1

COMPUTER BASICS

1.0. INTRODUCTION

Every organisation regardless of its size or purpose is concerned with processing of facts or data for its smooth and efficient functioning. With the ever-increasing amount of data to be processed in shortest possible time, organisations felt the need for faster, cheaper and more efficient methods of processing data. To fill this need various types of automated devices were developed and foremost among them was the introduction of the electronic computer in the later half of the 20th century. Nowadays, the computers have come up in such a big way that their presence is felt in every sphere of life like education, business, research, medicine, banking, airflight, etc., to mention a few. This chapter deals with basics of computer.

1.1. DATA, INFORMATION AND DATA PROCESSING

The word data is the plural of datum, which means facts. The term *data* includes all facts and figures or description of an idea, object, condition or situation. Every field of activity produces data—names of employees, marks obtained by students, details of purchase made, etc.

Information is processed data that is organized and meaningful to the person receiving it. Data is thus a raw material that is transformed into information by processing. For example, the temperature, the atmospheric pressure, humidity, etc., at a certain place represent data item. These data items when processed become more meaningful and predict the weather as Sunny day, Cloudy day, Rainy day, etc., which is an information.

Data processing is a process that converts the data into information.

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{Input}\\\text{(Data)}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Process}\\\text{(Data Processing)}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Output}\\\text{(Information)}\end{array}}
\end{tikzcd}$$

Fig. 1.1 : Information by processing data

With the development of electronic computer, the data processing is done by computer system in many ways. Thus, when we talk of data processing, it is in reference to the computer data processing.

1.2. WHAT IS A COMPUTER?

The computer is an automatic machine made up of electronic and electro-mechanical devices that processes data under program control to generate meaningful information with speed and accuracy. Thus, the computer can be called as "Electronic Data Processor" (EDP) or an "Automatic Data Processor".

1.3. CHARACTERISTICS OF COMPUTERS

(*a*) **Speed:** As the computer is electronic, its internal speed is virtually instantaneous. The speed of execution of operations by modern computer is several million operations per second. The time required for computers to execute basic operations, such as addition and subtraction, varies from a few microseconds (10^{-6}) for small computers to nanoseconds (10^{-9}) and for large ones even the picoseconds (10^{-12}).

(*b*) **Automatic operation:** Computers are automatic in operation. Once data and program are fed to a computer, operation of the computer is automatic in the sequence of steps defined by the program as opposed to mechanical or electronic calculator in which operator intervention is required.

(*c*) **Storage:** For automatic processing of data at a very high speed by the computer, it is necessary that both data and specified sequence of instructions to be performed be stored somewhere within the computer in advance. Modern supercomputers have several millions of words of primary memory. Large volume of data can be conveniently stored, accessed and altered.

(*d*) **Versatility:** Computers are extremely versatile, and are capable of performing almost any task, provided that the task can be reduced to a series of logical steps. The machine can be used to solve problems relating to various different fields like complex scientific problems, business problems, the problem of traffic at an airport, etc.

(*e*) **Diligence:** A computer, being a machine, does not suffer from boredom, tiredness or lack of concentration even if it has to work for long hours. Moreover, even after working for long hours, there is no loss of accuracy in its results. Thus, if a computer is to perform millions of calculations, it will perform the last calculation with the same accuracy and perfection as it will do the first one.

(*f*) **Reliability:** Today's computers are extremely reliable and results are always same as per design. There are two sources of errors: (*i*) Human error in program design and logic, (*ii*) Machine error but due to increased efficiency in error-detecting techniques, these seldom lead to false results.

1.4. TYPES OF COMPUTER

Computers are of two main types—Analog and Digital, although there are Hybrid computers with features of both.

1.4.1. Analog Computer

Analog is a Greek word which means establishing similarities between two quantities. The analog computers work on the principle of measurement. In analog computers the physical processes such as pressure, acceleration, power, force, viscosity, etc., are represented by electrical current or voltage signals. When physical parameter is continuously varying, its analogous electrical parameter also will be continuously varying. Such a continuously varying electrical voltage is fed as input to the analog computer which are then manipulated using various electronic modules such as inverters, comparators, summers, multipliers and integrators, etc., and the results are obtained. These results are measured and displayed by meters, oscilloscopes. The computing units of analog computers are able to respond immediately to changes which they detect in the input variables and can perform very complex arithmetical functions at high speed while the actual process in the operation. This ability to operate in real-time means that these computers have many applications in the scientific and industrial fields in stimulating various physical systems or automatically controlling industrial processes. The sequence of steps that the machine has to execute in solving the problem is permanently wired into the circuitry of the machine.

1.4.2. Digital Computer

Digital computers work on the principle of counting. These computer operate on discrete numbers represented by a finite sequence of digits. In other words, a digital computer accepts discrete numbers as input and after performing the desired processing (operations) on these numbers, produces discrete numbers as output.

Since digital computers work directly on the variables of the problem, rather than on some equivalent continuous variables, they are more accurate than analog computers.

In analog computers, the degree of accuracy depends upon the instrument and the operator.

Depending on the flexibility in operation, digital computers are either special purpose or general purpose.

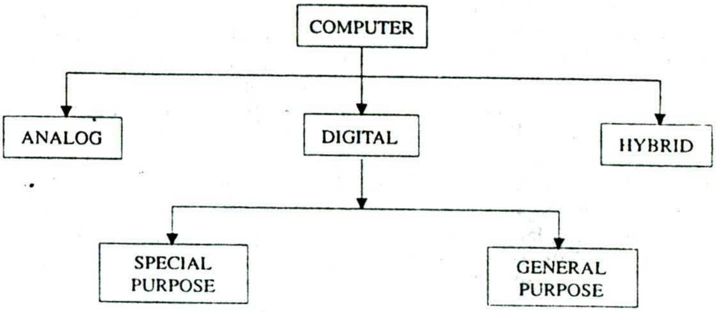


Fig. 1.2 : Types of computers

Special purpose computers (dedicated computers): They are designed to solve a restricted class of problems. They are designed and built to cater to the requirements of a particular task or application. It incorporates the instru·'ions needed into the design of internal circuitry so that it can perform the task ‹. a simple command quickly and efficiently. These dedicated systems may reduce the processing load on large computer.

Examples are: computers meant for process control in industry, computers meant for air traffic control, Robots, etc.

General purpose computers : They are designed to solve wide variety of problems to meet the needs of many different applications. The instructions needed to perform a particular task are not wired permanently into the internal storage. They are read from an input device and placed into the internal memory until they are needed. Examples are: payroll, sales analysis, etc.

1.4.3. Hybrid Computer

Hybrid computer is another type of computer which combines the features of both analog and digital computers. In many cases, a hybrid computer is an analog computer controlled by a digital computer instead of human beings. For example, in an intensive care unit, analog devices measure a patient's heart function, temperature and other vital signs. These measurements are then converted into numbers and supplied to digital components that monitor the patients vital signs and signals a nurse's station if abnormal readings are detected. Hybrid computers are used only for special applications. Main areas of their applications are aerospace and process control.

Normally, when we speak of a computer, it is understood as a digital computer. Nowadays, these are the most widely used machines. Essentially, a computer performs three functions:

(*i*) It accepts data (Input);
(*ii*) It processes data by performing desired ari:hmetic and logical operations (Processing); and
(*iii*) It generates data in the desired form (Output).

Five basic units are required for performing the above three functions.

1.5. DIFFERENCES BETWEEN ANALOG AND DIGITAL COMPUTERS

The digital computer is basically a counting device. Hence, presence, absence, and repetition rate of the signal are the important factors, and not the amplitudes of the signal.

The analog computer is fundamentally a measuring device. It operates simultaneously on the input data, interprets their changes, and indicates the resulting variations in the response of the system. Hence, the amplitude of the signal is an important factor.

In the case of digital computer, a problem changes results in the writing of a new program; the internal structure of the machine does not get altered. In the analog computer, inter-connections between the functional units represent the details of the problem. Hence, a new problem results in a new type of inter-connections within the machine.

The precision of a digital computer can be adjusted to suit the problem to any significant figure, whereas in analog computer, it depends on the accuracy of the components and the measuring device, and is usually of the order of 0.1 per cent.

Problems, involving large amount of data in discrete form, are most readily solved on digital computer. Differential equations and problems involving integration of continuous data are more readily solved on analog computer.

1.6. BASIC COMPONENTS (ORGANIZATION) OF A COMPUTER SYSTEM

Computer system consists of five basic units:

1.	Input Unit	:	Input data are fed into con.puter.
2.	Memory Unit	:	Both program and data are stored for processing.
3.	Arithmetic Logic Unit (ALU)	:	Data are actually processed.
4.	Output Unit	:	Output data are presented to the user.
5.	Control Unit	:	Controls all the operations of the computer.

The control unit, together with the ALU and memory units, constitutes the Central Processing Unit (CPU).

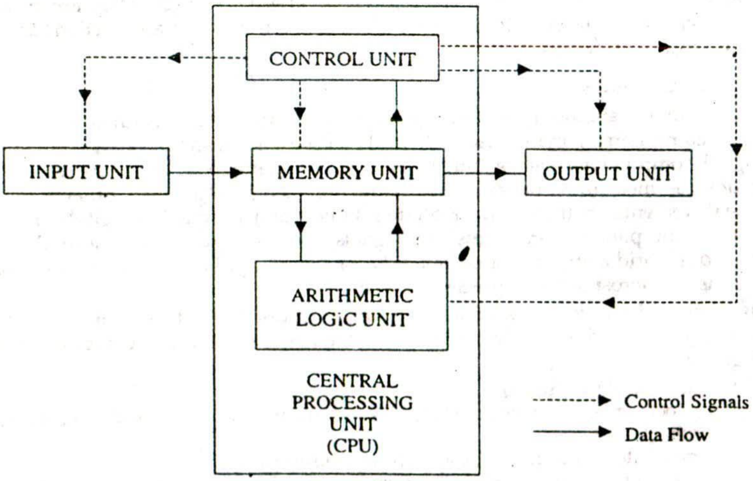


Fig. 1.3 : Block diagram of a digital computer

Input unit: An input unit is a device which accepts instructions and data in a form understandable to human beings, converts it into a machine readable form and transmits to the memory unit of the computer.

Some input devices require data to be stored in a medium like punched card, paper tape, magnetic tape, floppy disk, magnetic disk, etc., and data is read and transmitted by input devices like card reader, paper-tape reader, magnetic-tape reader, disk drives, etc., into a form understandable by the computer. The input can also be transmitted directly to the computer using a keyboard terminal. Currently available input devices are optical mark reader (OMR) and optical character reader (OCR), in which input is scanned by an array of photocells, converted into machine code and transmitted to the memory of the computer for processing. On identical principles bar-code readers read the information prepared in bar-code for application of computers in libraries, general stores, etc. Also, with the help of available magnetic ink character reader (MICR), information written/printed in magnetic ink is read and transmitted directly to the memory for processing. Electronic mouse, touch screens, light pens are also used as input device.

Memory unit: A computer system also has storage areas, often referred to as memory. The memory unit stores the information to be processed by the CPU. This information consists of the program as well as data. The memory can receive data, hold them and deliver them when instructed to do so. In modern computers, the internal memory consists of microelectronic semiconductor storage circuitry. The storage available in the memory is also known as main storage or primary storage. The data can be processed only when it is available in the main memory. Main memory is finite. It may be augmented by adding auxiliary or secondary storage, such as magnetic tapes, magnetic disks and drums which can store thousands of millions of characters. The information stored in the auxiliary storage can be transferred to the main memory for processing at a high speed.

Arithmetic logic unit (ALU): The ALU performs the actual processing of data including addition, subtraction, multiplication and also division. This unit also performs certain logical operations such as comparing two numbers to see one is larger than the other or if they are equal. Arithmetic or logic operation is performed by bringing the required operands into ALU. Suppose two numbers located in the main memory are to be added. They are brought into the arithmetic unit and temporarily stored in registers or in accumulators associated with this unit where the actual addition is carried out. The result is placed in one of the registers and subsequently transferred to the memory.

Control unit: The control unit directs and coordinates all activities of the computer system including the following:

1. Control of input/output devices.
2. Entry and retrieval of information from storage.
3. Routing of information between storage and the arithmetic logic unit.
4. Direction of arithmetic and logical operations.

Although control section does not process data, it acts as a central nervous system for the other data manipulating components of the computer. At the beginning of the processing the first program instruction is selected and fed into the control section from the program storage area. There it is interpreted, and from there signals are sent to other components to execute the necessary action.

Output unit: After a program is executed and the results computed, the results must be made available in a readable form. The computer system needs an output device to communicate the processed information to the user. The output device translates processed data from a machine coded form to a form that can be read and used by people. The most common types of output devices are the monitor, which resembles a television screen and the printer. Another common output device is the graphics plotter, which produces graphs, charts, or technical drawings on paper. An illustration is shown in Fig. 1.4.

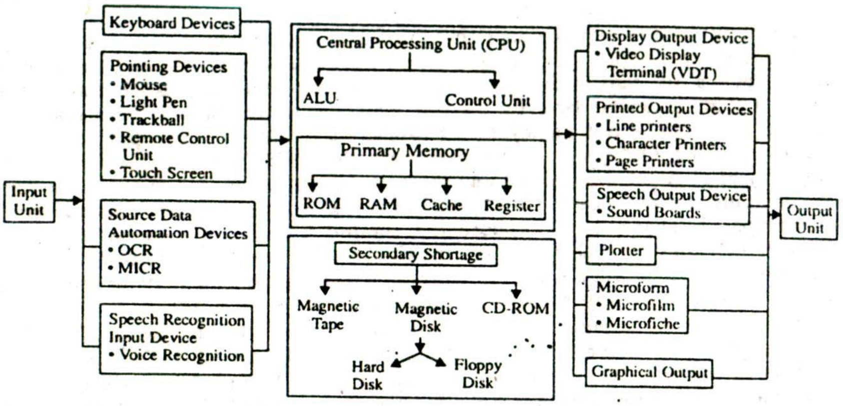

Input Unit
Keyboard Devices
Pointing Devices • Mouse • Light Pen • Trackball • Remote Control Unit • Touch Screen
Source Data Automation Devices • OCR • MICR
Speech Recognition Input Device • Voice Recognition
Central Processing Unit (CPU)
ALU
Control Unit
Primary Memory
ROM
RAM
Cache
Register
Secondary Shortage
Magnetic Tape
Magnetic Disk
CD-ROM
Hard Disk
Floppy Disk
Display Output Device • Video Display Terminal (VDT)
Printed Output Devices • Line printers • Character Printers • Page Printers
Speech Output Device • Sound Boards
Plotter
Microform • Microfilm • Microfiche
Graphical Output
Output Unit

Fig. 1.4: Basic components of a computer system

1.7. COMPUTER SYSTEM

A system is composed of a set of interacting parts which operate together to achieve some objective. Since a computer is composed of a set of parts like CPU, input/output devices, storage devices integrated together to process the data under program control. The term computer system rather than the term computer is used.

1.7.1. Computer Hardware and Software

The data are processed by a collection of electronic circuits and other devices that make up the computer system. The physical equipment and components which one can see, touch and feel in the computer system are called hardware. Mechanical, magnetic, electrical or optical devices used in the computer system are examples of computer hardware.

Computer hardware as a machine cannot solve a given problem on its own. The physical components are to be properly instructed to work in the desired way. Sets of

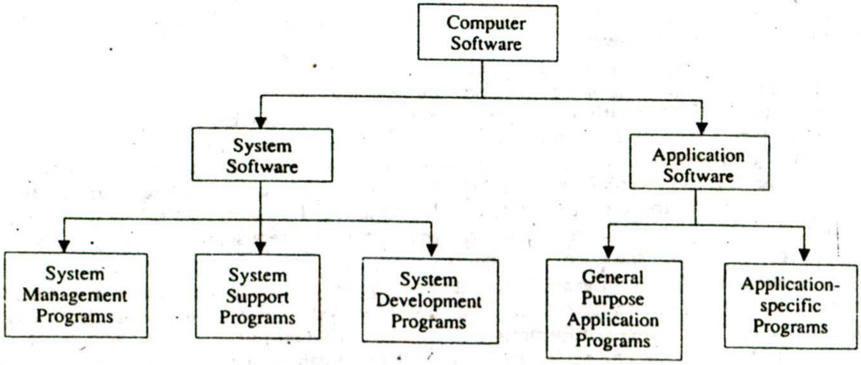

Computer Software
System Software
Application Software
System Management Programs
System Support Programs
System Development Programs
General Purpose Application Programs
Application-specific Programs

Fig. 1.5: An overview of computer software

programmed instructions which enable the hardware units to perform tasks constitute a computer's software which is, in a sense, the interface between a computer hardware and its users. Software is mainly divided into two categories: (*i*) Application software, and (*ii*) System software.

Application software: It refers to the programs that the programmer's write to accomplish (*i*) General purpose application programs such as word processing, electronic spreadsheet, graphics, etc., and (*ii*) Application specific programs such as inventory control, payroll, railway reservation, etc.

System software: The system software refers to the programs written for a specific computer to aid programmers/users of that computer. It controls all processing activities and make sure that the resources and the power of the computer are used in a most efficient way. It includes (*i*) System management programs such as operating systems, database management systems, etc., (*ii*) System support programs such as system utilities, system security monitors, and (*iii*) System development programs like language processors, Computer Aided Software Engineering (CASE) packages, etc.

Distinction between a Computer and an Electronic Calculator **(AMIE, W '97)**

A calculator only computes, whereas a computer can perform other functions which a calculator cannot. These differences are slowly decreasing with the advent of advanced programmable calculators.

The following areas distinguish a computer from a calculator:

Electronic Calculator	*Computer*
1. Generally non-programmable.	1. Always programmable.
2. Contains no secondary storage.	2. Generally have some secondary storage. External storage capacity is almost limitless.
3. No operating system/command interpreter needed.	3. Existence of operating system alongwith command interpreter is essential.
4. A small built-in display window.	4. A separate device, like a TV, enables it to display data and other information.
5. Does not have the capability of text processing.	5. Text processing is exclusively done.

REVIEW QUESTIONS SET

1. What are data? What is information? Explain the difference between these two terms.
2. Define a computer. What do you understand by data processing?
3. What is a computer? Explain briefly the working principle of various types of computers?
4. Define the term 'hardware' and 'software'.
5. What is meant by the following:
   (*a*) Computer system (*b*) Peripheral devices (*c*) Computer configuration.
6. List the basic components of a computer system and explain them.
7. Draw the block diagram of a computer and explain briefly the functions of each unit. **(AMIE, S '96, W '97)**
8. Differentiate the following:
   (*a*) General purpose *vs* Special purpose computer (*b*) Digital *vs* Analog computer.
9. What are the major characteristics of a computer? Explain briefly each of them.
10. Write short notes on:
    (*a*) Application software, and (*b*) System software.
11. Name the major elements and state their functions in a general purpose digital computer. With a neat sketch, indicate how these elements are connected. **(AMIE, W '93)**
12. How do you classify the digital computers according to their applications?
13. How the programming languages are classified according to applications? Give salient features of each application. **(AMIE, W '97)**

2

HISTORY, GENERATIONS AND CLASSIFICATION OF COMPUTERS

2.0. INTRODUCTION

This chapter deals with historical development of computers, computer generations which are characterised by hardware and software technology and the classification of computers as micro, mini, mainframe and supercomputer.

2.1. HISTORY OF COMPUTING

Although the abacus is not a computer, the history of computing really began with this device. It was used in China and Japan for thousands of years before Christ. The abacus is a manual device combining two fundamental concepts. First, numerical information can be represented in a physical form. Second, this information can be manipulated in the physical form to produce the required result.

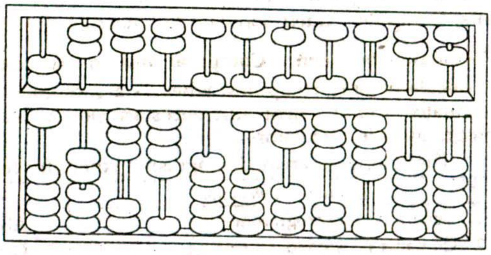


Fig. 2.1: Abacus

The abacus is essentially a collection of beads strung on parallel rods fixed in a frame. There are two portions. The beads in the upper portion count five each and those in the lower portion count one each. Arithmetic calculations are performed by manipulating those beads.

2.1.1. Mechanical Calculators

The first machine to add numbers mechanically was invented by Pascal, the French mathematician and philosopher in 1643. His machine consisted mainly of a row of toothed wheels. These teeth were numbered from 0 to 9. The machine could add eight columns of numbers. It sets a milestone in the development of computers. Later in the same century German mathematician Leibnitz added the facility of multiplication and division as well.

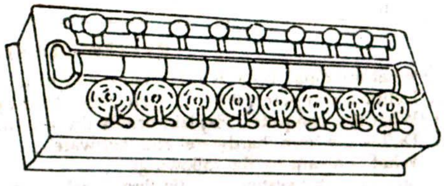


Fig. 2.2: Mechanical calculator

2.1.2. Charles Babbage—His Engines

Charles Babbage, a Professor of Mathematics at Cambridge University, England, attempted in 1812 to build a difference engine, a machine that could add, subtract, multiply, divide and perform a sequence of steps automatically. Babbage called his machine a difference engine because he attempted to use it to compute mathematical tables by adding differences.

Babbage failed to get the necessary fund for his machine and in 1833 the project was dropped. Babbage was also thinking of making a analytical engine with store (memory), mill (arithmetic) and sequence of machanism (control). He did not gain necessary fund for this project even though his concepts were sound in every respect. It is generally stated that the technology in Babbage time just did not permit the development of instruments with the precision required by his analytical engine.

2.1.3. Punched Card Machine

In weaving, the chief problem weavers face was the control of a number of shuttles for creating designs. The whole operation was cumbersome and expensive. Jacquard devised a method. He took a card and punched holes in it wherever the shuttles had to go through. He punched different sets of holes on different cards which resulted in producing different patterns.

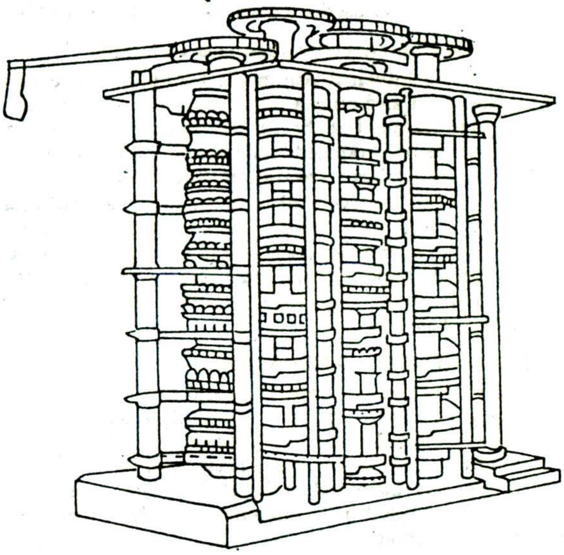


Fig. 2.3 : Babbage's difference engine

A major development occurred in 1886 when Herman Hollerith devised a system based on the principle of punching holes into cards, similar to Jacquard's idea. Hollerith had the idea that these holes could be sensed by a machine, a new way to handle large volume of data. Jacquard and Charles Babbage had used punched cards and operated them by mechanical devices. The first card machine which was electrically activated was used by Hollerith to compute the statistics of the 1890 United States census. Till 1960s the punched card system was the chief mode of processing data.

2.1.4. First Digital Computer

In 1937, Howard A. Aiken of Harvard University began work on the design of fully automatic calculating machine using the concepts of Babbage and punch cards in collaboration with the IBM (International Business Machine). Seven years later in January 1944 the design became a reality and was named MARK I. This was considered to be the first digital computer. MARK I could accept data from punch cards, store them in memory, make calculations by means of automatically controlled electromagnetic relays and arithmetic counters which were mechanical. It could be programmed, *i.e.*, given a set of commands to carry out certain operations. It performed arithmetic and logical operations and solved scientific problems.

2.1.5. First Electronic Computer

ENIAC (Electronic Numerical Integrator and Calculator): The innovation of very high speed vacuum tube, a built in device, led to the first all electronic computer in the year 1947. It contains vacuum tubes, registers, capacitors and switches and it was much faster than the MARK I.

2.1.6. First Computer to use Stored Program

EDSAC (Electronic Delayed Storage Automatic Computer): Earlier machines could be programmed but the idea of storing instructions in the computer memory was not there. Von Neumann, referred to as the father of modern computers, was the first to introduce the concept of stored program around the same time. EDSAC is the first computer to use the stored program concept and was designed and completed in 1949 at the Cambridge University, England. The program was set into the storage unit by means of paper tape. EDSAC also used vacuum tubes and was a little faster than the ENIAC.

2.1.7. First Commercially Produced Computer

UNIVAC I (Universal Automatic Computer): The first computer to use magnetic tape for data input and output was UNIVAC I. It was built in 1946. The first UNIVAC computer was installed at the United States Bureau of the Censor in 1951. It was the first computer to be produced commercially. It could process numeric as well as alphabetic data. Vacuum tubes were used in this system also.

After the discovery of the transistor, vacuum tubes were replaced by smaller size transistors. Many drawbacks of the vacuum tubes were eliminated with the use of transistors. With further advancement in electronics technology, transistors have been replaced by extremely superior solid state devices. The fabrication of large number of circuits on very small silicon chips has led to the reduction in the size, cost of computers and enhancement of speed.

2.2. COMPUTER GENERATIONS

Electronic Numerical Integrator and Calculator (ENIAC) was the first general purpose electronic digital computer and was developed in 1946. Today it has become antique as science has developed to such an extent that we have a pocket computer which is more powerful in comparison. As efficiency in terms of speed, storage capacity and reliability of computers increased with time; the size, computing time and cost decreased. This growth is divided into different generations which are characterised by hardware and software technology.

2.2.1. First Generation (1942-55) (The era of the vacuum tube)

The first generation of computers was marked by the use of vacuum tubes and by the use of either electrostatic tubes (CRT) or mercury delay lines for storage. Punched card and punched paper tape were used for input and output of data. These include UNIVAC I, ENIAC, EDSAC, IBM 650 and 701. On the first generation machines, programs were written in machine language, and read into the computers memory as a stored program. Disadvantages with the use of vacuum tubes are: (*i*) size of the computer becomes large, thereby lot of space is required for their storage and is non-portable, (*ii*) constant maintenance is required as lifetime of tube generates maximum amount of heat thus air conditioning is required, (*iii*) switching time of the tubes is very high and the speed of the computer is slow.

2.2.2. Second Generation (1956-65)

The second generation machines were initially marked by either magnetic drum or magnetic (ferrite) core storage and later by the use of the transistor (a small piece of germanium metal suitably duped with impurities) in place of vacuum tubes. The transistor performs the same function as the vacuum tube but smaller, less expensive, generates almost no heat and requires little power. Second generation computers were substantially reduced in size, required less power and were more reliable. Examples are IBM 1401, Borroughs B 5000, CDC 1604.

People began to realise that the software was going to play a major role in computing. Standardized high level languages such as COBOL, FORTRAN and AIGOL were developed to take care of the rapid growth in the number of applications that were being computerised. With higher speed CPUs and the advent of magnetic tape and disk storage, operating systems were developed. Computing in real time (applications that require a response within a short period of time) was starting to become popular.

2.2.3. Third Generation (1966-75)

The third generation computers are characterized by miniaturised circuits, the integration of hardware with software and an orientation of data communication and the handling of more than one operation simultaneously. The speed is faster, prices are lower, *e.g.*, ICL 1900 series, system 4, IBM 360, Honeywell 6000. The transistors were replaced by small integrated circuits (ICs). Instead of having one transistor of its own, several transistors along with other components integrated on a single silicon chip are known as integrated circuits (ICs). These IC-based systems were more economical and powerful in all respect. The effect of increasing switching speed of transistors, the reliability, the reduction of power dissipation and size were the emergence of extremely powerful CPUs with the capacity of carrying out one million instructions per second. Time sharing became the buzzword of the third generation. The combination of hardware and software allows a central computer to serve many users at what appears to be at the same time.

In high level languages, improved FORTRAN IV was developed, COBOL 68 was standardized by the American National Standard Institute and PL/I of IBM was emerged as a powerful language.

2.2.4. Fourth Generation (1975 onwards)

After development of ICs, they were further integrated to form Large-scale Integration (LSI) and Very Lárge-Scale Integration (VLSI) and resulted in the development of microprocessors. The latest microprocessor of Intel's 8086 family, 1586 contains more than three million transistors. Microprocessor-based computers are characterized by their smaller size, lower cost, larger memory and faster speed. The computers of today are members of the fourth generation. Some of the computers which belong to this generation are PARAM, PC-AT 486, MIGBTY-FRAME 1, MACINTOSH, CRAY XMP 14, etc.

Packaged software-programs like electronic spread-sheets and data management packages that are already written, tested and able to be purchased from the shelf in retail computer stores is creating an independent software industry. Another important development is interactive graphic devices and language interfaces to graphic systems. The emergence of graphics has given a great impetus to computer-aided engineering design.

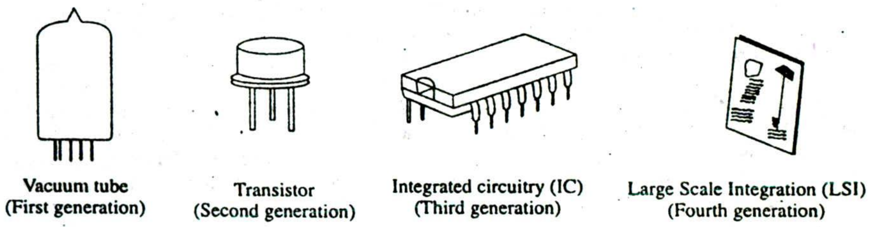


Fig. 2.4: Electronic devices of different generations

Table 2.1: Comparison of Computer Generations

S. No.	*Generations*	*I (1942-55)*	*II (1956-65)*	*III (1966-75)*	*IV (1975 Onwards)*
1.	Technologies	Vacuum tubes: acoustic memories, CRT memories	Transistors, ferrite cores magnetic disks	Integrated circuits (ICs)	LSI circuits and VLSI circuits, semiconductor memories
2.	Software	Machine Language (low-level), Assembly Language	COBOL, ALGOL FORTRAN (high-level language), Batch operating system	FORTRAN-IV COBOL-68, PL/I Time shared operating system	PASCAL, FORTRAN 77 ADA, COBOL 74
3.	Number of users	1	1	Multiple user community	Remote users
4.	Nature of inputs	Torn tapes	Punched cards, Magnetic tape, Disks, etc.	Cards, Magnetic tapes, Disks	VDUs, Floppies, Optical Disks
5.	Nature of Processing	Serial	Serial	Serial	Serial/Parallel
6.	Execution Speed measure	Milliseconds (thousandth of a second)	Microseconds (Millionth of a second)	Nanoseconds (billionth of a second)	Picoseconds (trillionth of a second)
7.	Represen-tative Computers	UNIVAC I, IBM 650, ENIAC	IBM 7094 CDC 1604	IBM 360 DECPOP-8/11 Burroughs 6700/7700 CDC 6000/7000	IBM 370 CRAY XMP 14 PC-AT 486 ETA-10
8.	Applications	Mostly scientific, simple business application	Extensive business application, Scientific research	Database management systems, on line system	Personal computer, Integrated CAD/CAM, Distributed system

2.2.5. Fifth Generation

The fifth generation computers are under development. Japan and USA have undertaken to design and develop such computers. It appears that the fifth generation computers will have at least three important characteristics: (*i*) mega-chip memories, (*ii*) the ability to extensive use of parallel processing, and (*iii*) artificial intelligence. The design approach of the CPU of these computers will be conceptually different from that of the earlier four generations of computers of Von Neuman architecture in which processor executes simple instructions in sequence. In new design, processing units may not be centralized but distributed in the computer system. All data may not be stored in the main memory. The data may flow through the processing units activating each of them as needed. These computers will be knowledge-based and will be used for Information Management, Natural Language Processing, Speech, Character and Image Recognition and such other artificial intelligence applications.

2.3. CLASSIFICATION OF COMPUTERS

Based on the size of primary storage, the capabilities in terms of processing speed, the range of applications and the number and type of peripheral devices, the computers may be divided into following different classes:

1. Micro Computers
2. Mini Computers
3. Mainframe Computers
4. Supercomputers

2.3.1. Micro Computers

It is the smallest and cheapest category of digital computers. It is called micro because of its miniature size and using of a microprocessor. A micro computer consists of a main microprocessor (a CPU on a chip), several support microprocessors and associated control, primary storage and a variety of input/output and secondary storage devices.

Computers of this category are supported by single user operating system. This category is further sub-divided into (*a*) Home computers, and (*b*) Personal computers.

(*a*) Home computers are used for entertainment, education, training and for home management. The word length of these computers is 8 bit. They have a keyboard integrated with the CPU in one box which is interfaced with an ordinary colour television to act as the VDU and an audio cassette recorder to act as the secondary storage device. Examples are TRS-80, UNICORN.

(*b*) Personal computers (PCs) are meant for professionals, small business units and for office automation systems. The following are three categories of PC:

(*i*) PC—This uses 8086 or 8088 microprocessor.

(*ii*) PC-XT—This is a PC with an extended technology. It uses 8088 microprocessor and a fixed inbuilt disk known as hard disk. PC and PC-XT are single user system.

(*iii*) PC-AT—This is a PC with an advanced technology and it uses 80286, 80386, 80486 or 1586 microprocessor and hard disk drive. Its CPU is powerful than PC-XT. Four terminals can be connected to make it multiuser. The following are the major features of these computers:

CPU speed	:	500 KIPS
Word length	:	8 to 16/32 bits
Storage capacity	:	256 KB to 16 MB
Secondary storage	:	Floppy disk, hard disk, magnetic tape, optical disk
Input/Output devices	:	Keyboard, electronic mouse, light pen, optical and voice input devices, video display monitors and printers are the most widely used output devices
Major areas of applications	:	Word processing, database management, accounting and financial analysis, engineering and scientific applications
Commonly used machines	:	IBM-PC, Apple, BBC Micro, Radio Shak TRS-80, etc.

2.3.2. Mini Computers

The mini computers are slightly bigger in size, memory and speed compared to microcomputers. Mini computers are multi-user systems. This means that more than one user can use the computer system at the same time.

The major features of these computer are given below:

CPU speed	:	10 to 30 MIPS

Word length	:	16 to 32 bits
Storage capacity	:	8 MB to 96 MB
Input/Output devices	:	Winchester hard disk, magnetic tapes, high speed line printers, plotters, etc.
Major areas of applications	:	Process control in industries, high-performance workstations with graphics input/output display, engineering and scientific research
Commonly used machines	:	PDP 11, HP 2000, IBM SYS/3, etc.

2.3.3. Mainframe Computers

Mainframe computers are larger than micros and minis, and usually have one or more central processors. Normally, mainframe manufacturers produce families of computers with models ranging in size from small to very large. Most models in family are compatible, *i.e.*, programs written for one model can be run on other. The major features of these computers are given below:

CPU speed	:	30 to 100 MIPS
Word length	:	32 to 64 bits
Storage capacity	:	8 MB to 256 MB
Input/Output devices	:	1000 MB to 10 GB hard disks, high speed magnetic tapes, fast line printer, laser printer, mini computers as front end processors
Major areas of applications	:	Research organisations, large industries and government organisations, large-scale on line reservation systems
Commonly used machines	:	IBM 4300 series, HP 9000 model, CYBER-170, BORROUGHS 7800

2.3.4. Super Computers

Supercomputers are very big in memory size, perform billion of operations per second. The high speed in these computers is due to use of a number of processors working in parallel and high storage densities are obtained by using magnetic bubble memories.

The main features of these computers are given below:

CPU speed	:	400 MIPS to 10,000 MIPS
Word length	:	64 bits to 96 bits
Storage capacity	:	256 MB and more
Input/Output devices	:	Very high speed 1000 MB hard disks, very high speed tapes, large size computers as front end processors, high speed laser printers
Major areas of applications	:	Defence research, weather forecasting, space research, etc.
Commonly used machines	:	CRAY XMP-14, CDC Cyber 205 family, CRAY YMP series, ETA 10, etc.

REVIEW QUESTIONS SET

1. What do the following names stand for : ENIAC, EDSAC, UNIVAC.
2. What is meant by the term generation in computer technology? How many computer generations are there till now? Explain each generation briefly.
3. Explain the disadvantages of using vacuum tubes. How have they been overcome?
4. How are third generation computers superior to second generation computers?

5. How do you differentiate between:
   (*a*) Micro
   (*b*) Mini
   (*c*) Mainframe computer?
   Write their field of applications.
6. What are the characteristics of a microcomputer?
7. Write short notes on :
   (*a*) Mini Computers
   (*b*) Mainframe Computers
   (*c*) Supercomputers
8. Write a short note on Personal Computer.

3

NUMBER SYSTEMS

3.0. INTRODUCTION

The knowledge of number systems is essential because the design and organisation of a computer is dependent upon the number systems. The binary system has proven the most natural and efficient system for computer use. This chapter describes number systems used in computer technology.

3.1. NUMBER SYSTEMS

A number system consists of a set of symbols and rules for representing any number. There are mainly two types of number systems: (*i*) Non-positional systems, and (*ii*) Positional systems.

3.1.1. Non-positional Systems

In the non-positional systems, the characters used are I for 1, II for 2, etc., and are of positional invariant, *i.e.*, each symbol represents the same value regardless of its position in the number. Since it is very difficult to perform arithmetic calculations with such a number system, positional number systems were developed.

3.1.2. Positional Systems

A number system with a specified number as the base is called a positional number system. Any positional number system requires only a finite number of symbols, called the digits, of the system to represent arbitrarily large number. The total number of digits in the system is called its base or radix. The value of each digit in such a number is determined by

(*a*) the digit itself,

(*b*) the position of the digit in the number, and

(*c*) the base of the number system.

In general, the positional and polynomial form of a number in fixed point form is

$$N_r = (\underbrace{\overset{\text{MSD}}{a_{n-1}} \cdots\cdots a_3 a_2 a_1 a_0}_{\text{Integer}} \overset{\text{Radix Point}}{\bullet} \underbrace{a_{-1}\, a_{-2}\, a_{-3} \cdots\cdots \overset{\text{LSD}}{a_{-m}}}_{\text{Fraction}})_r \qquad \ldots(1)$$

where the radix (or base) r is the total number of digits in the number system and a is a digit in the set defined for radix r. Here the radix point separates n integer digits on the left from m fraction digits on the right. Notice that a_{n-1} is the most significant (highest order) digit, called MSD and that a_{-m} is the least significant (lowest order) digit, denoted by LSD.

The value of the number in Eq. (1) is given in polynomial form by

$$N_r = \sum_{i=-m}^{n-1} a_i r^i \qquad ...(2)$$

$$= a_{n-1} r^{n-1} + \ldots\ldots + a_2 r^2 + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1}, + a_{-2} r^{-2} + \ldots\ldots\ldots\ldots + a_{-m} r^{-m}$$

where a_i is the digit in the *i*th position with a weight r^i.

Application of Eqs. (1) and (2) follows directly. For the decimal system $r = 10$ indicates that there are 10 distinguishable characters. Consider a number 4625.735 in decimal system. The number can be expressed as

$$(4625.735)_{10} = 4 \times 10^3 + 6 \times 10^2 + 2 \times 10^1 + 5 \times 10^0 + 7 \times 10^{-1} + 3 \times 10^{-2} + 5 \times 10^{-3}$$

where 10^3, 10^2, 10^1, 10^0, 10^{-1}, 10^{-2} and 10^{-3} are the place values of the digits from left. MSD and LSD for this number are 4 and 5, respectively. In order to identify the system, in which the numbers, is written, we write the base or radix along with the number. Thus, $(4625.735)_{10}$ is written in decimal system as its base is 10.

The same number in octal system (base 8) can be expressed in decimal equivalent as

$$(4625.735)_8 = 4 \times 8^3 + 6 \times 8^2 + 2 \times 8^1 + 5 \times 8^0 + 7 \times 8^{-1} + 3 \times 8^{-2} + 5 \times 8^{-3}$$

Here 8^3, 8^2, 8^1, 8^0, 8^{-1}, 8^{-2} and 8^{-3} are the place values of the various position of the digits.

The four number systems that are commonly used are shown in Table 3.1.

Table 3.1

Number system	*Base/Radix*	*Digit for the number system*
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Haxadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 A, B, C, D, E, F

3.1.2.1. Binary System

The binary number system, as the name suggests, consists of only two digits, namely 0 and 1. Since the system uses only two digits, its base is 2 and all the numbers in the system are written as a string of 0's and 1's. The binary digits 0 and 1 are generally referred to the common abbreviation bit. For example, a binary number is shown below:

$$\underset{\substack{\uparrow \\ \text{Most} \\ \text{significant} \\ \text{bit}}}{1} \quad 1 \quad . \quad 1 \quad 0 \quad \underset{\substack{\uparrow \\ \text{Least} \\ \text{significant} \\ \text{bit}}}{1}$$

The right most bit is called the least significant bit and the left most bit, the most significant bit. The binary numbers are usually written with the base as a subscript in the form 1101_2 or $(1101)_2$.

The weights in the binary system are the powers of the base 2 just as the weights in the decimal system are the power of the base 10. The weights of the digits of the integral part are 2^0, 2^1, 2^2, . . . from right to left and the weights of the digits of the fractional part are 2^{-1}, 2^{-2}, 2^{-3} . . . from right of the decimal. Thus, the binary number 10101 can be expressed in decimal equivalent as

$$\begin{aligned}(10101)_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 16 + 0 + 4 + 0 + 1 \\ &= (21)_{10}\end{aligned}$$

3.1.2.2. Octal Number System

The octal number system consist of 8 digits : 0, 1, 2, 3, 4, 5, 6, 7. Since the system uses 8 digits, the base of the system is 8 and each position in an octal number represents a power of the base (8). Thus, the octal number 7312 can be expressed in decimal equivalent as

$$\begin{aligned}(7312)_8 &= 7 \times 8^3 + 3 \times 8^2 + 1 \times 8^1 + 2 \times 8^0 \\ &= 7 \times 512 + 3 \times 64 + 1 \times 8 + 2 \times 1 \\ &= 3584 + 192 + 8 + 2 \\ &= (3786)_{10}\end{aligned}$$

3.1.2.3. Hexadecimal Number System

The hexadecimal number system consists of 16 single character digits or symbols, where A, B, C, D, E, F represents the decimal value 10, 11, 12, 13, 14, 15, respectively. Thus, the hexadecimal number D_3E_0 can be expressed in decimal equivalent as

$$\begin{aligned}(D3E0)_{16} &= D \times 16^3 + 3 \times 16^2 + E \times 16^1 + 0 \times 16^0 \\ &= 13 \times 4096 + 3 \times 256 + 14 \times 16 + 0 \times 1 \\ &= 53248 + 768 + 224 \\ &= (54240)_{10}\end{aligned}$$

Table 3.2 shows the first 20 digits of some number systems.

3.2. CONVERSION FROM ONE NUMBER SYSTEM TO OTHER

It is just not sufficient to represent numbers in various number systems but also there should be flexibility to convert numbers from one system to another. There are different methods to serve this purpose. Various methods of conversions are described here.

3.2.1. Conversion of Any Number System to Decimal System

A number represented in any number system can be converted in any other system. In the field of computer the input and output values are in decimal. Computer professionals are often required to convert number in other systems to decimal and *vice-versa*. A number represented in any number system can be converted to equivalent decimal number system by a method known as Polynomial Evaluation method.

Table 3.2: First Twenty Digits of Some Number System

Radix 10 (Decimal)	*Radix 2 (Binary)*	*Radix 3 (Ternary)*	*Radix 4 (Quarternary)*	*Radix 5 (Quintal)*	*Radix 8 (Octal)*	*Radix 16 (Hexadecimal)*
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	10	2	2	2	2	2
3	11	10	3	3	3	3
4	100	11	10	4	4	4
5	101	12	11	10	5	5
6	110	20	12	11	6	6
7	111	21	13	12	7	7
8	1000	22	20	13	10	8
9	1001	100	21	14	11	9
10	1010	101	22	20	12	A
11	1011	102	23	21	13	B
12	1100	110	30	22	14	C

(Contd.)

13	1101	111	31	23	15	D
14	1110	112	32	24	16	E
15	1111	120	33	30	17	F
16	10000	121	100	31	20	10
17	10001	122	101	32	21	11
18	10010	200	102	33	22	12
19	10011	201	103	34	23	13

In this method, each digit within a number is multiplied by the weight of its position and then all these values are added. The following examples will illustrate the method.

Example 3.1. *Convert the following binary numbers into equivalent decimal:*

(i) $(10101101)_2$ *(ii)* $(11010.111)_2$

Solution. (*i*)
$$(10101101)_2 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 128 + 0 + 32 + 0 + 8 + 4 + 0 + 1 = (173)_{10}$$

(*ii*)
$$(11010.111)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$
$$= 16 + 8 + 0 + 2 + 0 + 0.5 + 0.25 + 0.125 = (26.875)_{10}$$

Example 3.2. *Convert decimal equivalent of* $(736.5)_8$ *and* $(3FA.8)_{16}$. **[AMIE, S '94]**

Solution.
$$(736.5)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 5 \times 8^{-1}$$
$$= 448 + 24 + 6 + 0.625 = (478.625)_{10}$$
$$(3\,FA.8)_{16} = 3 \times 16^2 + F \times 16^1 + A \times 16^0 + 8 \times 16^{-1}$$
$$= 768 + 15 \times 16 + 10 \times 1 + 0.5$$
$$= 768 + 240 + 10 + 0.5 = (1018.5)_{10}$$

3.2.2. Conversion of Decimal System to Any Other Number System

The conversion from decimal to other number system may be done by a method known as Dibble-Dabble method. In this method, the integer and fraction parts of the number are converted separately.

3.2.2.1. Conversion of Integer Part

In order to convert integer part of a decimal number to an equivalent number of other system of base *b*, divide the number (integer part) and each succeeding quotient by *b* until a zero quotient is obtained. The sequence of remainders, in reverse order, yields the base *b* representation of the number, *i.e.*, the least significant digit is the first remainder obtained and its most significant digit is the last remainder.

3.2.2.2. Conversion of Fractional Part

In order to convert fractional part of a decimal number to an equivalent number of other system of base *b*, multiply the number (fractional part) successively by base *b*. In each step the integer part obtained after multiplication is noted separately and the new fractional part is again used for new multiplication. The process continues until a zero fractional part or a duplicate fractional part or sufficient number of digits have been obtained. Then the sequence of integer parts of the products gives the base *b* representation of the fractional part. Then the first integer is the MSD and the last integer is the LSD of the fractional part of the convert number. The following examples will illustrate the method:

Example 3.3. *Convert (a) the decimal number 41.6875 into a binary number, (b) the decimal number 153.513 into an octal number.* **[AMIE, S '96]**

Solution. (*a*) The given number is $(41.6875)_{10}$.

Here integer part is 41 and fractional part .6875

For integer part,

Base	Number		Remainder	
2	41			
2	20	—	1	LSD
2	10	—	0	
2	5	—	0	
2	2	—	1	
2	1	—	0	
	0	—	1	MSD

$(41)_{10} = (101001)_2$

For fractional part,

Number		*Base*		*Product*	*Fractional Part*	*Integer Part*	
0.6875	×	2	=	1.3750	.3750	1	MSD
0.3750	×	2	=	0.7500	.7500	0	
0.7500	×	2	=	1.5000	.5000	1	
0.5000	×	2	=	1.0000	.0000	1	LSD

$(.6875)_{10} = (.1011)_2$

Hence, $(41.6875)_{10} = (101001.1011)_2$

(*b*) The given number is $(153.513)_{10}$

Here integer part is 153 and fractional part is .513

For integer part,

Base	Number		Remainder	
8	153			
8	19	—	1	LSD
8	2	—	3	
	0	—	2	MSD

$(153)_{10} = (231)_8$

For fractional part,

Number		*Base*		*Product*	*Fractional Part*	*Integer Part*	
0.513	×	8	=	4.104	.104	4	MSD
0.104	×	8	=	0.832	.832	0	
0.832	×	8	=	6.656	.656	6	
0.656	×	8	=	5.248	.248	5	
⋮				⋮	⋮	⋮	
		$(.531)_{10}$	=	$(.4065\ldots)_8$			

Hence, $(153.531)_{10} = (231.4065\ldots)_8$.

***Example 3.4.** Convert $(0.2)_{10}$ to its octal form.*

Solution.

Number		*Base*		*Product*	*Fractional Part*	*Integer Part*	
0.2	×	8	=	1.6	.6	1	MSD
0.6	×	8	=	4.8	.8	4	
0.8	×	8	=	6.4	.4	6	
0.4	×	8	=	3.2	.2	3	
0.2	×	8	=	1.6	.6	1	

At the fourth step we again obtain .2 as the fractional part, hence the digits 1463 will repeat, giving $(0.2)_{10} = (0.1463\ 1463\ ....)_8$.

Example 3.5. *Convert* $(429.32)_{10}$ *into haxadecimal.*

Solution. The given number is $(429.32)_{10}$.

Here integer part is 429 and fractional part .32

For integer part,

Base	Number		Remainder	
16	429			
16	26	—	D	LSD
16	1	—	A	
	0	—	1	MSD

For fractional part,

Number		*Base*		*Product*	*Fractional Part*	*Integer Part*	
.32	×	16	=	5.12	.12	5	MSD
.12	×	16	=	1.92	.92	1	
.92	×	16	=	14.52	.52 Up to third place	E	

Hence, $(429.32)_{10} = (1\ A\ D.\ 5\ 1\ E)_{16}$ up to third place.

3.3. DIRECT METHOD OF CONVERTING THE NUMBER FROM ONE SYSTEM TO OTHER

Whenever the radix of one system can be expressed in powers of the radix of other system, the system can be converted directly by forming pair of numbers 2, 3 or 4 noting the power of the radix.

3.3.1. Conversion from Octal to Binary

The octal system is a system having base 8. Since $8 = 2^3$, each octal digit is a union of 3 bit representation. An octal number can be converted to binary number by converting each octal digit to its binary equivalent with 3 bits. The example below illustrate the method.

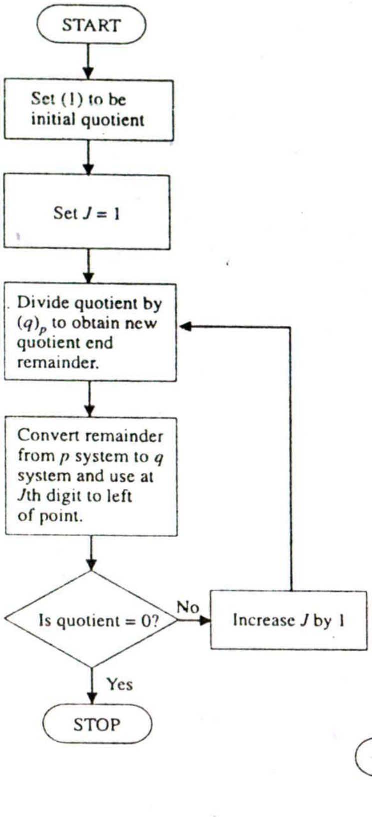
Example 3.6. *Convert to binary form (a)* $(617025)_8$ *,(b)* $(43.0276)_8$

Solution. (*a*) $(617025)_8 = (\ \underline{110}\ \ \underline{001}\ \ \underline{111}\ \ \underline{000}\ \ \underline{010}\ \ \underline{101}\ )_2$

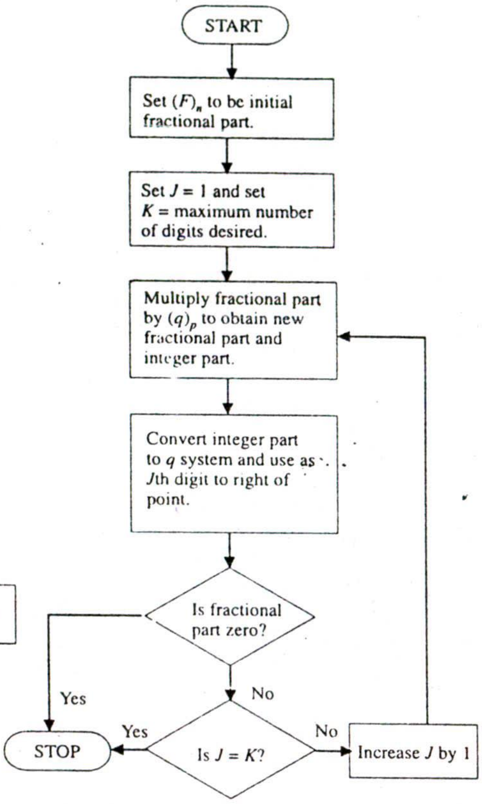
(*b*) $(43.0276)_8 = (\ \underline{100}\ \ \underline{011}\ .\ \underline{000}\ \ \underline{010}\ \ \underline{111}\ \ \underline{110}\ )_2$

3.3.2. Conversion from Binary to Octal

A binary number can be converted to octal number by partitioning the number into 3 bit groups formed from left to right for the fractional part of the number and from right


START
Set (I) to be initial quotient
Set J = 1
Divide quotient by (q)p to obtain new quotient end remainder.
Convert remainder from p system to q system and use at Jth digit to left of point.
Is quotient = 0?
No
Increase J by 1
Yes
STOP

(*a*) Integers


START
Set (F)n to be initial fractional part.
Set J = 1 and set K = maximum number of digits desired.
Multiply fractional part by (q)p to obtain new fractional part and integer part.
Convert integer part to q system and use as Jth digit to right of point.
Is fractional part zero?
Yes
No
STOP
Yes
Is J = K?
No
Increase J by 1

(*b*) Fraction

Fig. 3.1: Flow chart of radix conversion

to left for the integer part. If the number of bits in the integer part is not a multiple of 3, we insert leading 0s, as leading 0s have no significance for the integer part. If the number of bits in the fractional part is not a multiple of 3, then we introduce trailing 0s, as trailing 0s have no significance in the fractional part.

The following examples will illustrate the procedure.

Example 3.6. *Convert $(1101010)_2$ into octal*

Solution. $(1101010)_2 = \underline{001}\ \ \underline{101}\ \ \underline{010}$

(Group three digits from right)

$= (152)_8$

(Convert each group to an octal digit)

Example 3.7. *Convert $(11\ \ 11\ \ 0101\ \ \ 111\ .\ 10001)_2$ into octal.*

Solution. $(11110101111.10001)_2 =$ 011 110 101 111 . 100 010

(Group three digits from left of the binary point) (Group three digits from right of the binary point)

$= (3657.42)_8$
(Convert each group to an octal digit)

3.3.3. Conversion from Hexadecimal to Binary

The hexadecimal system is a system having base 16. Since $16 = 2^4$, each digit is an unique 4 bit representation. For conversion from hexadecimal to binary, each digit of hexadecimal number is to be replaced by its binary equivalent with 4 bits.

Example 3.8. *Convert $(2\ AB)_{16}$ into binary.*

Solution. $(2AB)_{16} = \underset{2}{\underline{0010}}\quad \underset{A}{\underline{1010}}\quad \underset{B}{\underline{1011}} = (0010101011)_2$

3.3.4. Conversion from Binary to Hexadecimal

A binary number can be converted to hexadecimal number by partitioning the number into 4 bit groups starting from left to right for the fractional part of the number and from right to left for the integer part. If the number of bits in the integer part is not a multiple of 4, we insert leading 0s and if the number of bits in the fractional part is not a multiple of 4, then we introduce trailing 0s.

The following examples illustrate the method.

Example 3.9. *Convert $(11011111)_2$ into hexadecimal system.*

Solution. $(11011111)_2 = \underline{1101}\quad \underline{1111}$
(Group four digits from right)
$= DF$
(Convert each group to hexadecimal digits)

Hence, $(11011111)_2 = (DF)_{16}$

3.3.5. Conversion from Hexadecimal to Octal and *vice-versa*

For the conversion of number from hexadecimal system to octal system, hexadecimal number is first converted to binary and then binary number to octal and for the conversion from octal to hexadecimal, convert the given octal number to binary and then binary to hexadecimal.

The following examples will illustrate the procedure.

Example 3.10. *Convert $(IE.C)_{16}$ to equivalent octal system.*

Solution. $(IE.C)_{16} = \underline{0001}\quad \underline{1110}\ .\ \underline{1100}$
$= (00011110.1100)_2$
$= \underline{011}\quad \underline{110}\quad \underline{110}$ (grouping into three)
$= (36.6)_8$

The octal equivalent of $(IE.C)_{16}$ is $(36.6)_8$.

Example 3.11. *Convert $(46.57)_8$ to its equivalent hexadecimal number.*

Solution. $(46.57)_8 = \underline{100}\ \ \underline{110}\ .\ \underline{101}\ \ .\ \underline{111}$
$= (100110.101111)_2$
$= 0010\ \ 0110\ .\ 1011\ \ 1100$ (grouping into four)
$= (26.BC)_{16}$

3.4. BINARY ARITHMETIC

Processors of computers perform arithmetic operations only on binary numbers. We should thus know how four basic operations are performed using binary numbers.

3.4.1. Addition

The rules of binary addition are

$$\begin{array}{rl}
0 + 0 = 0 & \\
0 + 1 = 1 & \\
1 + 0 = 1 & \\
1 + 1 = 0 & \text{with a carry of 1} \\
1 + 1 + 1 = 1 & \text{with a carry of 1}
\end{array}$$

Carryovers are performed in the same manner as in decimal arithmetic. Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be carried over. The exact procedure is illustrated with examples.

Example 3.12. *Perform* $(101)_2 + (001)_2$

Solution.

$$\begin{array}{lcccc}
\text{Carry} & & & 1 & \\
 & & 1 & 0 & 1 \\
 & + & 0 & 0 & 1 \\
\hline
 & & 1 & 1 & 0
\end{array}$$

Example 3.13. *Perform* $(11011)_2 + (101011)_2$

Solution.

$$\begin{array}{lccccccc}
\text{Carry} & & 1 & 1 & & 1 & 1 & \\
 & & & 1 & 1 & 0 & 1 & 1 \\
 & + & 1 & 0 & 1 & 0 & 1 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}$$

3.4.2. Subtraction

The rules for binary subtraction are

$$\begin{array}{rl}
0 - 0 = 0 & \\
1 - 0 = 1 & \\
1 - 1 = 0 & \\
0 - 1 = 1 & \text{with a borrow of 1 from the next column to the left} \\
 & \text{(value borrowed is equal to 10).}
\end{array}$$

Note that if the lower digit is larger than the upper digit, it is necessary to borrow from the column to the left, the value borrowed depends upon the base of the number and is always the decimal equivalent of the base, i.e., for binary system it is $2(10_2)$.

The exact procedure is illustrated with examples.

Example 3.14. *Perform* $(10110)_2 - (01001)_2$. **[AMIE, W '93]**

Solution.

$$\begin{array}{cccccc}
 & 0 & & & 0 & \\
 & 1 & 0 & 1 & 1 & 0 \\
- & 0 & 1 & 0 & 0 & 1 \\
\hline
 & 0 & 1 & 1 & 0 & 1
\end{array}$$

Explanation : To explain the case when we cannot borrow 1 from the next column because the column contains 0, let us look the decimal difference.

$$\begin{array}{ccccccc}
 & 6 & 9 & 9 & 9 & & \\
 & 7 & 0 & 0 & 0 & 4 & 3 \\
- & 4 & 8 & 4 & 3 & 5 & 1 \\
\hline
 & 2 & 1 & 5 & 6 & 9 & 2
\end{array}$$

We have borrowed from the sixth column for the second column since the third, fourth and fifth columns contained zeros. After borrowing, the third, fourth and fifth columns contain $10 - 1 = 9$. The same thing happens in binary subtraction except that after borrowing the zero columns contain $10 - 1 = 1$.

Example 3.15. *Perform* $(110.001)_2 - (11.111)_2$

Solution.

```
   0 0 1 1
   1 1 0. 0 0 1
 -   1 1. 1 1 1
 ---------------
   0 1 0. 0 1 0
```

3.4.3. Multiplication

Multiplication is nothing but successive addition. Most of the computers perform multiplication operations in binary using additive approach. Multiplication in the binary system also follows the same general rules as for decimal multiplication.

The rules for binary multiplication are

$$0 \times 0 = 0$$
$$1 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 1 = 1$$

The following examples illustrate binary multiplication.

Example 3.16. *Perform* $(10110)_2 \times (1101)_2$.

Solution.

```
            1 0 1 1 0
          × 1 1 0 1
          ----------
            1 0 1 1 0
          0 0 0 0 0
        1 0 1 1 0
      1 0 1 1 0
    -----------------
    1 0 0 0 1 1 1 1 0
```

Hence, $10110_2 \times 1101_2 = 100011110_2$

3.4.4. Division

Division can be carried out by repetitive subtractions.

Rules for binary division are

$$\frac{0}{0} = \text{No meaning}$$
$$\frac{1}{0} = \text{No meaning}$$
$$\frac{0}{1} = 0$$
$$\frac{1}{1} = 1$$

Following examples illustrate binary division.

Example 3.17. *Divide* *(i)* $(1100010)_2$ *by* $(111)_2$

(ii) $(100001)_2$ *by* $(110)_2$

Solution.

```
(i) 111) 1100010 (1110          (ii) 110) 100001 (101.1
         111                              110
         -----                            ------
         01010                            001001
           111                               110
         -----                            ------
         00111                             00110
           111                               110
         -----                            ------
           000                               000
```

Ans. $(1110)_2$ **Ans.** $(101.1)_2$

Example 3.18. *Divide* $(111011)_2$ *by* $(0111)_2$.

Solution.

```
0111 ) 111011 ( 1000.011011
       0111
       ------
        01100
          111
       -------
          1010
           111
       --------
          01100
            111
       --------
           1010
            111
       --------
```

Ans. $1000.\overline{011}$

3.5. COMPLEMENTS

Subtraction of two numbers in a computer is complicated and expensive in terms of circuit specially for repeated borrowing from one column to another. Complements can be used to reduce from subtraction to addition. It avoids the possibility of repeated borrowing from one column to another and, hence, borrow circuits are eliminated, and cost is reduced.

There are two types of complements, the radixminus-one complement ard r dix complement. In decimal system, they are called nines complement and tens complement. The nines complement of a number is obtained by subtracting each digit of the number from 9 and 10s complement of the number is its nines complement plus one. In binary system, they are called 1's complement and 2's complement.

3.5.1. 1's Complement

The 1's complement of a binary number can be obtained by subtracting each digit of the number from 1 which is same as replacing 1 by 0 and 0 by 1. For example, 1's complement of 110101 is 001010.

3.5.2. 2's Complement

The 2's complement of a binary number can be obtained from either of the following procedure:

(*i*) First complement each bit of the number (i.e., replace 1 by 0 and 0 by 1) to get 1's complement. Add 1 to this number to get 2's complement. For example, consider the number 13 whose binary representation is 1101. 1's complement is 0010. Now, adding 1 to this number gives 0011 which is 2's complement representation for –13.

(*ii*) Scan the binary numbers from right to left and complement all bits appearing after the first appearance of 1. For example, 2's complement of $10\underset{\substack{\uparrow\\\text{First appearance of 1 from right to left.}}}{1}0$ is 0 1 1 0.

3.5.3. Subtraction using Complements

If the number of digits of integer part in subtrahend is less than the number of digits in minuend, add 0 on the left hand of subtrahend to make equal number of digits and for fractional part add 0 on the right of subtrahend.

3.5.3.1. Using 1's Complement

Procedure : (*a*) Complement the subtrahend (the number to be subtracted)

(*b*) Add the complement with minuend (from the number to be subtracted) (*c*) Delete the higher order carry, if any, and add to the sum obtained in (*b*). If there is no carry (when a larger number is subtracted from a smaller number), recomplement the sum and attach a negative sign to obtain the result.

Example 3.19. *Perform* $(1101)_2 - (110)_2$ *using one's complement.*

Solution.

$$\begin{array}{rl} 1\,1\,0\,1 & \text{Minuend} \\ +\,\underline{1\,0\,0\,1} & \text{One's complement of subtrahend} \\ (1)\,0\,1\,1\,0 & \\ \llcorner\!\longrightarrow\ \underline{+\,1} & \\ 0\,1\,1\,1 & \end{array}$$

Ans. $(0111)_2$

Example 3.20. *Subtract* $(100011)_2$ *from* $(010010)_2$ *using one's complement.*

Solution.

$$\begin{array}{rl} 0\,1\,0\,0\,1\,0 & \text{Minuend} \\ +\,\underline{0\,1\,1\,1\,0\,0} & \text{One's complement of subtrahend} \\ 1\,0\,1\,1\,1\,0 & \end{array}$$

As there is no carry, the minuend is smaller than the subtrahend. Thus, it is required to complement the sum and attach a negative sign to obtain the result.

Ans. $-(010001)_2$

3.5.3.2. Using 2's Complement

The procedure for subtraction using 2's complement is almost identical with 1's complement except that the higher order carry is ignored.

The following examples will illustrate the procedure.

Example 3.21. *Perform* $(11010101)_2 - (10011010)_2$ *using 2's complement.*

Solution.

$$\begin{array}{rl} \text{1's complement of subtrahend} & 0\,1\,1\,0\,0\,1\,0\,1 \\ & \underline{+\,1} \\ \text{2's complement of subtrahend} & 0\,1\,1\,0\,0\,1\,1\,0 \end{array}$$

$$\begin{array}{rl} 1\,1\,0\,1\,0\,1\,0\,1 & \text{Minuend} \\ +\,\underline{0\,1\,1\,0\,0\,1\,1\,0} & \text{2's complement of subtrahend} \\ (1)\,0\,0\,1\,1\,1\,0\,1\,1 & \\ \uparrow\qquad\qquad\qquad & \\ \text{Neglect carry}\qquad\quad & \end{array}$$

Ans. $(00111011)_2$

Example 3.22. *Subtract* $(01000)_2$ *from* $(01111)_2$

(*i*) *using 1's complement*

(*ii*) *using 2's complement* **[AMIE, W '93]**

Solution. (*i*) Complement of subtrahend 10111

$$\begin{array}{rl} 0\,1\,1\,1\,1 & \text{Minuend} \\ +\,\underline{1\,0\,1\,1\,1} & \text{1's complement of subtrahend} \\ (1)\,0\,0\,1\,1\,0 & \\ \llcorner\!\longrightarrow\ \underline{+\,1} & \\ 0\,0\,1\,1\,1 & \end{array}$$

Ans. $(00111)_2$

(*ii*) 2's complement of subtrahend is 10111 (1's complement) + 1 = 11000

$$\begin{array}{rl} 0\,1\,1\,1\,1 & \text{Minuend} \\ +\,\underline{1\,1\,0\,0\,0} & \text{2's complement of subtrahend} \end{array}$$

```
(1)0 0 1 1 1
 ↑
Neglect carry
```

Ans. $(00111)_2$

Example 3.23. *Perform* $(101)_2 - (11011)_2$.

Solution.

$$\begin{array}{rl} 1\;0\;1 & \text{Minuend} \\ +\;\underline{0\;0\;1\;0\;1} & \text{2's complement of subtrahend} \\ 0\;1\;0\;1\;0 & \end{array}$$

As there is no final carry (the minuend is smaller than the subtrahend), taking 2's complement of 01010 and assigning a –ve sign provides the answer.

Ans. $-(10110)_2$

3.6. COMPARISON BETWEEN 1'S AND 2'S COMPLEMENTS

A comparison between 1's and 2's complements reveals the advantages and disadvantages of each. The 1's complement has the advantage of being easier to implement by digital components since the only thing that must be done is to change 0's into 1's and 1's into 0's. The implementation of the 2's complement may be obtained in two ways: (*a*) by adding 1 to the least significant digit of the 1's complement, and (*b*) by scanning the number from right to left and complementing all bits appeared after the first appearance of 1. During subtraction of two numbers by complements, the 2's complement is advantageous in that only one arithmetic addition operation is required. The 1's complement requires two arithmetic addition when an end around carry occurs. The 1's complement has the additional disadvantage of possessing two arithmetic zeros, one with all 0's and one with all 1's which may complicate matters while 2's complement has only one zero representation.

3.7. REASONS FOR USE OF BINARY SYSTEM IN THE DESIGN OF COMPUTER

The reasons of using binary system by the computer are:

(*i*) Electronic components naturally operate in a binary mode. A core is magnetized in anticlockwise (0 state) or clockwise (1 state); a switch is either open (0) or closed (1); electrical pulses are either absent (0) or present (1).

(*ii*) With the use of only two states, the circuit design is simplified, cost is reduced and reliability is improved.

(*iii*) Finally, the binary system is used because everything that can be done with a base 10 can also be done in binary.

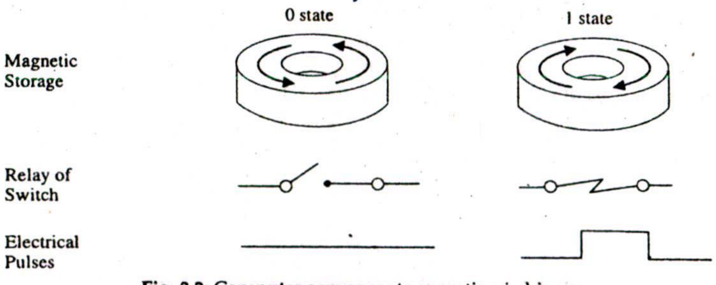


Fig. 3.2: Computer components operating in binary

3.8. ARITHMETIC IN BASE SYSTEM

Arithmetic in base system (other than decimal) is similar to decimal arithmetic. Four arithmetic operations with commonly used base system, other than two are discussed.

3.8.1. Addition

The sum of two numbers in *r* base/radix system can be obtained by (*i*) finding their decimal sum digit by digit, (*ii*) modifying the decimal sum if it exceeds or equal to radix by subtracting radix and carrying 1 to the next column.

Example 3.24. *Perform* $(67453)_8 + (35162)_8$

Solution. Carry

$$\begin{array}{cccccc l}
1 & 1 & & 1 & & & \\
 & 6 & 7 & 4 & 5 & 3 & \\
 & 3 & 5 & 1 & 6 & 2 & \\ \hline
 & 10 & 12 & 6 & 11 & 5 & \text{Decimal sum} \\
 & 8 - & 8 - & 0 - & 8 - & 0 & \text{Modification} \\ \hline
1 & 2 & 4 & 6 & 3 & 5 & \text{Octal sum}
\end{array}$$

Ans. $(124635)_8$

Example 3.25. *Perform* $(8\ 2\ C\ 5)_{16} + (9\ D\ 8\ 6)_{16}$

Solution. Carry

$$\begin{array}{ccccc l}
1 & 1 & 1 & & & \\
 & 8 & 2 & C & 5 & \\
 & 9 & D & 8 & 6 & \\ \hline
 & 18 & 16 & 20 & 11 & \text{Decimal sum} \\
 & -16 - & 16 - & 16 - & 0 & \text{Modification} \\ \hline
1 & 2 & 0 & 4 & B & \text{Hexadecimal sum}
\end{array}$$

Ans. $(1204B)_{16}$

3.8.2. Subtraction

Subtraction in base system can be performed by (*i*) using the borrowing method and the borrowing is in terms of *r*'s not in tens, (*ii*) using complements.

Example 3.26. *Perform* $(6214)_8 - (3527)_8$ *using borrowing method.*

Solution.

$$\begin{array}{rcccc}
 & 6 & 2 & 1 & 4 \\
- & 3 & 5 & 2 & 7 \\ \hline
 & 2 & 4 & 6 & 5
\end{array}$$

Explanation :

Column 1 : $4 + 8 \text{ (borrow)} - 7 = 12 - 7 = 5$

Column 2 : $1 + 8 \text{ (borrow)} - (2 + 1) = 9 - 3 = 6$

Column 3 : $2 + 8 \text{ (borrow)} - (5 + 1) = 10 - 6 = 4$

Column 4 : $6 - (3 + 1) = 2$

Example 3.27. *Perform* $(5113D)_{16} - (3A57)_{16}$ *using borrowing method.*

Solution.

$$\begin{array}{rccccc}
 & 5 & 1 & 1 & 3 & D \\
- & & 3 & A & 5 & 7 \\ \hline
 & 4 & D & 6 & E & 6
\end{array}$$

Explanation :

Column 1 : $13 - 7 = 6$

Column 2 : $3 + 16 \text{ (borrow)} - 5 = 19 - 5 = E$

Column 3 : $1 + 16 \text{ (borrow)} - (A + 1) = 17 - 11 = 6$

Column 4 : $1 + 16 \text{ (borrow)} - (3 + 1) = 17 - 4 = D$

Column 5 : $(5 - 1) = 4$

Example 3.28. *Perform* $(72A4)_{16} - (4E86)_{16}$ *using 16's complement.*

Solution. 15's complement of subtrahend can be found by subtracting each digit of subtrahend from 15.

15's complement of subtrahend = B179

16's complement of subtrahend = B179 + 1 = B17A

$$\begin{array}{rcccl}
 & 7 & 2 & A & 4 \quad \text{Minuend} \\
+ & B & 1 & 7 & A \quad \text{16's complement of subtrahend} \\
\hline
 & 18 & 4 & 17 & 14 \quad \text{Decimal sum} \\
 & 16- & 0- & 16- & 0 \quad \text{Modification} \\
\hline
 & 2 & 4 & 1 & E
\end{array}$$

Ans. $(241E)_{16}$

Example 3.29. *Perform* $(6214)_8 - (3527)_8$ *using 8's complement.*

Solution. 7's complement of subtrahend can be found by subtracting each digit of subtrahend from 7.

7's complement of subtrahend = 4250

8's complement of subtrahend = 4250 + 1 = 4251

$$\begin{array}{rcccl}
 & 6 & 2 & 1 & 4 \quad \text{Minuend} \\
+ & 4 & 2 & 5 & 1 \quad \text{8's complement of subtrahend} \\
\hline
 & 10 & 4 & 6 & 5 \quad \text{Decimal sum} \\
 & 8- & 0- & 0- & 0 \quad \text{Modification} \\
\hline
(1) & 2 & 4 & 6 & 5 \\
\uparrow & & & &
\end{array}$$

Ignore the final carry.

Ans. $(2465)_8$

3.8.3. Multiplication

Multiplication in base system is performed in the same way as in the decimal arithmetic but care should be taken while taking carryovers. This can be understood clearly as shown in the following examples.

Example 3.30. *Perform* $(47)_8 \times (13)_8$

Solution.

$$\begin{array}{rrrl}
 & 4 & 7 & \\
\times & 1 & 3 & \\
\hline
1 & 6 & 5 & \quad \text{1st partial product} \\
4 & 7 & & \quad \text{2nd partial product} \\
\hline
6 & 5 & 5 & \quad \text{Product}
\end{array}$$

Explanation : $7 \times 3 = (21)_{10} = (25)_8$, octal digit 5 is written in the first partial product and 2 is carried. Now, $4 \times 3 + 2$ (carry) $= (14)_{10} = (16)_8$, 16 is written. $7 \times 1 = (7)_{10} = (7)_8$, 7 is written for second partial product and no carry. $4 \times 1 = (4)_{10} = (4)_8$.

Hence, $(47)_8 \times (13)_8 = (655)_8$

Example 3.31. *Perform* $(AE2)_{16} \times (86)_{16}$

Solution.

$$\begin{array}{rrrrrl}
 & & A & E & 2 & \\
 & & \times & 8 & 6 & \\
\hline
 & 4 & 1 & 4 & C & \quad \text{Ist partial product} \\
5 & 7 & A & 0 & & \quad \text{2nd partial product} \\
\hline
5 & B & B & 4 & C & \quad \text{Product}
\end{array}$$

The same procedure explained in octal multiplication is followed, but multiples of 16 are considered to take carry and the partial products are added hexadecimally.

3.8.4. Division

Division in base system is performed in the same way as in the decimal division, i.e., by the method of trial subtraction.

***Example* 3.32.** *Divide $(3465)_8$ by $(32)_8$.*

Solution.

```
32 ) 3465 (106.75
     32
     ---
      265
    - 234
    -----
       310
     - 266
     -----
        220
      - 202
      -----
         16
```

Ans. $(106.75...)_8$

***Example* 3.33.** *Divide $(FF31)_{16}$ by $(ED)_{16}$.*

Solution.

```
ED )F F 3 1 ( 113
   - E D
     ---
     1 2 3
   -   E D
       ---
       3 6 1
     - 2 B 7
       -----
         A A
```

Hence, $(FF31)_{16} \div (ED)_{16}$ gives 113 as quotient and AA as remainder, respectively.

REVIEW QUESTIONS SET

1. What is meant by radix 7. How is a number system using radix *r* interpreted.
2. Describe binary, octal and hexadecimal number systems and symbols used for them.
3. Why do you suppose people adopted the decimal number system for everyday use? If you had to propose another number system so as to facilitate arithmetic computations. What radix would you choose? Justify your answer. What radix would you recommend for a computer? **(AMIE, S '94)**
4. 'Computers always need Radix conversion', why? **(AMIE, S '93)**
5. Assuming an arbitrary number system of radix 3, write down the first ten numbers in this system.
6. What is meant by complement *r*'s and $(r-1)$'s in a radix *r* system?
7. With examples, explain the meaning of 'radix' of a number system. Is it possible to have a number system without radix? If yes, give an example of such a number system. **(AMIE, W '93)**
8. State the reasons for use of binary system in the design of computer. **(AMIE, W '97)**
9. What are one's and two's complement schemes?
10. Why are subtractions using 2's complement used in modern computers? What are the difficulties with simple binary subtractions?
11. Convert the following binary numbers to decimal equivalent:
    (*a*) 1010 (*b*) 11011 (*c*) 11001 (*d*) 10011010
    (*e*) 111000101 (*f*) .0101 (*g*) 0.001101 (*h*) 11001.0101
    (*i*) 1011.0011 (*j*) 111011.101 (*k*) 11011.101 (*l*) 1010110.0101 **(AMIE, W '93)**
12. Convert the following octal numbers to equivalent decimal numbers:
    (*a*) 72 (*b*) 283 (*c*) 339.55 (*d*) 7715
13. Convert the following hexadecimal numbers to equivalent decimal numbers:
    (*a*) 64AC (*b*) A492 (*c*) A3FC (*d*) 3A9
    (*e*) IA5E (*f*) 2AF.A (*g*) 2CE.25
14. Convert the following decimal numbers to equivalent binary numbers:
    (*a*) 64 (*b*) 154 (*c*) 475 **(AMIE, '94)** (*d*) 15.4 **(AMIE, ' 94)**
    (*e*) 50.7 (*f*) .375 (*g*) 100.5
15. Convert the following decimal numbers to equivalent octal numbers :

(*a*) 63 (*b*) 332 (*c*) 146.25 (*d*) 0.25
(*e*) 632.97 (*f*) 0.0625 (*g*) 225.225 **(AMIE, S '94)**

16. Convert the following decimal numbers to equivalent hexadecimal numbers :
(*a*) 428 (*b*) 745 (*c*) 967 (*d*) 465.5
(*e*) 225.225 **(AMIE, S '94)**

17. Perform the following binary addition and check by converting the binary numbers to decimal:
(*a*) 101001 + 11010 (*b*) 1100 + 1001
(*c*) 1011.1010 + 1010 + 1000.011 (*d*) 110101 + 100101
(*e*) 1111 + 1111 (*f*) 110101 + 101111
(*g*) 101.11 + 110.10 (*h*) 1101.1011 + 10001.01
(*i*) 11011 + 10111 (*j*) 11001 + 1011 + 110011
(*k*) 11.101 + 110.01 + 111.101 + 1101.1 (*l*) 11011 + 111001 + 1001 + 11001.

18. Find the binary differences and check by converting the binary numbers to decimal:
(*a*) 10001 − 1111 (*b*) 111000 − 11001
(*c*) 110.001 − 11.111 (*d*) 1101 − 1010
(*e*) 100 − 011 (*f*) 101 − 011
(*g*) 0.11 − 0.101 (*h*) 1011.1 − 0100.11

19. Find the binary product of the following:
(*a*) 11010 × 1001 (*b*) 10101 × 1011
(*c*) 111 × 101 (*d*) 101.1 × 11.01
(*e*) 11.101 × 11.01 (*f*) 1111 × 111

20. Perform the binary divisions of the following:
(*a*) 1011 ÷ 11 (*b*) 100011 ÷ 101
(*c*) 11011 ÷ 11 (*d*) 101010 ÷ 111
(*e*) 11001 ÷ 101 (*f*) 100.0001 ÷ 10.1
(*g*) 100001 ÷ 110 (*h*) 11110 ÷ 110

21. Convert the following binary numbers to equivalent octal numbers:
(*a*) 101101 (*b*) 110110.011
(*c*) 1011.1011 (*d*) 1100101
(*e*) 11010011.011011

22. Convert the following binary numbers to equivalent hexadecimal numbers:
(*a*) 101101 (*b*) 1111110
(*c*) 1001111 (*d*) 0.01111110
(*e*) 1010110.01011001

23. Convert the following octal numbers to equivalent binary numbers:
(*a*) 376 (*b*) 56.34
(*c*) 562 (*d*) 3.75

24. Convert the following hexadecimal numbers to equivalent binary numbers:
(*a*) 5D (*b*) F2E
(*c*) 2BCD (*d*) 2AB

25. Convert the following octal numbers to equivalent hexadecimal numbers:
(*a*) 536 (*b*) 4753
(*c*) 714.06

26. Convert the following hexadecimal numbers to equivalent octal numbers:
(*a*) D9 (*b*) 58.3A (*c*) 31C.E8

27. Convert the following numbers as per instructions given against each:
(*a*) $(375.25)_{10} = (?)_2 = (?)_8$ (*b*) $(A09.26)_{16} = (?)_2 = (?)_{10}$
(*c*) $(101101.01)_2 = (?)_{16} = (?)_{10}$ (*d*) $(32.14)_6 = (?)_{10} = (?)_2$ **(AMIE, W '95)**
(*e*) $(476.275)_8 = (?)_{10} = (?)_{16}$ **(AMIE, W '97)**

28. Convert the following decimal numbers into octal and hexadecimals:
(*a*) 97.001 (*b*) 26.25 **(AMIE, W '94)**

29. Convert the decimal equivalent of
$(11010.111)_2$, $(736.5)_8$ and $(3FA.8)_{16}$ **(AMIE, S '94)**

30. Find 1's and 2's complements of the following binary numbers:
(*a*) 111 (*b*) 100.11 (*c*) 110110 (*d*) 110011 (*e*) -538.25_{10} = 2's complement binary
(AMIE, W '97)

31. Perform the following subtractions of binary numbers by the 2's complement method:
(*a*) 0.1110 – 0.0110 (*b*) 10101 – 11010 (*c*) 101 – 111
(*d*) 0.11 – 0.101 (*e*) 10101.001 – 10001.01 (*f*) 11000011 – 00010111

32. Write the first 12 numbers in the base 4 number system.

33. Perform
(*a*) $(BAD)_{16} + (432)_{16}$ (*b*) $(CAB)_{16} + (427)_{16}$ (*c*) $(7346)_8 + (5263)_8$
(*d*) $(72)_8 - (25)_8$ (*e*) $(7526)_8 - (3142)_8$ (*f*) $(67.E9)_{16} + (A.BCDE)_{16}$
(*g*) $76B5_{16} - 432C_{16}$ (*h*) $(45376)_8 + (36274)_8$ (*i*) $(6157)_8 - (4325)_8$

34. Find 9's and 10's complements of the following decimal numbers:
(*a*) 75 (*b*) 183

35. Find radix-minus one complement and the radix complements of
(*a*) $74B9_{16}$ (*b*) 50309_{16}

36. Perform
(*a*) $31.37_8 \times 6.2_8$, (*b*) $1A.23_{16} \times 8.F_{16}$ and (*c*) $A3E_{16} \times B4_{16}$.

37. Convert the following:
(*a*) $(3.75)_8$ to binary form, (*b*) $(689B)_{16}$ to decimal form,
(*c*) $(1101.1111)_2$ hexadecimal form, (*d*) $(634.64)_{10}$ to octal form. **(AMIE, W '97)**

ANSWERS TO REVIEW QUESTIONS SET

11. (*a*) 10 (*b*) 27 (*c*) 25 (*d*) 154 (*e*) 2742 (*f*) 0.3125 (*g*) 0.203125
(*h*) 25.3125 (*i*) 11.1875 (*j*) 59.625 (*k*) 27.625 (*l*) 86.3125

12. (*a*) 58 (*b*) 195 (*d*) 409.703 (*d*) 4045

13. (*a*) 25772 (*b*) 42130 (*c*) 41980 (*d*) 937 (*e*) 6750 (*f*) 687.625
(*g*) 718.207

14. (*a*) 1000000 (*b*) 10011010 (*c*) 111011011 (*d*) 1111.011001
(*e*) 110010.1011001 (*f*) 0.011 (*g*) 1100100.10

15. (*a*) 77 (*b*) 514 (*c*) 224.4 (*d*) 0.2 (*e*) 1170.76051
(*f*) 0.04 (*g*) 341.341

16. (*a*) 1AC (*b*) 2E9 (*c*) 3C7 (*d*) 1D 1.8 (*e*) E1.E1

17. (*a*) 1100011 (*b*) 10101 (*c*) 10100.00 (*d*) 1011010
(*e*) 11110 (*f*) 1100100 (*g*) 1100.01 (*h*) 11110.1111
(*i*) 110010 (*j*) 1110011 (*k*) 11111.000 (*l*) 1110110

18. (*a*) 10 (*b*) 11111 (*c*) 10.010 (*d*) 0011
(*e*) 011 (*f*) 100.01 (*g*) 0.001 (*h*) 110.11

19. (*a*) 11101010 (*b*) 11100111 (*c*) 100011 (*d*) 10001.111
(*e*) 1011.11001 (*f*) 1101001

20. (*a*) 11.1010 (*b*) 111 (*c*) 1001 (*d*) 110
(*e*) 101 (*f*) 1.101 (*g*) 0101 (*h*) 101

21. (*a*) 55 (*b*) 66.3 (*c*) 13.54 (*d*) 145 (*e*) 323.33

22. (*a*) 2D (*b*) 7E (*c*) 4F (*d*) 0.7E (*e*) 59.59

23. (*a*) 11111110 (*b*) 101110.0111 (*c*) 101110010 (*d*) 11.111101

24. (*a*) 1011101 (*b*) 111100101110 (*c*) 0010101111001101 (*d*) 001010101011

25. (*a*) 153 (*b*) 9EB (*c*) ICC.18

26. (a) 331 (*b*) 133.164 (*c*) 1434.72

27. (*a*) $(101110111.01)_2 = (567.2)_8$ (*b*) $(101000001001.001011)_2 = (2559.171)$
(*c*) $(2D.4)_{16} = (45.25)_{10}$ (*d*) $(125.29)_{10} = (111101.00101)_2$ (*e*) 320.3691406, 13E.5E8

28. (*a*) 141.000406 (*b*) 32.2

29. (*a*) 26.875 (*b*) 478.625 (*c*) 1018.5

30. (*a*) 000, 001 (*b*) 011.00, 011.00 (*c*) 001001, 001010 (*d*) 001100, 001101
(*e*) 1011100110.10

31. (*a*) .1000 (*b*) – 101 (*c*) – 010
(*d*) 0.001 (*e*) 11.111 (*f*) – 00100110

33. (*a*) FDF (*b*) 12D2 (*c*) 14631 (*d*) 45
(*e*) 4364 (*f*) 72.A 5DE (*g*) 3389 (*h*) 103672 (*i*) 1632

34. (*a*) 24, 25 (*b*) 183

35. (*a*) 8B46, 8B47 (*b*) A2CF6, A2CF7

36. (*a*) 137. 216_8 (*b*) $E9.98D_{16}$ (*c*) 73398_{16}

37. (*a*) $(11.111101)_2$ (*b*) $(26779)_{10}$ (*c*) $(13.E)_{16}$ (*d*) $(1172.5075)_8$

4
DATA REPRESENTATION

4.0. INTRODUCTION

Data are usually represented by using the alphabets A to Z, numbers 0 to 9 and various other symbols. This form of representation is used to formulate problem and fed to the computer. The processed output is required in the same form. This form of representation is called external data representation. However, the computer can understand data only in the form of 0 and 1. The method of data representation in a form suitable for storing in the memory and for processing by the CPU is called the internal data representation on digital computer.

Data, in general, are of two types: numeric and non-numeric (character data). The numeric data deals only with numbers and arithmetic operations and non-numeric data deals with characters, names, addresses, etc., and non-arithmetic operations.

4.1. REPRESENTATION OF SIGNED AND UNSIGNED NUMBERS

Numeric data represented in either of the two forms, integer and real. Integers are the numbers which do not have decimal point. These integers may be positive or negative. For example, 8, –11, 0, 157 are termed as integers. A real number consists of an integer part and a fractional part. A real number may be positive or negative. For example, 10.56, – 23.07, etc., are real numbers.

4.1.1. Representation of Pcsitive Integers

In the binary system, we represent the sign of a number using an extra bit, known as sign bit, at the extreme left of the number. By convention, the bit 0 is used to represent the (+) sign. For instance, +7 is represented by 0, 111. The comma separates the sign bit from the number. This method of representation is known as a signed magnitude representation.

The storage format of a typical 16 bit word size computer is shown in Fig. 4.1. In a 16 bit word length computer, a positive integer could be stored with values from 0, 000000000000000 to 0, 111111111111111, i.e., from 0 to $2^{15} - 1$. In general, for a n bit word length computer, the largest positive integer that can be stored is $2^{n-1} - 1$.

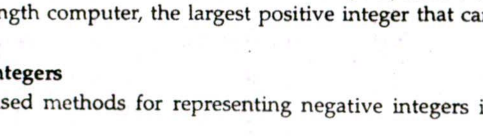


Fig. 4.1: A typical 16 bit binary integer storage

4.1.2. Representation of Negative Integers

There are usually three commonly used methods for representing negative integers in binary terms. These are:

1. Signed Magnitude Representation.
2. Signed 1's Complement Representation.
3. Signed 2's Complement Representation.

Signed magnitude representation: In this representation, the negative number bit pattern differs from the corresponding positive number bit pattern by only digit '1' in the sign bit location. In *n*-bit computer words, the range of numbers representable is between - $(2^{n-1} - 1)$ to $(2^{n-1} - 1)$. For example, in an 8 bit computer word, the number $(+15)_{10}$ whose binary equivalent is 1111, is represented as 0, 0001111 and $(-15)_{10}$ as 1, 0001111. All other numbers between $-(2^{8-1} - 1) = -127$ and $+(2^{8-1} - 1) = +127$ have unique representation. Zero has forms 0, 0000000 (+ 0) and 1, 0000000 (– 0).

Signed 1's complement representation : In binary number system, the 1's complement of 1 is 0 and 0 is of 1. Hence, $(+15)_{10}$ in 1's complement into 8 bit representation is represented as 0, 0001111 and $(-15)_{10}$ as 1, 1110000. The range of numbers represented is $-(2^{n-1} - 1)$ to $(2^{n-1} - 1)$, same as signed magnitude representation. Here also 0 has two representations 0, 0000000 (+ 0) and 1, 1111111 (– 0).

Signed 2's complement representation : It is 1's complement plus 1. Hence, $(+15)_{10}$ in 2's complement in 8 bit word is represented as 0, 0001111 and $(-15)_{10}$ as 1, 1110001 (which is two's complement of + 15). All numbers between -2^{n-1} and $(2^{n-1} - 1)$ have unique representation. This form do not represent 0 in dual form and hence can represent – 128 to + 127 in a 8 bit computer word.

The signed magnitude system is easier to interpret but computer arithmetic with this is not efficient. The circuits for handling numbers are simplified if 1's or 2's complement systems are used and as a result one of these is almost always adopted.

The representations and range of integer numbers in an 8 bit computer word are shown below :

Decimal	*Signed magnitude*		*1's complement*		*2's complement*	
–128	—		—		1000	0000
–127	1111	1111	1000	0000	1000	0001
–126	1111	1110	1000	0001	1000	0010
–125	1111	1101	1000	0010	1000	0011
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
–2	1000	0010	1111	1101	1111	1110
–1	1000	0001	1111	1110	1111	1111
– 0	1000	0000	1111	1111	—	
0	0000	0000	0000	0000	0000	0000
1	0000	0001	0000	0001	0000	0001
2	0000	0010	0000	0010	0000	0010
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
+127	0111	1111	0111	1111	0111	1111

Fig. 4.2: Representations and range of numbers for one byte

Note 1: In 1's and 2's complements, all positive integers are represented as in sign magnitude system.

Note 2: When all the bits of the computer word are used to represent the number and no bit is used for signed representation, it is called unsigned representation of the number.

4.1.3. Addition in Signed 2's Complement

2's complement signed binary coding does not call any special logic when performing arithmetic operation provided an arithmetic operation does not generate an answer that is too large to accommodate in the available space (overflow). The sign bit is treated as though it is a part of the number, the answer indicates whether it is positive or negative. If the sign bit is 1, then the answer is negative and by taking 2's complement of the answer, the pure binary representation of the answer is found. The following examples will illustrate the addition of signed binary numbers.

Example 4.1. *Add (+ 3) and (+ 7).*

Solution.

$$\begin{array}{rl}
+3 & 0,0\ 0\ 1\ 1 \\
+7 & 0,0\ 1\ 1\ 1 \\
\hline
+10 & 0,1\ 0\ 1\ 0
\end{array}$$

Sign bit is 0, so the sum is positive.

Example 4.2. *Add (+14) and (–9) using 8 bit for storage representation.*

Solution.

$$\begin{array}{rll}
+14 & 0,0\ 0\ 0\ 1\ 1\ 1\ 0 & \\
-\ 9 & 1,1\ 1\ 1\ 0\ 1\ 1\ 1 & \text{(2's complement of }-9) \\
\hline
+\ 5 & 10,0\ 0\ 0\ 0\ 1\ 0\ 1 &
\end{array}$$

Sign bit 0, so the sum is positive (carry is neglected)

Example 4.3. *Add (– 12) and (+ 8).*

Solution.

$$\begin{array}{rll}
-12 & 1,1\ 1\ 1\ 0\ 1\ 0\ 0 & \text{(2's complement of }-12) \\
+\ 8 & 0,0\ 0\ 0\ 1\ 0\ 0\ 0 & \\
\hline
-4 & 1,1\ 1\ 1\ 1\ 1\ 0\ 0 &
\end{array}$$

Sign bit 1, so the sum is negative and 2's complement form

2's complement of 1111100 = 0000100 = 4

Ans. – 4

Example 4.4. *Add (–13) and (–11).*

Solution.

$$\begin{array}{rll}
13 & 0,0\ 0\ 0\ 1\ 1\ 0\ 1 & \\
11 & 0,0\ 0\ 0\ 1\ 0\ 1\ 1 & \\
-13 & 1,1\ 1\ 1\ 0\ 0\ 1\ 1 & \text{(2's Complement of }-13\text{ with sign bit)} \\
-11 & 1,1\ 1\ 1\ 0\ 1\ 0\ 1 & \text{(2's Complement of }-11\text{ with sign bit)} \\
\hline
 & 11,1\ 1\ 0\ 1\ 0\ 0\ 0 &
\end{array}$$

Sign bit indicates the negative result (carry ignored) in 2's complement form.

2's Complement of 1101000 = 0011000

Ans. – 24

4.2. BINARY CODED DECIMAL (BCD)

When a decimal number is represented by its equivalent binary number, we call it straight binary coding. If each digit of a decimal number is represented by its binary equivalent, the result is a code called binary-coded decimal. As there are ten digits to be encoded in decimal system, we require 4 bits for encoding each of the digit. For example, the decimal number 23 is represented as

2	3	Decimal
0010	0011	B C D

BCD code is a weighted code, each bit is assigned a weight and from left to right, the weights are 8, 4, 2 and 1. This is known as weighted 8 – 4 – 2 – 1 BCD. There are other weighted BCD codes but the 8 – 4 – 2 – 1 code is usually referred to as BCD code. The other weighted BCD codes always mention the weightage to distinguish them from the 8421 BCD code. The bit 0110, for example, can be interpreted by the weights to represent the decimal digit 6 as $0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 6$.

To convert a given BCD number to decimal number, all bits are to be divided into groups of 4 starting from the decimal point (if any) towards left and right adding 0 at extremities (if necessary) and each group of 4 bits is replaced by corresponding decimal number.

For example, $(10101.1001)_{BCD} = (0001\ 0101.1001)_{BCD} = (15.9)_{10}$

and $(137.409)_{10} = (0001\ 0011\ \ 0111.0100\ \ 0000\ \ 1001)_{BCD}$

4.2.1. Advantages and Disadvantages of BCD System

The advantages of using BCD representation for decimal numbers over binary number system are in terms of saving the efforts of conversion. This is very important as user generally provides input in decimal form and expects his results also in decimal form.

There is no round of error in BCD representation, but there may be in straight binary representation, for example, which will produce some error in computer processing.

The disadvantages of BCD representation of a number (except for single digit decimal number) that it takes more digits in BCD than in straight binary coding.

Performing arithmetic operations with straight binary coding is easier than the BCD coding.

Circuits required for BCD arithmetic operations are more complex than those of binary circuits and require more time in executing numeric computation.

4.2.2. BCD Addition

It is known that the adder circuits perform additions in pure binary fashion only where a carry is generated when a pair of 1s are added, in decimal operation. If we add 7 and 6, a carry is generated when the sum exceeds 9. Since BCD notation uses 4 binary bits, a carry into the fifth bit position will occur only when the 4 bits are greater than 1111 or 15. But BCD notation goes only up to 9, which is represented by 1001, and the next number is 0001 0000, thereby a carry would require an addition of a further decimal number 6 (0110 in BCD).

BCD addition procedure:

1. Add, using ordinary binary addition, the BCD code groups for each digit position.
2. For those positions where the sum is 9 or less, no correction is needed. The sum is in proper BCD form.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum to get the proper BCD result. This case always produces a carry into the next digit position, either from the original addition (step 1) or from the correction addition.

The following examples will illustrate the procedure.

Example 4.5. Perform 26 + 37 using BCD number system. **(AMIE, S '93)**

Solution.	0010	0110	BCD for 26
	0011	0111	BCD for 37
	0101	1101	Perform addition
		+ 0110	Add 6 for BCD correction
	0110	0011	BCD for 63

Example 4.6. Add 236 and 194 using BCD code. **(AMIE, W '93)**

Solution.	0010	0011	0110	BCD for 236
	0001	1001	0100	BCD for 194
	0011	1100	1010	Perform addition
		+ 0110	0110	Add 6 for BCD correction
	0100	0011	0000	BCD for 430

Example 4.7. Perform 342 + 739 in BCD system.

Solution.		0011	0100	0010	BCD for 342
		0111	0011	1001	BCD for 739
		1010	0111	1011	Perform addition
		+ 0110		0110	Add 6 for BCD correction
	0001	0000	1000	0001	BCD for 1081

4.3. OTHER BCD WEIGHTED CODES

The other types of BCD weighted codes are given in Table 4.1. Of these, $8-4-\bar{2}-\bar{1}$ code has two bits, namely, $\bar{2}$ and $\bar{1}$ which have negative weights. The decimal digit can be converted to weighted codes by starting from the left-most weight down to the right-most weight. Express the decimal digit as the sum of weights. Write 1 against the weights which appear in the sum and 0 against the weights which do not appear in the sum.

Example 4.8. *Convert $(458)_{10}$ to 4–2–2–1 BCD code.*

Solution.	*Digit*	*Weight*	*Code*
		4 – 2 – 2 – 1	
	4	4 + 0 + 0 + 0	1000
	5	4 + 0 + 0 + 1	1001
	8	4 + 2 + 2 + 0	1110

Hence, 4 –2–2–1 BCD representation of 458 is 1000 1001 1110. Note that encoding is not unique in the system, e.g., 2 → 0010 and 2 → 0100.

Table 4.1

Weights / *Digits*	*2421 Code*	*4221 Code*	*5221 Code*	*$84\bar{2}\bar{1}$ Code*
0	0000	0000	0000	0000
1	0001	0001	0001	0111
2	0010	0010	0100	0110
3	0011	0011	0101	0101
4	0100	1000	0110	0100
5	0101	1001	1000	1011
6	1100	1010	1001	1010
7	1101	1011	1100	1001
8	1110	1110	1101	1000
9	1111	1111	1110	1111

4.4. NON-WEIGHTED CODES

The non-weighted codes are of two types, namely, (*i*) Non-error detecting codes, and (*ii*) error detecting codes. Non-error detecting codes are: (*i*) excess-3 code, and (*ii*) gray code.

4.4.1. Excess-3 Code

This is a binary code in which each decimal number is expressed in excess of three, i.e., 3 is added to the decimal number and then coded into binary digits, e.g., to get the excess-3 code of 4, 3 is added to it which gives 7. The BCD code of 7, namely, 0111 will be 4 in the excess-3 code. The excess-3 code is said to be self-complementing code (a code is said be self-complementing if the code word of the 9's complement of *N*, *i.e.*, 9-*N* can be obtained from the code word of *N*, by interchanging all the 1's and 0's). For example, representation of 7 in excess-3 is given by 0111 + 0011 = 1010 while decimal (9–7) is represented by 0010 + 0011 = 0101 which can be obtained by interchanging 0 and 1 in 1010. Note that BCD code is not self-complementing. There exists only four positively weighted self-complementing codes, namely, 2–4–2–1, 3–3–2–1, 4–3–1–1 and 5–2–1–1. In addition, there exist, 13 self-complementing codes with positive and negative weights.

Table 4.2

Decimal Number	*Excess-3 Code*	*Complement of Excess-3 Code*	*Decimal value of the Complement*
0	0011	1100	9
1	0100	1011	8
2	0101	1010	7
3	0110	1001	6
4	0111	1000	5
5	1000	0111	4
6	1001	0110	3
7	1010	0101	2
8	1011	0100	1
9	1100	0011	0

4.4.2. Gray Code

The gray code is an unweighted code not suited to arithmetic operations, but useful for input/output devices, analog to digital converters. Each gray code differs from its neighbour by a single bit. (Codes which have such a property is known as cyclic code). For instance, in going from 7 to 8, the gray code numbers change from 0100 to 1100. The codes for 0 and 1 are the same as binary code.

4.4.2.1. Conversion from Binary to Gray

The steps of conversion are:

(*a*) The most significant digit of gray code is the same as the most significant digit of binary code;

(*b*) Add each pair of adjacent bits (starting from most significant digit) of the binary code to get the next gray digit disregarding the carries.

Example 4.9. *Convert the binary number 1100 to gray code number.*

Solution.

$$\begin{tikzcd}
1 \arrow[d] \arrow[r, bend left, no head] \arrow[dr] & 1 \arrow[r, bend left, no head] \arrow[dr] & 0 \arrow[r, bend left, no head] \arrow[dr] & 0 & \text{binary} \\
1 & 0 & 1 & 0 & \text{gray}
\end{tikzcd}$$

add 1 + 1 = 0
(ignoring carry)
add 1 + 0 = 1
add 0 + 0 = 0

Hence, the required number is 1010.

Table 4.3: Complete 4 bit Gray Code

Decimal number	*Gray*				*Binary*			
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

4.4.2.2. Conversion from Gray to Binary

Gray coded numbers may be converted to binary with the help of the following procedure:

(*a*) Starting from the MSD of the gray code, go on keeping 0s until encounter with the first 1. Keep it 1, and

(*b*) Go on keeping 1, until encounter with another 1, then

(*c*) Replace it by 0 and repeat the steps (*a*) and (*b*) until LSD of the gray code is reached.

4.4.3. Error Detecting Code

Computers are very reliable. However, if one bit in a string of 7 or 8 bits is lost during data input, processing or output operations, an incorrect code will be created. Errors may be caused by dust particles on storage media, by improper humidity levels or by electrical disturbances during data transmission between units. Computer designers have developed a method for detecting errors by adding an extra bit usually preceding the zone bits to each 7 or 8 bit character presented in storage. This additional bit is known as parity bit. This checking feature is called a parity check. There are two types of parity checking codes, namely, (*a*) Odd parity code, and (*b*) Even parity code. In the odd parity method, the value of the parity bit is chosen so that the total number of 1's in the code group (including the parity bit) is an odd number. The even parity method is designed in exactly the same way except that the parity bit is chosen so that the total number of 1's (including the parity bit) is an even number.

For example, suppose that the group is 1000011. This is ASCII character C. The code group has three 1's. Therefore, we will add a parity bit of 1 to make the total number of 1 as an even number if it is designed for even parity code. The new code group including the parity bit thus becomes

$$\underset{\uparrow\text{ added parity bit}}{\boxed{1}}\ 1\ 0\ 0\ 0\ 0\ 1\ 1$$

If the code group contains even numbers of 1s to begin with the parity bit is given a value of 0.

It may be noted that if there is a one failure in more than one bit positions, then the proper parity may be maintained and parity checking system will fail. For detecting failure in multiple bits, more additional bits are to be provided and this will increase the hardware cost of the system.

4.5. REAL NUMBERS REPRESENTATION

There are two basic techniques to represent real numbers in a computer. These are called (*a*) the fixed-point representation, (*b*) the floating-point representation.

4.5.1. Fixed-point Representation

Since a real number has two integer parts, i.e., a part preceding a decimal point known at integer part and a part following a decimal point known as fractional part. The representation of real numbers can be done by assuming a fixed position between two parts. All the arithmetic operations are done treating them as integers. After the operations are performed, one may need to transfer any carry and borrow generated in the fractional part to the integer part. Fig. 4.3

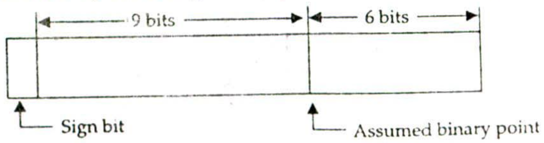


Fig. 4.3

shows the format for fixed-point representations of real numbers when a 16 bit word is employed to store numbers.

Fixed point system has two major disadvantages. First, the range of numbers that can be represented is restricted by the number of digits or bits used. Second, the need of keeping track of the binary point particularly in multiplication and division operations so that it can be correctly positioned in the final result. Thus, in the fixed-point of representation the user has to keep the track of radix point which is a tedious job. Hence, a system of representation which automatically keep track of the position of the radix point is required. Such a system of representation is called floating point representation of numbers.

4.5.2. Floating Point Representation

A convenient notation for representing real number is the floating point notation. The notation is based on the relation $y = a \times r^p$, where y is the number to be represented, a is the mantissa, r is the base of the number system (r = 10 for decimal and r = 2 for binary) and p is the power the base is raised, called the exponent. For example, 14.2×10^4, -7.8×10^{-12} are real numbers in floating point notation and 14.2 and – 7.8 are called mantissa and 400, –12 are the exponent of base 10, respectively. Such a representation is not unique, e.g., $567 = 0.0567 \times 10^4 = 0.567 \times 10^3 = 56.7 \times 10^1 = 56700 \times 10^{-2}$. Observe that the only difference between the equivalent forms is in the position of the decimal point and the exponent of 10.

In computer memory, real numbers are represented in normalised floating point notation in which any non-zero decimal number D can be uniquely expressed as $D = M \times 10^n$, where the decimal point appears directly infront of the first non-zero digit in M ($.1 \le M < 1$) for positive D and $-1 < M \le -.1$ for negative D. Decimal number 15.62 and 0053 can be represented in normalized form as $.1562 \times 10^2$ and $.5300 \times 10^2$, respectively. In binary normalized floating point form the mantissa M would have a value from .5 (decimal) to 1 i.e., the binary point appears before the first 1 bit. The binary numbers –11 and – 010 can be expressed in normalized form as -0.1100×2^2 and 0.100×2^{-1}. Table 4.4 gives some binary numbers in normalized floating point form, each mantissa being written with exactly 5 bits.

Table 4.4

Number	*Normalized form*	*Mantissa*	*Exponent*
11.33	0.1133×10^2	0.1133	2
0.0044	0.4400×10^{-2}	0.400	– 2
– 0.000077	-0.7700×10^{-4}	– 0.7700	–4
10.1	0.1010×2^2	0.1010	2
– 1111	-0.1111×2^4	– 0.1111	4
0.001110	0.1110×2^{-2}	0.1110	– 2

A word in a floating point representation may be divided into two block of bits: the first block contains the mantissa of the number and second, the exponent of the number to be represented. The format of representation in floating point form in 16 bit word is shown in Fig. 4.4.

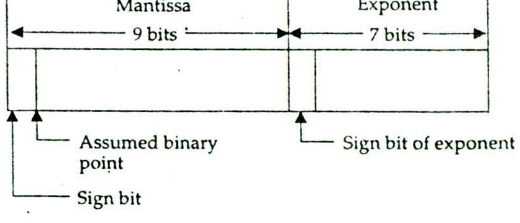


Fig. 4.4

In a normalized floating point mode the largest magnitude number which may be stored in a 16 bit computer word using 9 bits for mantissa and 7 bits for exponent is

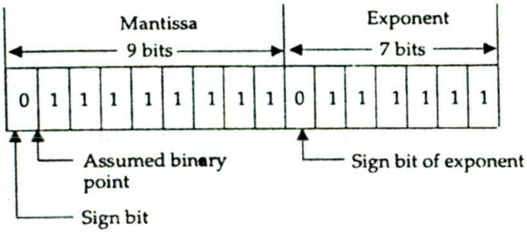


Fig. 4.5: Maximum stored number.

$$0.11111111 \times 2^{0111111} = (1 - 2^{-8}) \times 2^{2^6 - 1}$$
$$= (1 - 2^{-8}) \times 2^{63} = 10^{19}$$

and the minimum value is

$$0.10000000 \times 2^{1111111} = (2^{-1}) \times 2^{-(2^6 - 1)}$$
$$= (2^{-1}) \times 2^{-63} = 2^{-64} = 10^{-19}$$

The number of bits to be used for the mantissa is determined by the number of significant digits required in computation.

It is possible to use a radix other than 2. For example, $.7556 \times 8^6$ is represented as in Fig. 4.6 as the mantissa and ·xponent are same as in $.11110110111 \times 2^6$. The difference is in the assumption of the radix which must be assumed together with fixed point position of the mantissa and must be used consistently.

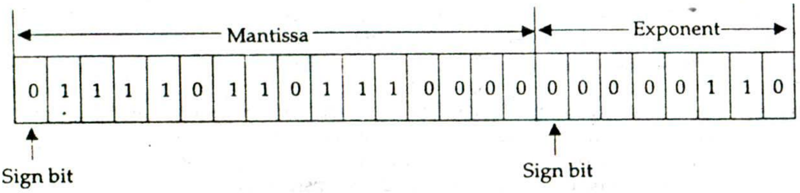


Fig. 4.6

To avoid negative sign in exponent, most computers represent exponent (n) by its excess representation known as bissed exponent $n + 2^{t-1}$ when t is the number of bits in the exponent block. This means that the true exponent is stored after adding a constant. This constant is determined according to the method that the minimum number is increased to zero. Table 4.5 shows the relationship between true exponent and its excess representation.

Table 4.5

True Exponent	– 64	– 63	– 62 1	0	1......63
Excess Representation	0	1	2.....63	64	65.....127

The decimal fraction –15.0 can be represented in normalized binary exponential form as $-.1111 \times 2^4$. The true exponent is 4, its excess representation is $4 + 64 = 68 = 1000100$. Hence, decimal number –15.5 can be stored in 16 bit word as shown in Fig. 4.7.

Another way in which the exponent bit may be interpreted is to assume a base other than 2 for the exponent. If we assume a base of 16 for the exponent, then the largest magnitude floating point number that may be represented in this format is

$$0.11111111 \times 16^{0111\ 111} = 0.11111111 \times 16^{63}$$

The range obtained by this representation is considerably large compared to base 2 representation of the exponent. But there is some loss of significance. If the exponent is increased by 1 the mantissa is to be shifted left by one hexadecimal digit, i.e., 4 bits. When 2 is used as the exponent base, increasing exponent by 1 will lead to shifting the mantissa left by one bit position.

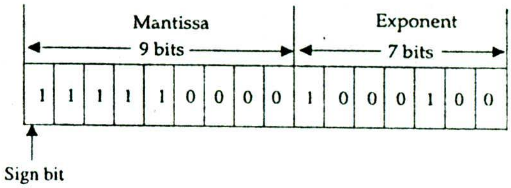


Fig. 4.7

4.5.3. Floating Point Arithmetic

Arithmetic operations with floating point number representation are more complicated than arithmetic operations with fixed point numbers and their execution takes longer time and requires more complex hardware circuitry.

4.5.3.1. Addition

(*a*) If two numbers to be added have the equal exponent, the mantissas are added the same exponent issued and the sum is renormalized. if necessary.

$0.4466 \times 10^4 + 0.777 \times 10^4 = 1.2236 \times 10^4 = .1224 \times 10^5$

(renormalized)

(*b*) If two numbers have different exponent then the operand with the larger exponent is kept as it is and the mantissa of the operand with the smaller exponent is shifted right by a number of places equal to the difference in the two exponents so that they have equal exponent. With each shift, the exponent must be changed. For each left shift of mantissa the exponent is decreased by one and for each right shift the exponent is increased by one, and for each increase of exponent causes the last digit in the mantissa to be chopped off, as number of digits of mantissa that a location can accommodate is usually fixed in a particular computer. For example, if the arithmetic unit can accommodate only a 4 digit mantissa, then

$$.3344 \times 10^2 + .8777 \times 10^{-1}$$
$$= .3344 \times 10^2 + .0008 \times 10^2 = .3352 \times 10^2$$

In this case, two exponents are not equal. The difference between the exponents is $2 - (-1) = 3$. Thus, the mantissa .8777 is shifted right three places. Each shift cause the last digit in the mantissa to be chopped as the arithmetic unit in this case can accommodate only a 4 digit mantissa.

4.5.3.2. Subtraction

The operation of subtraction is nothing but adding a negative number. The principles are the same as addition. The following examples will illustrate the procedure.

Example 4.10. *Subtract* $.8432 \times 10^{-4}$ *from* $.5451 \times 10^3$

Solution. $.5451 \times 10^{-3} - .8432 \times 10^{-4}$

$= .5451 \times 10^{-3} - .0843 \times 10^{-3} = (.5451 - .0843) \times 10^{-3}$

$= .4608 \times 10^{-3}$

Example 4.11. *Subtract* $.1101 \times 2^4$ *from* $.110111 \times 2^5$. **[AMIE (AN) Model Paper]**

Solution. $.110111 \times 2^5 - .1101 \times 2^4$

$= .110111 \times 2^5 - .01101 \times 2^5 = (.110111 - .01101) \times 2^5$

$= .11101 \times 2^4$ (renormalized)

Example 4.12. *Subtract* $0.6\ B \times 16^2$ *from* $0.7A \times 16^3$. **[AMIE (AD), Model Paper]**

Solution. $0.7A \times 16^3 - 0.6B \times 16^2$

$= 0.7A \times 16^3 - 0.06 \times 16^3 = 0.64 \times 16^3$

4.5.3.3. Multiplication and Division

(a) Multiplication rule :

(i) Multiply the mantissas
(ii) Add the exponents
(iii) Normalize and truncate, if necessary.

(b) Division rule :

(i) Divide the mantissas
(ii) Subtract the exponents
(iii) Normalize

i.e., if the numbers are $n_1 = f_1.r^{e_1}$ and $n_2 = f_2.r^{e_2}$

$$n_1 \times n_2 = (f_1 \times f_2)\, r^{e_1+e_2} \text{ and } n_1/n_2 = (f_1/f_2) \times r^{e_1-e_2}$$

Example 4.13. Multiply 0.3355×10^2 by 0.4466×10^3.

Solution.

$$\begin{aligned} & (0.3355 \times 10^2) \times (0.4466 \times 10^3) \\ &= (0.3355 \times 0.4466) \times 10^{2+3} \\ &= 0.14983480 \times 10^5 = 0.1498 \times 10^5 \end{aligned}$$

Example 4.14. Divide 0.2526×10^7 by 0.8352×10^4.

Solution.

$$\begin{aligned} & (0.2526 \times 10^7) \div (0.8352 \times 10^4) \\ &= (0.2526 \div 0.8352) \times 10^{7-4} \\ &= 0.3024425 \times 10^3 = 0.3024 \times 10^3 \end{aligned}$$

4.6. OVERFLOW AND UNDERFLOW

Normally, logic system of a computer is so designed that the size of the word is fixed. If the magnitude of the number after arithmetic operations exceeds the capability of the storage of the words, an overflow occurs and errors are bound to arise. Therefore, it is essential to devise a procedure for identifying such erroneous results.

4.6.1. Integer Arithmetic

An overflow occurs when the numbers to be added are both positive or both negative and the results exceeds the storing capacity of the register. An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result (positive or negative) which is smaller than the larger of the two original numbers. The limit on the maximum number of positive and negative integers that are possible in a word of a given size can be determined by the formula.

Number of positive integers $= 2^{B-1} + 1$, and

Number of negative integers $= 2^{B-1}$

where B is the number of bits.

For example, in a 8-bit word, the number of positive integers $= 2^{8-1} - 1 = 127$, *i.e.*, the number 0 to 127 can be represented in a 8-bit word and similarly negative number –128 to 0 can be represented in a 8-bit word. The following examples will show the addition of two positive or two negative numbers when there is an overflow.

Example 4.15. Add + 85 and +67 using 8 bit register.

Solution.

+ 85	0,1010101
+ 67	0,1000011
+152	1,0010000

In this, carry from MSB of the addition of numbers is 1 and carry from the sign bit is 0 ,i.e., they differ and also the sign bit is 1 which indicates negative result. Thus, the result is not correct. (Note that 152 is the maximum positive number that can be represented in a 8-bit register).

Example 4.16. Add – 87 and – 65.

Solution.

– 87	1,010 1001	(2's complement of – 87)
– 65	1,011 1111	(2's complement of – 65)
–152	(1) 0,110 1000	

In this, carry from MSB of the addition of the number is 0 and carry from the sign bit is 1, i.e, they differ and also the sign bit is 0 which indicates positive result. Thus, the result is not correct. The answer is wrong as the decimal number –152 lies outside the range of 8 bit arithmetic.

From the above two examples, it is clear that the overflow conditions occur when the carry of MSB of numbers (i.e., carry into the sign bit) is not equal to the carry out of the sign bit. If the two carries are applied to an exclusive OR gate, an overflow would be detected when the output of the gate is 1.

It is also important to note that there is an overflow if the addition of two positive numbers produce a negative sign bit or addition of two negative numbers produce a positive sign bit. In other words, the overflow produces an erroneous sign reversal.

Example 4.17. *Perform the arithmetic operations (+35) + (+ 40) and (– 35) + (– 40) with binary numbers in signed 2's complement representation. Use seven bits to accommodate each number together with its sign. Show that overflow occurs in both cases that the last two carries are unequal and that there is a sign reversal.* **(AMIE, S '94)**

Solution.

$$\begin{array}{rr}
+35 & 0,\,100011 \\
\underline{+40} & \underline{0,\,101000} \\
+75 & 1,\,001011
\end{array}$$

There is a carry (1) from MSD from addition of two numbers and no carry (0) from the sign bit. Also 1 in sign bit indicates negative result. Hence, there is an overflow and the result is not correct.

$$\begin{array}{rrl}
-35 & 1,\,011101 & \text{(2's component of } -35) \\
\underline{-40} & \underline{1,\,011000} & \text{(2's component of } -40) \\
-75 & (1)0,\,110101 &
\end{array}$$

There is no carry (0) from MSD from addition of two numbers and a carry (1) from the sign bit. Also, 0 in sign bit indicates positive result. Hence, there is an overflow and the result is not correct.

From the above cases, it is clear that there is an overflow if two numbers are positive but the result of addition has a negative sign bit and if two numbers are negative but the result of addition has a positive sign bit, i.e., overflow produces an erroneous sign reversal.

4.6.2. Floating Point Arithmetic

When two floating-point numbers of the same sign are added, a carry may be generated out of high-order bit position. For example, if $.643 \times 10^{99}$ is added to $.4854 \times 10^{99}$, the sum of mantissa exceeds 1. Thus, the mantissa is shifted right and exponent is increased by 1 resulting a value of 100 for the exponent. If the exponent part of the word of computer cannot store more than two digits (largest number that can be stored in a memory location). The condition is called an overflow condition and the arithmetic unit will intimate an error condition. The overflow can also occur in multiplication and division as in the following examples:

$$(1111 \times 10^{51}) \times (4444 \times 10^{50}) = .4937 \times 10^{101} \quad \text{overflows}$$
$$(.9887 \times 10^{1}) \div (1000 \times 10^{99}) = .9837 \times 10^{101} \quad \text{overflows}$$

An underflow condition occurs when the result is smaller than the smallest number which could be stored in the word of a computer. For example, if the word contains only two digits of exponent, then

$$.5462 \times 10^{-99} = .5427 \times 10^{-99}$$
$$.0035 \times 10^{-99} = .35 \times 10^{-101} \quad \text{Underflows}$$

Underflow can also occur in multiplication and division.

4.7. REPRESENTATION OF CHARACTERS

Apart from storage and processing of numeric data, computer system store and process character data also. Information processing using computers requires, generally, 26 capital and 26 small English letters, 10 digits and many special characters (32) like +, –, (.), etc. Total number of characters to be coded is thus 26 + 26 + 10 + 32 = 94. To represent them on computer we require a string sequence of 0 and 1 and minimum 7 bits will be required to encode the characters which can code $2^7 = 128$ characters.

Coding of characters has been standardised to facilitate exchange of recorded data between computers. The two popular standard codes used for modern data processing are:

(*a*) Extended Binary Coded Decimal Interchange Code (EBCDIC).

(*b*) American Standard Code for Information Interchange (ASCII).

Data represented in ASCII format can be easily converted in EBCDIC format, and *vice versa* to suit requirements of a particular computer system.

4.7.1. EBCDIC

The EBCDIC (pronounced as eb – si – dic) is an 8 bit code primarily used by IBM and IBM compatible computer system. In addition to the various characters, it allows a large variety of printable special characters and non-printable control characters. The control characters are used to control such activities as printer vertical spacing, movement of cursor or the terminal screen, etc. All 256 (2^8) bit combinations have not yet been assigned characters, so the code can still grow as new requirements develop. The representation of each character is divided into two 4 bit portion, a zone portion on the left and numeric portion on the right.

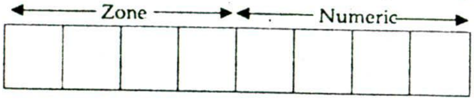


Fig. 4.8

Table 4.6 shows the EBCDIC coding scheme used to represent alphabets, digits and a few special characters. It is observed that a digit has its binary representation as the numeric portion of its code.

Table 4.6: EBCDIC Coding Scheme

Character	*Zone*	*Numeric*	*Hexadecimal Equivalent*	*Character*	*Zone*	*Numeric*	*Hexa decimal Equivalent*	*Character*	*Zone*	*Numeric*	*Hexadecimal Equivalent*
				S	1110	0010	E2	blank	0100	0000	40
A	1100	0001	C1	T		0011	E3	.		1011	4B
B		0010	C2	U		0100	E4	<		1100	4C
C		0011	C3	V		0101	E5	(	↓	1101	4D
D		0100	C4	W		0110	E6	+	0100	1110	4E
E		0101	C5	X		0111	E7	&	0101	0000	50
F		0110	C6	Y	↓	1000	E8	$		1011	5B
G		0111	C7	Z	1110	1001	E9	*		1100	5C
H	↓	1000	C8					)	↓	1101	5D
I	1100	1001	C9					;	0101	1110	5E
J	1101	0001	D1					–	0110	0000	60
K		0010	D2	0	1111	0000	F0	/		0001	61
L		0011	D3	1		0001	F1	,		1011	6B
M		0100	D4	2		0010	F2	%		1100	6C
N		0101	D5	3		0011	F3	>	↓	1110	6E
O		0110	D6	4		0100	F4	?	0110	1111	6F
P		0111	D7	5		0101	F5	:	0111	1010	7A
Q	↓	1000	D8	6		0110	F6	#		1011	7B
R	1101	1001	D9	7		0111	F7	@	↓	1100	7C
				8	↓	1000	F8	=	0111	1110	7E
				9	1111	1001	F9				

The internal coded representation of the string MANJU in EBCDIC is

1101 0100	1100 0001	1101 0101	1101 0001	1110 0100
M	A	N	J	U

4.7.2. ASCII

These codes are of two types : ASCII-7 and ASCII-8. ASCII-7 is a 7 bit code that allows 128 (2^7) different characters. Table 4.7 shows the ASCII-8 coding scheme.

Table 4.7: ASCII-8 Coding Scheme

Character	*Zone*	*Numeric*	*Hexadecimal Equivalent*	*Character*	*Zone*	*Numeric*	*Hexa decimal Equivalent*	*Character*	*Zone*	*Numeric*	*Hexadecimal Equivalent*
0	0101	0000	50	A	1010	0001	A1	P	1011	0000	B0
1		0001	51	B		0010	A2	Q		0001	B1
2		0010	52	C		0011	A3	R		0010	B2
3		0011	53	D		0100	A4	S		0011	B3
4		0100	54	E		0101	A5	T		0100	B4
5		0101	55	F		0110	A6	U		0101	B5
6		0110	56	G		0111	A7	V		0110	B6
7		0111	57	H		1000	A8	W		0111	B7
8		1000	58	I		1001	A9	X		1000	B8
9	0101	1001	59	J		1010	AA	Y		1001	B9
				K		1011	AB				
				L		1100	AC	X	1011	1010	BA
				M		1101	AD				
				N		1110	AE				
				O	1010	1111	AF				

4.7.3. Collating Sequence

The value of an alphanumeric or alphabetic data element is usually the name of some object. In order to arrange them in some desired order, it is necessary to have some assigned ordering system among the characters used by the computer. The order in which an alphanumeric code causes the characters in the code to be arranged based on their numeric codes, is called the collating sequence for the code and it varies from computer to computer.

In EBCDIC, the zones values of characters A through 9 increases from the equivalent of decimal 12 to 15. This means a computer using EBCDIC code for its internal representation of characters will treat numeric characters to be greater than alphabetic characters. On the other hand, a computer using ASCII-B code will place numbers ahead of letters during ascending order.

For example, 23 > 2C > C2 in EBCDIC but in ASCII-8, C2 > 2C > 23.

4.7.4. Differences between EBCDIC and ASCII-8

There is no clear superiority of either EBCDIC and ASCII-8, but there are two important differences. The collating sequence for EBCDIC has the numerals flow the letters: for ASCII-8, the reverse is true. Hence, documents coded and sorted under one system would be in a different order if they were coded and sorted by the other. Second, the ASCII codes for the alphabet progress consecutively by +1, whereas EBCDIC has two gaps (between I and J and between R and S) which proves annoying in certain programming situations.

4.7.5. Unicode

Although ASCII and EBCDIC contain some foreign language symbols, both are clearly insuficient in a global computer market. **Unicode** solves this problem for most languages by expanding the number of available bits to 16. Because 16 bits is enough to code more than 65,000 characters, Unicode can represent many, if not most, of the world's languages. Some languages are not represented because more research is needed to determine how best to encode their scripts.

REVIEW QUESTIONS SET

1. What is a sign bit?
2. What are mantissa and exponent of a real number?
3. What do you understand by a floating point number? Give some examples.
4. What is normalised floating point number? Give some examples and write their mantissa and exponents.
5. What is signed magnitude scheme? Give one example of this scheme.
6. What is the advantage of using normalized mantissa? Compare the advantages and disadvantages of BCD arithmetic with respect to binary arithmetic.
7. What are fixed and floating point representations? Describe their comparative advantages and disadvantages.
8. Why excess representation is preferred for exponent in floating point representation?
9. Which signed representation in binary number system is most efficient in arithmetic? Explain.
10. What is the range of numbers represented if 10 bits are used (*a*) in signed magnitude form, (*b*) in 2's complement form.
11. Discuss overflow and underflow phenomena which occurs in a digital computer.
12. What is a weighted code?
13. What is the difference between ASCII and EDCDIC codes?
14. What is collating sequence?
15. What is a parity bit? What are the two kinds of parity?
16. Convert the decimal numbers to excess-3 code.
    (*a*) 5 (*b*) 62 (*c*) 55 (*d*) 395
17. Convert excess-3 code to decimal form:
    (*a*) 1011 (*b*) 0101 1011 1000
18. Convert the following gray codes to binary form:
    (*a*) 0111 (*b*) 1000 (*c*) 11100100011
19. Convert the following binary numbers to gray code:
    (*a*) 1000110111 (*b*) 110100110
20. Convert the following 5421 code number into decimal:
    0010 1000 1010 **(AMIE, S '94)**
21. Encode the following decimal numbers into BCD numbers:
    (*a*) 45 (*b*) 732
22. Use normalized floating point representation to find
    1011011.0101 + 1101101.0101
23. Add (*a*) 8 and 9, (*b*) 15 and 17, (*c*) 120 + 215 using BCD representation.
24. Represent the decimal number 8620 (*a*) in BCD, (*b*) in excess −3 code, (*c*) in 2 − 4 − 2 − 1 code.
25. Represent the number $(+47.5)_{10}$ with normalized integer mantissa of 13 bits and an exponent of 7 bits:
    (*a*) As a binary number (radix 2);
    (*b*) As a binary coded octal (radix 8);
    (*c*) As a binary coded hexadecimal (radix 16).
    Show that the mantissa is same in all the three cases (except for left or right zero), but that the value of exponent changes. Determine the largest positive number that the 20 bit register can hold in each of the three representations. What is the advantage of using radix 8 (or 16) over radix 2 for floating point numbers in registers. **(AMIE, S '96)**

ANSWERS TO REVIEW QUESTIONS SET

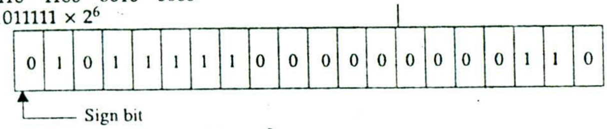
1. A standard convention has been formulated to represent signed numbers in computers. The sign of a number is represented by using an extra bit (0 or 1) at the extreme left of the number. The symbol 0 is used to represent '+' sign and '1' to represent '–' sign. These binary bits are called the sign bits.

16. (*a*) 1000 (*b*) 1001 0101 (*c*) 1000 1000 (*d*) 0110 1100 1000
17. (*a*) 8 (*b*) 285
18. (*a*) 0101 (*b*) 1000 (*c*) 10111000010
19. (*a*) 1100101100 (*b*) 101110101
20. 2, 5, 7

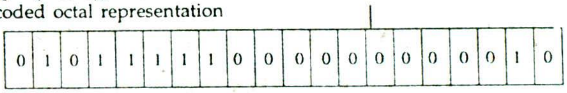
21. (*a*) 0100 0101 (*b*) 0111 0011 0010

24. (*a*) 1000 0110 0010 0000
(*b*) 1011 1001 0101 0011
(*c*) 1110 1100 0010 0000

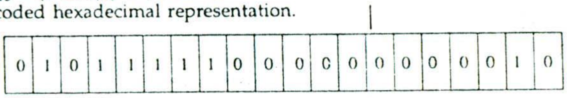
25. (*a*) $.1011111 \times 2^6$



(*b*) $(+\ 47.5)_{10} = (+\ 57.4)_8 = .574 \times 8^2$
Binary coded octal representation



(*c*) $(+\ 47.5)_{10} = (+\ 2F.8)_{16} = .2F8 \times 16^2$
Binary coded hexadecimal representation.



Largest numbers are of the order 2^{63}, 8^{63} and 16^{63}.

5

BOOLEAN ALGEBRA AND LOGIC CIRCUITS

5.0. INTRODUCTION

For centuries mathematicians felt that there was a connection between mathematics and logic circuits but no one before George Boole could find this missing link. In 1854 he invented symbolic logic known as Boolean algebra. Boolean algebra provides a mathematical basis which is essential for proper understanding of logic circuits. This chapter deals with the rules of Boolean algebra and their applications in the design of logic circuits.

5.1. AXIOMS OF BOOLEAN ALGEBRA

For any elements a, b and c of the set B on which two binary operations + and . and a unary operation denoted – or ' are defined, and 0 and 1 denote two distinct elements of B. Then

1. Commutative Laws
   (a) $a + b = b + a$ (b) $a \cdot b = b \cdot a$
2. Distributive Laws
   (a) $a + (b.c) = (a+b) \cdot (a+c)$ (b) $a.(b+c) = (a.b) + (a.c)$
3. Identity Laws
   (a) $a + 0 = a$ (b) $a.1 = a$
4. Complement Laws
   (a) $a + \bar{a} = 1$ (b) $a \cdot \bar{a} = 0$

5.2. BASIC THEOREMS

Let a, b, c be any three elements in a Boolean algebra B. Then

(i) Idempotent Laws
(a) $a + a = a$ (b) $a \cdot a = a$

(ii) Boundedness Laws
(a) $a + 1 = 1$ (b) $a \cdot 0 = 0$

(iii) Absorption Laws
(a) $a + (a \cdot b) = a$ (b) $a \cdot (a + b) = a$

(iv) Associative Laws
(a) $(a + b) + c = a + (b + c)$ (b) $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

(v) (Uniqueness of Complement)
If $a + x = 1$ and $a \cdot x = 0$, then $x = \bar{a}$

(vi) (Involution Law): $\overline{(\bar{a})} = a$

(vii) (a) $\bar{0} = 1$ and (b) $\bar{1} = 0$

De Morgan's Laws: (a) $\overline{(a + b)} = \bar{a} \cdot \bar{b}$
(b) $\overline{(a \cdot b)} = \bar{a} + \bar{b}$

5.3. LOGICAL ADDITION (OR OPERATION)

Each variable in Boolean algebra has either of two values: true or false (1 or 0). For instance, in logic equation, $A + B = C$, each of the variables A, B and C may have only the values 0 or 1. We can define the + *ve* symbol by listing all possible combinations for A and B and the resulting values of $A + B$.

Table 5.1 : Truth Table of Logical Addition (OR Operations)

INPUT		OUTPUT
A	B	$C = A + B$
0	0	0
1	0	1
0	1	1
1	1	1

Table 5.1 is a logical addition table (truth table of logical addition) and could represent binary addition table except for the last entry. The + symbol, therefore, does not have the normal meaning but is a logical addition and is referred to as OR operation. The equation $A + B = C$ can be read as A OR B equals C. This concept can be extended to any number of variables. To avoid ambiguity, a number of other symbols have been recommended as replacements for the + sign, for example, U and V.

5.4. LOGICAL MULTIPLICATION (AND OPERATION)

A second important operation in Boolean algebra is logical multiplication and is referred to as AND operation. The logical multiplication of two variables A and B is expressed as $A.B$ and is read as A and B. The truth table for logical multiplication of two variables is shown in Table 5.2.

Table 5.2 : Truth Table of Logical Multiplication (AND Operation)

INPUT		OUTPUT
A	B	$C = A . B$
0	0	0
1	0	0
0	1	0
1	1	1

5.5. COMPLEMENTATION

Boolean algebra uses an operation called complementation and the symbol used for this is – or ′ and this can be defined as $\overline{0} = 1$ and $\overline{1} = 0$. Thers $\overline{A}$ means the complement of A and read as NOT A. The term or terms overlined are said to be negated and the process of complementing is called negation.

Table 5.3 : Truth table for logical complementation

INPUT	OUTPUT
A	$B = \overline{A}$
0	1
1	0

5.6. EXAMPLES TO ILLUSTRATE LOGICAL OPERATIONS

Electrical switches are good examples to illustrate OR, AND and many Boolean theorems.

A switch has only two states: either closed or open. When the two switches are connected in parallel, the current will flow in the circuit when either switch is in closed position. The current will not flow at all when both switches are in open position. If the flowing of current is taken as ON and not flowing as OFF, and if we assume closed = 1, open = 0, ON = 1 and OFF = 0, then behaviour of two switches connected in parallel can be tabulated as shown in Table 5.4.

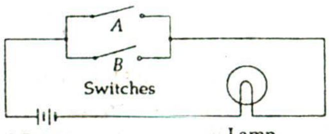


Fig. 5.1: Parallel circuit

Table 5.4 : Behaviour of Two Switches in Parallel Circuit

Switch A	*Switch B*	*Bulb C*
Open (0)	Open (0)	OFF (0)
Closed (1)	Open (0)	ON (1)
Open (0)	Closed (1)	ON (1)
Closed (1)	Closed (1)	ON (1)

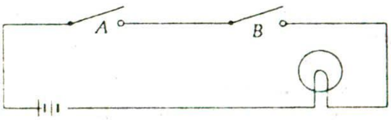


Fig. 5.2: Series circuit

This is precisely the property described by the truth table for logical addition (OR operation).

Similarly, when two switches connected in series as shown in Fig. 5.2, the lamp will light up when both *A* and *B* are closed. Table 5.5 shows the behaviour of two switches in series circuit.

Table 5.5 : Behaviour of Two Switches in Series Circuit

Switch A	*Switch B*	*Bulb C*
Open (0)	Open (0)	OFF (0)
Closed (1)	Open (0)	OFF (0)
Open (0)	Closed (1)	OFF (0)
Closed (1)	Closed (1)	ON (1)

This is precisely the property described by the truth table for logical multiplication (AND operation).

5.7. LOGIC GATE

A logic gate is simply an electronic circuit that operates on one or more input signals to produce an output signal. Gates are digital (two-state) circuits and often called logic circuits because they can be analyzed with Boolean algebra. For example, the electronic circuit which performs OR operation is called OR gate, the circuit which performs AND operation is called AND gate and so on.

5.7.1. OR Gate

The OR gate has two or more inputs but only one output. An input signal applied to gate has only two stable states either 1 (high) or 0 (low). In case of a 2 input OR gate the output 1 (high) if at least any one of the inputs is high. The output is 0 if all inputs are 0. The logic symbol (the symbol used to represent a circuit performing specific function) for OR gate of two variables is shown in Fig. 5.3.

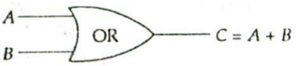


Fig. 5.3: Logic symbol for OR gate

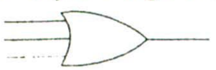


Fig. 5.4: Symbols for 3 input OR gate

Logic symbols for OR gates with more than two inputs are shown in Fig. 5.4.

5.7.2. AND Gate

AND gate has two or more inputs but only one output. In case of a 2-input gate the output is 1 (high) only if both inputs are 1 (high), otherwise the output is 0 (low). Logic symbol for AND gate of two inputs is shown in Fig. 5.5.

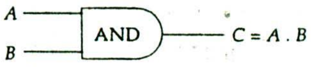


Fig. 5.5: Logic symbol for AND gate

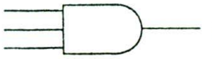


Fig. 5.6: Logic symbol for 3 input AND gate

Logic symbols for AND gates with two more inputs are shown in Fig. 5.6.

Suppose A and B are assigned the sequence of bits as A = 101010, B = 111001. Then OR gate will produce the sequence $A + B$ = 111011 and AND gate will produce the sequence $A.B$ = 101000.

Both + and . obey the associative law, i.e.,

$$(A + B) + C = A + (B + C), \text{ and}$$
$$(A \,.\, B) \,.\, C = A \,.\, (B \,.\, C)$$

This means that one can write $A + B + C$ and $A.B.C$ without ambiguity; no matter in what order the operation is performed. The truth table for three input OR gate and AND gate are shown in Tables 5.6 and 5.7, respectively.

Table 5.6 : Truth Table for Three Input OR Gate

INPUT			OUTPUT
A	*B*	*C*	*D*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 5.7 : Truth Table for Three Input AND Gate

INPUT			OUTPUT
A	*B*	*C*	*D*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
0	1	0	0
0	1	1	0
1	1	1	1

5.7.3. NOT Gate

A NOT gate has only one input, and only one output signal. It is also called INVERTER. Its output is the complement of the input signal. The output is 1 (high) if the input is 0 (low). The output is 0 when input is 1. Table 5.8 shows the truth table of NOT gate and Fig. 5.7 shows logic symbol for a NOT gate.

Table 5.8 : Truth Table for NOT Gate

INPUT A	OUTPUT $B = \overline{A}$
0	1
1	0

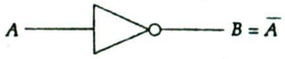


Fig. 5.7: Logic symbol for NOT gate

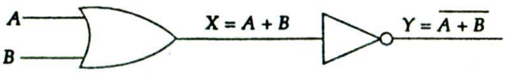
5.7.4. NOR and NAND Gate

Boolean algebra has two binary operators called AND and OR, and a unary operator NOT. Other binary operators can be defined in terms of them. The NOR function is the complement of OR function and its name is an abbreviation of not OR and uses an OR symbol followed by a small circle. Similarly, NAND is the complement of AND and is an abbreviation of not – AND, and indicated by a symbol which consists of an AND symbol followed by a small circle. Table 5.9 shows the truth table for NOR gate. It shows that for every combination of the input signals, the output of the NOR gate is the complement of the output of the OR gate.

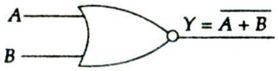
Table 5.9 : Truth Table for NOR Gate

INPUT		OUTPUT	
		OR	NOR
A	B	$X = A + B$	$Y = \overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

The circuit shown in Fig. 5.8 is a combination of OR gate followed by an inverter The output of the OR gate is $A + B$ and the output of the combination, that is, the NOR gate, is $\overline{A + B}$.



(*a*) Logic diagram of NOR function



(*b*) Logic symbol for NOR gate

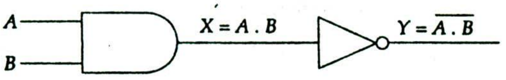
Fig. 5.8

Table 5.10 shows the truth table for NAND gate. It shows that for every combination of the input signals, the output of the NAND gate is the complement of the output of AND.

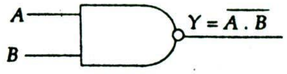
Table 5.10 : Truth table for NAND gate

INPUT		OUTPUT	
		AND	NAND
A	B	$X = A . B$	$Y = \overline{A . B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Figure 5.9 shows the logic diagram of NOR function and logic symbol for NOR gate.



(*a*) Logic diagram of NAND function



(*b*) Logic symbol for NAND gate

Fig. 5.9

5.7.5. Exclusive-OR (XOR) Gate

An OR gate recognises words with one or more 1's. The EXCLUSIVE-OR gate is different; it recognises only words that have an odd numbers of 1s. Table 5.11 is the truth table for a 2-input XOR gate which shows the input-output relationship.

Table 5.11 : Truth Table for 2-input XOR Gate

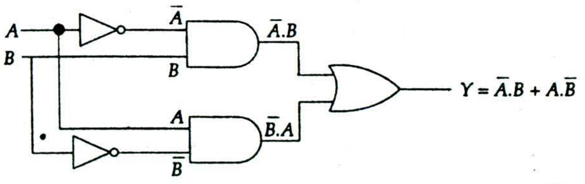
INPUT		OUTPUT
A	B	$C = \overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

In Boolean algebra, the sign ⊕ stands for XOR operation

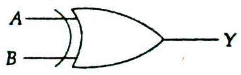
$$C = \overline{A} . B + A . \overline{B} = A \text{ XOR } B$$

$$C = A \oplus B$$

A logic circuit of XOR operation is shown in Fig. 5.10.



(*a*) Two-input XOR gate



(*b*) Symbol for two-input XOR gate

Fig. 5.10

One can see from Table 5.11 that the output is 1 only with an odd number of binary 1 inputs, with an even number of binary 1 inputs the output is 0. An XOR gate, irrespective of the number of inputs, recognises only those words which have an odd number of binary 1s. defective

Now consider a 4 input XOR gate which can be implemented by using three XOR gates as shown in Fig. 5.11.

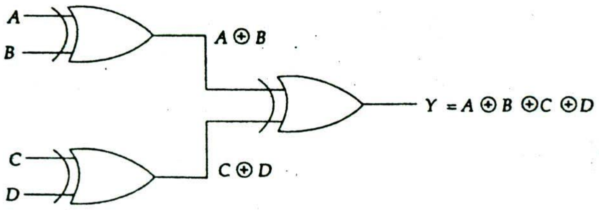


Fig. 5.11: Four-input XOR gate

Table 5.12: Truth Table for 4-input XOR Gate

INPUT				OUTPUT
A	*B*	*C*	*D*	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

We notice again from Table 5.12 that the output is 1 only with an odd number of binary 1 inputs and is 0 only with even number of 1 inputs. This property of XOR gate is used to check the parity (even or odd) of any word.

5.7.6. EXCLUSIVE-NOR (XNOR) Gate

XNOR gate is equivalent to an Exclusive OR gate followed by an inverter. The truth table for XNOR gate is given in Table 5.13. The output is 1 only when both inputs are either 0 or 1.

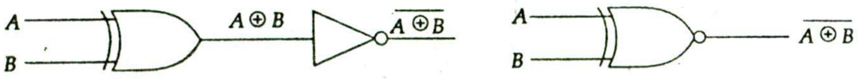
Table 5.13 : Truth Table for 2-input XNOR Gate

INPUT		OUTPUT
A	*B*	$Y = AB + \bar{A}\,\bar{B}$
0	0	1
0	1	0
1	0	0
1	1	1

XNOR operation can be written as follows:

$$\begin{aligned} Y &= AB + \bar{A}\ \bar{B} \\ &= A \text{ XNOR } B \\ &= \overline{A \oplus B} \\ &= A \odot B \end{aligned}$$

The logic diagram and logic symbol is shown in Fig. 5.12.



(*a*) XNOR gate (*b*) Symbol for XNOR gate

Fig. 5.12

5.8. INTERCONNECTING GATES

The gates can be interconnected to form gating or logic networks. The Boolean algebra expression corresponding to a given gating network can be derived by systematically progressing from input to output on the gates. Figure 5.13 shows a gating network with three inputs X, Y, and Z, and an output expression.

5.9. BOOLEAN FUNCTIONS

A digital computer operates on electrical signals which have only two states: high (1) and low (0). A signal that does not change its state in time is called a constant signal and a signal which changes in time is known as a variable signal. The variables which have only two values 1 and 0 are called Boolean variables. AND, OR and NOT are basic operations performed on Boolean variables. As in ordinary algebra, we have the concept of a function of Boolean variables or an expression containing Boolean variable. For example, consider the equation $X = A.B + C(\overline{D} + E)$. Here the variable X is a function of *A*, *B*, *C*, *D* and *E*. This is written as $X = f(A, B, C, D, E)$, *A*, *B*, *C*, *D* and *E* are Boolean variables. The right hand side of the above equation is known as Boolean expression. Each occurrence of a variable or its complement in an expression is called *literal.*

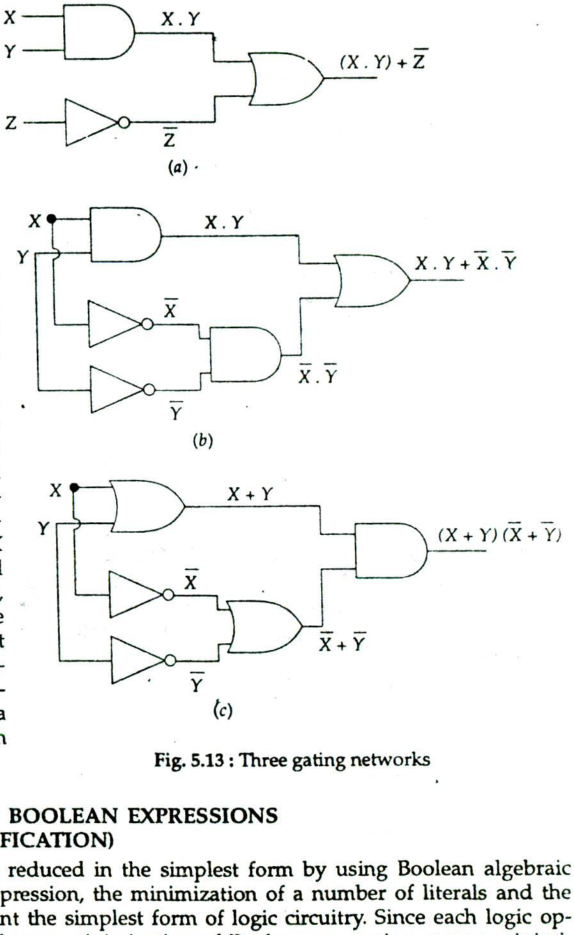


Fig. 5.13 : Three gating networks

5.10. SIMPLIFICATION OF BOOLEAN EXPRESSIONS (ALGEBRAIC SIMPLIFICATION)

A Boolean expression can be reduced in the simplest form by using Boolean algebraic laws. For a given Boolean expression, the minimization of a number of literals and the number of terms will represent the simplest form of logic circuitry. Since each logic operator represents a logic hardware, minimization of Boolean expression means minimization in cost.

Example 5.1. *Simplify the following Boolean functions*:

1. $\overline{X}\,\overline{Y}Z + \overline{X}YZ + X\overline{Y}$
2. $XY + \overline{X}Z + YZ$
3. $Z(Y + Z)(X + Y + Z)$
4. $(X + \overline{Y} + \overline{Z})(X + \overline{Y} + Z)$

Solution.

1. $\overline{X}\,\overline{Y}Z + \overline{X}YZ + X\overline{Y}$
   $= \overline{X}Z\,(\overline{Y} + Y) + X\overline{Y}$
   $= \overline{X}Z + X\overline{Y}$
2. $XY + \overline{X}Z + YZ$
   $= XY + \overline{X}Z + YZ\,(X + \overline{X})$
   $= XY + \overline{X}Z + XYZ + \overline{X}YZ$
   $= XY\,(1 + Z) + \overline{X}Z\,(1 + Y)$
   $= XY + \overline{X}Z$
3. $Z\,(Y + Z)\,(X + Y + Z)$
   $= (ZY + ZZ)\,(X + Y + Z)$
   $= (ZY + Z)\,(X + Y + Z)$
   $= Z\,(Y + 1)\,(X + Y + Z)$
   $= Z\,(X + Y + Z)$
   $= ZX + ZY + ZZ$
   $= ZX + ZY + Z$
   $= Z\,(X + Y + 1)$
   $= Z$
4. $(X + \overline{Y} + \overline{Z})\,(X + \overline{Y} + Z)$
   $= XX + X\overline{Y} + XZ + \overline{Y}X + \overline{Y}\,\overline{Y} + \overline{Y}Z + \overline{Z}X + \overline{Z}\,\overline{Y} + \overline{Z}Z$
   $= X + X\overline{Y} + XZ + \overline{Y}X + \overline{Y} + \overline{Y}Z + \overline{Z}X + \overline{Z}\,\overline{Y} + 0$
   $= X\,(1 + \overline{Y} + Z + \overline{Y} + \overline{Z}) + \overline{Y}\,(1 + Z + \overline{Z})$
   $= X.1 + \overline{Y}.1$
   $= X + \overline{Y}$

5.11. SUM OF PRODUCTS FORMS OF BOOLEAN EXPRESSIONS

A Boolean expression E is said to be in a sum of products form if E is a sum of two or more fundamental products of variables none of which is included in another. The variables may or may not be complemented. The following are examples of sum of products expression:

(*a*) $X\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}Z$
(*b*) $X + X\overline{Y} + \overline{Y}Z$
(*c*) $AB + CD$
(*d*) $ABC + A\overline{B}C + \overline{A}BC + \overline{A}B\overline{C}$

Sometimes a product term may consist of a single variable. Each expression consists of two or more AND (product) terms that are ORed together.

5.12. PRODUCT OF SUMS FORMS OF BOOLEAN EXPRESSIONS

A Boolean expression E is said to be in a product of sums form if E consists of several sum terms logically multiplied. The variables may or may not be complemented. The following are examples of product of sums expression:

(*a*) $(X + Y)\,(\overline{X} + \overline{Y})$
(*b*) $(A + B)\,(C + D)$
(*c*) $(A + B + C)\,(A + \overline{B} + C)\,(A + \overline{B} + \overline{C})$

Each expression consists of two or more OR (sum) terms that are ANDed together.

5.13. CANONICAL AND STANDARD FORMS

When each term of a Boolean expression contains all variables in normal or complemented form it is said to be in the canonical form. When a sum of products form of Boolean expression is in canonical form, each product term is called a *minterm*. When a product of sums form of Boolean expression is in canonical form, each sum term is called a *maxterm*. The canonial form of a sum of products expression is also called minterm canonical form or standard sum of products and the canonical form of a product of sums expression is also called maxterm canonical form or standard product of sums.

In case of n variables, the maximum possible number of minterms and maxterms are 2^n. The 2^n different terms may be obtained by a method similar to one shown in Table 5.14.

Table 5.14

X	Y	Z		*Minterms*		*Maxterms*
0	0	0	$\overline{X}$	$\overline{Y}$	$\overline{Z}$	$X + Y + Z$
0	0	1	$\overline{X}$	$\overline{Y}$	Z	$X + Y + \overline{Z}$
0	1	0	$\overline{X}$	Y	$\overline{Z}$	$X + \overline{Y} + Z$
0	1	1	$\overline{X}$	Y	Z	$X + \overline{Y} + \overline{Z}$
1	0	0	X	$\overline{Y}$	$\overline{Z}$	$\overline{X} + Y + Z$
1	0	1	X	$\overline{Y}$	Z	$\overline{X} + Y + \overline{Z}$
1	1	0	X	Y	$\overline{Z}$	$\overline{X} + \overline{Y} + Z$
1	1	1	X	Y	Z	$\overline{X} + \overline{Y} + \overline{Z}$

Each minterm is obtained from an AND term of n variables, with each variable being complemented if the corresponding bit of the binary number is 0 and normal if 1, and each maxterm is obtained from an OR term of n variables with each variable being normal if the corresponding bit is 0 and complemented if 1.

The following are the examples of the canonical sum of products Boolean expression:

(*i*) $X\,Y + \overline{X}\,Y + X\,\overline{Y}$

(*ii*) $X\,Y\,Z + \overline{X}\,Y\,Z + X\,Y\,\overline{Z} + \overline{X}\,\overline{Y}\,\overline{Z}$

The examples of canonical product of sums are:

(*i*) $(X + Y)\,(\overline{X} + Y)\,(X + \overline{Y})$

(*ii*) $(X + Y + Z)\,(\overline{X} + Y + Z)\,(X + Y + \overline{Z})\,(\overline{X} + \overline{Y} + \overline{Z})$

5.14. BOOLEAN FUNCTIONS AND TRUTH TABLES

A table which lists the value of the dependent variable for each set of values of the independent variables is known as a truth table. The steps of obtaining truth table from a given Boolean function are given below:

1. Form a table with one column for each of the independent variables and one column (preferably last column) for the dependent variable.
2. If there are n independent variables, start taking 0s as values of all independent variables and form successive rows using the natural binary counting sequence till the count become $(2^n - 1)$.
3. Evaluate the value of the dependent variable by substituting the value of the independent variable and enter the evaluated value in the appropriate row.

The truth table of Boolean function $F\,(A, B, C) = A + BC$ is shown in Table 5.15.

Table 5.15 : Truth Table of $F(A, B, C) = A + BC$

A	B	C	$A + BC$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

5.15. BOOLEAN EXPRESSION FROM TRUTH TABLE

The Boolean expression from a given truth table can be obtained either in sum of products form or product of sums form. The steps described below can be applied to obtain the sum of products form of Boolean expression:

1. Inspect the column corresponding to dependent variable from the top row and pick the row with a 1 entry.
2. A term in the Boolean expression corresponding to this row is obtained by 'AND'ing all the independent variables in the truth table.
3. Repeat steps 1 and 2 till all the entries in the dependent variable column is completed. Obtain an Boolean expression by 'OR'ing the terms corresponding to the 1 entries of dependent variable.

The method for obtaining the product of sums form of Boolean expression is similar to the one given above, instead of picking the rows of 1 entry in dependent variable, the rows of 0 entry are to be picked up. The individual term is formed by taking the sum of the independent variables. The expression is obtained by ANDing the individual terms.

We shall now consider the truth table given in Table 5.16 and find out the corresponding Boolean expression in sum-of-products and product of sums forms.

Table 5.16

Inputs			*Ouput*	*Product Terms*	*Sum Terms*
A	B	C			
0	0	0	1	$\overline{A}\,.\,\overline{B}\,.\,\overline{C}$	
0	0	1	1	$\overline{A}\,.\,\overline{B}.\,C$	
0	1	0	1	$\overline{A}\,.\,B.\,\overline{C}$	
0	1	1	0		$A + \overline{B} + \overline{C}$
1	0	0	1	$A\,.\,\overline{B}.\,\overline{C}$	
1	0	1	1	$A\,.\,\overline{B}\,.\,C$	
1	1	0	0		$\overline{A} + \overline{B} + C$
1	1	1	0		$\overline{A} + \overline{B} + \overline{C}$

The sum-of-products expression will be as follows:

$$\begin{aligned} F &= \overline{A}\,.\,\overline{B}\,.\,\overline{C} + \overline{A}\,.\,\overline{B}\,.\,C + \overline{A}\,.\,B\,.\,\overline{C} + A\,.\,\overline{B}\,.\,\overline{C} + A\,.\,\overline{B}\,.\,C \\ &= \overline{A}\,.\,\overline{B}\,(\overline{C} + C) + A\,.\,\overline{B}\,(\overline{C} + C) + \overline{A}\,.\,B\,.\,\overline{C} \\ &= \overline{A}\,.\,\overline{B} + A\,.\,\overline{B} + \overline{A}\,.\,B\,.\,\overline{C} = \overline{B} + \overline{A}\,.\,B\,.\,\overline{C} = \overline{B} + \overline{A}\,.\,\overline{C} \end{aligned}$$

The product-of-sums expression will be as follows:

$$F = (A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C}) = (A + \overline{B} + \overline{C})(\overline{A} + \overline{B})$$

$$= \overline{B}(A + \overline{A}) + \overline{B}(1 + \overline{C}) + \overline{A} . \overline{C} = \overline{B} + \overline{A} . \overline{C}$$

This expression happens to be the same as for the sum-of-products solution.

5.16. IMPLEMENTING LOGIC EXPRESSION WITH LOGIC GATES

A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR and NOT gates. The logic diagram include an inverter circuit for every variable present in its complement form. An AND - OR gates can be realised by using NAND gates and OR-AND gates by NOR gates. To find simpler circuits, one must know how to manipulate to obtain equal and simple expression.

(*a*) Logic Expression in Sum of Products Form

Take two simple logic expressions:

(*i*) $F_1 = AB + CD$

(*ii*) $F_2 = \overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,Z + X\,\overline{Y}'$

Expression (*i*) will require two AND gates and one OR gate and (*ii*) will require three AND gates and one OR gate. Figs. 5.14 and 5.15 show the implementation of the logic expression with logic gates.

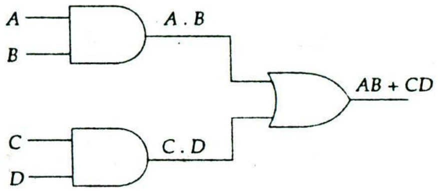


Fig. 5.14 : Implementation of $F_1 = AB + CD$ with AND - OR network

(*b*) Logical Expression in Product of Sums Form

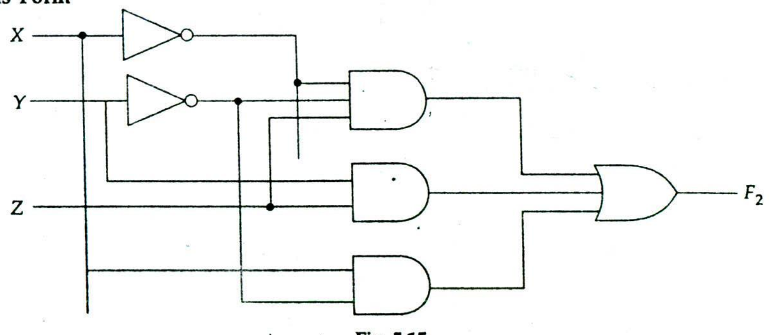


Fig. 5.15

Take an example.

$F_1 = (A + B)\cdot(C + D)$

The expression will require two OR gates and one AND gate. Fig. 5.16 shows the implementation of the given expression using OR-AND network.

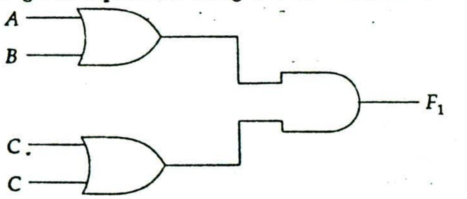
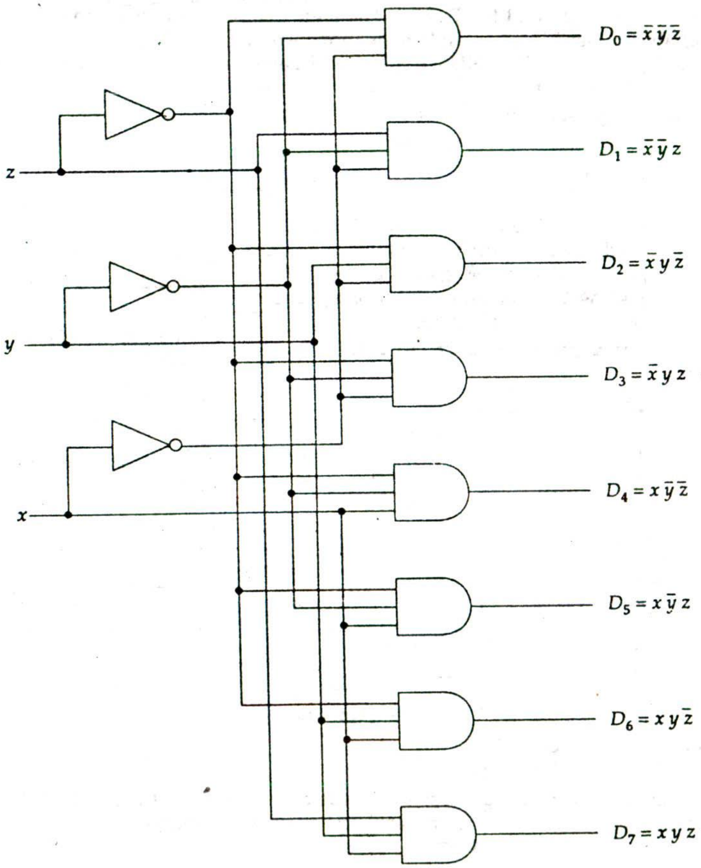


Fig. 5.16 : Implementation of $F_1 = (A + B) . (C + D)$ with OR-AND network

5.17. DECODER

A decoder is a combinational circuit that converts binary information from n input lines to a maximum 2^n unique output lines. The primary application of a decoder is that of addressing, where the n-bit input X is interpreted as an address used to select one of 2^n output lines. The decoders are called n to m lines decoders when $m \leq 2^n$. Another name for it is a 1-of-m decoder because only 1 of m output lines has a high voltage. As an example, consider 3 - to - 8 lines decoder circuit of Fig. 5.17.



$D_0 = \bar{x}\,\bar{y}\,\bar{z}$
$D_1 = \bar{x}\,\bar{y}\,z$
$D_2 = \bar{x}\,y\,\bar{z}$
$D_3 = \bar{x}\,y\,z$
$D_4 = x\,\bar{y}\,\bar{z}$
$D_5 = x\,\bar{y}\,z$
$D_6 = x\,y\,\bar{z}$
$D_7 = x\,y\,z$
z
y
x

Fig. 5.17 : A 3 - to - 8 line decoder

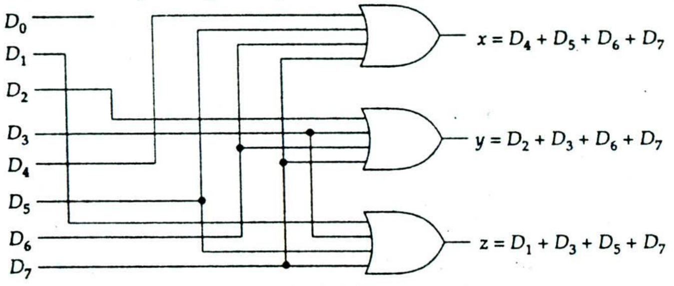
The three inputs are decoded into eight outputs. This is a binary to octal decoder, a circuit that converts from binary to octal. The truth table of the circuit is given in Table 5.17. Note that output variables are mutually exclusive because only one output can be equal to 1 at any time.

Table 5.17 : Truth Table of a 3-to-8 Line Decoder

Input			*Output*							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

5.18. ENCODER

An encoder does the reverse operation of a decoder, that is, it gets one of the active input signal and converts it into the coded output signal. An encoder has 2^n (or less) input lines and n output lines. An example of an encoder is shown in Fig. 5.18. The octal-to-binary encoder consists of eight inputs, one for each of the eight digits and three outputs that generate the corresponding binary number.



D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$

Fig. 5.18 : Octal-to-binary encoder

Note that D_0 is not connected to any OR gate; the binary output must be all 0s in this case.

5.19. MULTIPLEXER

Multiplexer means many - to - one. A multiplexer is a circuit with many inputs but only one output. The selection of a particular input line is controlled by a set of control signal lines. Normally, there are 2^n input lines and n control lines whose bit combinations determine which input is selected. A multiplexer is also called a data selector, since it selects one of many input and steers the binary information to the output line. A multiplexer is often abbreviated as MUX.

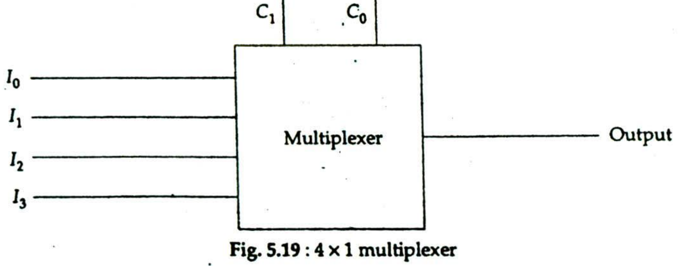


Fig. 5.19 : 4 × 1 multiplexer

Consider a multiplexer which has 4 input lines, 2 control lines and one output line. Using the two control signals one must be able to select and transmit any one of the input lines to the output line.

For example, when $C_0 = 0$ and $C_1 = 0$, one may want the input I_0 to be transmitted to the output. If $C_0 = 1$ and $C_1 = 0$, the I_1 value must be transmitted to the output. Similarly, other expected possibilities are shown in Table 5.18.

Table 5.18

C_1	C_0	*Expected output*
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

It is routine to verify that the circuit shown in Fig. 5.20 works as a 4 × 1 multiplexer explained above.

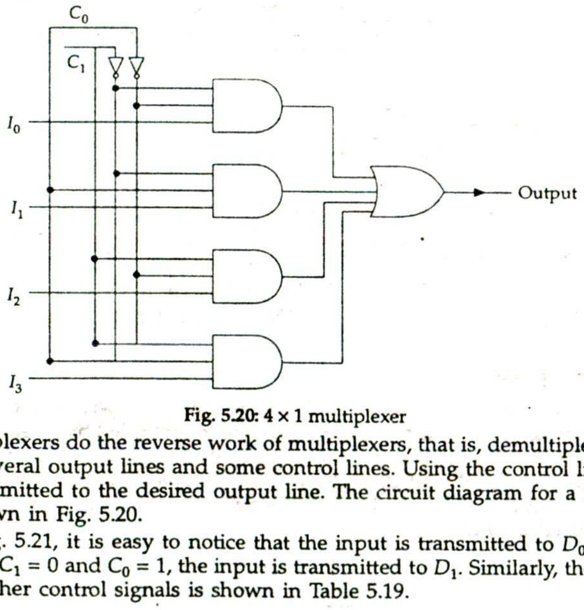


Fig. 5.20: 4 × 1 multiplexer

Demultiplexers do the reverse work of multiplexers, that is, demultiplexers have one input line, several output lines and some control lines. Using the control lines, the input data are transmitted to the desired output line. The circuit diagram for a 1 × 4 demultiplexer is shown in Fig. 5.20.

From Fig. 5.21, it is easy to notice that the input is transmitted to D_0 when $C_0 = 0$, $C_1 = 0$. When $C_1 = 0$ and $C_0 = 1$, the input is transmitted to D_1. Similarly, the transmission for various other control signals is shown in Table 5.19.

Table 5.19

Demultiplexer		
C_1	C_0	*Output*
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

5.20. ARITHMETIC CIRCUITS

A computer's circuits can respond only to binary numbers. Arithmetic operations, such as addition, subtraction, multiplication, division, etc., are performed in the binary form in a digital computer. Logic circuits of some basic arithmetic operations are discussed in the following sections.

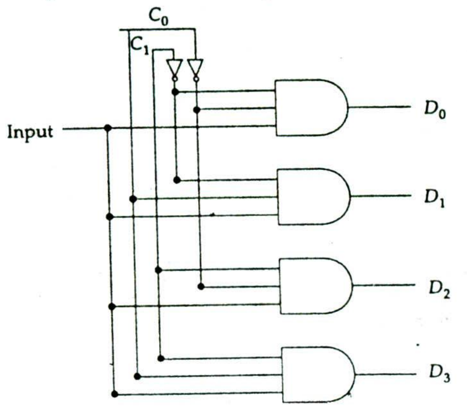


Fig. 5.21

5.20.1. Half Adder

A half adder is a logic circuit that adds 2 bits. Table 5.20 shows the addition of two bits. Columns 1 and 2 of the table give the values of two input bits, column 3 gives the sum of these two bits and column 4, the carry bit. This table is called a half adder truth table. Even though the inputs and outputs are binary numbers, they may be taken as depicting truth values of 0 and 1 for developing the Boolean expressions for C and S. By inspection of table, we obtain

$$S = \overline{A} . B + A . \overline{B} = A \oplus B$$
$$C = A . B$$

Table 5.20 : A Half Adder Truth Table

INPUT		OUTPUT	
A	*B*	S (SUM)	C (CARRY)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A logic circuit which uses logic gates to implement the half adder is shown in Fig. 5.22.

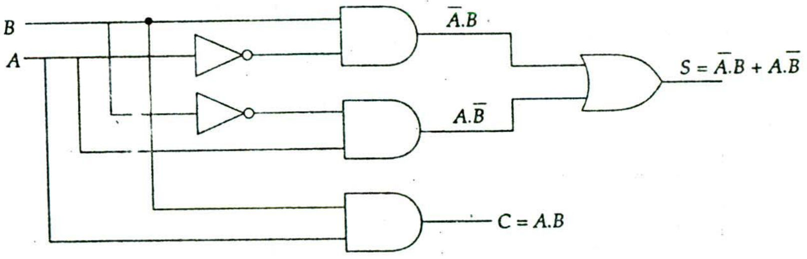


Fig. 5.22: A logic circuit realising half adder

It can also be seen from Table 5.20 that the sum of two binary digits can be represented by the output of an XOR gate and the carry output can be represented by the output of an AND gate, *i.e.*, if the same two inputs are applied to XOR and AND gates, the output of XOR gate will represent the sum and the AND will represent the carry.

Logic circuits using XOR gate and block diagram of half adders are shown in Figs. 5.23 and 5.24, respectively.

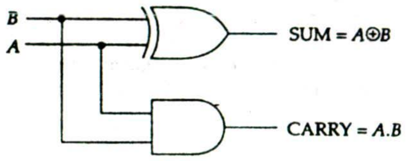


Fig. 5.23: Logic circuit for half adder using XOR gate

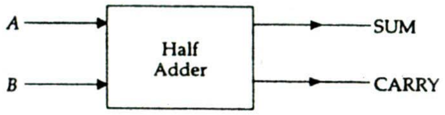


Fig. 5.24: Block diagram of half adder

5.20.2. Full Adder

When adding two binary numbers, we may have a carry from one column to the next. The carry coming out from one column is to be added to the next column. A half adder cannot add 3 bits as it has only 2 input terminals.

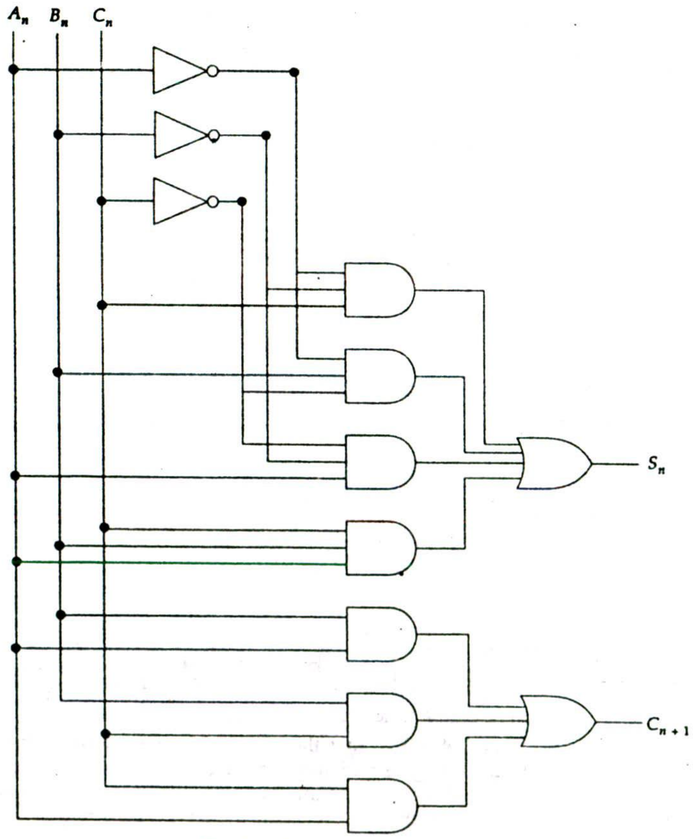


Fig. 5.25: Logic circuit for full adder

A full adder is a logic circuit that can add 3 bits at a time. Again, there are two outputs: sum and carry. Table 5.21 gives the full adder truth table.

Table 5.21 : Full Adder Truth Table

INPUTS			OUTPUT	
A_n	B_n	C_n (CARRY IN)	S_n (SUM)	C_{n+1} (CARRY OUT)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The Boolean expression for S_n and C_{n+1} are obtained by looking at the 1s in the S_n and C_{n+1} columns, respectively and writing the Boolean terms corresponding to them. SUM is 1 when the number of input is odd, CARRY is 1 when two or more inputs are 1s.

$$\begin{aligned}
S_n &= \overline{A}_n . \overline{B}_n . C_n + \overline{A}_n . B_n . \overline{C}_n + A_n . \overline{B}_n . \overline{C}_n + A_n . B_n . C_n \\
&= \overline{A}_n (\overline{B}_n C_n + B_n\overline{C}_n) + A_n (\overline{B}_n\overline{C}_n + B_n C_n) \\
&= \overline{A}_n (B_n \oplus C_n) + A_n (B_n \oplus C_n) \\
&= A_n \oplus B_n \oplus C_n \\
C_{n+1} &= \overline{A}_n . B_n . C_n + A_n . \overline{B}_n . C_n + A_n . B_n . \overline{C}_n + A_n . B_n . C_n \\
&= \overline{A}_n . B_n . C_n + A_n . B_n . C_n + A_n . \overline{B}_n . C_n + A_n . B_n . C_n \\
&\quad + A_n . B_n . \overline{C}_n + A_n . B_n . C_n \qquad \text{[Using Rule 5 (a)]} \\
&= B_n . C_n (A_n + \overline{A}_n) + A_n . C_n (\overline{B}_n + B_n) + A_n . B_n (C_n + \overline{C}_n) \\
&= B_n . C_n + A_n . C_n + A_n . B_n \qquad \text{[Using Rule 4 (a)]}
\end{aligned}$$

The logic circuit for the full adder is shown in Fig. 5.25. Also, a full adder using 3 input XOR gate and a block diagram of a full adder are shown in Figs. 5.26 and 5.27, respectively.

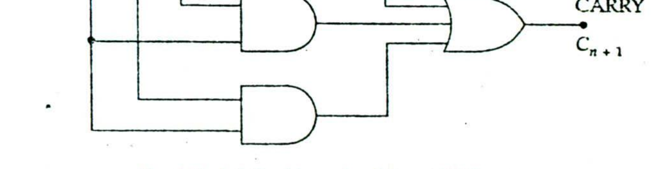


Fig. 5.26: A full adder using 3 input XOR gate

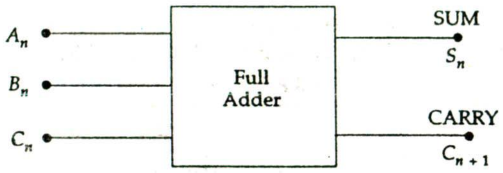


Fig. 5.27 : Block diagram of a full adder

5.20.3. Circuit Diagram of a Full Adder using Two Half Adders and OR Gate (AMIE, W '95)

By connecting two half adders and an OR gate, we get a full adder. Figure 5.28 shows a full adder using two half adders.

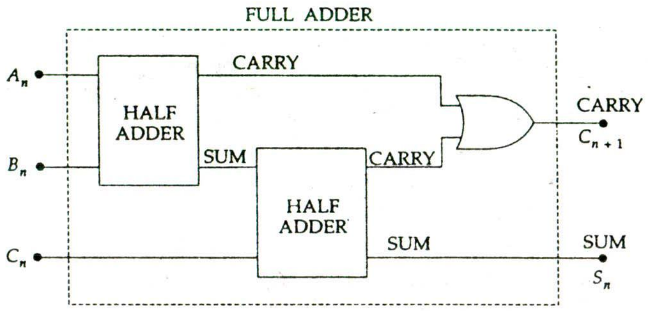


Fig. 5.28 : A full adder realized from two half adders

For instance, suppose $A = 1$, $B = 1$, and $C = 0$. Figure 5.29 (*a*) shows the full adder with these inputs. The first half adder (HA) has a sum of 0 with a carry of 1. The second half adder has a sum of 0 with a carry of 0. Therefore, the final output is a sum of 0 with a carry of 1.

If the inputs are $A = 1$, $B = 1$, and $C = 1$, Fig. 5.29 (*b*) shows the full adder with these inputs. The final output is a sum of 1 with a carry of 1.

Fig. 5.29: Examples of working a full adder

5.20.4. Half Subtractor

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this method, the subtraction operation becomes an addition operation.

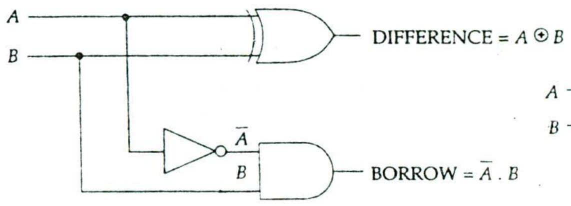
A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. The truth table for the input-output relationships for a half-subtractor have been summarized in Table 5.22. The difference output, D_n, resembles with XOR function and borrow (B_{n+1}) are derived directly from the truth table.

Table 5.22

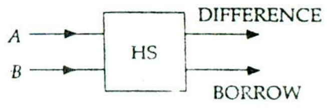
INPUT		OUTPUT	
A (*MINUEND*)	*B* (*SUBTRAHEND*)	*Dn* (*DIFFERENCE*)	B_{n+1} (*BORROW*)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D_n = \overline{A}B + \overline{B}A = A \oplus B$$
$$B_{n+1} = \overline{A}B$$

The logic circuit and block diagram for half subtractor are shown in Figs. 5.30 (*a*) and (*b*), respectively.



(*a*) Logic circuit of half subtractor



(*b*) Block diagram of half subtractor

Fig. 5.30

5.20.5. Full Subtractor

A half subtractor handles only 2 bits at a time and can be used for the least significant column of a subtraction problem. A full subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a may have been borrowed by a lower significant stage. This circuit has three inputs and two outputs. The truth table for the circuit is shown in Table 5.23.

Table 5.23 : A Full Subtractor Truth Table

INPUT			OUTPUT	
A	*B*	C_n (*BORROW IN*)	D_n (*DIFFERENCE*)	C_{n+1} (*BORROW OUT*)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The simplified Boolean function for two outputs of the full subtractor is

$$D_n = \overline{A}_n \overline{B}_n C_n + \overline{A}_n B_n \overline{C}_n + A_n \overline{B}_n \overline{C}_n + A_n B_n C_n = A_n \oplus B_n \oplus C_n$$

$$C_{n+1} = \overline{A}_n B_n + \overline{A}_n C_n + B_n C_n$$

By connecting two half-subtractors and an OR gate, we get a full subtractor. Figures 5.31 and 5.32 show a full subtractor realised from two half-subtractors and a block diagram.

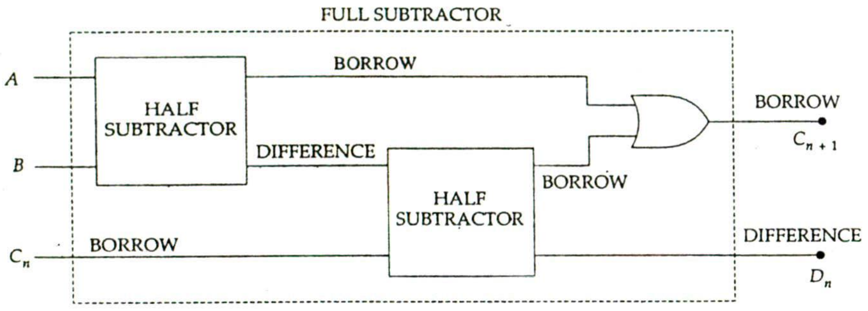


Fig. 5.31: A full subtractor

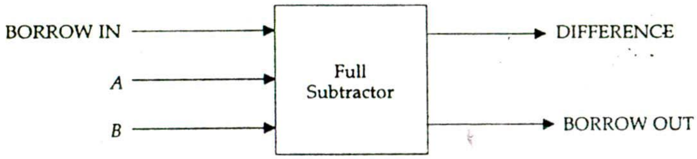


Fig. 5.32: Block diagram for full subtractor

5.20.6. Binary Adder

A binary adder is a logic circuit that can add two binary numbers. The addition of two binary numbers each of K bits can be accomplished using one half adder (HA) and K-1 full adders (FAs). The right most block represents a half adder. Each of the full adders has three inputs and two outputs. The carry output of each adder goes to the carry input of the next adder to the left. Consider, for example, $K = 4$, and two binary numbers to be added $A_3\ A_2\ A_1\ A_0$ and $B_3\ B_2\ B_1\ B_0$. The answer is

$$\begin{array}{rccccc} & & A_3 & A_2 & A_1 & A_0 \\ + & & B_3 & B_2 & B_1 & B_0 \\ \hline & C_4 & S_3 & S_2 & S_1 & S_0 \end{array}$$

Figure 5.33 shows a parallel 4 bit binary adder.

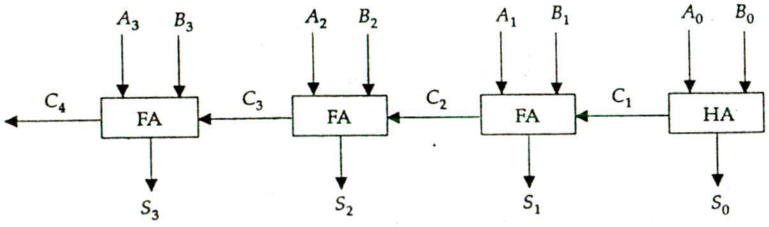


Fig. 5.33: Parallel 4 bit binary adder

Suppose $A = 1100$ and $B = 1011$. The half adder produces a sum of 1 and carry of 0, the first full adder produces a sum of 1 and a carry of 0, the second full adder produces a sum of 1 and a carry of 0, and the third full adder produces a sum of 0 and a carry of 1 .The overall output is 10111.

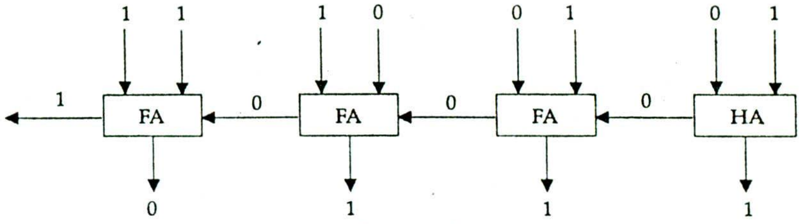


Fig. 5.34 : Adding 12 and 11 to get 23

A four bit binary adder can also be built by combining four full adders (Fig. 5.35), the first full adder acts as a half adder as its carry input is held at logic 0 level.

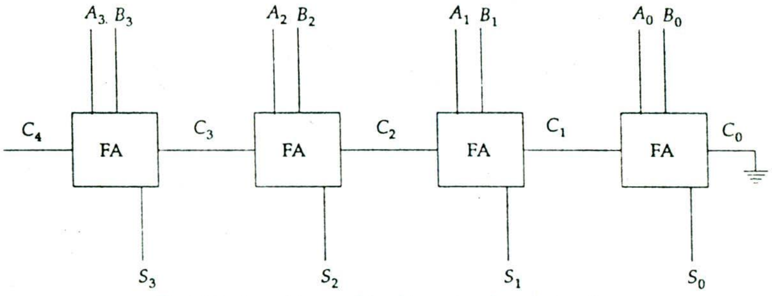


Fig. 5.35 : Four-bit parallel adder using four full adders

5.20.7. Binary Subtractor

A subtraction of one k-bit binary number B from the other k-bit binary number A can be performed using one half-subtractor (HS) and $k - 1$ full subtractors (FSs). Let, for example, $k = 4$, $A = A_3\ A_2\ A_1\ A_0$ and $B = B_3\ B_2\ B_1\ B_0$. The answer $A - B$ can be written as

$$\begin{array}{rcccc} & A_3 & A_2 & A_1 & A_0 \\ - & B_3 & B_2 & B_1 & B_0 \\ \hline & D_3 & D_2 & D_1 & D_0 \end{array}$$

A four bit parallel subtractor, using a half-subtractor and three full subtractors, is shown in Fig. 5.36.

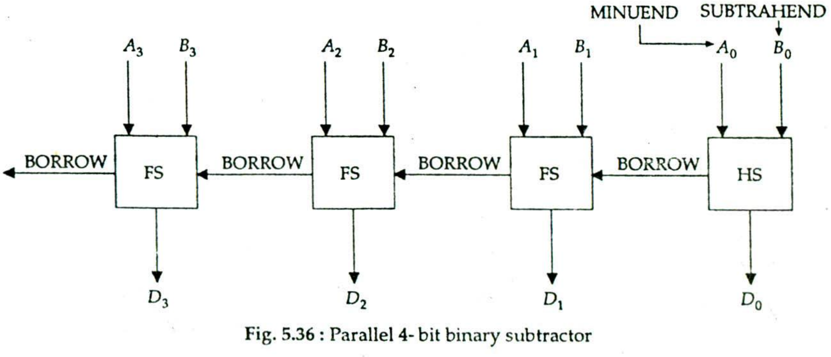


Fig. 5.36 : Parallel 4-bit binary subtractor

Controlled inverter

In two input XOR gate, one of its input labelled as INVERT be held low, output is the same as the other input (A, say), i.e., A passes to the output unchanged. When INVERT be held high, output is the complement of A.

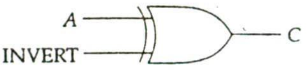


Fig. 5.37

5.20.8. Half Adder/Subtractor

Since the sum and difference outputs of an adder/subtractor can both be derived from XOR gate, a single logic circuit, which is used for addition as well as subtraction, can be developed by making use of XOR gate and INVERT function.

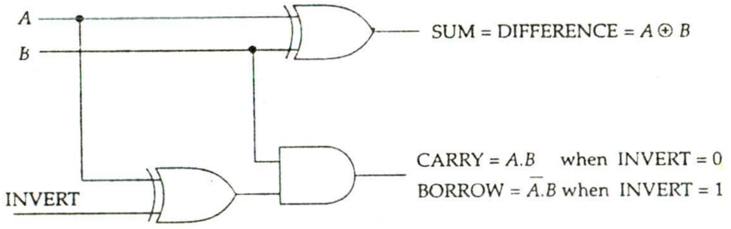

A
B
INVERT
SUM = DIFFERENCE = $A \oplus B$
CARRY = $A.B$ when INVERT = 0
BORROW = $\bar{A}.B$ when INVERT = 1

Fig. 5.38: Half adder/substractor

5.20.9. 2's Complement Adder/Subtractor (AMIE, S '93)

A 2's complement adder/subtractor is a logic circuit that can add or subtract binary numbers. Fig. 5.39 shows a logic circuit for 2's complement adder/subtractor.

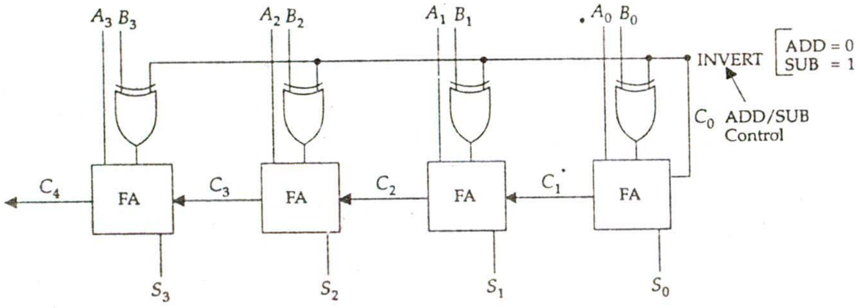


Fig. 5.39 : A four-bit adder /subtractor with 2's complement ADD/SUB control

Addition

When INVERT is low (0) the B bits pass through the inverter without any change. Therefore, the full adder (FA) produces the sum

$$S = A + B.$$

Subtraction

When INVERT is high (1) the controlled inverter produces the 1's complement of B. Furthermore, the high INVERT adds 1 ($C_0 = 1$) to the first full adder. This addition of 1 to the 1's complement of B forms the 2's complement of B.

The output of the full adders is

$$S = A + B'$$

which is equivalent to $S = A - B$.

REVIEW QUESTIONS SET

1. Explain AND and OR operations with suitable examples of logic statements and electrical switches.
2. Explain Boolean variables and Boolean functions.
3. Show that the exclusive OR function
   $x = A \oplus B \oplus C \oplus D$
   is an odd function. **(AMIE, S '96)**
4. What is a truth table? What are truth tables of the following functions :
   (*a*) $A + \overline{B}\,\overline{C}$ (*b*) $AB + \overline{A}\,\overline{B}$ (*c*) $A.B + B + C$
5. Simplify the following Boolean expressions :
   (*a*) $XY + X\overline{Y}$ (*b*) $(X + Y)(X + \overline{Y})$
   (*c*) $XYZ + \overline{X}Y + XY\overline{Z}$ (*d*) $ZX + Z\overline{X}Y$
   (*e*) $A(A + B)(A + AB)$ (*f*) $A(\overline{A} + B)(AB + \overline{B})$
   (*g*) $\overline{A}(AB + \overline{B})$ (*h*) $\overline{A}B\overline{C} + ABC + \overline{A}BC + AB\overline{C}$
6. What is a half adder? Write truth table for a half adder and develop its logic circuit.
7. What is a full adder? How is a full adder built using half adders?
8. What is a NOR gate, a NAND gate? Under what input conditions is the output of a NOR gate equal to 1? For what input conditions is the NAND gate output equal to 0?
9. Develop the expressions of the sum and carry for adding two 2 bit numbers. **(AMIE, W '95)**
10. Give the schematic diagram of a binary 2's complement adder/subtractor. Briefly explain the operation of the circuit.
11. What do you mean by block diagram? Draw the block diagram of serial adder and illustrate the function with two 15-bit operands. **(AMIE, W '94)**
12. Discuss decoder and multiplexer with suitabie examples. **(AMIE, W '94)**
13. What are (*a*) sum of products form, and (*b*) product of sums form of logic expressions. Explain with suitable examples.
14. What is canonical form of logic expressions? Explain minterms and maxterms.
15. Explain the functions of encoder and decoder.

ANSWERS TO REVIEW QUESTIONS SET

4. Given only last column
   (*a*) (1, 1, 1, 0, 1, 1, 1, 1) (*b*) (1, 0, 0, 1)
   (*c*) (0, 1, 1, 1, 0, 1, 1, 1)
5. (*a*) X (*b*) X (*c*) Y (*d*) Z (X + Y) (*e*) A (*f*) AB (*g*) A B (*h*) B.