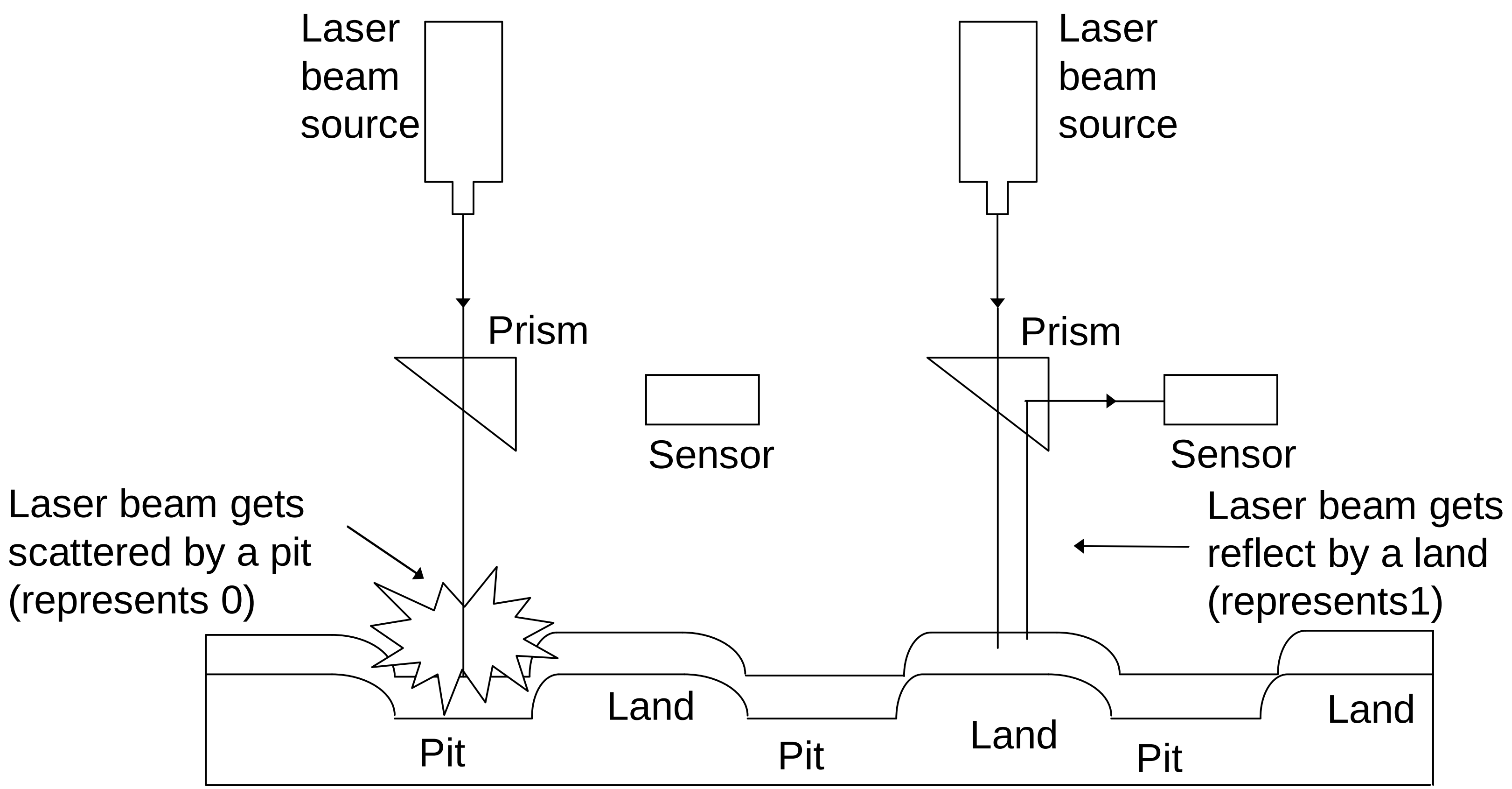
Optical Disk – Storage Capacity

Storage capacity of an optical disk

= Number of sectors
´ Number of bytes per sector

The most popular optical disk uses a disk of 5.25 inch diameter with storage capacity of around 650 Megabytes

Optical Disk – Access Mechanism



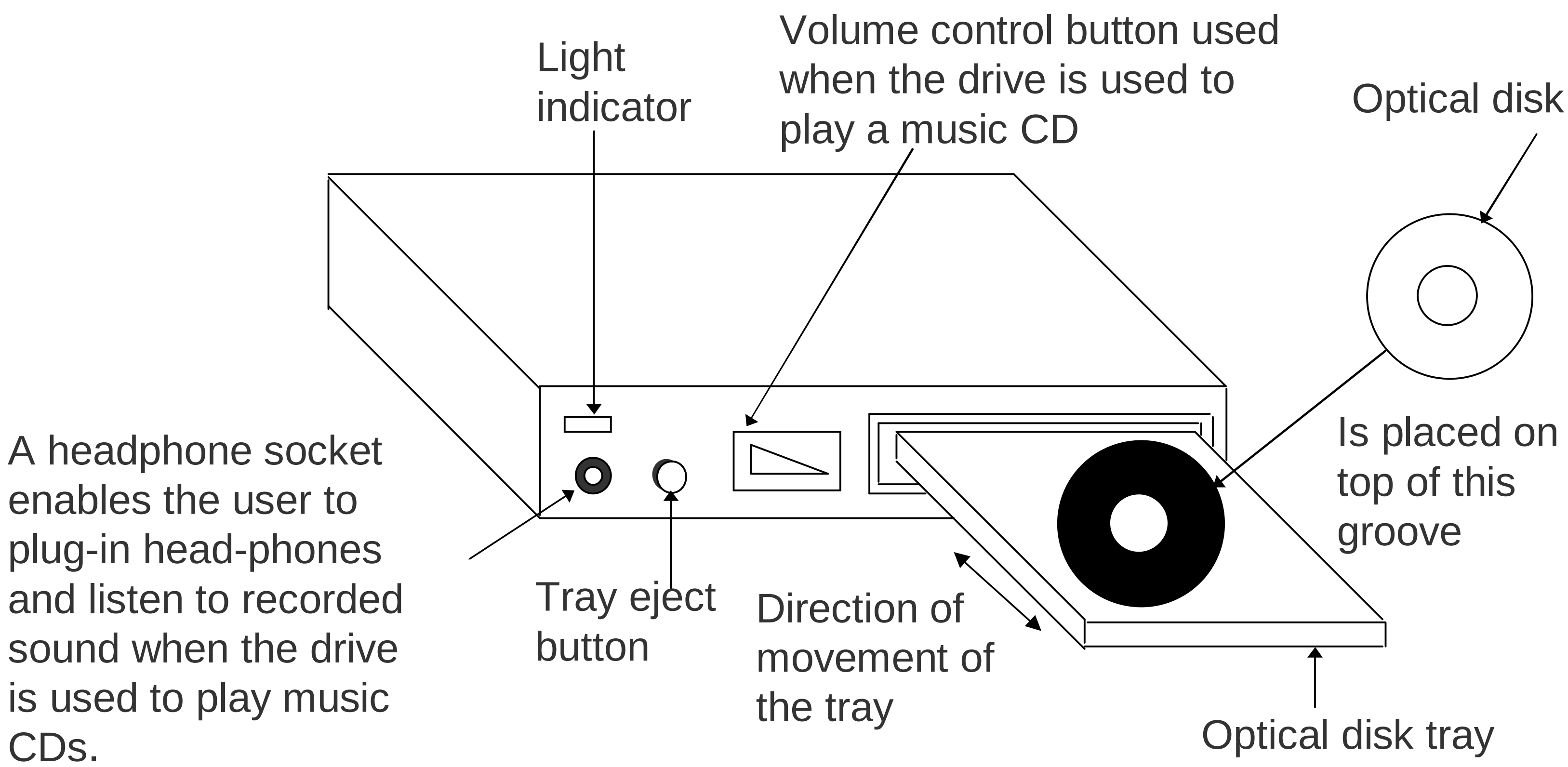
Optical Disk – Access Time

- With optical disks, each sector has the same length regardless of whether it is located near or away from the disk's center
- Rotation speed of the disk must vary inversely with the radius. Hence, optical disk drives use a constant linear velocity (CLV) encoding scheme
- Leads to slower data access time (larger access time) for optical disks than magnetic disks
- Access times for optical disks are typically in the range of 100 to 300 milliseconds and that of hard disks are in the range of 10 to 30 milliseconds

Optical Disk Drive

- Uses laser beam technology for reading/writing of data
- Has no mechanical read/write access arm
- Uses a constant linear velocity (CLV) encoding scheme, in which the rotational speed of the disk varies inversely with the radius

Optical Disk Drive



Types of Optical Disks

The types of optical disks in use today are:

CD-ROM

- Stands for Compact Disk-Read Only Memory
- Packaged as shiny, silver color metal disk of 5¼ inch (12cm) diameter, having a storage capacity of about 650 Megabytes
- Disks come pre-recorded and the information stored on them cannot be altered
- Pre-stamped (pre-recorded) by their suppliers, by a process called *mastering*

(Continued on next slide)

Types of Optical Disks

(Continued from previous slide..)

- Provide an excellent medium to distribute large amounts of data in electronic dorm at low cost.
- A single CD-ROM disk can hold a complete encyclopedia, or a dictionary, or a world atlas, or biographies of great people, etc
- Used for distribution of electronic version of conference proceedings, journals, magazines, books, and multimedia applications such as video games
- Used by software vendors for distribution of software to their customers

Types of Optical Disks

WORM Disk / CD-Recordable (CD-R)

- Stands for Write Once Read Many. Data can be written only once on them, but can be read many times
- Same as CD-ROM and has same storage capacity
- Allow users to create their own CD-ROM disks by using a CD-recordable (CD-R) drive that can be attached to a computer as a regular peripheral device
- Data to be recorded can be written on its surface in multiple recording sessions

(Continued on next slide)

Types of Optical Disks

(Continued from previous slide..)

- Sessions after the first one are always additive and cannot alter the etched/burned information of earlier sessions
- Information recorded on them can be read by any ordinary CD-ROM drive
- They are used for data archiving and for making a permanent record of data. For example, many banks use them for storing their daily transactions

Types of Optical Disks

CD-Read / Write (CD-RW)

- Same as CD-R and has same storage capacity
- Allow users to create their own CD-ROM disks by using a CD-recordable (CD-R) drive that can be attached to a computer as a regular peripheral device
- Data to be recorded can be written on its surface in multiple recording sessions
- Made of metallic alloy layer whose chemical properties are changed during burn and erase
- Can be erased and written afresh

Types of Optical Disks

Digital Video / Versatile Disk (DVD)

- Looks same as CD-ROM but has capacity of 4.7 GB or 8.5 GB
- Designed primarily to store and distribute movies
- Can be used for storage of large data
- Allows storage of video in 4:3 or 16:9 aspect-ratios in MPEG-2 video format using NTSC or PAL resolution
- Audio is usually Dolby® Digital (AC-3) or Digital Theater System (DTS) and can be either monaural or 5.1 Surround Sound

Advantages of Optical Disks

- The cost-per-bit of storage for optical disks is very low because of their low cost and enormous storage density.
- The use of a single spiral track makes optical disks an ideal storage medium for reading large blocks of sequential data, such as music.
- Optical disk drives do not have any mechanical read/write heads to rub against or crash into the disk surface. This makes optical disks a more reliable storage medium than magnetic tapes or magnetic disks.
- Optical disks have a data storage life in excess of 30 years. This makes them a better storage medium for data archiving as compared to magnetic tapes or magnetic disks.

Advantages of Optical Disks

- As data once stored on an optical disk becomes permanent, danger of stored data getting inadvertently erased/overwritten is removed
- Due to their compact size and light weight, optical disks are easy to handle, store, and port from one place to another
- Music CDs can be played on a computer having a CD-ROM drive along with a sound board and speakers. This allows computer systems to be also used as music systems

Limitations of Optical Disks

- It is largely read-only (permanent) storage medium. Data once recorded, cannot be erased and hence the optical disks cannot be reused
- The data access speed for optical disks is slower than magnetic disks
- Optical disks require a complicated drive mechanism

Uses of Optical Disks

- For distributing large amounts of data at low cost
- For distribution of electronic version of conference proceedings, journals, magazines, books, product catalogs, etc
- For distribution of new or upgraded versions of software products by software vendors

(Continued on next slide)

Uses of Optical Disks

(Continued from previous slide..)

- For storage and distribution of a wide variety of multimedia applications
- For archiving of data, which are not used frequently, but which may be used once in a while
- WORM disks are often used by end-user companies to make permanent storage of their own proprietary information

Memory Storage Devices

Flash Drive (Pen Drive)

- Relatively new secondary storage device based on flash memory, enabling easy transport of data from one computer to another
- Compact device of the size of a pen, comes in various shapes and stylish designs and may have different added features
- Plug-and-play device that simply plugs into a USB (Universal Serial Bus) port of a computer, treated as removable drive
- Available storage capacities are 8MB, 16MB, 64MB, 128MB, 256MB, 512MB, 1GB, 2GB, 4GB, and 8GB

Memory Storage Devices

Memory Card (SD/MMC)

- Similar to Flash Drive but in card shape
- Plug-and-play device that simply plugs into a port of a computer, treated as removable drive
- Useful in electronic devices like Camera, music player
- Available storage capacities are 8MB, 16MB, 64MB, 128MB, 256MB, 512MB, 1GB, 2GB, 4GB, and 8GB

Mass Storage Devices

- As the name implies, these are storage systems having several trillions of bytes of data storage capacity
- They use multiple units of a storage media as a single secondary storage device
- The three commonly used types are:
  1. *Disk array*, which uses a set of magnetic disks
  2. *Automated tape library*, which uses a set of magnetic tapes
  3. *CD-ROM Jukebox*, which uses a set of CD-ROMs
- They are relatively slow having average access times in seconds

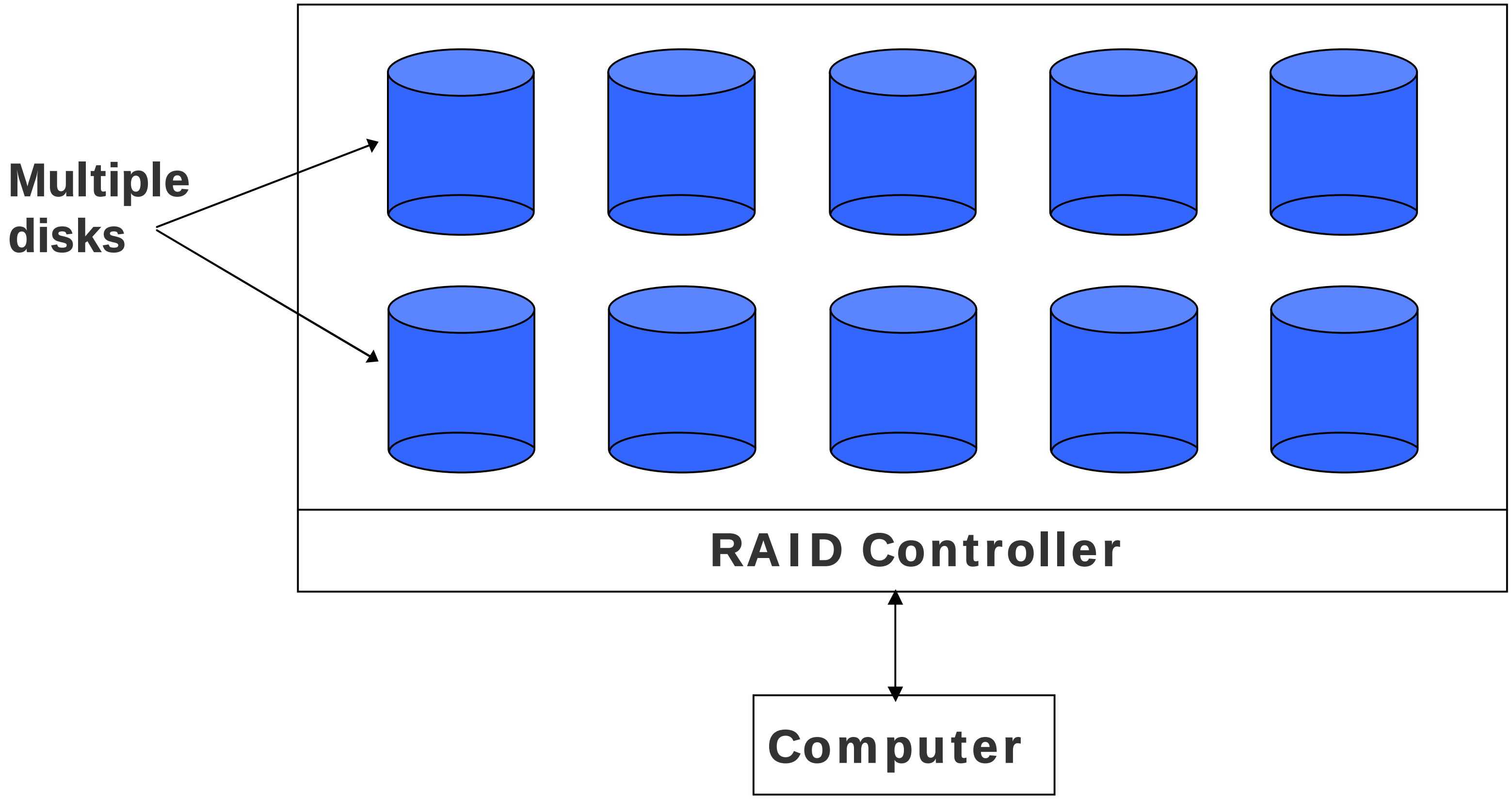
Disk Array

- Set of hard disks and hard disk drives with a controller mounted in a single box, forming a single large storage unit
- It is commonly known as a *RAID (Redundant Array of Inexpensive Disks*)
- As a secondary storage device, provides enhanced storage capacity, enhanced performance, and enhanced reliability

Disk Array

- Enhanced storage capacity is achieved by using multiple disks
- Enhanced performance is achieved by using parallel data transfer technique from multiple disks
- Enhanced reliability is achieved by using techniques such as mirroring or striping
- In *mirroring*, the system makes exact copies of files on two hard disks
- In *striping*, a file is partitioned into smaller parts and different parts of the file are stored on different disks

A RAID Unit



Automated Tape Library

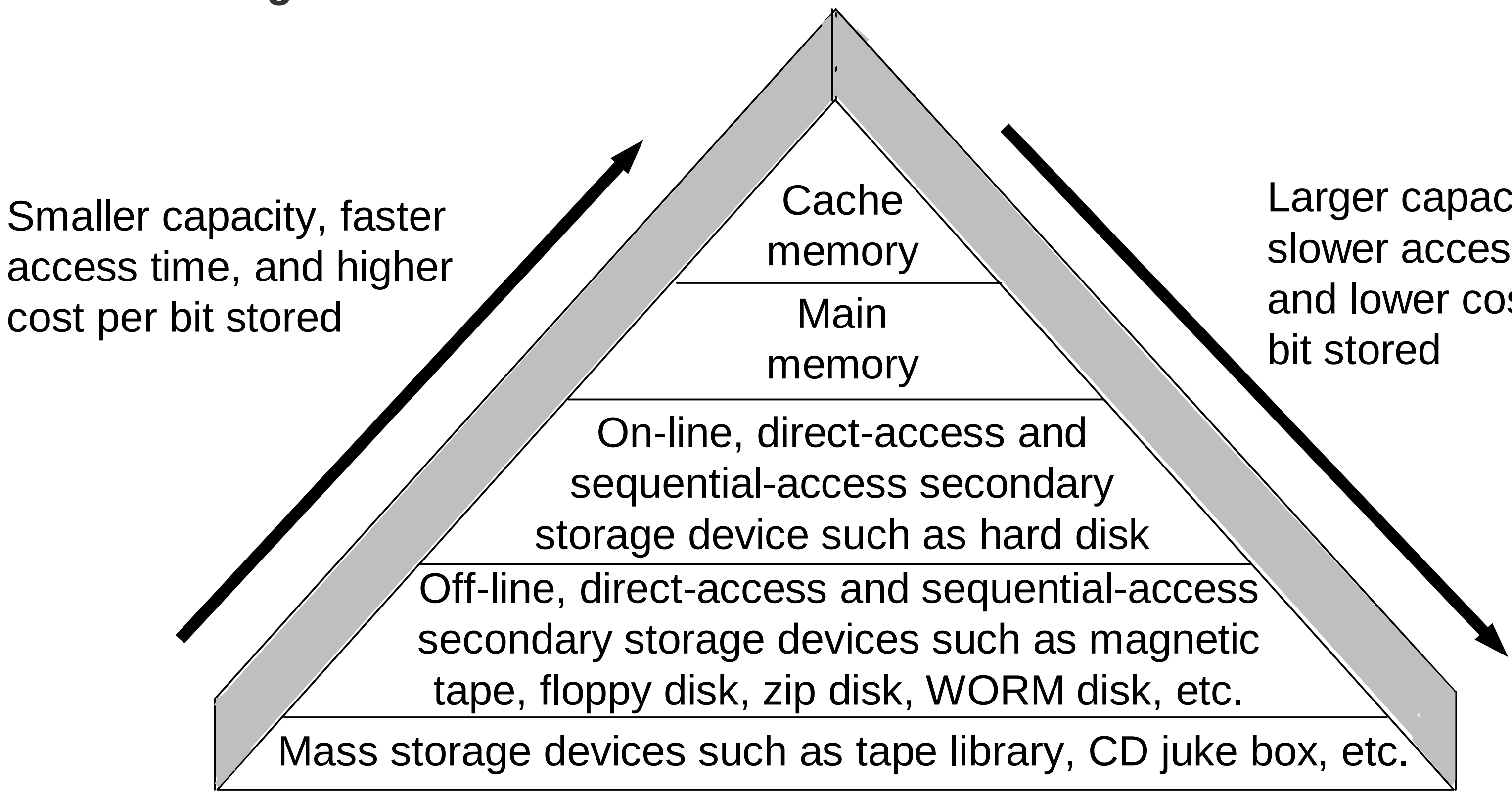
- Set of magnetic tapes and magnetic tape drives with a controller mounted in a single box, forming a single large storage unit
- Large tape library can accommodate up to several hundred high capacity magnetic tapes bringing the storage capacity of the storage unit to several terabytes
- Typically used for data archiving and as on-line data backup devices for automated backup in large computer centers

CD-ROM Jukebox

- Set of CD-ROMs and CD-ROM drives with a controller mounted in a single box, forming a single large storage unit
- Large CD-ROM jukebox can accommodate up to several hundred CD-ROM disks bringing the storage capacity of the storage unit to several terabytes
- Used for archiving read-only data in such applications as on-line museums, on-line digital libraries, on-line encyclopedia, etc

Storage Hierarchy

As a single type of storage is not superior in speed of access, capacity, and cost, most computer systems make use of a hierarchy of storage technologies as shown below.



Key Words/Phrases

- Automated tape library
- Auxiliary memory
- Block
- Blocking
- Blocking factory
- CD-ROM
- CD-ROM jukebox
- Check bit
- Cylinder
- Data transfer rate
- Direct access device
- Disk array
- Disk controller
- Disk drive
- Disk formatting
- Disk pack
- DVD
- Even parity
- File Allocation Tube (FAT)
- Floppy disk
- Hard disk
- Inter-block gap (IBG)
- Inter-record gap (IRG)
- Land
- Latency
- Magnetic disk
- Magnetic tape
- Magnetic tape drive
- Mass storage devices
- Master file
- Odd parity
- Off-line storage
- On-line storage
- Optical disk
- Parallel representation
- Parity bit
- Pit

(Continued on next slide)

Key Words/Phrases

(Continued from previous slide..)

- QIC Standard
- Record
- Redundant Array of Inexpensive Disks (RAID)
- Secondary storage
- Sector
- Seek time
- Sequential access device
- Storage hierarchy
- Tape controller
- Track
- Transaction file
- Winchester disk
- WORM disk
- Zip disk