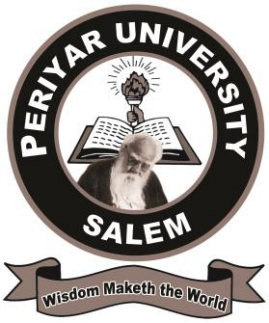


PERIYAR INSTITUTE OF DISTANCE EDUCATION
(PRIDE)

PERIYAR UNIVERSITY
SALEM - 636 011.

B.Sc. COMPUTER SCIENCE
FIRST YEAR
PAPER – I : DIGITAL COMPUTER FUNDAMENTALS

Prepared by :
N.RAJENDRAN M.C.A.,M.Phil.,
HOD, Department of Computer Science,
Vivekanandha College of Arts and Sciencs for Women,
Elayampalayam,Tiruchengode(Tk.),
Namakkal(Dt.),Tamilnadu.

INTRODUCTION

Dear Students

A digital computer is an electronic computing machine that uses the binary digits (bits) 0 and 1 to represent all forms of information internally in digital form. The term digital computer or simply computer embraces calculators, computer workstations, control computers for application such as domestic appliances, industrial processes and data processing systems.

Totally this book covers five units. The first unit deals with types of computers, characteristics of computers, generations of computers, and number system.

The second unit deals with concepts of Boolean algebra, gates, laws of Boolean algebra, De Morgan's theorems, derivation of Boolean expression, sum of products and products of sums, map method for simplifying expressions and subcubes and covering.

The third unit deals with anatomy of digital computers, memory units, input devices, output devices, and auxiliary storage devices.

The fourth unit deals with combinational logic, adders, subtractors, decoders, encoders, multiplexer, demultiplexer , flip-flops-register, shift register and counters.

The fifth unit deals with computer design, system configuration, computer instructions, design of computer registers, design control and computer console.

PRIDE would be happy in you could make use of this learning material to enrich your knowledge and skills to serve the society.

B.Sc. COMPUTER SCIENCE

FIRST YEAR

PAPER – I : DIGITAL COMPUTER FUNDAMENTALS

UNIT I

UNIT II

UNIT III

UNIT IV

UNIT V

Digital Computer Fundamentals

UNIT – I

Introduction to computers: Introduction–Types of Computers-Characteristics of Computers-Word Length-speed-Storage-Accuracy-Versatility-Automation-Diligence. Five generations of modern Computers: First Generation Computers-Second Generation Computers-Third Generation Computers-Fourth Generation Computers- Fifth Generation Computers. Classifications of digital computer system: Introduction- Microcomputers-Personal Computers-Workstations - Portable Computers – Minicomputers – Mainframes - Supercomputers - Network Computers. Number system: Introduction-Decimal Number System-Binary number System-Binary to decimal Conversion-Decimal to Binary Conversion-Binary Addition-Binary Subtraction - Complements - 9's Complement - 10's Complement - 1's Complement-2's Complement Octal Number System-Hexadecimal Number System.

UNIT – II

Boolean Algebra and Gate Networks: Fundamental concepts of Boolean Algebra-Logical Multiplication –AND Gates and OR Gates-Complementation and Inverters – Evaluation of Logical Expressions – Evaluation of an Expression containing parentheses – Basic Laws of Boolean Algebra – Simplification of expressions – De Morgan's theorems – Basic Duality of Boolean Algebra – Derivation of a Boolean Expression – Interconnecting Gates-Sum of products and products of sums – Derivation of products of sums expressions – Derivation of three Input variable expression – NAND gates and NOR gates – The Map method for simplifying expressions – Sub cubes and covering – Product of sums. Expressions – Don't care.

UNIT – III

Anatomy of a Digital: Functions and Components of a Computer Central Processing Unit-Control Unit – Arithmetic Logic Unit-Memory – Registers Addresses – How the CPU and Memory Work. Memory units : Introduction – RAM – ROM – EPROM – EEPROM – Flash memory. Input Devices: Introduction-Keyboard-Mouse-Types of Mice-Connections-Mouse Pad-Trackball – Joystick – Digitizing Tablet – Scanners – Digital Camera – MICR-OCR-OMR- Barcode Reader – speech input Devices-continuous speech- Discrete Word-Touch Screen – Touch Pad-Light Pen. Output Devices: Introduction-Monitor – Classification of Monitors – Based on Color – Classification of Monitors Based on signals – characteristics of a Monitor – Video Standards-Printer-Plotter-Sound Cards and Speakers – Auxiliary storage Devices : Introduction – Magnetic Tape – Hard disk – Floppy Disk – CD-ROM-CD-R Drive-CD-RW Disks.

UNIT-IV

Combinational logic, adders, subtractors, decoders, encoders, multiplexer, demultiplexer – Flip-flops-Register – Shift register – Counters

UNIT-V

Computer design – System configuration – Computer instructions – Design of computer registers – Design control – Computer console.

TEXTBOOKS:

1. "Fundamentals of Computer Science and Communication Engineering"
   Alexis Leon, Mathews Leon,
   Vikas Publishing House,
   New Delhi, 1998
   (Unit 1 , III & IV)
2. " Digital Computer Fundamentals"
   Thomas C.Bartee,
   T.M.H, New Delhi,
   6th Edition 1991 (Unit-II)
3. "Digital logic and Computer Design"
   M. Morris Mano

UNIT-I

INTRODUCTION TO COMPUTERS

UNIT-I

INTRODUCTION TO COMPUTERS

1.0 INTRODUCTION

A computer is a programmable machine. The two principal characteristics of a computer are:

- It responds to a specific set of instructions in a well-defined manner.
- It can execute a prerecorded list of instructions (a program)

Modern computers are electronic and digital. The actual machinery – wires, transistors, and circuits – is called hardware; the instructions and data are called software.

All general – purpose computers require the following hardware components:

- **Central processing unit (CPU)** The "heart" of the computer, the component that actually executes instructions.
- **Memory** Enables a computer to store, at least temporarily, data and programs.
- **Input device** Usually a keyboard or mouse, the input device is the conduit through which data and instructions enter a computer.
- **Output device** A display screen, printer, or other such devices that lets you see what the computer has accomplished.
- **Mass storage device** Allows a computer to permanently retain large amounts of data. Common mass storage devices include disk drives and tape drives.

In addition to these components, many others make it possible for the basic components of a computer to work together efficiently. For example, every computer requires a bus that transmits data from one part of the computer to another.

1.1 TYPES OF COMPUTERS

Computers can be classified by their size and power as follows:

- **Personal computer** A small, single – user computer based on a microprocessor. In addition to the microprocessor, a personal computer has a keyboard for entering data, a monitor for displaying information, and a storage device for saving data.
- **Workstation** A powerful, single-user computer. A workstation is like a personal computer, but it has a more powerful microprocessor and a higher-quality monitor.
- **Minicomputer** A multi-user computer capable of supporting 10 to hundreds of users simultaneously.
- **Mainframe** A powerful multi-user computer capable of supporting many hundreds of users simultaneously.
- **Supercomputer** An extremely fast computer that can perform hundreds of millions of instructions per second.

1.2 CHARACTERISTICS OF COMPUTERS

All computers have certain common characteristics irrespective of their type and size.

Word Length

A digital computer operates on binary digits – 0 and 1. It can understand information only in terms of 0s and 1s. A binary digit is called a bit. A group of 8 bits is called a byte. The number of bits that a computer can process at a time in parallel is called its word length. Commonly used word lengths are 8, 16, 32 or 64 bits. Word length is the measure of the computing power of a computer.

Speed

Computers can calculate at very high speeds. A microcomputer, for example, can execute millions of instructions per second over and over again without any mistake. As the power of the computer increases, the speed also increases. For example, supercomputers can operate at speeds measured in nanoseconds and even in picoseconds – one thousand to one million times faster than microcomputers.

Storage

Computers have their main memory and auxiliary memory systems. A computer can store a large amount of data. With more and more auxiliary storage devices, which are capable of storing huge amounts of data, the storage capacity of a computer is virtually unlimited. The factor that makes computer storage unique is not that it can store vast amount of data, but the fact that it can retrieve the information that the user wants in a few seconds.

Accuracy

The accuracy of a computer system is very high. Errors in hardware can occur, but error detecting and correcting techniques will prevent false results. In most cases, the errors are due to the human factor rather than the technological flaws. For example, if a program is wrongly coded, if the data is corrupted, or if the program logic is flawed, then irrespective on which computer you run it, you will always get wrong results. Another area where mistakes can creep in is during data entry. People often make mistakes when data is keyed-in and the computer accepts whatever that is keyed-in. So if a wrong input is given, the output also will be wrong – GIGO (Garbage In Garbage Out).

Versatility

Computers are very versatile machines. They can perform activities ranging from simple calculations to performing complex CAD modeling and simulation to navigating missiles and satellites. In other words, they are capable of performing almost any task, provided the task can be reduced to a series of logical steps. Computers can communicate with other computers and can receive and send data in various forms like text, sound, video, graphics, etc. This ability of computer to communicate to one another has led to the development of computer networks, **Internet, WWW** and so on.

Automation

The level of automation achieved in a computer is phenomenal. It is not a simple calculator where you have to punch in the numbers and press the 'equal to' sign to get the result. Once a task is initiated, computers can proceed on its own till its completion. Computers can be programmed to perform a series of complex tasks involving multiple programs. Computers will perform these things flawlessly. They will execute the programs in the correct sequence, they will switch on/off the machines at the appropriate time, they will monitor the operational parameters, they will send warning signals or take corrective actions if the parameters exceed the control level, and so on. Computers are capable of theses levels of automation, provided they are programmed correctly.

Diligence

Diligence means being constant and earnest in effort and application. Human beings suffer fro weakness like tiredness, lack of concentration, etc. Humans have feelings, they become sad, depressed, bored, and negligent and it will reflect on the work they do. Moreover, human beings cannot perform the same or similar tasks over and over again with the same precision, accuracy and enthusiasm as the first time. After some time, people will become bored and tedium will set in. This will affect the performance. Being a machine, a computer does not have any of these human weaknesses. They won't get tired or bored. They will not go into depression or loose concentration. They will perform the tasks that are given to them, irrespective of whether it is interesting, creative, monotonous or boring, irrespective of whether it is the first time or the millionth time, with exactly the same accuracy and speed.

1.3 FIVE GENERATIONS OF MODERN COMPUTERS

The computers are classified into different generations – from first generation to fifth generation computers. The classification and the time period is given bellow:

1. First Generation (1945-1956)
2. Second Generation (1956-1963)
3. Third Generation (1964-1971)
4. Fourth Generation (1971 – Present)
5. Fifth Generation (Present and Beyond)

FIRST GENERATION (1945-1956)

John Presper Eckert (1919-1995) and John W.Mauchly(1907-1980), built the first digital computer using parts called vacuum tubes. They named their new invention ENIAC. Consisting of 18,000 vacuum tubes, 70,000 resistors and 5 million soldered joints.

Von Neumann designed the Electronic Discrete Variable Automatic Computer (EDVAC) in 1947 with a memory to hold both a stored program as

well as data. This “stored memory” technique as well as the “conditional control transfer”, that allowed the computer to be stopped at any point and then resumed, allowed for greater versatility in computer programming. The key element to the Von Neumann architecture was the central processing unit, which allowed all computer functions to be coordinated through a single source. In 1951, UNIVSC I (Universal Automatic Computer), built by Remington Rand, became one of the first commercially available computer.

SECOND GENERATION COMPUTERS (1956-1963)

Second – generation computers replaced machine language with assembly language, allowing abbreviated programming codes to replace long, difficult binary codes. There were a number of commercially successful second-generation computers used in businesses, universities, and government. These second-generation computers were also of solid state design, and contained transistors in place of vacuum tubes.

It was the stored program and programming language that gave computers the flexibility to finally be cost effective and productive for business use. The stored program concept meant that instructions to run a computer for a specific function (known as a program) were held inside the computer’s memory, and could quickly be replaced by a different set of instructions for a different function. A computer could print customer invoices and minutes later design products or calculate pay cheques. More sophisticated high-level languages such as COBOL (Common Business-Oriented Language) and FORTRAN (Formula Translator) came into common use during this time, and have expanded to the current day. These languages replaced cryptic binary machine code with words, sentences, and mathematical formulae, making it much easier to program a computer. New types of careers (programmer, analyst, and computer system expert) and the entire software industry began with second generation computers. Paralleling the development of second-generation system was the creation of a new industry, built around the idea of integrating transistors and other components into circuits that could be placed on small chips of silicon.

THIRD GENERATION COMPUTERS (1964-1971)

Though transistors were clearly an improvement over the vacuum tube, they still generated a great deal of heat, which damaged the computer’s sensitive internal parts. The quartz rock eliminated this problem. The IC combined three electronic components onto a small silicon disc, which was made from quartz. Scientists later managed to fit even more components on a single chip, called a semiconductor. Third generation development included the use of an operating system that allowed machines to run many different programs at once with a central program that monitored and coordinated the computer’s memory.

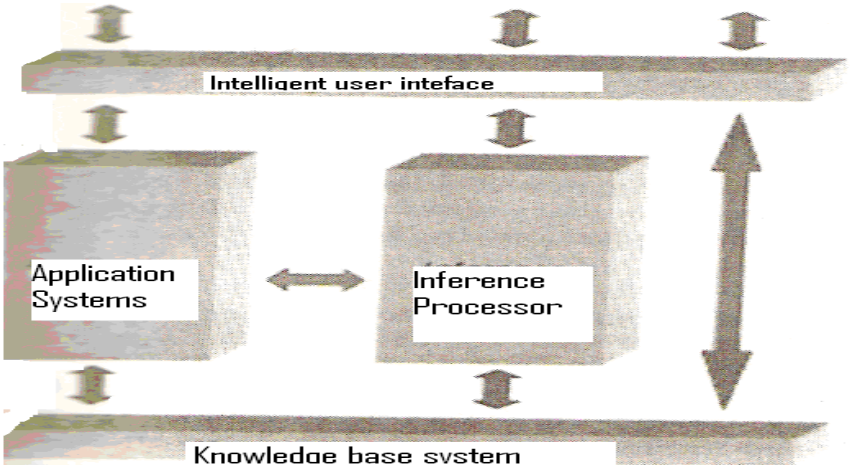
FOURTH GENERATION COMPUTERS (1971-PRESENT)

After the integrated circuits, the only place to go was down – in size, that is. Large – scale integration (LSI) could fit hundreds of components onto one chip. Very large scale integration (VLSI) squeezed hundreds of thousands of components onto a chip. Ultra – Large-scale integration (ULSI) increased that number into the millions. The ability to fit so much onto an area about half the size of one-rupee coin helped diminish the size and price of computers. It also increased their power, efficiency and reliability. One microprocessor could be manufactured and then programmed to meet any number of demands. Soon everyday household items such as microwave ovens, television sets and automobiles incorporated microprocessors.

There minicomputers came complete with user-friendly software packages that offered even non-technical users an array of applications, most popularly word processing and spreadsheet programs.

FIFTH GENRATION (PRESENT AND BEYOND)

Many advances in the science of computer design and technology are coming together to enable the creation of fifth-generation computers. Another advance is superconductor technology, which allows the flow of electricity with little or no resistance, greatly improving the speed of information flow. Computers today have some attributes of fifth generation computers. For example, expert systems assist doctors in making diagnoses by applying the problem-solving steps a doctor might use in assessing a patient's needs.



Schematic diagram of a 5th generation computer

Fifth generation computers aim to be able to solve highly complex problems, ones, which require reasoning, intelligence and expertise when solved by people. They are intended to be able to cope with large subsets of natural languages, and draw on very large knowledge basis. Fifth generation computers are being designed to use by people who are not necessarily computer experts. In order to achieve these very ambitious aims, fifth generation computers will not have a single processor, or a small number of tightly coupled processors as computers do today. As said before they are being

designed to contain a large number of processors, grouped into three major subsystems: a knowledge base system, an inference mechanism and an intelligent user interface.

The knowledge base system has a very large store of knowledge with a set of processors, which access and update it. It is likely that knowledge bases will evolve from current work in relational databases.

Operations on knowledge bases require the manipulation of large numbers of individual elements: this manipulation will be done in parallel by the arrays of knowledge processing elements.

The inference mechanism draws reasoned conclusions from the knowledge base. Much of its processing will be drawing logical inferences of the :

If <condition> then <action>

variety. Accordingly, the processing power of fifth generation computers is expressed in logical inferences per second (lips). The target is in the range 50 to 1000 million lips (compared with a current performance of 10 to 100 thousand lips). Most of this improved performance is planned to achieve via highly parallel architectures, such as the data-flow and graph reduction architectures.

The intelligent user interface is the point of contact between a fifth generation computer and its user. Many of these will be based on communication in a large subset of a natural language. Others will make extensive use of advanced graphics, including image processing. The intention is to build a user interface, which is close to the natural way of thinking of the user, rather than close to the way of working of the computer.

1.4 CLASSIFICATION OF DIGITAL COMPUTER SYSTEMS

1.4.1 INTRODUCTION

Computer systems are classified as Microcomputers, Minicomputers, Mainframes and Supercomputers.

1.4.2 MICROCOMPUTERS

The most familiar kind of computer is the microcomputer. In the past, microcomputers have been considered to be of two types – Personal Computers and Workstations.

Personal Computers (PCs)

PCs were desktop or portable machines. These machines ran comparatively easy – to – use applications software such as the word processors, spreadsheets, etc. They were usually easier to use and more affordable than workstations. However, they had less sophisticated video display screens, operating systems and networking capabilities. They did not have the processing power that workstations did. Examples of personal computers are Acer’s Aspire, Compaq Presario, etc.

Workstations

Workstations expensive, powerful machines used by engineers, scientists, and other professionals who processed a lot of data. Workstations use high-resolution colour graphics and operating systems such as UNIX that permitted multitasking. Workstations also use powerful networking links to other computers. The most significant distinguishing factor, however, is the powerful processor, which could churn out results much faster than the PCs. The more powerful workstations are called supermicros. Examples of well-known workstations are those made by Sun, Apollo, Hewlett-Packard, NeXT and IBM.

PCs are now as powerful as many of those used in workstations. More powerful microprocessors and increased graphics and communications capabilities now let end users run software that previously ran only on more powerful machines.

Portable Computers

Personal computing market is seeing the miniaturization phenomena. Computers are becoming smaller yet more powerful. There are three categories of portable computers: Laptops or Notebook PCs, Subnotebooks and Personal Digital Assistants.

Laptops / Notebooks Laptops may be either AC-powered, battery-powered, or both. These computers are ideal for users who have to work away from their offices.

The user of these computers might be an executive on the move, a student, a journalist, a salesperson, etc.

Subnotebooks Subnotebooks are for frequent flyers and life–on-the-road professionals. Subnotebook users give up a full display screen and keyboard in exchange for less weight.

These computers fit easily into any briefcase. They typically have an external floppy disk drive and monochrome monitor, although of late colour models are available. An example of a colour sub notebooks is Toshiba Protégé.

Personal Digital Assistants (PDAs) PDAs are much smaller than the sub notebooks. They combine pen input, writing recognition, personal organization tools, and communication capabilities in a very small package.

Typical users are executives, businessmen, etc. who use these machines for their day-to-day activities – scheduling, organization, etc. An example PDA is Apple's Newton**.**

1.4.3 MINICOMPUTERS

Minicomputers, also known as mid range computers. They were used to control machines in a manufacturing unit. They are widely used as general – purpose computers. The more powerful minicomputer models are called superminis. The increasing power of microcomputer gives minutes. One of the

popular minicomputer systems is the VAX made by Digital Equipment Corporation.

Minicomputers work well in what are known as Distributed Data Processing (DDP). That is, a company's processing power is decentralized, or distributed across different computers.

An example of such computer architecture is the Client/Server model, in which end users can process at their own microcomputers. End users can also access and share the resources of the server, which usually is a minicomputer. For example, an executive could use the server to search the company's centralized database and retrieve selected data. He / she could then use a spreadsheet on his/her microcomputer to analyze the data.

1.4.4 MAINFRAMES

Mainframe computers can process several million – program instructions per second. Large organizations rely on these room-size systems to handle large programs with lots of data.

Mainframes are mainly used by insurance companies, banks, airline and railway reservation systems, etc. An advanced mainframe made by IBM is S/390**.**

1.4.5 SUPERCOMPUTERS

Supercomputers are the fastest calculating devices. A desktop microcomputer processes data and instructions in millionths of a second, or microseconds. A supercomputer, by contrast, can operate at speeds measured in nanoseconds and even in picoseconds. One thousand to one million times as fast as microcomputers.

Most supercomputers are used by government agencies. These machines are for applications requiring very large programs and huge amounts of data that must be processed quickly. Examples of such task are weather forecasting, oil exploration, weapons research, and large-scale simulation. The chief difference between a supercomputer and a mainframe is that a supercomputer channels all its power into executing a few programs as fast as possible, whereas a mainframe uses its power to execute many programs concurrently. Supercomputers use a technology called massively parallel processing. These supercomputers consist of thousands of integrated microprocessors. One massively parallel computer built by Intel Corporation is capable of performing 8.6 billion mathematical calculations per second.

1.4.6 NETWORK COMPUTERS

Network computers are computers with minimal memory, disk storage and processor power designed to connect to a network, especially the Internet. Network computers is that many users who are connected to a network. Network computers designed to connect to the Internet are sometimes called Internet boxes, Net PCs and Internet appliances.

1.5 NUMBER SYSTEM

1.5.1 INTRODUCTION

We use the decimal numbers or the decimal number system for our day-to-day activities. In the decimal number system there are ten digits – 0 through 9. But computers understand only 0s and 1s – the machine language. We can use the decimal numbers, the alphabets and special characters like +,-,*,?,/, etc. for programming the computer. Inside the computer, these decimal numbers, alphabets and the special characters are converted into 0s and 1s. So, that the computer can understand what we are instructing it to do.

1.5.2 DECIMAL NUMBER SYSTEM

The base or radix of a number system is defined as the number of digits it uses to represent the numbers in the system. Since decimal number system uses 10 digits – 0 through 9 – its base or radix is 10. The decimal number system is also called base 10 number system. The weight of each digit of a decimal number system depends on its relative position within the number. For example, consider the number 3256.

$3256 = 3000 + 200 + 50 + 6$

or, in other words,

$3256 = 3 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$

From the above example, we can see that the weight of the n^{th} digit of the number from the right hand side is equal to n^{th} digit x 10^{n-1} which is again equal to n^{th} digit x $(base)^{n-1}$.

1.5.3 BINARY NUMBER SYSTEM

The base or radix of the binary number system is 2. It uses only two digits – 0 and 1. Data is represented in a computer system by either the presence or absence of electronic or magnetic signals in its circuitry or the media it uses. This is called a binary, or two-state representation of data since the computer is indicating only two possible states or conditions.

For example, consider the binary number 10100.

$10100 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 16 + 0 + 4 + 0 + 0 = 20$

Decimal	Binary	Decimal	Binary
0	0	11	1011
1	1	12	1100
2	10	13	1101
3	11	14	1110
4	100	15	1111
5	101	16	10000
6	110	17	10001
7	111	18	10010
8	1000	19	10011
9	1001	20	10100
10	1010		

Binary –decimal Conversion

To convert a binary number to its decimal equivalent we use the following expression:

The weight of the n^{th} bit of a number from the right hand side = n^{th} bit x 2^{n-1}

After calculating the weight of each bit, they are added to get the decimal value as shown in the following examples:

$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$

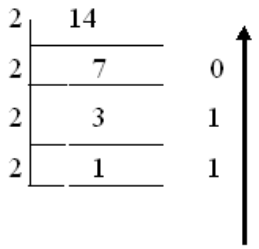
$1010 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8 + 0 + 2 + 0 = 10$

$1111 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 4 + 2 + 1 = 15$

$1.001 = 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1 + 0 + 0 + .125 = 1.125$

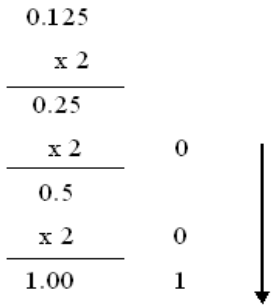
Decimal – binary Conversion

Decimal numbers are converted into binary by a method called Double Dabble Method. In this method, the mantissa part of the number is repeatedly divided by two and noting the reminders, which will be either 0 or 1. This division is continued till the mantissa becomes zero. The reminders, which are noted down during the division is read in the reverse order to get the binary equivalent. This can be better illustrated using the following example.



The number is written from below, that is 1110. So the binary equivalent of 14 is 1110.

If the decimal number has a fractional part, then the fractional part is converted into binary by multiplying it with 2. Only the integer of the result is noted and the fraction is repeatedly multiplied by 2 until the fractional part becomes 0. This can be explained using the following example.



Here the number is written from top - .001. So the binary equivalent of 0.125 is .001

Therefore, from the above two examples, we can conclude that the binary equivalent of the decimal number 14.125 is 1110.001.

Binary Addition

The addition of numbers in the binary system is shown in the table 5.2 and is illustrated by the examples.

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	Carry --> 1 0

The addition of 101101 and 1111 (which are 45 and 15 in the decimal system) is done as follows:

Binary	Decimal
101101	45
+ 001111	+ 15
111100	60

Similarly, the addition of 1111011 and 11011 (which are 123 and 27 in the decimal system) is :

Binary	Decimal
1111011	123
+ 0011011	+ 27
10010110	150

The subtraction of Binary Numbers is given the table.

Binary Subtraction

The subtraction of 1111 from 101101 (which are 15 and 45 in the decimal system) is done as follows:

Binary	Decimal
101101	45
- 001111	- 15
11110	30

Similarly the subtraction of 11011 from 1111011(which are 27 and 123 in the decimal system) is:

Binary	Decimal
1111011	123
+ 0011011	- 27
1100000	96

When you are subtraction a larger number from a smaller number, the result obtained will be the 2's complement. If we subtract 45 (101101) from 15 (1111), we should get – 30. But when we do the binary subtraction, instead of getting - 11110 (-30) we will get the 2's complement of 11110 which is 10.

Binary	Decimal
001111	15
- 101101	- 45
Ignore Carry -> 1 00010	-30
10	

1.5.4 COMPLEMENTS

Computers use complemented numbers or complements to perform subtraction. In the binary number system there are two types of complements –

1's complement and 2's complement. Similarly, in the decimal number system also their two types of complements-9's complement and the 10's complement.

9's Complement

The 9's complement of a decimal number is obtained by subtracting each digit of the number from 9.

Example 9's complement of 2 is 9-2=7

9's complement of 123 is 999-123=876

10's Complement

The 10's complement of decimal number is obtained by adding 1 to 9's complement of that number.

Example 10's complement of 2 is 9-2=7+1=8

10's complement of 123 is 999-123=876+1=877

1's Complement

To get the 1's complement at a number replace 0 by 1 & 1 by 0. For example, the 1's complement of 1010 is 0101, that of 1111 is 0000 and so on.

2's Complement

To get the 2's complement of a number, add 1 to the 1's complement of the number. For example, the 2's complement of 1010 is 0110, that of 1111 is 0001, etc.

Just as adding a number to the 10's complement of another number is equivalent to subtraction the second number from the first in the decimal system, adding a number to the 2's complement of another number is equivalent to subtracting the second number from the first in the binary system. For example, if you want to subtract 101 from 1111, you add the 2's complement of 101 to 1111 as shown below :

10's complement method	Normal method
1111	1111
+ 1011	- 0101
Ignore Carry - >1 1010	1010
1010	

1.5.5 SIGNED AND UNSIGNED NUMBER REPRESENTATIONS

We put a plus (+) or minus (-) sign before the number to represent its sign. In computers such notations cannot be employed and therefore, a different method is used. To represent a positive number a 0 is placed before the binary number. Similarly, to represent a negative number, a 1 is placed before the binary number. For example +15 and -15 are represented by **0**1111

and **1**1111 respectively. There is only one way to represent a positive number, but there are different ways to represent a negative number. These are:

- Signed – magnitude representation
- Signed – 1's complement representation
- Signed – 2's complement representation

The number 15 can be represented in the above three ways as **1**1111, **1**0000 and **1**0001 respectively. Since 15 is represented by 4 bits and a separate bit is used to represent sign, in a computer, the most significant bit (MSB) can be used to represent the sign of the number.

For example, 8-bit computers will represent- 15 as **1**0001111, 10000000 and **1**0000001 for signed- magnitude, signed -1's complement and signed-2's complement respectively. 7 bits are used to represent the number and the MSB is used to represent the sign of the number.

When all the bits of the computer word (in an 8-bit computer, the length of a word is 8 bits) are used to represent the number and no bit is used for sign representation, it is called unsigned representation of numbers.

1.5.6 FIXED – POINT REPRESENTATION OF NUMBERS

In the fixed – point number representation system, all numbers are represented as integers or fractions. Signed integer or BCD numbers are referred to as fixed-point numbers because they contain no information regarding the location of the decimal point or the binary point. The binary or decimal point is assumed to be at the extreme right or left of the number.

If the binary or decimal point is at the extreme right of the computer word, then all numbers are positive or negative integers. If the radix point is assumed to be at the extreme left, then all numbers are positive or negative fractions.

Consider that you have to multiply 23.15 and 33.45. This will be represented as 2315 x 3345. The result will be 7743675. The decimal point has to be placed by the user to get the correct result, which is 774.3675. So in the fixed-point representation system, the user has to keep track of the radix point, which can be a tedious job.

1.5.7 FLOATING – POINT REPRESENTATION OF NUMBERS

In most computing applications, fractions are used very used frequently. So a system of number representation, which automatically keeps track of the position of the binary or decimal point, is better than the fixed-point representation. Such a system is the floating-point representation of numbers.

A number, which has both an integer part and a fractional part, is called a real number or a floating-point number. These numbers can be dither positive or negative. Examples of real numbers (decimal) are 123.23, -56.899, 0.008, etc. The real number 123.23 can be written as 1.2323×10^2 or 0.12323×10^3. Similarly the numbers 0.008 and 1345.66 can be represented as 0.8×10^{-2} and 1.34566×10^3 respectively. This of representation is called the *scientific*

representation. Using this scientific form, any number can be expressed as a combination of a mantissa and an exponent, or in other words, the number 'n' can be expressed as 'n=mr^e' where 'm' is the **mantissa, '**r' is the **radix** of the number system and 'e' is the **exponent.**

In a computer also the real or floating – point number is represented by two parts – mantissa and exponent. Mantissa is a signed fixed point number and the exponent indicates the position of the binary or decimal point. For example, the number 123.23 is represented in the floating-point system as:

Sign		Sign	
0	.12323	0	03
Mantissa		Exponent	

The zero in the left most position of the mantissa and exponent indicates the plus sign. The mantissa can be either a fraction or an integer, which is dependent on the computer manufacturer. Most computers use the fractional system of representation for mantissa. The decimal point shown above is an assumed decimal point and is not stored in the register.

The exponent of the above example, +3, indicates that the actual decimal point is three digits to the right of the assumed one. In the above example, the mantissa is shown as a fraction. As mentioned , we can use an integer as the mantissa. The following example shows how it is done.

Sign		Sign	
0	12323	1	02
Mantissa		Exponent	

In the above representation, the sign of the exponent is negative and it indicates that the actual decimal point lies two decimal positions to the left of the assumed point (in this case, the assumed decimal point is placed at the extreme right of the integer or 12323).

A negative number say-123.23 can be expressed as follows

Sign		Sign	
1	.12323	0	03
Mantissa		Exponent	

A negative fraction, say-0.0012323 can be represented as follows

Sign		Sign	
1	.12323	1	02
Mantissa		Exponent	

1.5.8 BINARY CODED DECIMAL (BCD)

The BCD is the simplest binary code that is used to represent a decimal number. In the BCD code, 4 bits represent a decimal number. For example, 2 is represented as 0010. If a decimal number consists of more than 1 digit, each decimal digit is represented individually by its 4-bit binary equivalent for example, 123 is represented as 0001 0010 0011. There is a difference between the equivalent of a decimal number and its BCD code. For example, the binary equivalent of 45 is 101101 and its BCD code is 0100 0101. Computers perform subtraction-using complements and there is difficulty in forming complements when numbers are representing in BCD. For example 1's complement of 2 (0010) is 1101, which is 13 in the decimal system and is not an acceptable BCD code. To overcome this difficulty, other BCD codes such as Excess -3 are used.

1.5.9 GRAY CODE

The Gray code is binary code. It is used in shift encoder, which indicates the angular position of a shift in digital form. The bits of arranged in such a way that only one bit changes at a time when we make a change from one number to the next. Its use reduces the error in reading shift position. The largest possible errors will be one least significant digit. The gray code is often used in computer controlled machines such as lathes, etc,. Photoelectric coders or shift position encoders are used as sensors.

Decimal	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

The table 5.3 shows the Gray codes for the decimal numbers 0 through 15.

1.5.10 EXCESS-3 CODE

As mentioned above, to overcome the disadvantages of BCD in forming complements, other systems like Excess-3 are used. Adding 3 to the decimal number and forming the binary coded number form this code. For instance, to form the Excess-3 representation of 5, first 3 is added to 5 yielding 8, and normal BCD is used, which is 1000. Similarly, the decimal number 123 coded in Excess-3 will be 0011 0100 0101. The table 5.4 shows the BCD and Excess-3 codes for decimal number 0 through 9.

Decimal	BCD	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

1.5.11 ASCII CODE

ASCII stands for American Standard Code for Information Interchange. ASCII code is used extensively in small computers, peripherals, instruments and communications devices. It is a seven-bit code. Microcomputers using 8-bit word length use 7 bits to represent the basic code. The 8th bit is used for parity or it may be permanently 1 or 0.

With 7 bits, up to 128 characters can be coded. A letter, digit or special symbol is called a character. It includes upper and lower case alphabets, numbers, punctuation mark and special and control characters.

ASCII-8 Code

A newer version of ASCII is the ASCII-8 code, which is an 8-bit code. With 8 bits, the code capacity is extended to 256 characters.

1.5.12 EBCDIC CODE

EBCDIC stands for Extended BCD Interchange Code. It is the standard character code for large computers. It is an 8-bit code without parity. A 9th bit can be used for parity. With 8 bits up to 256 characters can be coded.

In ASCII-8 and EBCDIC, the first 4 bits are known as zone bits and the remaining 4 bits represent digit values. In ASCII, the first 3 bits are zone bits and the remaining 4 bits represent digit values.

Some examples of ASCII and EBCDIC values are shown in the table 5.5.

Character	ASCII	EBCDIC
0	00110000	11110000
1	00110001	11110001
..	..	..
9	00111001	11111001
A	01000001	11000001
B	01000010	11000010
..	..	..
Z	01011010	11101001

1.5.13 BITS, BYTES AND WORDS

A byte is a basic grouping of bits (binary digits) that the computer operates on as a single unit. It consists of 8 bits and is used to represent a character by the ASCII and EBCDIC coding systems. For example, each storage location of computers using EBCDIC or ASCII-8 codes consist of electronic circuit elements or magnetic or optical media positions that can represent at least 8 bits. Thus each storage location can hold one character. The capacity of a computer's primary storage and its secondary storage devices is usually expressed in terms of bytes.

A word is a grouping of bits (usually larger than a byte) that is transferred as a unit between primary storage and the registers of the ALU and

control unit. Thus, a computer with a 32-bit word length might have registers with a capacity of 32 bits, and transfer data and instructions within the CPU in groupings of 32 bits. It should process data faster than computers with a 16-bit or 8-bit word length.

1.5.14 OCTAL NUMBER SYSTEM

Refers to the base-8 number system, which uses just eight unique symbols (0,1,2,3,4,5,6, and 7).

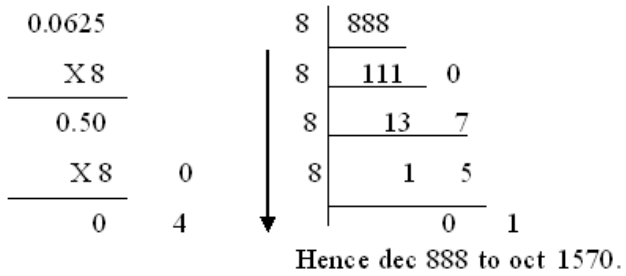
In octal format, each digit represents three binary digits, as shown in the table 5.6.

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

With this table, it is easy to translate between octal and binary. For example, the octal number 3456 is 011 100 101 110 in binary. To convert an octal number to decimal, the same method used for binary- decimal conversion is used, the only difference is that, instead of 2 the base is 8. For example, the octal number 24.25 is 20.328125 in decimal as shown below.

$24.25 = 2 \times 8^1 + 4 \times 8^0 + 2 \times 8^{-1} + 5 \times 8^{-2} = 16 + 4 + 0.25 + 0.078125 = 20.328125$

To convert a decimal number to octal system, it is repeatedly divided by 8 as illustrated in the following example, where the number 888 is converted to octal system (1570).



Hence dec 888 to oct 1570.

Here the number is written from top – .04. So the binary equivalent of 0.0625 is .04.

Therefore, from the above two examples, we can conclude that the binary equivalent of the decimal number 888.0625 is 1570.04.

1.5.15 HEXADECIMAL NUMBER SYSTEM

Hexadecimal number system uses 16 as the base or radix. This base -16 number system consists of 16 unique symbols: the numbers 0 to 9 and the letters A to F. For example, the decimal number 15 is represented as F in the hexadecimal numbering system as shown in the table 5.7.

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

The hexadecimal system is useful because it can represent every byte (8bits) as two consecutive hexadecimal digits. Compared to binary numbers, hexadecimal numbers are easier for humans to read.

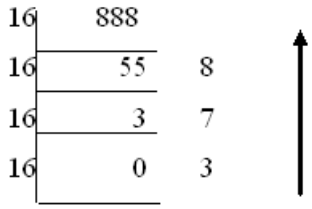
To convert a value from hexadecimal to binary, you merely translate each hexadecimal digit into 4-bit binary equivalent. For example, the hexadecimal number 3F7A translates to the following binary number: 0011 1111 0111 1010.

To convert a hexadecimal number to decimal the same method used for binary-decimal conversion is used, the only difference is that instead of 2 the base is 16. for example, the hexadecimal number 24.25is 36.14453125 in decimal as shown below:

$24.25 = 2x16^{1}+4x16^{0}+2x16^{-1}+5x16^{-2} = 32+4+0.125+0.01953125$

$= 36.14453125$

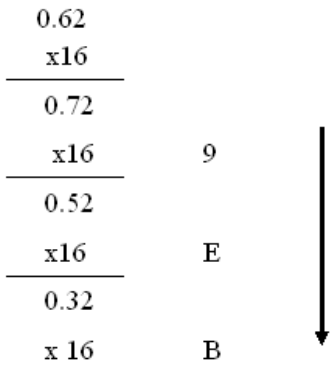
To convert a decimal number to hexadecimal system, it is repeatedly divided by 16 as illustrated in the following example. To convert 888 to hexadecimal system.



The number is written from below, that is 378. So the hexadecimal equivalent of 888 is 378.

If the decimal number has a fractional part, then the fractional part is converted into hexadecimal by multiplying it with 16. Only the integer of the result is noted and the fraction is repeatedly multiplied by 16 until the fractional part has become 0.

This can be explained using the following example.



Therefore, from the above two examples, we can conclude that the hexadecimal equivalent of the decimal number 888.62 is 378.9EB approximately.

UNIT – I

Self Assessment Questions

Fill in the blank

1. _____ is a programmable instructions.
2. The Actual machinery in a computer is called _____ and the instructions and data are called _____.
3. The number of bits that a computer can process at a time in parallel is called its _____.
4. The name of the first digital computer is _____ and built the first digital computer using parts called_____.
5. The second generation computers contained _____ in place of vacuum tubes.
6. The use of operating systems in computer was introduced in the _____ generation computers.
7. _____ Computers aim to solve highly complex problems, ones which require reasoning, intelligence and expertise when solved by people.
8. The Three categories of digital computers are _____, _____, and _____.
9. The base of the octal number system is _____.
10. BCD stands for _____ _____ _____.
11. Binary number system uses only two digital _____ and _____.
12. The 9's complement of a decimal number is obtained by subtracting each digit from _____.

True or False

1. Main frame is a powerful multi user computer capable of supporting many hundreds of users simultaneously.
2. GIGO stands for Garbage In Garbage out
3. PDA stands for portable digital assistant.
4. Second generation computers replaced machine language with assembly language.
5. Third generation computers replaced transistors with vacuum tubes.
6. The radix of the hexadecimal number system is 8.
7. When add 1 to the 9's complement you will get the 10's complement.

Multiple Choice

1. A set of prerecorded instructions executed by a computer is called the

   a). Action b) Hardware c) Program d) None of the above

2. Which is the part that transmits data from one part of the computer to another?

   a). Bus b) CPU c) Hard disk d) none of the above

3. LSI stands for

   a) Light sensitive Instrument b) Large scale Integration

   c) Logical sample Integration d) none of the above

4. The BCD of 123 is

   a)0001 010 0011 b) 0001 0011 0010 c)0011 0010 0001

   d)none of the above

5. The Binary equivalent of 20 is

   a) 1111 b). 10100 c) 10101 d) 10011

Unit Questions

1. What are the different types of Computers?
2. What are major components of a computer?
3. What are the characteristics of computer?
4. What are the different generations of computers?
5. What are the features of the third generation computers?
6. What are the different categories of digital computers?
7. What are the different types of portable computers?
8. What is the difference between a minicomputer and a microcomputer?
9. Explain the decimal number system and the binary number system?
10. Explain 1. 1's complement 2. 2's complement
    3. 9's complement 4. 10's complement
    with Examples.
11. Explain (i). Hexadecimal (ii). Octal number system
    (iii). BCD with Examples.
12. What is the Excess-3 and Gray codes?

Self Assessment Answers

Fill in the blanks

1. Computer
2. Hardware, Software
3. Word Length
4. ENIAC, Vacuum
5. Transistors
6. III
7. V Generation
8. Mini, Mainframe & Super computers
9. 8
10. Binary Coded Decimal
11. 0
12. 9

True / False

1. True
2. True
3. False
4. True
5. False
6. False
7. True

Multiple Choice

1. C
2. A
3. B
4. A
5. B

NOTES

……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..

UNIT –II

UNIT-II

2.0 BOOLEAN ALGEBRA AND GATE NETWORKS

Modern digital computers are designed and maintained, and their operation is analyzed, by using techniques and symbology from a field of mathematics called *modern algebra*. Algebraists have studied for over a hundred years mathematical systems called *boolean algebras.*

2.1 FUNDAMENTAL CONCEPTS OF BOOLEAN ALGAEBRA

When a variable is used in an algebraic formula, it is generally assumed that the variable may take any numerical value. For instance, in the formula 2X - 5Y = Z, we assume that X, Y, and Z may range through the entire field of real numbers.

The variable used in boolean equations have a unique characteristic, however; they may assume only one of two possible values. These two values may be represented by the symbols 0 and 1. If an equation describing logical circuitry has several variables, it is still understood that each of the variables can assume only the value 0 or 1. For instance, in the equation X + Y = Z, each of the variables X, Y, and Z may have only the values 0 or 1.

This concept will become clearer if a symbol is defined, the + symbol. When the + symbol is placed between two variables, say X and Y , since both X and Y can take only the role 0 or 1, we can define the + symbol by listing all possible combinations for X and Y and the resulting values of X + Y.

The possible input and output combinations may be arranged as follows:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

This is a *logical* addition table and could represent a standard binary addition table except the last entry. When both X and Y represent 1s, the value of X + Y is 1. The + symbol therefore, does not have “normal” meaning, but is a logical addition or logical OR symbol. The equation X + Y = Z can be read “X or Y equals Z” or “X plus Y equals Z”. This concept may be extended to any number of variables. For instance, in the equation A + B + C + D = E, even if A, B, C, and D all had the value of 1, E would represent only a 1.

To avoid ambiguity, a number of other symbols have been recommended as replacement for the + sign. Some of these[3] are U, v and V. computer people still use the + sign, however, which was the symbol originally proposed by Boole.

2.2 LOGICAL MULTIPLICATION

A second important operation in Boolean algebra we call *logical multiplication* or the *logical* AND *operation*[4]. The rules of this operation can be given by simply listing all values that might occur:

$$0.0 = 0$$

$$0.1 = 0$$

$$1.0 = 0$$

$$1.1 = 1$$

Thus, for instance, if we write Z = X **.** Y and find X = 0 and Y = 1, then Z = 0. Only when X and Y are both 1s would Z be a 1.

Both + and. Obey a mathematical rule called the *associative law.* This law says, for +, that (X + Y)+Z = X + (Y + Z) and for . , that X.(Y.Z) = (X.Y).Z. This means that we can write X + Y + Z without ambiguity, for no matter in what order the operation is performed, the result is the same.

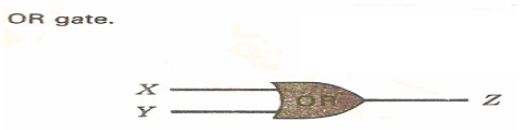
2.3 AND GATES and OR GATES

The + and **.** operations are physically realized by two types of electronic circuits, called *OR gates* and *AND gates*.

A *gate* is simply an electronic circuit which operates on one or more input signals to produce an output signal. One of the simplest and most frequently used gates is called the OR gate, and the block diagram symbol for the OR gate is shown figure 1, as is the table of combinations for the inputs and outputs for the OR gate.

Similarly, the AND gate in figure 2 ANDs or logically multiplies input values, yielding an output Z with value X . Y, so that Z is a 1 only when both X and Y are 1s.

Figure 1.



INPUT		OUTPUT
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2.

AND gate

INPUT		OUTPUT
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

2.4 COMPLEMENTATION AND INVERTERS

The two operations defined so far have been what algebraists would call *binary operations* in that they define an operation on two variables. There are also *singular* or *unary, operations,* which define an operation on a single variable. A familiar example of unary operation is -, for we can write -5 or -10 or –X, meaning that we are to take the negative of these values.

In Boolean algebra we have an operation called *complementation,* and the symbol we use is $\overline{\ }$. Thus we write $\overline{X}$, meaning "take the complement of X," or $\overline{(X+Y)}$, meaning "take the complement of X + Y." the complement operation can be defined quite simply:

$$\overline{0} = 1$$

$$\overline{1} = 0$$

The complement of a value can be taken repeatedly. For instance, we can find $\overline{\overline{X}}$. For X = 0 it is $\overline{0} = 1$ $\overline{\overline{0}} = \overline{1} = 0$, and for X = 1 it is $\overline{1} = 0$ $\overline{\overline{1}} = \overline{0} = 1$.

The complementation operation is physically realized by a gate or circuit called an *inverter.* Figure 3 shows an inverter and the table of combinations for its input and output.

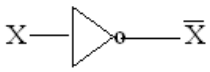


Figure 3.

INPUT	OUTPUT
X	$\overline{X}$
0	1
1	0

2.5 EVALUATION OF LOGICAL EXPRESSION

The tables of values for the three operations just explained are sometimes called *truth tables,* or *tables of combinations*. To study a logical expression, it is very useful to construct a table of values for the variables and then to evaluate the expression for each of the possible combinations of variables in turn. Consider the expression X + YZ. There are three variables in this expression : X, Y and Z, each of which can assume the value 0 or 1. The possible combinations of values may be arranged in ascending order,[5], as in Table 1.

One of the variables, Z, is complemented in the expression X + YZ. So $\overline{a}$ column is now added to the table listing values of Z (see Table 2).

A column is now added listing the values that YZ assumes for each value of the X, Y, and Z. This column will contain the value 1 only when both Y is a 1and Z is a 1 (see Table 3)

Now the ORing, or logical addition, of the values of X to the values which have been calculated for YZ is performed in a final column (see Table 4).

The final column contains the value of X +YZ for each set of inputs values which X, Y, and Z may take. For instance, when X = 1, Y = 0, and Z = 1, the expression has the value of 1.

TABLE 1

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

TABLE 2

X	Y	Z	$\overline{Z}$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

TABLE 3

X	Y	Z	$\overline{Z}$	$Y\overline{Z}$
0	0	0	1	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

X	Y	Z	$\overline{Z}$	$Y\overline{Z}$	$X + Y\overline{Z}$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	0	0	1

Table 4

2.6EVALUATION OF AN EXPRESSION CONTAINING PARENTHESES

The following example illustrations the procedure for constructing a truth table for the expression X + Y(X + Y). There are only two variables in the expression, X and Y. First a table of the values which X and Y may assume is constructed.(see table 5)

Now, since the expression contains both X and $\overline{Y}$, two columns are added listing complements of the original values of the variables (see table 6).

The various values of $X + \overline{Y}$ are now calculated (see table 7).

The values for $X + \overline{Y}$ are now multiplied (ANDed) by the values of Y in the table, forming another column representing $Y(X + \overline{Y})$(see table 8).

Finally the values for $Y(X + \overline{Y})$ are added (ORed) to the values for X which are listed, forming the final column and completing the table(see table 9).

Inspection of the final column of the table indicates that the values taken by the function X + Y(X + Y) are identical with the values found in the table

for ORing X and Y. This indicates that the function X + Y(X + Y) is equivalent to the function X + Y. This equivalence has been established by trying each possible combination of values in the variables and noting that both expressions then have the same value. This is called a *proof by perfect induction*. If a logic circuit were constructed for each of the two expressions, both circuits would perform the same function, yielding identical outputs for each combination of inputs.

Table 5

X	Y
0	0
0	1
1	0
1	1

X	Y	$\overline{X}$	$\overline{Y}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

Table 6

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} + \overline{Y}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

Table 7

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} + \overline{Y}$	$Y(\overline{X} + \overline{Y})$
0	0	1	1	1	0
0	1	1	0	1	1
1	0	0	1	1	0
1	1	0	0	0	0

Table 8

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} + \overline{Y}$	$Y(\overline{X} + \overline{Y})$	$X + Y(\overline{X} + \overline{Y})$
0	0	1	1	1	0	0
0	1	1	0	1	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	1

Table 9

Some fundamental relations of Boolean algebra have been presented. A complete set of the basic operations is listed below.

	TABLE BOOLEAN ALGEBRA RULES
0	$0 + X = X$
1	$1 + X = 1$
2	$X + X = X$
3	$X + \overline{X} = 1$
4	$0 . X = 0$
5	$1 . X = X$
6	$X . X = X$
7	$X . \overline{X} = 0$
8	$\overline{\overline{X}} = X$
9	$X + Y = Y + X$
10	$X . Y = Y . X$
11	$X + (Y + Z) = (X + Y) + Z$
12	$X (YZ) = (XY) Z$
13	$X (Y + Z) = XY + XZ$
14	$X + XZ = X$
15	$X (X + Y) = X$
16	$X + Y (X + Z) = X + YZ$
17	$X + \overline{X}Y = X + Y$
18	$XY + YZ + \overline{Y}Z = XY + Z$

Table 10

A list of useful relations is presented in table 10. Most of the basic rules by which Boolean algebra expressions may be manipulated are contained in this table. Each rule may be proved by using the proof by perfect induction. An example of this proof for rule 3 in table 10 is as follows: the variable X can have only the value 0 or 1. If X has the value 0, then 0 + 0 = 0; if X has the value 1, then 1 + 1 = 1. Therefore X + X = X.

The same basic technique may be used to prove the remainder of the rules. Rule 9 states that double complementation of a variable results in the original variable. If X equals 0, then the first complement is 1 and the second will be 0, the original value.If the original value for X is 1, then the first complement will be 0 and the second 1, the original value. Therefore X = X

Rules 10 and 11, which are known as the *commutative laws,* express the fact that the order in which a combination of terms is performed does not affect the result of the combination. Rule 10 is the commutative law of addition, which states that the order of addition or ORing does not affect the sum (X + Y = Y + X). Rule 11 is the commutative law of multiplication (XY = YX), which states that the order of multiplication or ANDing does not affect the product.

Rules 12 and 13 are the *associative laws.* Rule 12 states that in the logical addition of several terms, the sum which will be obtained if the firs term is added to the second and then the third term is added will be the same as the sum obtained if the second term is added to the third and then the first term is added [X. + (Y + Z) = (X+Y)+Z]. Rule 13 is the associative law of logical multiplication, stating that in a product with three factors, *any* two may be multiplied, followed by the third [X(YZ) = (XY)Z].

Rule 14, the *distributive law*, states that the product of a variable (X) times a sum (Y + Z) is equal to the sum of the products of the variable multiplied by each term of the sum [X(Y + Z) = XY + XZ]

2.7 SIMPLIFICATIONS OF EXPRESSIONS

The rules given may be used to simplify boolean expressions, just as the rules of normal algebra may be used to simplify expressions. Consider the expression

$$(X + Y)(X + Y)(X + Z)$$

The first terms consist of X + Y and X + Y; these terms may be multiplied and, since X + XY + XY = X and YY = 0, reduced to X.

The expression has been reduced now to X(X + Z), which may be expressed as XX + XZ(rule 14). And since XX is equal to 0, the entire expression (X + Y)(X + Y)(X + Z) may be reduced to XZ.

Another expression that may be simplified is XYZ + XYZ + XYZ. First the three terms XYZ + XYZ + XYZ may be written X(YZ + YZ + YZ), by rule 14. Then, by using rule 14 again, X[Y(Z + Z) + YZ]; and since Z + Z equals 1, we have X(Y + YZ).

The expression X(Y + YZ) may be further reduced to X(Y + Z) by using rule 18. The final expression can be written in two ways: X(Y + Z) or XY + XZ. The first expression is generally preferable if the equation is to be constructed as an electronic circuit, because it requires only one AND circuit and one OR circuit.

2.8 DE MORGAN'S THEOREMS

The following two rules are known as De Morgan's theorems:

$$\overline{(X + Y)} = \overline{X} \cdot \overline{Y}$$

$$\overline{(X \cdot Y)} = \overline{X} + \overline{Y}$$

The complement of any boolean expression, or a part of any expression, may be found by means of these theorems. In these rules, two steps are used to form a complement:

1 The + symbols are replaced with · symbol and · symbols with + symbols.

2 Each of the terms in the expression is complemented.

The use of De Morgan's theorem may be demonstrated by finding the complement of the expression X + YZ. First, note that a multiplication sign has been omitted and the expression could be written X + (Y.Z). To complement this, the addition symbol is replaced with a multiplication symbol and the two terms are complemented, giving $\overline{X}$. ($\overline{Y.Z}$); then the remaining term is complemented $\overline{X}(\overline{Y} + \overline{Z})$. The following equivalence has been found: $\overline{(X + YZ)} = \overline{X}(\overline{Y} + \overline{Z})$.

The complement of $W\overline{X}$ + YZ may be formed by two steps:

1 The addition symbol is changed

2 The complement of each term is formed:

$$(W \overline{\cdot} \overline{X})(Y \overline{\cdot} Z) \overline{}$$

This becomes $(W + X)(\overline{Y} + \overline{Z})$.

Since W and Z were already complemented, they become uncomplemented

by the theorem $\overline{\overline{X}} = X$.

It is sometimes necessary to complement both sides of an equation. This may be done in the same way as before:

$$WX + YZ = 0$$

Complementing both sides gives

$$\overline{(WX + YZ)} = \overline{0}$$

$$(W + \overline{X})(\overline{Y} + \overline{Z}) = 1$$

2.9 BASIC DUALITY OF BOOLEAN ALGEBRA

De Morgan's theorem expresses a basic duality which underlies all boolean algebra. The postulates and theorems which have been presented can all be divided into pairs. For example, (X + Y) + Z = X + (Y + Z) is the *dual* of (XY)Z = X(YZ), and X + 0 = X is the dual of X . 1 = X.

Often the rules or theorems are listed in an order which illustrates the duality of the algebra. In proving the theorems or rules of the algebra, it is then necessary to prove only one theorem, and the dual of the theorem follows necessarily. For instance, if you prove that X +XY = X, you can immediately add the theorem X(X + Y) = X to the list of theorems as the dual of the first expression.[8] In effect all boolean algebra is predicated on this two-for-one basis.

2.10 DERIVATION OF A BOOLEAN EXPRESSION

When designing a logical circuit, the logical designer works from two sets of known values: (1) the various which the inputs to the logical network can take and (2) the desired outputs for each input condition. The logical expression is derived from these sets of values.

Consider a specific problem. A logical network has two inputs X and Y and an output Z. The relationship between inputs and outputs is to be as follows:

1. When both X and Y are 0s, the output Z is to be 1.
2. when X is 0 and Y is 1, the output Z is to be 0.
3. when X is 1 and Y is 0, the output Z is to be 1.
4. when X is 1 and Y is 1, the output Z is to be 1.

These relations may be expressed in tabular form, as shown in table 11

TABLE		
INPUTS		OUTPUT
X	Y	Z
0	0	1
0	1	0
1	0	1
1	1	1

Table 11

It is now necessary to add another column to the table. This column will consist of a list of *product terms* obtained from the values of the input variables. The new column will contain each of the input variables listed in each row of the table, with the letter representing the respective input complemented when the input value for this variable is 0 and not complemented when the input value is 1. The terms obtained in this manner are designated as product terms. With two input variables X and Y, each row of the table will contain a product term consisting of X and Y, with X or Y complemented or not, depending on the input values for that row (see table 12)

Whenever Z is equal to 1, the X and Y product term from the same row is removed and formed into a *sum-of-products* expression. Therefore, the product terms from the first, third, and fourth rows are selected. These are $\bar{X}\bar{Y}$, $\bar{X}Y$, and $X\bar{Y}$.

INPUTS		OUTPUT	PRODUCT
X	Y	Z	TERMS
0	0	1	$\bar{X}\bar{Y}$
0	1	0	$\bar{X}Y$
1	0	1	$X\bar{Y}$
1	1	1	XY

Table 12

X	Y	$\overline{Y}$	$X + \overline{Y}$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

Table 13

There are now three items, each the product of two variables. The logical sum of these products is equal to the expression desired. This type of expression is often referred to as a *canonical expansion* for the function. The complete expression in normal form is

$$\overline{X}\overline{Y} + X\overline{Y} + XY = Z$$

The left-hand side of the expression may be simplified as follows:

$$\overline{X}\overline{Y} + X\overline{Y} + XY = Z$$

$$\overline{X}\overline{Y} + X(\overline{Y} + Y) = Z$$

$$\overline{X}\overline{Y} + X = Z$$

And finally, by rule 18 in table 10, $X + \overline{Y} = Z$.

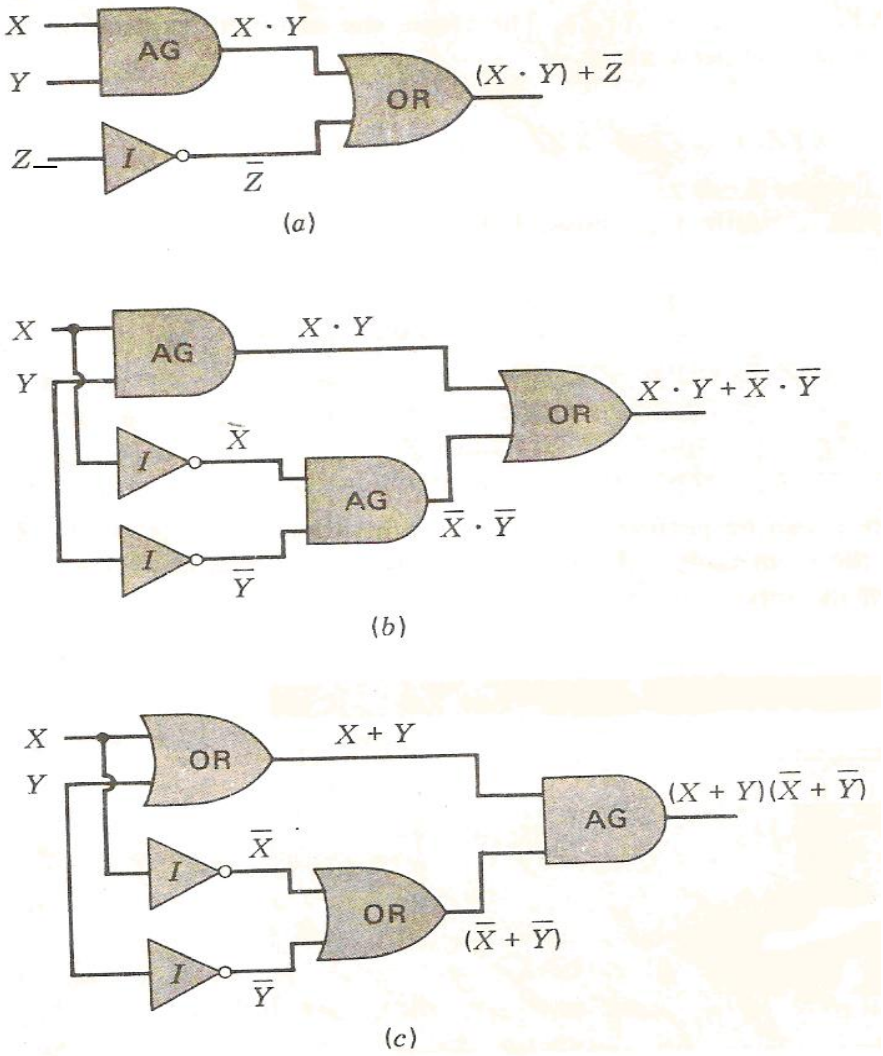
The truth table may be constructed to check the function that has been derived (see table 13). The last column of this table agrees with the last column of the truth table of the desired function, showing that the expressions are equivalent.

2.11 INTERCONNECTING GATES

The OR gates, AND gates, and inverters described 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. The below figure 4(a) shows a gating network with three inputs X, Y, and z and an output expression (X.Y) + Z. A network that forms (X.Y) + (X.Y) and another network that forms (X + Y).(X + Y) are shown in figure 4 (b) and (c).

Figure 4

Three gating networks.



.

2.12 SUM OF PRODUCTS AND PRODUCT OF SUMS

An important consideration in dealing with gating circuits and their algebraic counterparts is the *form* of the boolean algebra expression and the resulting form of the gating network. Certain types of Boolean algebra expressions lead to gating networks, which are more desirable from most implementation viewpoints.

We now define the two most used and usable forms for Boolean expressions.

First let us define terms

1. *Product term* A product term is a single variable or the logical product of several variables. The variables may or may not be complemented.

2. *Sum term* A sum term is a single variable or the sum of several variables. The variables may or may not be complemented.

For example, the term X.Y.Z is a product term ; X + Y is a sum term; X is both a product term and a sum term; X + Y . Z is neither a product term nor a sum term; X + Y is a sum term; X.Y.$\overline{Z}$ is a product term; $\overline{Y}$ is a both a sum term and a product term.

We now define two most important types of expressions.

1. *sum of product expressions* A sum-of-products expression is a product term or several product terms logically added.
2. *product-of-sums expressions* A product-of-sums expression is a sum term or several sum terms logically multiplied.

   For example, the expression X.$\overline{Y}$ + X.Y is a sum-of-products expression; (X + Y)($\overline{X}$ + Y) is a product-of-sums expression. The following are all sum-of-product expressions:

$$X$$
$$X.Y + Z$$
$$\overline{X}.\overline{Y} + \overline{X}.\overline{Y}.\overline{Z}$$
$$X + Y$$

The following are product-of-sums expressions:

$$(X + Y).(X + \overline{Y}).(\overline{X} + \overline{Y})$$
$$(X + Y + Z).(X + \overline{Y}).(\overline{X} + \overline{Y})$$
$$X + Z$$
$$\overline{X}$$
$$(X + Y)X$$

One prime reason for liking sum-of-products or product-of-sums expressions is their straightforward conversion to very nice gating networks.

Figure 3.7 shows several gating networks. Figure 3.7(a) shows sum-of-product networks, and figure 3.7(b) shows product-of-sums networks.

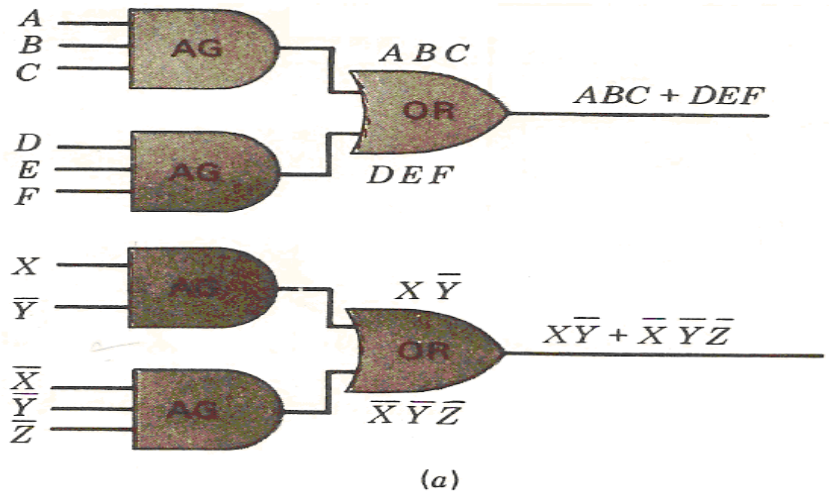
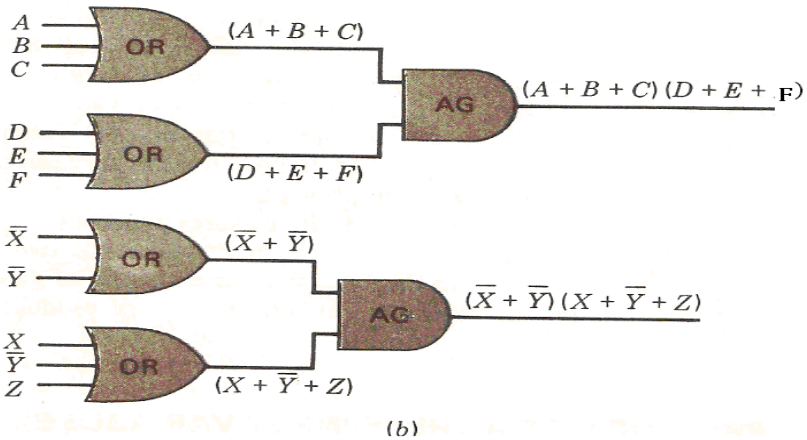


Figure3.7



2.13 DERIVATION OF RPODUCT-OF-SUMS EXPRESSION

The method for arriving at the desired expression is as follows:

1 Construct a table of the input and output values.

2 Construct an additional column of sum terms containing complemented and uncomplemented variables (depending on the values in the input columns) for each row of the table. In each row of the table, a sum term is formed. However, in this case, if the input value for a given variable is 1, the variable will be complemented; and if 0, not complemented.

3 The desired expression is the product of the sum terms from the rows in which the output is 0.

The use of these rules is illustrated by working examples in this and the following sections.

Table 16 contains the input and output values which describe a function to be

TABLE 16

INPUT		OUTPUT
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

realized by a logical network.

A column containing the input variables in sum-term form is now added in each row. A given variable is complemented if the input value for the variable is 1 in the same row and is not complemented if the value is 0(see table 17). Each sum term is, therefore, simply the complement of the product term which occurs in the same row in the previous table for sum-of-products expressions. Notice that the sum term X + Y in the third row of Table 3.20 is the complement of the product term $X\underline{Y}$ used in the sum-of-products derivation.

A product-of-sum expression is now formed by selecting those sum terms for which the output is 0 and multiplying them. In this case, 0s appear in the second and third rows, showing that the desired expression is $(\overline{X} + \overline{Y})(\overline{X} + Y)$. A sum-of-products expression may be found by multiplying the two terms of this expression, yielding XY + XY. I this case the same number of gates would be required to construct circuits corresponding to both the sum-of-products and the product-of-sum expressions.

2.14 DERIVATION OF A THREE-INPUT-VARIABLE EXPRESSION

Consider Table 18, expressing an input-to-output relationship for which expression is to be derived. Two columns will be added this time, one containing the sum-of-products terms and the other the product-of-sums terms(see table 19). The two expressions may be written in the following way:

Sum-of-products:

$$\overline{X}Y\overline{Z} + \overline{X}YZ + XY\overline{Z} = A$$

TABLE 17

INPUTS		OUTPUT	SUM
$\overline{X}$	Y	Z	TERMS
0	0	1	$\overline{X} + Y$
0	1	0	$\overline{X} + \overline{Y}$
1	0	0	$\overline{\overline{X}} + Y$
1	1	1	$\overline{\overline{X}} + \overline{Y}$

Table 18

INPUTS			OUTPUT
$\overline{X}$	Y	$\overline{Z}$	A
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Product-of-sums:

$$(X + Y + Z)(X + Y + \overline{Z})(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z})(\overline{X} + \overline{Y} + \overline{Z}) = A$$

The two expressions may be simplified as shown:

SUM OF PRODUCTS

$$\begin{aligned}
(\overline{X}\,\overline{Y}\overline{Z}) + (\overline{X}YZ) + (X\overline{Y}\overline{Z}) &= A \\
\overline{X}(Y\overline{Z} + YZ) + (X\overline{Y}\overline{Z}) &= A \\
\overline{X}Y + X\overline{Y}\overline{Z} &= A \\
Y(\overline{X} + X\overline{Z}) &= A \\
\overline{X}Y + Y\overline{Z} &= A
\end{aligned}$$

PRODUCT OF SUMS

$$\begin{aligned}
(X + Y + Z)(X + Y + \overline{Z})(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z})(\overline{X} + \overline{Y} + \overline{Z}) &= A \\
(X + Y)(\overline{X} + Y)(\overline{X} + \overline{Z}) &= A \\
Y(\overline{X} + \overline{Z}) &= A
\end{aligned}$$

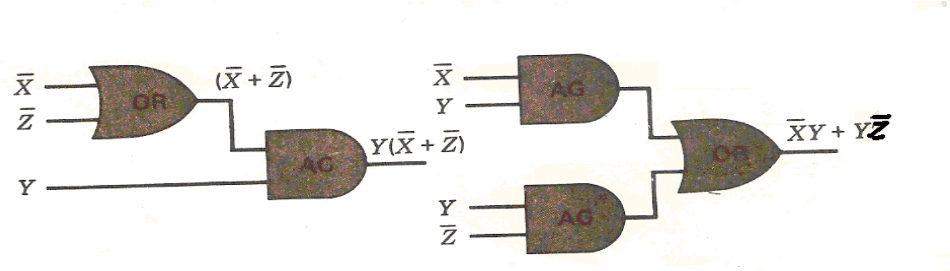
The two final expressions clearly can be seen to be equivalent, Notice, however, that the shortest sum-of-products expression, which is $XY + \overline{Y}Z$, requires two AND gates and an OR gate(fig 3.8), while the shortest product-of-sums expression,

$Y(\overline{X} + \overline{Z})$, requires only a single AND gate and a single OR gate.

Figure 3.8

TABLE 19

INPUT			OUTPUT	PRODUCT	
X	Y	Z	A	TERMS	SUM TERMS
0	0	0	0	$\bar{X}\bar{Y}\bar{Z}$	$X+Y+Z$
0	0	1	0	$\bar{X}\bar{Y}Z$	$X+Y+\bar{Z}$
0	1	0	1	$\bar{X}Y\bar{Z}$	$X+\bar{Y}+Z$
0	1	1	1	$\bar{X}YZ$	$X+\bar{Y}+\bar{Z}$
1	0	0	0	$X\bar{Y}\bar{Z}$	$\bar{X}+Y+Z$
1	0	1	0	$X\bar{Y}Z$	$\bar{X}+Y+\bar{Z}$
1	1	0	1	$XY\bar{Z}$	$\bar{X}+\bar{Y}+Z$
1	1	1	0	XYZ	$\bar{X}+\bar{Y}+\bar{Z}$



2.15 NAND GATES AND NOR GATES

Two other types of gates, NAND gates and NOR gates, are often used in computers.

A NAND gate is shown in figure3.9. The inputs are A, B and C, and the output from the gate is written A ~~+~~B ~~+C~~. ~~th~~e output will be a 1 if A is a 0 or B is a 0 or C is a 0, and the output will be a 0 only is A and B and C are all 1s.

The operation of the gate can be analyzed using the equivalent block diagram circuit shown fig 3.9, which has an AND gate followed by an inverter. If the inputs are A, B and C, the output of the AND gate will be A.B.C, and the complement of this is $\overline{(A.B.C)} = \bar{A} + \bar{B} + \bar{C}$, as shown in the figure.

Figure 3.9

NAND gate

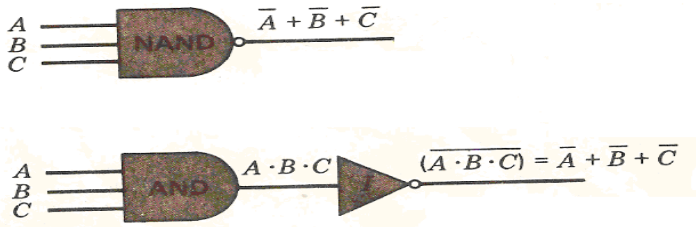
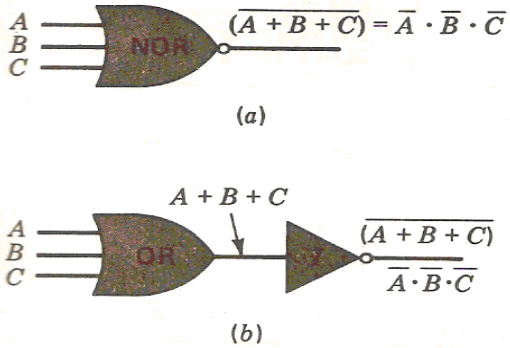


Figure 3.10.

NOR gate



INPUT			OUTPUT
A	B	C	
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

The NOR gate can be analyzed in a similar manner. Figure3.10 shows the NOR gate block diagram symbol with inputs, A, B, C and output $\overline{A}\,\overline{B}\,\overline{C}$. This shows the NOR gate's output will be 1 only when all three inputs are 0s. If any input represents a 1, the output of a NOR gate will be a 0.

Below the NOR gate block diagram symbol in fig 3.10 is an equivalent circuit showing an OR gate and an inverter.[9] The inputs A, B and C are ORed by the OR gate, giving A + B + C, which is complemented by he inverter, yielding

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

2.16 MAP METHOD FOR SIMPLYFYING EXPRESSIONS

There are several other ways to represent or list function values, and the use of certain kinds of maps, also permits minimization of the expression

The particular type of map we use is called *karnaugh map* figure 3.15 shows the layouts for Karnaugh maps of two to four variables. The diagram in each case lists the 2^n different product terms which can be formed in exactly *n* variables, each in a different square. For a function of *n* variables, a product term in exactly these *n* variables is called a *minterm.Thus for* are three variables X, Y and Z there are 2^3, or 8, different *minterms*, which are $\overline{X}\,\overline{Y}\,\overline{Z}$, $\overline{X}\overline{Y}Z$, $\overline{X}Y\overline{Z}$, $\overline{X}YZ$, $X\overline{Y}\overline{Z}$, $X\overline{Y}Z$, $XY\overline{Z}$, and XYZ. For four variables there are 2^4, or 16, terms;

for five variables there are 32 terms; etc. As a result, a map of *n* variables will have 2^n squares, each representing a single minterm. The minterm in each box, or cell, of the map is the product of the variables listed at the abscissa and ordinate of the cell. Thus XYZ is at the intersection of XY and Z.

Given a Karnaugh map form, the map is filled in by placing 1s in the squares, or cells, for each term which leads to a 1 output.

As an example, consider a function of three variables for which the following input values are to be 1:

$$X = 0, Y = 1, Z = 0$$
$$X = 0, Y = 1, Z = 1$$
$$X = 1, Y = 1, Z = 0$$
$$X = 1, Y = 1, Z = 1$$

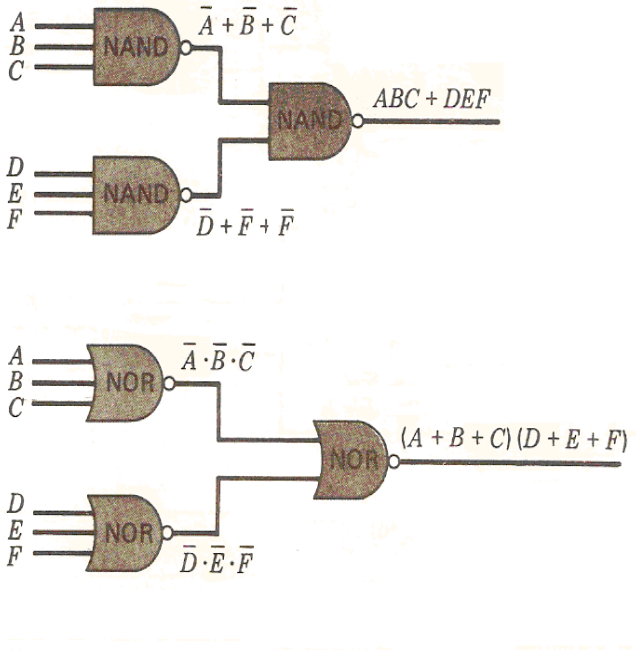


FIGURE 3.14

NAND and NOR

This function is shown in fig3.16(a) in both table-of-combinations and Karnaugh map form.

Figure 3.15.

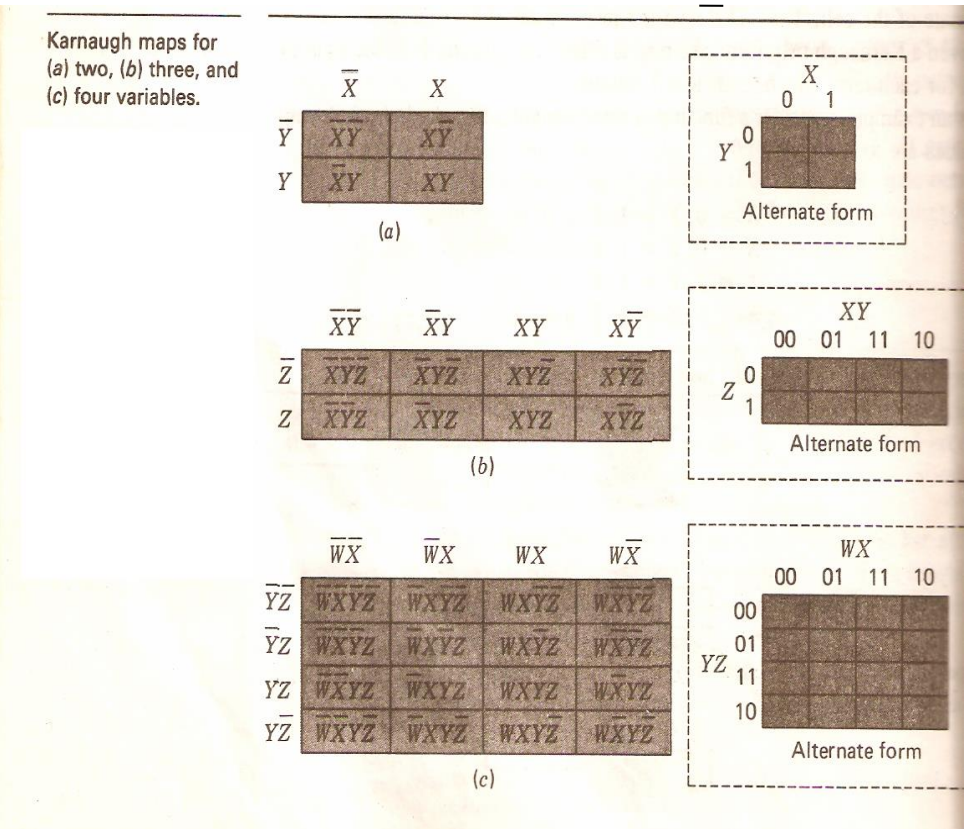


Figure 3.16. MAP METHOD FOR SIMPLIFYING EXPRESSIONS

X	Y	Z	FUNCTION VALUES
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a)

	$\bar{X}\bar{Y}$	$\bar{X}Y$	XY	$X\bar{Y}$
$\bar{Z}$	0	1	1	0
Z	0	1	1	0

2.17 SUBCUBES AND COVRING

A *subcube* is a set of exactly 2^m adjacent cells containing 1s. for $m = 0$ the subcube consists of a single cell. For $m = 1$ a subcube consists of two adjacent cells;for instance,the cells containing XYZ and XYZ form a subcube, as shown kin fig 3.17(a), as do XYZ and XYZ (since the map is rolled).

For *m* = 2 a subcube has four adjacent cells, and several such subcubes are shown in fig 3.17 (c). Notice that here we have omitted 0s for clarity and filled in only the 1s for the function. This policy will be continued.

Finally, subcubes containing eight cells (for m=3) are shown in figure3.17(d).

To demonstrate the use of map and subcubes in minimizing Boolean algebra expressions, we need to examine a rule of Boolean algebra:

$$AX + A\overline{X} = A$$

In the above equation, A can stand for more than one variable. For instance, let A = WY; then we have

$$(WY)X + (WY)\overline{X} = WY$$

Or let $A = W\overline{Y}\overline{Z}$; then we have

$$W\overline{Y}\overline{Z}\overline{X} + W\overline{Y}\overline{Z}X = W\overline{Y}\overline{Z}$$

The basic rule can be proved by factoring

$$AX + A\overline{X} = A(X + \overline{X})$$

THEN since $X + \overline{X} = 1$, we have

$$AX + A\overline{X} = A(X + \overline{X}) = A \cdot 1 = A$$

Each of the examples given can be checked similarly; for instance,

$$W\overline{Y}\overline{Z}\overline{X} + W\overline{Y}\overline{Z}X = W\overline{Y}\overline{Z}(\overline{X} + X) = W\overline{Y}\overline{Z} \cdot 1 = W\overline{Y}\overline{Z}$$

THIS rule can be extended. Consider

$$WX\overline{Y}\overline{Z} + WX\overline{Y}Z + WXY\overline{Z} + WXYZ$$

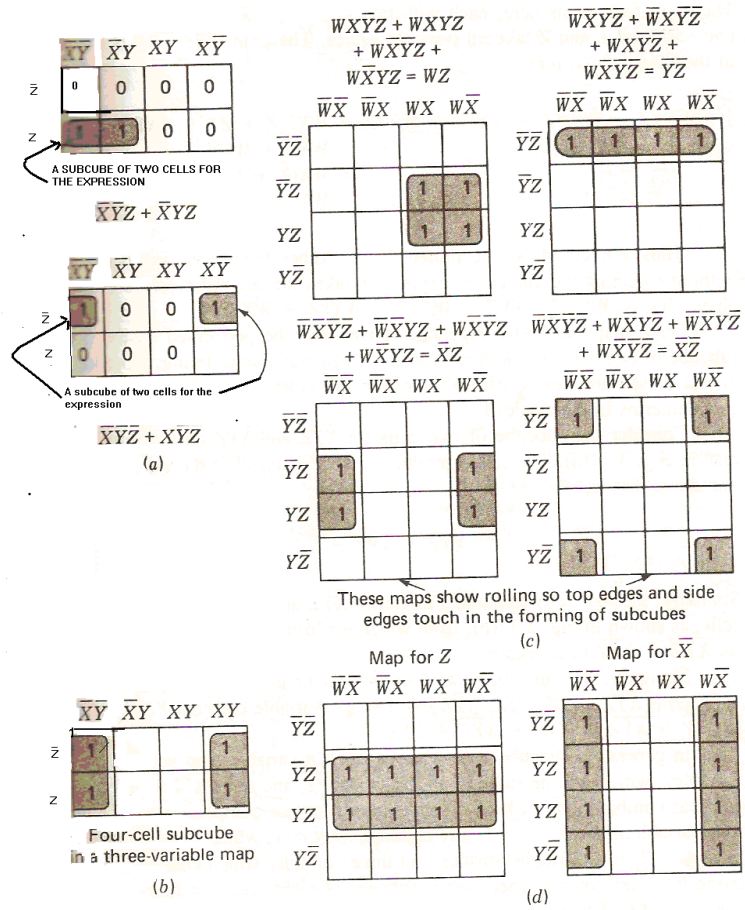


FIGURE 3.17

Subcubes with two, four, and eight cells. Blank cells are assumed to contain 0s.

There are four terms here, each with two variables WX constant while the other Two variables Y and Z take all possible values. The term WX is equal to the sum of the other terms, for

$$W X \overline{Y}\,\overline{Z} + W X \overline{Y} Z + W X Y \overline{Z} + W X Y Z = W X \overline{Y}\,(\overline{Z} + Z) + W X Y\,(Z + \overline{Z})$$

$$= W X \overline{Y} + W X Y$$

$$= W X (\overline{Y} + Y)$$

$$= W X$$

Thus WX could be substituted for the other four terms in an expression Without changing the values the expression takes for any input values to the Variables, that is, $WX = WX\overline{Y}\overline{Z} + WX\overline{Y}Z + WXY\overline{Z} + WXYZ$.

The set of minters in an expression does not necessarily form a single subscale, however, and there are two cases to be dealt with. Call a maximal *subcube* the largest subcube that can be found around a given minterm. Then the two cases are as follows:

1 All maximal subcubes are nonintersecting: that is no cell in a maximal subcube is a part of another maximal subcube. Several examples are shown in

Fig.3.18

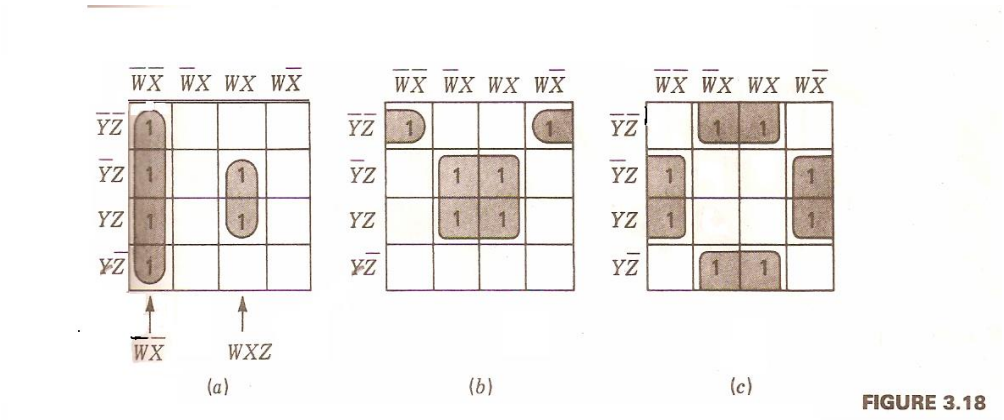


FIGURE 3.18

2.The maximal sub cubes intersect; that is , cells in one maximal subcube are also in other maximal subcubes. Figure 3.19 shows examples of this.

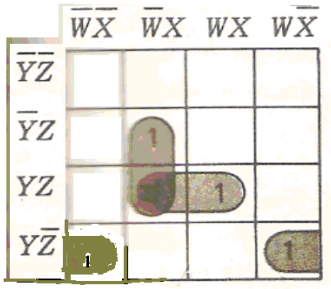
Case 1 is the more easily dealt with. In this case, the product terms corresponding to the maximal subcubes are selected, and the sum of these forms a minimal sum-of-products expression. (In switching theory, the product term corresponding to a maximal subcube is called a *prime implicant.)*

Figure 3.18(a) shows an example of this in four variables. There is a subcube of two cells containing WXYZ and WXYZ which can be covered by the product term WXZ. There is also a sub cube of four cells containing WXYZ, WXYZ, WXYZ, and WXYZ, which can be covered by WX. The minimal expression is, therefore, $WX + WXZ$.

Two other examples are shown in figure 3.18(b) and (c).In each case the subcubes do not intersect or share cells and so the product term(prime implicant)which corresponds to a given maximal subcube can be readily derived and the sum of these for a given map forms the minimal expression.

When the subcubes intersect, the situation can be more complicated. The first principle to note is this: Each cell containing a 1 (that is, each 1 cell) must be contained in some subcube which is selected.

Figure 3.19(a) shows a map with an intersecting pair of subcubes plus another subcube. The minimal expression is, in this case, formed by simply adding the three product terms associated with the three maximal subcubes. Notice that a

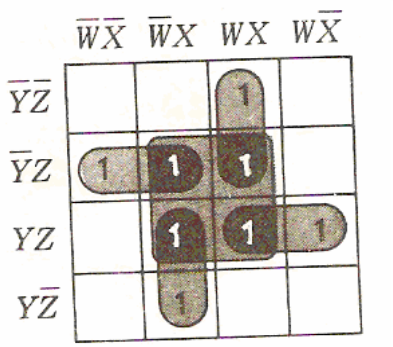


(a)

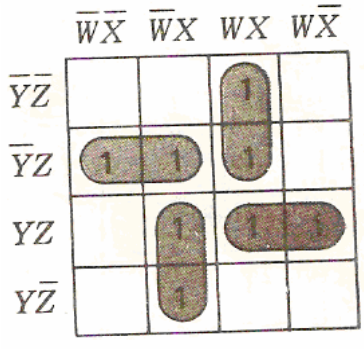
Figure 3.19

\

FIGURE 3.20



(a)



(b)

Single term, WXYZ, is shared between two subcubes and, because of this, is effectively in the minimal expression twice. This is permissible because of the idempotent rule of Boolean algebra, A + A = A, which states that repetition of terms does not change functional equivalence.

As long as the maximal subcubes can be readily found and there are no options in subcube selection, the minimization problem is straightforward. In some case the problem is more complicated. Figure 3.20 shows an expression with a subcube of four cells in the center of the map, which is maximal. The selection of this maximal subcube does not lead to a minimal expression, however, because the four cells with

1s around this subcube must be covered also .In each case these 1 cells can be found to have a single adjacent cell and so to be part of maximal subcubes consisting of 2 cells. In fig.3.20 (a), $W\overline{X}\overline{Y}\overline{Z}$ is in a cell adjacent to only WXYZ and so forms part of a 2 cell. Figure 3.20(b shows another way to form subcues for the map, and this leads to the minimal expression $WXY + \overline{W}YZ + W\overline{X}Y + WYZ$.

The finding of minimal expressions for such maps is not direct, but follow these rules:

1 Begin with cells that are adjacent to no other cell. The minters in these cells cannot be shortened and must be used as they are.

2 Find all cells that are adjacent to only one other cell. These form subcubes of two cells each.

3 Find those cells that lead to maximal subcubes of four cells. Then find Subcubes of eight cells, ect.

4 the minimal expression is formed from a collection of as few cubes as possible, each of which is as possible, that is, each of which is a maximal subcube.

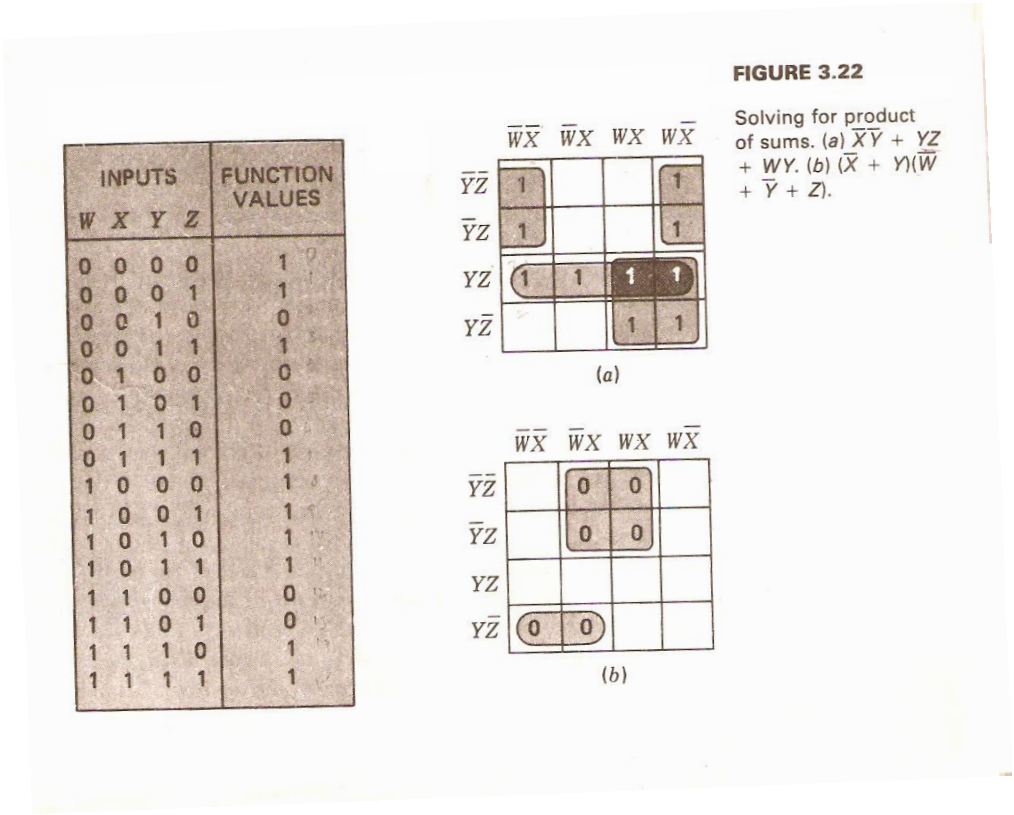
2.18 PRODUCT-OF-SUMS EXPRESSIONS-DON'T-CARES

The technique for product-of-sums expressions is almost identical with the design procedure using sum-of-products. The basic rule can be stated quite simple:

Solve for 0s, and then complement the resulting expression.

Let us take example.figure 3.22(a) shows a table of combinations and a karnaugh map for a four-variable problem. In fig. 3.22(a) the sum-of-products expression is derived and in minimal form is found to be $XY + \overline{Y}\overline{Z} + WY$.

In fig .3.22(b) the same problem is solved for the 0s,which gives $X\overline{Y}+W\overline{Y}Z$..Since we have solved for 0s,we have solved for the complement of the desired problem. If the output is called F, then we have solved for F .we then write $\overline{F} = XY + W\overline{Y}Z$.



INPUTS W	X	Y	Z	FUNCTION VALUES
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

FIGURE 3.22

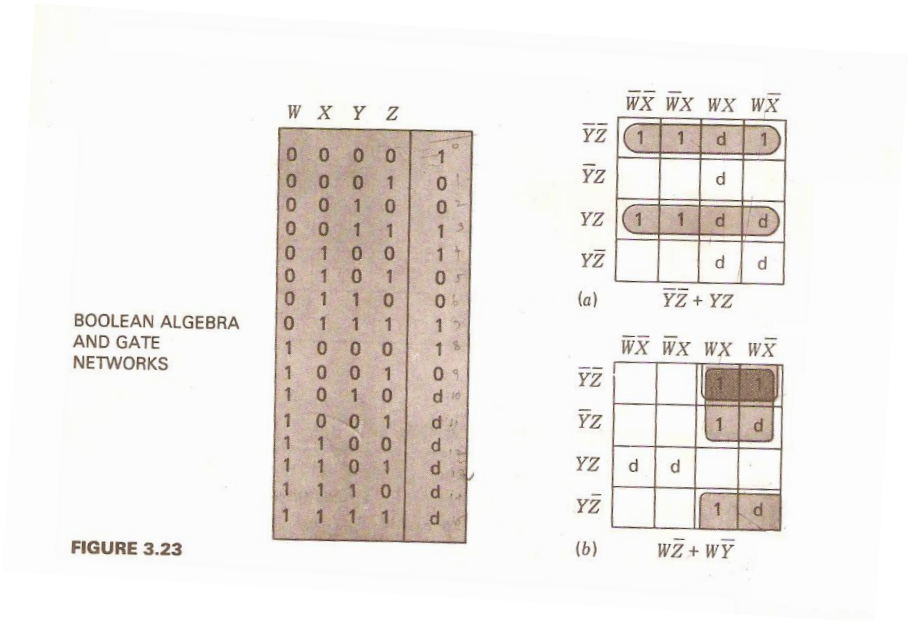
Solving for product of sums. (*a*) $\bar{X}\bar{Y} + YZ + WY$. (*b*) $(\bar{X} + Y)(\bar{W} + \bar{Y} + Z)$.

Now, what is wanted is F: so both sides of this expression are complemented,

And we have

$$F = (X + Y)(W + Y + Z)$$

This expression is in product-of-sums form and is somewhat simpler then the sum-of-products expression.



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

FIGURE 3.23

If sum-of-products and product-of-sums expression are equally easy to Implement, then a given problem must be solved in both forms and the simpler solution chosen. There is no way to determine which will be simpler other than by a complete working of the problem.

There is another frequently encountered situation in which certain outputs are not specified in a problem .such outputs are called don't -*care* outputs,for the designer does not care what the outputs are for these particular inputs.

Figure 3.23(a) shows such a problem with 6 of the possible 16 output values Listed as d's (don't cares). This is a part of a BCD translator, and so these particular six input combinations are never used.

Since don't care output values are of no importance, they may be filled in with 1s and 0s in any way that is advantageous .figure 3.23(a) shows a karnaugh map of the table of combinations in the figure, with d's in the appropriate places. In solving this table, a d may be used as either a 1 or a 0; so the d's are used to enlarge or complete a subcube whenever possible, but otherwise are ignored (that is, made 0*). The d's need not be covered by subcubes selected, but are used only to enlarge subcubes containing 1s, which must be covered.*

In fig.3.23(a) ,the vertical string of four d's in the WX column is of use twice, once in filling out, or completing, the row of 1s and once in competing the third row. These subcubes give the terms YZ and YZ; so the minimal sum-of-products expression is $Y\overline{Z} + \overline{Y}Z$. Notice that if all the d's were made os, the solution would require more terms.

Another problem is worked in fig.3.23 (b). for this problem the solution is W~~Z~~ +W~~Y~~.Notice that two of the d's are made 0s.In effect, the d's are chosen so that they lead to the best solution.

UNIT – II

Self Assessment Questions

Fill in the blank

1. The + and . operations are physically realized by two types of electronic circuits called _____ and _____.
2. The complementation operation is physically realized by a gate or circuit called an _____.
3. NAND gate is a combination of _____ and _____ gates.
4. The _____ on the Output of the NAND and NOR gates represents complementation.

True or False

1. In the derivation of Boolean expression when both X and Y are Os, the output Z is to be 1.
2. The symbol $\overline{\ }$ is not used to indicate complementation .
3. A subcube is a set of exactly 2^m adjacent cells containing 1^s.

Multiple Choice

1. Gating networks can now be made with many gates in a single contains by using the manufacturing technique.

   a). MSI b) LSI c) CSI d) SSI

2. 

   This Logical diagram is for the gate.

   a)AND b) OR c) NOT d) NOR

3. According to Boolean Algebra rules X + XZ = ?

   a) X b)Y c) XZ d) XY.

Unit Questions

1. Prepare a truth Table for the following Boolean expressions

   a) $A(B\overline{C} + \overline{B}C)$

   b) $ABC(A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C})\overline{\ }$

2. Simplify the following expressions

   a). $AB + A\overline{B} + A\overline{C} + A\overline{C}\overline{\ }$

   b). $XY(\overline{X}YZ\overline{\ } + XY\overline{Z}\overline{\ } + \overline{X}\overline{Y}\overline{Z})\overline{\ }$

3. Prove the two basic De Morgan theorems, using the proof by perfect induction.
4. Explain the AND and OR gates with neat diagram.
5. Explain Sum of Products and Products of Sums.
6. What is NAND gate.
7. Explain NOR gate with neat diagram.
8. Draw the Logical diagram for the following.

   a). $A\overline{BC}\,\overline{+} + AB$

   b). $(\overline{X} + \overline{Y})\,(X + \overline{Y\,}\,\overline{+} + Z)$

   c). (A+B+C) (D+E+F)

Self Assessment Answers

Fill in the blanks

1. OR, AND
2. Inverter
3. AND,NOR
4. Bubble

True / False

1. True
2. False
3. True

Multiple Choice

1. B
2. D
3. A
4.

UNIT-III

Anatomy of a Digital Computer

3.0 Functions and Components of a Computer

- 3.0.1 Central Processing Unit
  - 3.0.1.1 Control Unit
  - 3.0.1.2 Arithmetic - Logic Unit
- 3.0.2 Memory
  - 3.0.2.1 Registers
  - 3.0.2.2 Addresses
- 3.0.3 How the CPU and Memory Work

3.1 Memory Units

- 3.1.1 Introduction
- 3.1.2 RAM
- 3.1.3 ROM
- 3.1.4 PROM
- 3.1.5 EPROM
- 3.1.6 EEPROM
- 3.1.7 Flash Memory

3.2 Input Devices

- 3.2.1 Introduction
- 3.2.2 Keyboard
- 3.2.3 Mouse
  - 3.2.3.1 Types of Mice
  - 3.2.3.2 Connections
  - 3.2.3.3 Mouse Pad
- 3.2.4 Trackball
- 3.2.5 Joystick
- 3.2.6 Digitizing Tablet
- 3.2.7 Scanners
- 3.2.8 Digital Camera
- 3.2.9 Magnetic Ink Character Recognition
- 3.2.10 Optical Character Recognition
- 3.2.11 Optical Mark Recognition

UNIT-III

ANATOMY OF A DIGITAL COMPUTER

3.0 FUNCTIONS AND COMPONENTS OF A COMPUTER

To function properly, the computer needs both hardware and software. Hardware consists of the mechanical and electronic devices. The software consists of programs, the operating systems and the data that reside in the memory and storage devices. A computer does mainly the following four functions:

- Receive input – Accept information from outside through various input devices like the keyboard, mouse, etc.
- Process information – perform arithmetic or logical operations on the information.
- Produce output – Communicate information to the outside world through output devices like monitor, printer, etc.
- Store information – Store the information in storage devices like hard disk, floppy disks, etc.

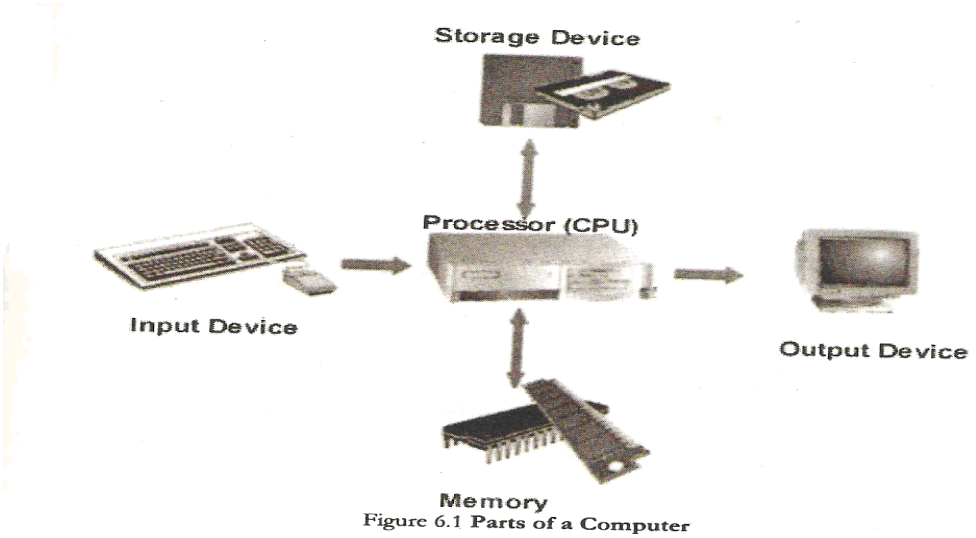


Figure 6.1 Parts of a Computer

Computer hardware falls into two categories: processing hardware, which consists of the central processing unit (CPU), and the peripheral devices.

3.0.1 CENTRAL PROCESSING UNIT (CPU)

The part of the computer that executes program instructions is known as the processor or central processing unit (CPU). The CPU has two parts – the control unit and the arithmetic – logic unit (ALU).

3.0.1.1 Control unit

The control unit tells the rest of the computer system how to carry out a program's instructions. It directs the movement of electronic signals between memory – which temporarily holds data, instructions and processed information – and the ALU. It also directs these control signals between the CPU and input/ output devices.

3.0.1.2 Arithmetic – Logic Unit (ALU)

Arithmetic Logic Unit, usually called the ALU, performs two types of operations – arithmetic and logical. Arithmetic operations are the fundamental mathematical operations consisting of addition, subtraction, multiplication and division. Logical operations consist of comparisons. That is, two pieces of data are compared to see whether one is equal to., less than, or greater than the other.

3.0.2 MEMORY

Memory – also known as the primary storage or main memory – is a part of the microcomputer that holds data for processing, instructions for processing the data (the program) and information. Part of the contents of the memory is held only temporarily, that is, it is stored only as long as the microcomputer is turned on. When you turn the machine off, the contents are lost. The capacity of the memory to hold data and program instructions varies in different computers hold approximately 6,40,000 characters of data or instructions only. But modern microcomputers can hold millions, even billions of characters in their memory.

3.0.2.1 Registers

Computers also have several additional storage locations called registers. These appear in the control unit and ALU and make processing more efficient. Registers areas hold data and instructions temporarily during processing. They are parts of the control unit and ALU rather than the memory.

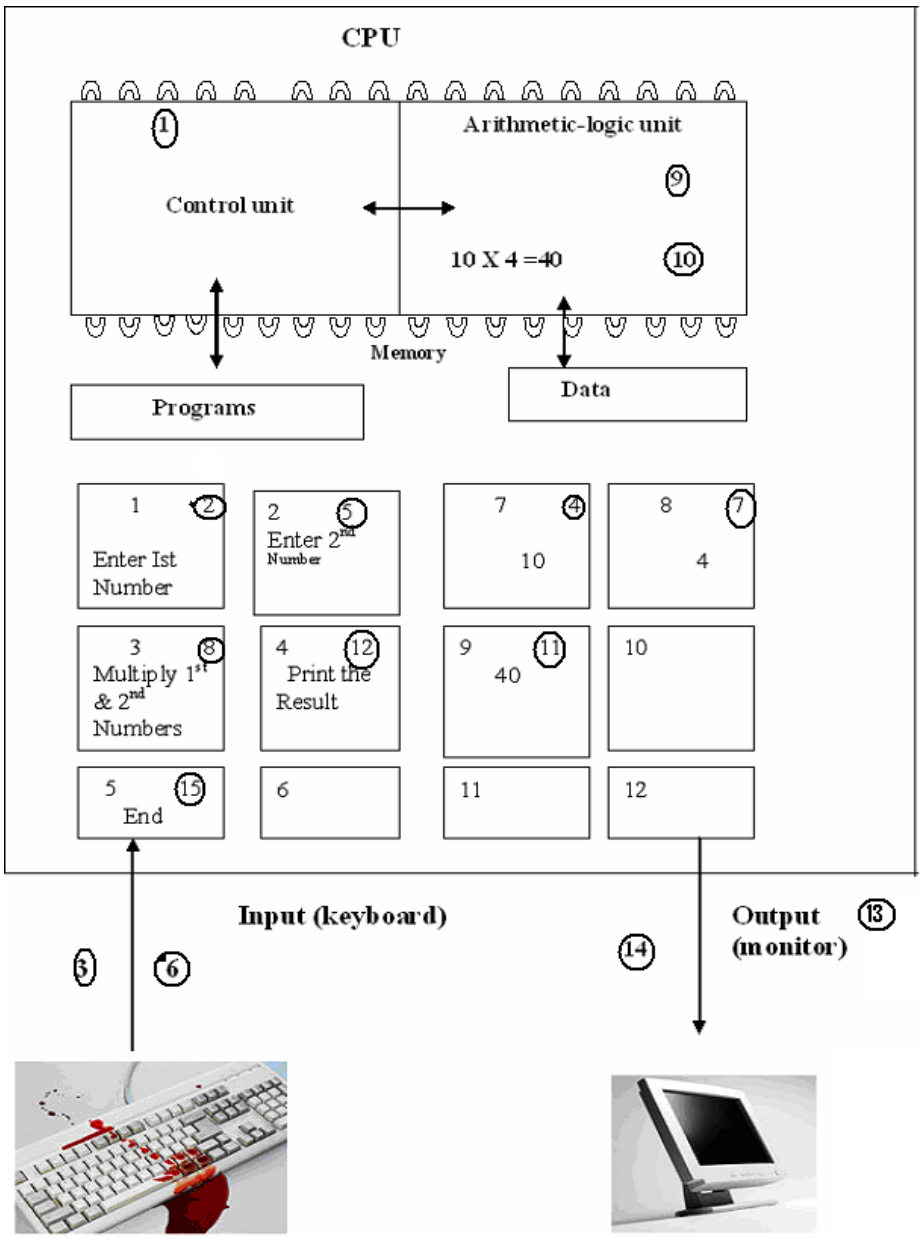
3.0.2.2 Addresses

To locate the characters of data or instructions in the main memory, the computer stores them in locations known as addresses. A unique number designates each address. Addresses can be compared to post office mailboxes. Their numbers stay the same, but contents continuously change.

3.0.3 HOW THE CPU AND MEMORY WORK

The various steps involved for multiplying two numbers is explained below:

1. The control unit recognizes that the programs has been loaded into memory. It begins to execute the first step in the program.
2. The program tells the user, "Enter 1st Number."
3. The user types the number 10 on the keyboard. An electronic signal is sent to the CPU.
4. The control unit recognizes this signal and routes the signal to an address in memory – address 7.
5. After completing the above instruction, the next instruction tells user, "Enter 2nd Number."
6. The user types the number 4 on the keyboard. An electronic signal is sent to the CPU.
7. The control unit recognizes this signal and routes it to memory address 8.
8. The next program instruction is executed – "Multiply 1st and 2nd Numbers."
9. To execute this instruction, the control unit informs the ALU that two numbers are coming and the ALU is to multiply them. The control unit next sends to the ALU a copy of the contents of address 7(10) and address 8 (4).
10. ALU performs the multiplication: 10 X 4 = 40.
11. The control unit sends a copy of the multiplied result (40) back to memory, to address 9.



Working of the CPU and Memory

12. The next program instruction is executed: "Print the Result.'
13. To execute this instruction, the control unit sends the contents of the address 9(40) to the monitor.
14. Monitor displays the value 40.
15. Final instruction is executed: "End." The program is complete.

3.1 MEMORY UNITS

3.1.1 INTRODUCTION

Memory units are the internal storage areas in a computer. The term "memory": identifies data storage that comes in the form of chips, and the word "storage" is used for memory that exists on tapes or disks. Every computer comes with a certain amount of physical memory, usually referred to as the main memory or the RAM. The main memory can hold a single byte of information in each location. A computer that has 1 megabyte of memory, therefore, can hold about 1 million bytes (or characters)of information.

There are several different types of memory.

3.1.2 RAM

Pronounced ramm, acronym for random access memory, a type of computer memory that can be accessed randomly; that is, any byte memory can be accessed without touching the preceding bytes. RAM is the most type of memory found in computers and other devices, such as printers. RAM refers to read and write memory.

Most Ram is volatile when the power is turned off, whatever data was in Ram is lost.

There are two types of RAM:

- Dynamic RAM->It needs to be refreshed thousands of times per second.
- Static RAM->It needs to be refreshed less often, which makes it faster; but it is also more expensive than dynamic RAM.

3.1.3 ROM

Pronounced ROM, acronym for read-only memory, a computer memory on which data has been prerecorded. Once data has been written onto a ROM chip, it cannot be removed and can only be read. Unlike the main memory (RAM), ROM retains its contents even when the computer is turned off. ROM is referred to as being nonvolatile, whereas RAM is volatile.

Most personal computers contain a small amount of ROM that stores critical programs such as the program that boots the computer. In addition, ROMs are used extensively in calculators and peripheral devices such as laser printers, whose fonts are often stored in ROMs. A variation of a ROM is the PROM (programmable read-only memory). PROMs are manufactured as blank chips on which data can be written.

3.1.4 PROM

Pronounced prom, an acronym for programmable read-only memory. A PROM is a memory chip on which data can be written only once. Once a program has been written onto a PROM, it remains there forever. Unlike the main memory, PROMs retain their contents when the computer is turned off.

The difference between a PROM and a ROM (read-only memory) is that a PROM is manufactured as blank memory, whereas a ROM is

programmed during the manufacturing process. To write data onto a PROM chip, you need a special device called a PROM programmer or a PROM burner. The process of programming a PROM is sometimes called burning the PROM.

3.1.5 EPROM

Acronym for Erasable Programmable Read –Only Memory, and pronounced ee-prom, EPROM is a special type of memory that retains its contents until it is exposed to ultraviolet light. The ultraviolet light clears its contents, making it possible to reprogram the memory.

An EPROM differs from a PROM in that a PROM can be written to only once and cannot be erased. EPROM are used widely in personal computers because they enable the manufacture to change the contents of the PROM before the computer is actually shipped. This means that bugs can be removed and new versions installed shortly before delivery.

3.1.6 EEPROM

Acronym for electrically erasable programmable read-only memory. Pronounced double-ee-prom, an EEPROM is a special type of PROM that can be erased by exposing it to an electrical charge. Like other types of PROM, EEPROM retains its contents even when the power is turned off. Also like all other types of ROM, EEPROM is not as fast as RAM.

EEPROM is similar to flash memory (sometimes called flash EEPROM). The principle difference is that EEPROM requires data to be written or erased one byte at a time whereas flash memory allows data to be written or erased in blocks. This makes flash memory faster.

3.1.7 FLASH MEMORY

Flash memory is a special type of EEPROM that can be erased and reprogrammed in blocks instead of one byte at a time. Many modern PCs have their BIOS (Basic Input Output System) stored on a flash memory ship so that it can easily be updated if necessary. Such a BIOS is sometimes called a flash BIOS. Flash memory is also popular in moderns because it enables the modern manufacturer to support new protocols as they become standardized.

3.2 INPUT DEVICES

3.2.1 INTRODUCTION

An input device is any machine that feeds data into a computer. For example, a keyboard is an input device, whereas a display monitor is an output device. Input devices other than the keyboard are sometimes called alternate input devices. Mice, trackballs, and light pens are all alternate input devices.

3.2.2 KEYBOARD

Keyboard is an input device consisting of a set typewriter-like keys that enable you to enter data into a computer. Computer keyboards are similar to electric-typewriter keyboards but contain additional keys. The keys on computer keyboards are often classified as follows:

- Alphanumeric keys – letters and numbers
- Punctuation keys – comma, period, semicolon, and so on.
- Special keys – function keys, control keys, arrow keys, caps Lock key, and so on.

There are actually three different PC keyboards: the **original PC keyboard**, with 84 keys; the **AT keyboard**, also with 84 keys; and the **enhanced keyboard**, with 101 keys.

In addition to these keys, IBM keyboards contain the following keys: Page UP, Page Down, Home, End, Insert, Pause, Num Lock, Scroll Lock, Break, Caps Lock, Print Screen.

3.2.3 MOUSE

Mouse is a device that controls the movement of the cursor or pointer on a display screen. A mouse is a small object you can roll along a hard, flat surface. As you move the mouse, the pointer on the display screen moves in the same direction. Mice contain at least one button and sometimes as many as three, which have different functions depending on what program is running.

In particular, the mouse is important for graphical user interfaces because you can simply point to options and objects and click a mouse button. Such applications are often called point- and – click programs. The mouse is also useful for graphics programs that allow you to draw pictures by using the mouse like a pen, pencil, or paintbrush.

3.2.3.1 Types of Mice

There are three basic of Mice.

Mechanical Has a rubber or metal ball on its underside that can roll in all directions. Mechanical sensors within the mouse detect the direction the ball is rolling and move the screen pointer accordingly.

Optomechanical Same as a mechanical mouse, but uses optical sensors to detect motion of the ball.

Optical Uses a laser to detect the mouse's movement. You must move the mouse along a special mat with a grid so that the optical mechanism has a frame of reference. Optical mice have no mechanical moving parts. They respond more quickly and precisely than mechanical and optomechanical mice, but they are also more expensive.

3.2.3.2 Connections

Mice connect to PCs in one of three ways:

- Serial mice connect directly to an RS-232C serial port or a PS/2 port. This is the simplest type of connection.
- Bus mice connect to the bus through an interface card. This is somewhat more complicated because you need to configure and install an expansion board.

- Cordless mice aren't physically connected at all. Instead they rely on infrared or radio waves to communicate with the computer.

3.2.3.3 Mouse pad

Mouse pad is a pad over which you can move a mouse.

3.2.4 TRACKBALL

Trackball is another pointing device. Essentially, a trackball is a mouse lying on its back. To move the pointer, you rotate the ball with your thumb, your fingers, or the palm of your hand. There are usually one to three buttons next to the ball, which you use just like mouse buttons.

The advantage of trackballs over mice is that the trackball is stationary so it does not require much space to use it. In addition, you can place a trackball on any type of surface, including your lap. For both these reasons, trackballs are popular pointing devices for portable computers.

3.2.5 JOYSTICK

A lever that moves in all directions and controls the movement of a pointer or some other display symbols. A joystick is similar to a mouse, except that with a mouse cursor stops moving as soon as you stop moving the mouse.

With a joystick, the pointer continues moving in the direction the joystick is pointing. To stop the pointer, you must return the joystick to its upright position. Joystick are used mostly for computer games, but they are also used occasionally for CAD/CAM systems and other applications.

3.2.6 DIGITIZING TABLET

This is an input device that enables you to enter drawings and sketches into a computer. A digitizing tablet consists of an electronic tablet and a cursor or pen.

A cursor (also called a puck) is similar to a mouse, except that it has window with crosshairs for pinpoint placement and it can have as any as 16 buttons

A pen (also called a stylus), which looks like a simple ballpoint pen but uses an electronic head instead of ink. The tablet contains electronic that enable it to detect movement of the cursor or pen and translate the movements into digital signals that it sends to the computer.

Digitizing tablet are also called digitizers, graphics tablets, touch tablets, or simply tablets.

3.2.7 SCANNERS

Scanner is an input device that can read text or illustrations printed on paper and translate the information into a form that the computer can use. A scanner works by digitizing an image – dividing it into a grid of boxes and representing each box with either a zero or a one, depending on whether the box is filled in. The resulting matrix of bits, called a bit map, can then be stored in a file, displayed on a screen, and manipulated by programs.

Optical scanners do not distinguish text from illustrations; they represent all images as bit maps. Therefore, you cannot directly edit text that has been scanned. To edit text read by an optical scanner, you need an optical character recognition (OCR) system to translate the image into ASCII characters. Most optical scanners sold today come with OCR packages.

Scanners differ from one another in the following respects:

Scanning technology Most scanners use charge – coupled device (CCD) arrays, which consists of tightly packed rows of light receptors that can detect variations in light intensity and frequency. The quality of the CCD array is probably the single most important factor affecting the quality of the scanner. Industry – strength drum scanners use a different technology that relies on a photomultiplier tube (PMT) but this type of scanner is much more expensive than the more common CCD – based scanners.

Resolution The denser the bit map, the higher the resolution. Typically, scanners support resolutions from 72 to 600 dots per inch (dpi).

Bit depth The number of bits used to represent each pixel. The greater the bit depth, the more colors or grayscales can be represented. For example, a 24-bit color scanner can represent 2 to the 24^{th} power (16.7 million) colors. Note, however, that a large color range is useless if the CCD arrays are capable of detecting only a small number of distinct colors.

Size and shape Some scanners are small hand-held devices that you move across the paper. This hand- held scanners are often called half-page scanners, which are adequate for small pictures and photos, but they are difficult to use if you need to scan an entire page of text or graphics. Larger scanners include machines into which you can feed sheets of paper. These are called sheet-fed scanners. Sheet-fed scanners are excellent for loose sheets of paper, but they are unable to handle bound documents. A second type of large scanner, called a flatbed scanner, is like a photocopy machine. It consist of a board on which you lay books, magazines, and other documents that you want to scan.

3.2.8 DIGITAL CAMERA

Images can be input into a computer using a digital camera. These images can then be manipulated in many ways using the various imaging tools available. The digital camera takes a still photograph, stores it, and then sends it as digital input into the computer. The images are then stored as digital files.

3.2.9 MAGNETIC INK CHARATER RECOGNITION (MICR)

Magnetic Ink Character Recognition (MICR) allows the computer to recognize characters printed using magnetic ink. MICR is a direct-entry method used in banks. This technology is used to automatically read those frustrating – looking numbers on the bottom of the cheque. A special –purpose machine known as a reader/sorter reads characters made of ink containing magnetized particles. A related technology is the magnetic strip, used on the back of credit cards and bank debit cards, that allows readers such as Automated Teller Machines (ATMs) to read account information and facilitate

monetary transactions. Another example of magnetic strip technology is in ID cards, which can be used for a variety of functions from attendance monitoring to restricting access to specific locations.

3.2.10 OPTICAL CHARACTER RECOGNITION (OCR)

Often abbreviated OCR, optical character recognition refers to the branch of computer science that involves reading text from paper and translating the images into a form that the computer can manipulate (for example, into ASCII codes). An OCR system enables you to take a book or a magazine article and feed it directly into an electronic computer file.

All OCR systems include an optical scanner for reading text, and sophisticated software for analyzing images. Most OCR systems use a combination of hardware (specialized circuit boards) and software to recognize characters, although some inexpensive systems do it entirely through software. Advanced OCR systems can read text in a large variety of fonts, but they still have difficulty with handwritten text.

3.2.11 OPTICAL MARK RECOGNITIO0N (OMR)

Optical Mark Recognition (OMR) also called mark sensing is a technology where an OMR device senses the presence or absence of a mark, such as a pencil mark. OMR is used in tests such as aptitude tests.

3.2.12 BAR CODE READER

You are probably familiar with the bar code readers in supermarkets, bookshops, etc. Bar code readers are photoelectric scanners that read the bar codes, or vertical zebra striped marks, printed on product containers. Supermarkets use a bar code system called the Universal Product Code (UPC).

The bar code identifies the product to the supermarket's computer, which has a description and the latest price of the product. The computer automatically tells the POS (Point Of Sales) terminal what the price is.

3.2.13 SPEECH INPUT DEVICES

Speech or voice input devices convert a person's speech into digital form. These input devices, when combined with appropriate software, form voice recognition systems. These systems enable users to operate microcomputers using voice commands.

Some of these systems must be 'trained ' to the particular user's voice. This is done by his/her spoken words to patterns previously stored in the computer. More advanced systems that can recognize the same word spoken by many different people have been developed. However, until recently the list of words has been limited. A newly developed voice recognition system like IBM Voice Type identifies more than 30,000 words and adapts to individual voices. There are even systems that will translate from one language to another, such as from English Japanese.

There are two types of voice recognition systems:

- Continuous Speech
- Discrete – Word

Continuous Speech

Continuous speech recognition systems are used to control a microcomputer's operations and to issue commands to special application programs. For example, rather than using the keyword to save a spreadsheet file, the user could simply say "Save the file". Two popular systems are Apple Computer's Plain Talk and IBM's continuous speech series.

Discrete – word

A common activity in business is preparing memos and other written documents. Discrete – word recognition systems allow users to dictate directly into a microcomputer using a microphone. The microcomputer stores the memo in a word processing file where it can be revised later or directly printed out. IBM Voice Type Simply Speaking is an example.

3.2.14 TOUCH SCREEN

Touch screen is a type of display screen that has a touch-sensitive transparent panel covering the screen. Instead of using a pointing device such as a mouse or light pen, you can use your finger to point directly to objects on the screen.

Although touch screens provide a natural interface for computer novices, they are unsatisfactory for most applications because the finger is such a relatively large object. It is impossible to point accurately to small areas of the screen. In addition, most users find touch-screens tiring to the arms after long use.

3.2.15 TOUCH PAD

A small, touch – sensitive pad used as a pointing device on some portable computers.

By moving a finger or other object along the pad, you can move the pointer on the display screen.

3.2.16 LIGHT PEN

Light pen is an device that utilizes a light – sensitive detector select objects on a display screen.

A light pen is similar to a mouse, expect that with a light pen you can move the pointer and select objects on the display screen by directly pointing to the objects with the pen.

3.3 OUTPUT DEVICES

3.3.1 INTRODUCTION

Output is anything that comes out of a computer. An output device is any machine capable of representing information from a computer. Output devices include display screens, loudspeakers, printers, plotters, etc.

3.3.2 MONITOR

Monitor is another term for the display screen. The term monitor, however, usually refers to the entire box, whereas display screen can mean just the screen.

3.3.2.1 CLASSIFICATION OF MONITORS – BASED ON COLOUR

There are many ways to classify monitors. The most basic is in terms of colour capabilities, which separates monitors into three classes.

Monochrome

Monochrome monitors actually display two colours, one for the background and one for the foreground. The colours can be black and white, green and black, or amber and black. and white, green and black, or amber and black.

Gray – scale

A gray – scale monitor is a special type of monochrome monitor capable of displaying different shades of gray.

Colour

Colour monitors can display anywhere from 256 to over 1 million different colours. Colour monitors are sometimes called RGB monitors because they accept three separate signals – red, green, and blue. All colour computer monitors are RGB monitors. An RGB monitor consists of a vacuum tube with three electron guns – one each for red , green, and blue at one end and the screen at the other end. The three electron guns fire electrons at the screen, which contains a phosphorous coating. When the electron beams excite the phosphors, they glow. Depending on which beam excites them, they glow red, green, or blue. Ideally, the three beams should converge for each point on the screen so that each pixel is a combination of the three colours.

Colour and gray – scaling monitors are often classified by the number of bits they use to represent each pixel. For example, an 8- bit monitor each pixel with 8 bits. The more bits per pixel, the more colours and shades of gray the monitor can display.

3.3.2.2 CLASSIFICATION MONITORS – BASED ON SIGNALS

Another common way of classifying is in terms of the type of signal they accept; analog or digital.

Digital Monitor

A digital monitor accepts digital signals rather than analog signals. All monitors (except flat- panel displays) use CRT technology, which is essentially analog. The term digital, therefore, refers only to the type of input received from the video adapter. A digital monitor then translates the digital signals into analog signals that control the actual display.

Although digital monitors are fast and produce clear images, they cannot display variable colours continuously. Consequently, only low-quality

video standards such as MDA (Monochrome Display Adepter), CGA (Colour/Graphics Adapter), and EGA (Enhanced Graphics Adapter), specify digital signals. VGA (Video Graphics Array) and SVGA (Super VGA), on the other hand, require an analog monitor. Some monitors are capable of accepting either analog or digital signals.

Analog Monitor

This is the traditional type of colour display screen that has been used for years in televisions. In reality, all monitors based on CRT technology (that is, all monitors except flat- panel displays) are analog. Some monitors, however, are called digital monitors because they accept digital signals from the video adapter. EGA monitors, for example, must be digital because the EGA standard specifies digital signals. Digital monitors must nevertheless translate the signals into an analog form before displaying images. Some monitors can accept both digital and analog signals.

Some monitors have fixed frequency, which means that they accept input at only one frequency. Another type of monitor, called a multiscanning monitor, automatically adjusts to the frequency of the signals being sent to it. This means that it can accept input from different types of video adapters. Like fixed – frequency monitors, multiscanning monitors accept TTL, analog, or both types of input.

3.3.2.3 CHARACTERISTICS OF A MONITOR

Size

The most important aspect of a monitor is its screen size. Like televisions, screen sizes are measured in diagonal inches, the distance from one corner to the opposite corner diagonally. A typical size for small VGA monitors is 14 inches. Monitors that are 16 or more inches diagonally are often called full- page monitors. In addition to their size, monitors can be either portrait (height greater than width) or landscape (width greater than height). Larger landscape monitors can display two full pages, side by side.

Resolution

The resolution of a monitor indicates how densely the pixels are packed. Pixel is short for *Picture Element.* A pixel is a single point in a graphic image. Graphics monitors display pictures by dividing the display screen into thousands (or millions) of pixels, arranged in rows and columns. The number of bits used to represent each pixel determines how many colours or shades of gray can be displayed. For example an 8- bit colour monitor uses 8 bits for each pixel, making it possible to display 2 to the 8^{th} power (256) different colours or shades of gray.

On colour monitors, each pixel is actually composed of three dots – a red, a blue, and a green one. The quality of a display monitor largely depends on its resolution, how many pixels it can display, and how many bits are used to represent each pixel. VGA monitors display 640 by 480, or about 300,000 pixels. In contrast, SVGA monitors display 1,024 by 768, or nearly 800,000

pixels. Most modern monitors can display 1024 by 768 pixels, the SVGA standard. Some high – end models can display 1280 by 1024, or even 1600 by 1200.

Bandwidth

The amount of data that can be transmitted in a fixed amount of time. For digital devices, the bandwidth is usually expressed in bits or bytes per second (bps). For analog devices, the bandwidth is expressed in cycles per second, or Hertz (Hz).

Refresh Rate

Display monitors must be refreshed many times per second. The refresh rate determines hoe many times per second the screen is to be refreshed (redrawn). The refresh rate for a monitor is measured in hertz (Hz) and is also called the vertical frequency or vertical refresh rate. The old standard for monitor refresh rates was 60 Hz, but a new standard developed by VESA sets the refresh rate at 75Hz for VGA and SVGA monitors. The faster the refresh rate, the less the monitor flickers.

Interlaced or Non – interlaced

Interlacing is a display technique that enables a monitor to provide more resolution in expensively. With interlacing monitors, the electron guns draw only half the horizontal lines with each pass (for example, all odd lines on one pass and all even lines on the next pass). Because an interlacing monitor refreshes only half the lines at one time, it can display twice as many lines per refresh cycle, giving it greater resolution. Another way of looking at it is that interlacing provides the same resolution as non-interlacing, but less expensively. A shortcoming of interlacing is that the reaction time is slower, so programs that depend on quick refresh rates (animation and video, for example), may experience flickering or streaking. Given two monitors that offer the same resolution, the non – interlacing one will generally be better.

Dot – pitch

A measurement that indicates the vertical distance between each pixel on a display screen. Measured in millimeters, the dot pitch is one of the principal characteristics that determine the quality of display monitors. The dot pitch of colour monitors for personal computers ranges from about 0.15 mm 0.30mm. Another term for dot pitch is phosphor pitch.

Convergence

Convergence refers to how sharply an individual colour pixel on a monitor appears. Each pixel is composed of three dots – a red, blue, and green one. If the dots are badly misconverged, the pixel will appear blurry. All monitors have some convergence errors, but they differ in degree.

3.3.3 VIDEO STANDARDS

There are a variety of video standards that define resolution and colours for displays. Support for a graphics standard is determined by both the monitor

and the video adapter. The monitor must be able to show the resolution and colours defined by the standard, and the video adapter must be capable of transmitting the appropriate signals to the monitor.

Listed here, in approximate order of increasing power and sophistication, are the more popular video standards for PCs. Note that many of these numbers represent only the minimums specified in the standards. Many suppliers of video adapters provide greater resolution and more colours. For more information, refer to the entries for the specific graphics systems given in table 9.1.

Table 9.1 Popular Video Standards for PCs

Standard	Resolution	Simultaneous Colours
VGA	640 x 480	16
	320 x 200	256
SVGA	800 x600	16
	1024 x 768	256
	1280 x 1024	256
	1600 x 1200	256
8514/A	1024 x 768	256
XGA	640 x 480	65536
	1024 x 768	256
TI 34010	1024 x 768	256

VGA

Abbreviation of video graphics array, a graphics display system for PCs developed by IBM. VGA has become one of the de facto standards for PCs. In text mode, VGA systems provide a resolution of 720 by 400 pixels. In graphics mode, the resolution is either 640 by 480 (with 16 colours) or 320 by 200 (with 256 colours). The total palette of colours is 262,144.

SVGA

Short for Super VGA, a set of graphics standards designed to offered greater resolution than VGA. There are several varieties of SVGA, each providing a different resolution:

- 800 by 600 pixels
- 1024 by 768 pixels
- 1280 by 1024 pixels
- 1600 by 1200 pixels

All SVGA standards support a palette of 16 million colours, but the number of colours that can be displayed simultaneously is limited by the amount of video memory installed in a system.

8514/A

A high – resolution video standard for PCs developed by IBM in 1987. It was designed to extend the capabilities of VGA. The 8514/A standard provides a resolution of 1,024 by 768 pixels, which gives it about 2.5 times the pixels of VGA (640 by 480). Like VGA, 8514/A provides a palette of 262,000 colours, of which 256 can be displayed at one time. On monochrome displays, 8514/A provides 64 shades of gray.

XGA

Short for extended graphics array, a high – resolution graphics standard introduced by IBM in 1990. XGA was designed to replace the order 8514/A video standard. It provides the same resolutions (640 by 480 or 1024 by 768 pixels), but supports more simultaneous colours (65 thousand compared to 8514/A's 256 colours). In addition, XGA allows monitors to be non – interlaced.

TI 34010

TI 34010 is a video standard from Texas Instruments that supports a resolution of 1,024 by 768. TI 34010 conforms to TI's Graphics Architecture (TIGA). Unlike IBM's 8514/A, which supports the same resolution. TI 34010 is non–interlaced.

3.3.4 PRINTER

Printer is a device that prints text or illustrations on paper and in many cases on transparencies and other media. There are many different types of printers. In terms of the technology utilized, printers fall into the following categories.

Daisy – wheel Printer

Daisy – wheel printers are a type of printer that produces letter- quality type. A daisy – wheel printer works on the same principles as a ball – head typewriter. The daisy wheel is a disk made of plastic or metal on which characters stand out in relief along the outer edge. To print a character, the printer rotates the disk until the desired letter is facing the paper. Then a hammer strikes the disk, forcing the character to hit an ink ribbon, leaving an impression of the character on the paper. You can change the daisy wheel to print different fonts.

Daisy – wheel printers cannot print graphics, and in general they are noisy and slow, printing from 10 t about 75 characters per second.

Dot – matrix Printer

Dot – matrix printers create characters by striking pins against an ink ribbon. Each pin makes a dot, and combinations of dots form characters and

illustrations. Dot- matrix printers are inexpensive and relatively fast, but they do not produce high – quality output.

Dot – matrix printers vary in two important characteristics:

- **Speed** – Given in characters per second (cps), the speed can vary from about 50 to over 500 cps.
- **Print quality** – Determined by the number of pins (the mechanisms that, print the dots), it can vary from 9 to 24. The best dot– matrix printers (24 pins) can produce near letter- quality type, although you can still see a difference if you look closely.

Ink – jet Printer

Ink – jet printers' work by sparing ionized ink at a sheet of paper. Magnetized plates in the ink's path direct the ink onto the paper in the paper in the desired shapes. Ink – jet printers are capable of producing high quality print approaching that produced by laser printers. A typical ink- jet printer provides a resolution of 300 dots per inch. The price of ink – jet printers is lower than that of laser printers. However, they are also considerably slower. Another drawback of ink – jet printers is that they require a special type of ink that is apt to smudge on inexpensive copier paper.

Laser Printer

Laser printer utilizes a laser b4eam to produce an image on a drum. The light of the laser alters the electrical charge on the drum wherever it hits. The drum is then rolled through a reservoir of toner, which is picked up by the charged portions of the drum. Finally, the toner is transferred to the paper through a combination of heat and pressure.

Because an entire page is transmitted to a drum before the toner is applied. Laser printers are sometimes called page printers. There are two other types of page printers that fall under the category of laser printers even though they do not use lasers at all. One uses an array of LEDs to expose the drum, and the other uses LCDs. Once the drum is charged, however, they both operate like a real laser printer.

One of the chief characteristics of laser printers is their resolution – how many dots per inch (dpi) they lay down. The available resolutions range from 300 dpi at the low end to 1,200 dpi at the high end. By comparison, offset printing usually prints at 1,200 or 2,400 dpi. Some laser printers achieve higher resolutions with special techniques known generally as resolution enhancement.

In addition to the standard monochrome laser printer, which uses a single toner, there also exist colour laser printers that use four toners to print in full colour. Colour laser printers tend to be about five to ten times as expensive as their monochrome siblings.

Laser printers produce very high – quality print and are capable of printing an almost unlimited variety of fonts. Most laser printers come with a

basic set of fonts, called internal or resident fonts, but you can add additional fonts in one of two ways:

- **Font cartridges** – Laser printers have slots in which you can insert font cartridges, ROM boards on which fonts have been recorded. The advantage of font cartridges is that they use none of the printer's memory.
- **Soft fonts** – All laser printers come with a certain amount of RAM memory, and you can usually increase the amount of memory by adding memory boards in the printer's expansion slots. You can then copy fonts from a disk to the printer's RAM. This is called downloading fonts. A font that has been downloaded is often referred to as a soft font, to distinguish it from the hard fonts available on font cartridges. The more RAM a printer has, more fonts that can be downloaded at one time.

LCD & LED Printers

Similar to a laser printer but uses liquid crystals or light – emitting diodes rather than a laser to produce an image on the drum.

Line Printer

Line printers are high speed printers capable of printing an entire line at one time. A fast line printer can print as many as 3,000 lines per minute. The disadvantages of line printers are that they can print only one font, they cannot print graphics, the print quality is low, and they are very noisy.

Thermal printer

Thermal printers are printers that produce images by pushing electrically heated pins against special heat- sensitive paper. Thermal printers are inexpensive and are used in most calculators and many fax machines. They produce low- quality print, and the paper tends to curl and fades after a few weeks or months.

Printers are also classified according to the following characteristics:

- **Quality of type** – Type output produced by printers is said to be either letter quality (as good as a typewriter), near letter quality, or draft quality. Only daisy - wheel, ink – jet, and laser printers produce letter-quality type. Some dot –matrix printers claim letter – quality print, but if you look closely, you can see the difference.
- **Speed** – Measured in characters per second (cps) or pages per minute (ppm), the speed of printers varies widely. Daisy – wheel printers tend to be the slowest, printing about 30 cps. Line printers are fastest (up to 3,000 lines per minute). Dot – matrix printers can print up to 500 cps, and laser printers range from about 4 to 20 text pages per minute.
- **Impact or non – impact** – Impact printers include all printers that work by striking an ink ribbon. Daisy – wheel, dot – matrix, and line printers

are impact printers. Non – impact printers include laser printers and ink – jet printers. The important difference between impact and non – impact printers is that impact printers are much noisier but are useful for making multiple copies like carbon copies.

- **Graphics** – Some printers (daisy – wheel and line printers) can print only text. Other printers can print both text and graphics.
- **Fonts** – Some printers, notably dot –matrix printers, are limited to one or a few fonts. In contrast, laser and ink – jet printers are capable of printing an almost unlimited variety of fonts. Daisy – wheel printers can also print different fonts, but you need to change the daisy wheel, making it difficult to mix fonts in the same document.

3.3.5 PLOTTER

Plotter is a device that draws pictures on paper based on commands from a computer. Plotters differ from printers in that they draw lines using a pen. As a result, they can produce continuous lines, whereas printers can only simulate lines by printing a closely spaced series of dots. Multicolour plotters use different – coloured pens to draw different colours.

In general, poltters are considerably more expensive than printers. They are used in engineering applications where precision is mandatory.

3.3.6 SOUND CARDS & SPEAKERS

An expansion board that enables a computer to manipulate an output sounds. Sound cards are necessary for nearly all CD-ROMs and have become commonplace on modern personal computers. Sound cards enable the computer to output sound through speakers connected to the board, to record sound input from a microphone connected to the computer, and, manipulate sound stored on a disk.

Nearly all-sound cards support MIDI, a standard for representing music electronically. In addition, most sound cards are Sound Blaster – compatible, which means that they can process commands written for a Sound Blaster care, the *de facto* standard for PC sound.

Sound cards use two basic methods to translate digital data into analog sounds:

- FM (Frequency Modulation) Synthesis mimics different musical instruments according to built – in formulas.
- Wavetable Synthesis relies on recordings of actual instruments to produce sound. Wavetable synthesis produces more accurate sound, but is also more expensive.

3D Audio

3D audio is a technique for giving more depth to traditional stereo sound. Typically, 3D sound, or 3D audio, is produced by placing a device in a room with stereo speakers. The device dynamically analyzes the sound coming

from the speakers and sends feedback to the sound system so that it can readjust the sound to give the impression that the speakers are further apart.

3D audio devices are particularly popular for improving computer audio where speakers tend to be small and close together. There are a number of 3D audio devices that attach to a computer's sound card.

3.4 AUXILIARY STORAGE DEVICES

3.4.1 INTRODUCTION

Auxiliary storage also known as auxiliary memory or secondary storage, is the memory that supplements the main storage. This is a long – term, non – volatile memory. The term non –volatile means it stores and retains the programs and data even after the computer is switched off. Unlike RAM which looses the contents when the computer is turned off, and ROM, to which it is not possible to add anything new, auxiliary storage devices allows the computer to record information semi – permanently, so it can be read later by the same computer or by another computer. Auxiliary storage devices are also useful in transferring data or programs from one computer to another. They also function as back – up devices which allows to back – up the valuable information that you are working on. So even if by some accident your computer crashes and the data in it is unrecoverable, you can restore it from your back – ups. The most common types of auxiliary storage devices are magnetic disks, floppy disks, hard disks, etc.

There are two types of auxiliary storage devices. This classification is based on the type of data access: sequential and random. Based on the type of access they are called sequential – access media and random – media. In the case of sequential – access media, the data stored in the media can only be read in sequence and to get to a particular point on the media you have to go through all the preceding points.

Magnetic tapes are examples of sequential access media. In contrast, disks are random access also called direct access media because a disk drive can access any point at random without passing through intervening points. Other examples of direct access media are magnetic disks, optical disks, zip disks etc.

3.4.2 MAGNETIC TAPE

Magnetic tape is a magnetically coated strip of plastic on which data can be encoded. Tapes for computers are similar to the tapes used to store music. Some personal computers, in fact, enable you to use normal cassette tapes.

Storing data on tapes is considerably cheaper than storing data on disks. Tapes also have large storage capacities, ranging from a few hundred kilobytes to several gigabytes.

Accessing data on tapes, however, is much slower than accessing data on disks. Tapes are sequential access media, which means that to get to a

particular point on the tape, the tape must go through all the preceding points. In contrast, disks are random access media because a disk drive can access any point at random without passing through intervening points.

Because tapes are so slow, they are generally used only for long – term storage and backup. Data to be used regularly is almost always kept on a disk. Tapes are also used for transporting large amounts of data.

Tapes come in a variety of sizes and formats (see table 10.1). Tapes are sometimes called streamers or streaming tapes.

Type	Capacity	Description
Half – inch	60 MB – 400 MB	Half – inch tapes come both as 9 track reels and as cartridges. These tapes are relatively cheap, but require expensive tape drives.
Quarter – inch	40 MB – 5 GB	Quarter – inch cartridges (QIC tapes) are relatively inexpensive and support fast data transfer rates. QIC mini cartridges are even less expensive, but their data capacities are smaller and their transfer rates are slower.
8-mm Helical – scan	1GB – 5GB	8-mm helical – scan cartridges use the same technology as VCR tapes and have the greatest capacity. But they require expensive tape drivers and have relatively slow data transfer rates.
4-mm DAT	2GB-24G B	DAT (Digital Audio Tape) cartridges have the greatest capacity but they require expensive tape drivers and have relatively slow data transfer rates.

Helical – scan Cartridge

A type of magnetic tape that uses the same technology as VCR tapes. The term helical scan usually refers to 8-mm tapes, although 4-mm tapes (called DAt tapes) uses the same technology. The 8-mm helical – scan tapes have data capacities from 2.5GB to 5GB.

DAT Cartridge

DAT (Digital Audio Tape) is a type of magnetic tape that uses an ingenious scheme called helical scan to record data. A DAT cartridge is slightly larger than a credit card and contains a magnetic tape that can hold from 2 to 24 gigabytes of data. It can support data transfer rates of about 2 MBps (Million bytes per second). Like other types of tapes, DATs are sequential – access media. The most common format for DAT cartridge is DDS (digital data storage) which is the industry standard for digital audio tape (DAT) formats. The latest format, DDS-3, specifies tapes that can hold 24 GB (the equivalent of over 40 CD-ROMs) and support data transfer rates of 2 MBps.

3.4.3 WINCHESTER DISK

The term Winchester comes from an early type of disk drive developed by IBM that stored 30 MB and had a 30-millisecond access time; so its inventors named it a Winchester in honor of the 30-caliber rifle of the same name. Although modern disk drives are faster and hold more data, the basic technology is the same, so Winchester has become synonymous with hard disk.

3.4.4 HARD DISK

Hard disk is a magnetic disk on which you can store computer data. The hard is used to distinguish it from a soft, or floppy, disk. Hard disks hold more data and are faster than floppy disks. A hard disk, for example, can store anywhere from 10 megabytes to several gigabytes, whereas most floppies have a maximum storage capacity of 1.4 megabytes.

A single hard disk usually consists of several platters. Each platter requires two read/write heads, one for each side. All the read/write heads are attached to a single access arm so that they cannot move independently. Each platter has the same number of tracks, and a track location that cuts across all platters is called a cylinder. For example, a typical 84-megabyte hard disk for a PC might have two platters (four sides) and 1,053 cylinders

In general, hard disks are less portable than floppies, although it is possible to buy removable hard disks. There are two types of removable hard disks : disk packs and removable cartridges..

3.4.5 FLOPPY DISK

Floppy disk is a soft magnetic disk. It is called floppy because it flops if you wave it (at least, the $5\ ^{1/4}$ – inch variety does). Unlike most hard disks, floppy disks (often called floppies or diskettes) are portable, because you can remove them from a disk drive. Disk drives for floppy disks are called

floppy drives. Floppy disks are slower to access than hard disks and have less storage capacity, but they are less expensive and are portable.

Floppies come in two basic sizes:

- **5 1/4 – inch –** The common size for PCs made before 1987. This type of floppy is generally capable of storing between 100K and 1.2 MB (megabytes) of data. The most common sizes are 360 K and 1.2MB.
- **3 1/2– inch –** Floppy is something of a misnomer for these disks, as they are encased in a rigid envelope. Despite their small size, microfloppies have a larger storage capacity than their cousins – from 40K to 1.4MB of data. The most common sizes for PCs are 720K (double - density) and 1.44MB (high- density). Macintoshes support disks of 400K, 800K, and 1.2MB.

3.4.6 ZIP DISK

These are high – capacity floppy disk drives developed by the Iomega Corporation. Zip disks are slightly larger than the conventional floppy disks, and are about twice as thick. They can hold 100MB of data. Because they're relatively inexpensive and durable, they have become a popular media for backing up hard disks and for transporting large files.

3.4.7 JAZ DISK

These are removable disk drivers developed by the Iomega Corporation. The Jaz drive has a 12-ms average seek time and a transfer rate 5.5 Mbps.

The removable cartridges hold 1GB of data. The fast rates and large storage capacity make it a viable alternative for backup storage as well as everyday use.

3.4.8 SUPER DISK

This is a new disk storage technology developed by the Imation Corporation that supports very high-density diskettes. SuperDisk diskettes are etched with a servo pattern at the factory. This pattern is then read by the SuperDisk drive to precisely slign the read/write head. The result is that a SuperDisk diskette can have 2,490 tacks, as opposed to the 135 tacks that conventional 3.5 – inch 1.44 MB diskettes use. This density translates into 120MB capacity per diskette.

3.4.9 OPTICAL DISK

Optical Disks are a storage medium from which data is read and to which it is written by lasers. Optical disks can store much more data – up to 6 gigabytes (6 billion bytes)- than magnetic media, such as floppies and hard disks. There are three basic of optical disks:

- **CD-ROM** - Like audio CDs, CD-ROMs come with data already encoded onto them. The data is permanent and can be read any number of times, but CD-ROMs cannot be modified.

- **WORM** – This term stands for write – once, read many. With a WORM disk drive, you can write data onto a WORM disk, but only once. After that, the WORM disk behaves just like a CD-ROM.
- **Erasable** - Optical disks that can be erased and loaded with new data, just like magnetic disks. These are often referred to as EO (erasable optical) disks.

3.4.10 CD – ROM

CD – ROM, which is pronounced as 'see – dee - rom', is the abbreviation of Compact Disc – Read – Only Memory. CD – ROM is a type of optical disk capable of storing large amounts of data - up to 1GB, although the most common size is 630 MB (megabytes). A single CD – Rom has the storage capacity of 700 floppy disks, enough memory to store about 300,000 text pages.

CD – ROMs are recorded by the vendor, and once recorded, they cannot be erased and filled with new data. To read a CD, you need a CD – ROM player. Also called a CD – ROM drive, a CD – ROM player is a device that can read information from a CD – ROM.

There are a number of features that distinguish CD – ROM players, the most important of which is probably their speed. CD– Rom players are generally classified as single – speed or some multiple of single – speed. For example, a 4X player access data at four times the speed of a single – speed player. Within these groups, however, there is some variation. Also, you need to be aware of whether the CD – ROM uses the CLV (Constant Linear Velocity) or CAV (Constant Angular Velocity) technology.

Two more precise measurements are the drives seek time and data transfer rate. The seek time, also called the access time, measures how long, on average; it takes the drive to access a particular piece of information. The data transfer rate measures how much data can be read and sent to the computer in a second.

Most CD –ROMs connect via a SCSI bus. Other CD – ROMs connect to an IDE or enhanced IDE interface, which is the one used by the hard disk drive. CD – ROMs are particularly well suited to information that requires large storage capacity. This includes color graphics, sound, and especially video.

3.4.11 CD – R Drive

CD – R drive which is short for Compact Disk – Recordable drive, is a type of disk drive that can create CD- ROMs and audio CDs. This allows the users to "master" a CD – Rom or audio CD for publishing. Until recently, CD – R drives were quite expensive, but prices have dropped dramatically.

A feature of many CD – R drives, called multi session recording, and enables you to keep adding data to a CD – ROM over time. This is extremely important if you want to use the CD – R drive to create backup CD – ROMs.

To create CD – ROMs and audio CDs, you'll need not only a CD – R drive, but also a CD – R software package.

Often, it is the software package, not the drive itself that determines how easy or difficult it is to create CD – ROMs. CD – R drives can also read CD – ROMs and play audio CDs.

3.4.12 CD – RW Disks

CD – RW disk is short for CD Rewritable disk and this is a new type of CD disk that enables you to write onto it in multiple sessions. One of the problems with CD – R disks is that you can only write to them once. With CD – RW drives and disks, you can treat the optical disk just like a floppy or hard disk, writing data onto it multiple times.

The first CD – RW drives became available in mid – 1997. They can read CD– ROMs and can write onto today's CD – R disks, but they cannot write on CD – ROMs. Many experts believe that they'll be a popular storage medium.

UNIT – III

Self Assessment Questions

Fill in the blank

1. Computer hardware falls into two categories _____ and _____ devices.
2. Computers have several additional storage locations called _____
3. Every computer comes with a certain amount of physical memory, usually referred to as _____
4. The memory unit that holds instructions for starting up the computer is _____
5. _____ is the input device consisting of a set of keys that enables you to enter data into a computer.
6. The three basic types of mice are _____ , _____ and _____.
7. SVGA stands for _____ graphics array.
8. Colour monitors are some times called _____ monitors.
9. There are two types of auxiliary storage devices _____, _____.
10. CD – ROM stands for _____.

True or False

1.Hardware is the mechanical and electronic device

2.Static RAM is slower than Dynamic RAM.

3.An EEPROM can be erased by exposing it to an electrical charge.

4.Track ball is another pointing device.

5.Light pen is not an Input device..

6.A Gray scale monitor is a special type of colour monitor capable of displaying different shades of gray.

7. On colour monitors, each pixel is composed of 3 dots - red, blue, green.

8 Auxiliary storage is long term, volatile memory.

9 Zip disk can hold 100 MB,

10CD –R drive is short for compact disk Recordable drive.

Multiple Choice

1. The two parts of the CPU are

   a). Control Unit Memory b) Addresses and registers

   c). Addresses and ALU d) Control Unit and ALU

2. The different types of memory units are

   a). RAM b) PROM c) ROM d)All of the above

3. Which of the following loses it contents when the computer is turned off?

   a)RAM b)ROM c) PROM d) All of the above

4. Which is the technology used in the evaluation of aptitude tests
   a)OCR b)OMR c) MICR d)MCR

5. The different types of printers are

   a) Daisy – wheel b). Dot matrix c) Laser & Ink – jet

   d) All of the above

6. Which of the following is a sequential access device.

   a). Hard disk b) Floppy disk c) Magnetic tape d) all of the above

7. CD – RW stands for

   a) CD – Recordable b) CD – Reusable

   c) CD – Rewritable d) None of the above

Unit Questions

1. What are the major functions of a computer?
2. Explain how the CPU and memory work.
3. What are the different types of memory?
4. What is the difference between RAM & ROM
5. What are the different kinds of input devices.
6. Explain MICR, OCR and OMR
7. What are the major output devices for a computer?
8. What do you mean by VGA, SVGA & XGA
9. What are the major auxiliary storage devices for a computer?
10. What is the difference between random access device and sequential access device?
11. Describe Zip disk, Jaz disk and Super disk.
12. Explain CD –ROM, CD – R Drive & CD – RW Disks.

.

Self Assessment Answers

Fill in the blanks

1. CPU, Pheripheral
2. Registers
3. The Main Memory (Or) The RAM
4. ROM
5. Keyboard
6. Mechanical, Optomechanical, Optical
7. SVGA
8. RGB
9. Magnetic Tapes, Magnetic Disks
10. Compact Disc – Read Only Memory

True / False

1. True
2. False
3. True
4. True
5. False
6. True
7. True
8. False
9. True
10. True

Multiple Choices

1. D
2. D
3. A
4. B
5. D
6. C
7. C

UNIT- IV

COMBINATIONAL LOGIC

UNIT IV

COMBINATIONAL LOGIC

4-1 INTRODUCTION:

A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the inputs and generate signals to the outputs.

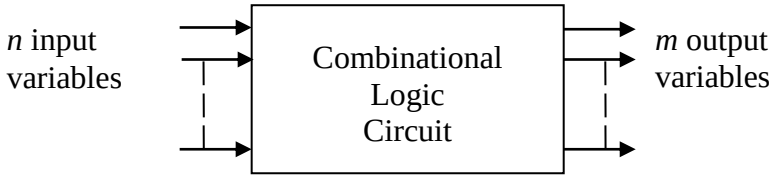


Figure 4-0 Block diagram of a combinational circuit

A block diagram of a combinational circuit is shown in Fig.4-0. The *n* input binary variables come from an external source; the *m* output variables go to an external destination.

4-2 ADDERS

Digital computers perform a variety of information-processing tasks. Among the basic functions encountered are the various arithmetic operations. The most basic arithmetic operation, no doubt, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely, 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10. The first three operations produce a sum whose length is one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a *carry*. When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher-order pair of significant bits. A combinational circuit that performs the addition of two bits is called a *half-adder*. One that performs the addition of three bits (two significant bits and a previous carry) is a *full-adder*. The name of the former stems from the fact that two half-adders can be employed to implement a full-adder. The two adder circuits are the first combinational circuits we shall design.

4.2.1 Half-Adder

From the verbal explanation of a half-adder, we find that this circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. It is necessary to specify two output variables because the result may consist of two binary digits. We arbitrarily assign symbols *x* and *y* to the two inputs and *S* (for sum) and *C* (for carry) to the outputs.

Now that we have established the number and names of the input and output variables, we are ready to formulate a truth table to identify exactly the function of the half-adder. This truth table is shown below:

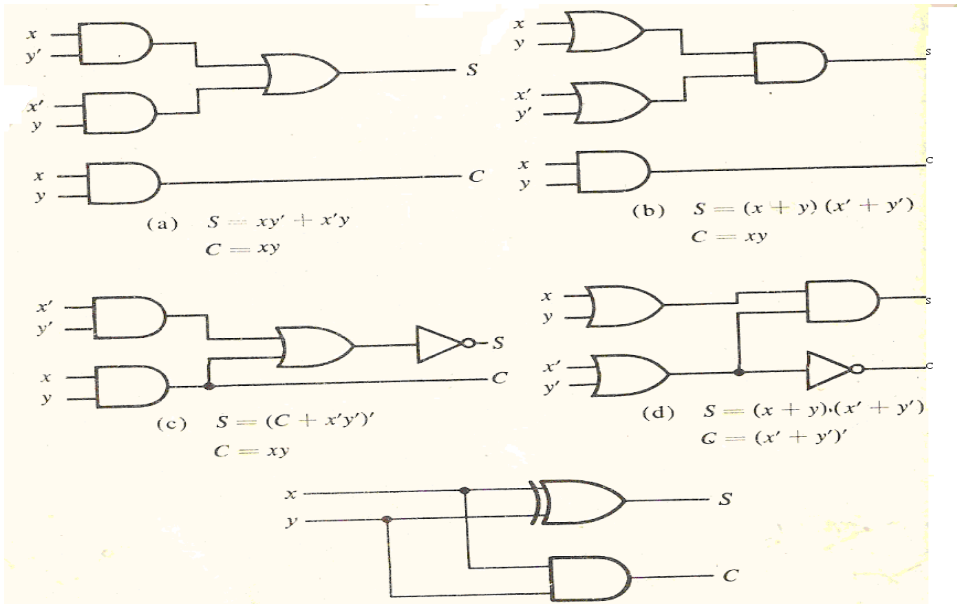
x	*y*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The carry output is 0 unless both inputs are 1. The *S* output represents the least significant bit of the sum.

The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum of products expressions are:

$$S = x'y + xy'$$

$$C = xy$$



(e) $S = x \oplus y$

$C = xy$

Figure 4-1 : Various implementation of a half-adder

The logic diagram for this implementation is shown in Fig. 4-1(a), as are four other implementations for a half-adder. They all achieve the same result as far as the input-output behavior is concerned. They illustrate the flexibility available to the designer when implementing even a simple combinational logic function such as this.

Fig 4-1(a), as mentioned above, is the implementation of the half-adder in sum of products. Figure 4-1(b) shows the implementation in product of sums:

$$S = (x + y)(x' + y')$$
$$C = xy$$

To obtain the implementation of Fig. 4-1(c), we note that S is the exclusive-OR of x and y. The complement of S is the equivalence of x and y:

$$S' = x y + x' y'$$

but $C = xy$, and therefore we have:

$$S = (C + x' y')$$

In Fig. 4-1(d) we use the product of sums implementation with C derived as follows :

$$C = xy = (x' + y')'$$

The half-adder can be implemented with an exclusive-OR and an AND gate as shown in Fig. 4-1(e). This form is used later to show that two half-adder circuits are needed to construct full-adder circuit.

4.2.2 Full-Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y, represent the two significant bits to be added. The third input, z, represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary 2 or 3 needs two digits. The two outputs are designated by the symbols S for sum and C for carry. The binary variable S gives the value of least significant bit of the sum. The binary variable C gives the output carry. The truth table of the full-adder is as follows:

x	***y***	***z***	***C***	***S***
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The eight rows under the input variables designate all possible combinations of 1's and 0's that these variables may have. The 1's and 0's for the output variables are determined from the arithmetic sum of the input bits. When all input bits are 0's, the output is 0. The *S* output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The *C* output has a carry of 1 if two or three inputs are equal to 1.

The input and output bits of the combinational circuit have different interpretations at various stages of the problem. Physically, the binary signals of the input wires are considered binary digits added arithmetically to form a two-digit sum at

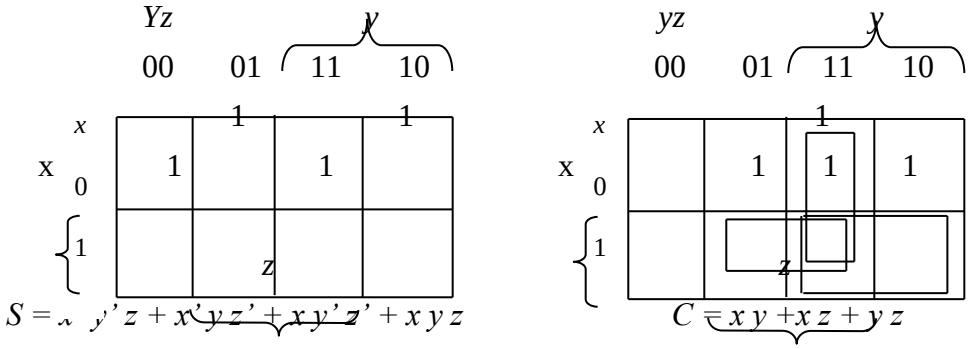


Figure4-2 Maps for full-adder

the output wires. On the other hand, the same binary values are considered variables of Boolean functions when expressed in the truth table or when the circuit is implemented with logic gates. It is important to realize that two different interpretations are given to the values of the bits encountered in this circuit.

The input-output logical relationship of the full-adder circuit may be expressed in two Boolean functions, one for each output variable. Each output Boolean function requires a unique map for its simplification. Each map must have eight squares, since each output is a function of three input variables. The maps of Fig. 4-2 are used for simplifying the two output functions. The 1's in the squares for the maps of *S* and *C* are determined directly from the truth table. The squares with 1's for the *S* output do not combine in adjacent squares to give a simplified expression in sum of products. The C output can be simplified to a six-literal expression. The logic diagram for the full-adder implemented in sum of products is shown in Fig.4-4. This implementation uses the following Boolean expressions:

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

Other configurations for a full-adder may be developed. The product-of-sums implementation requires the same number of gates as in Fig.4-4, with the

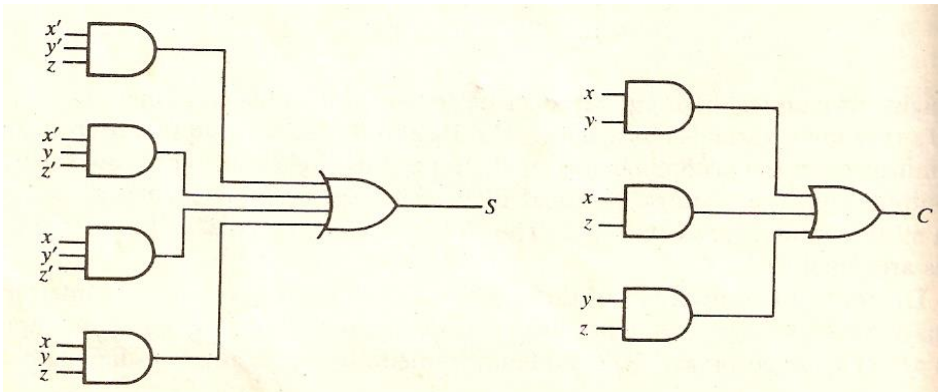


Figure 4-4 Implementation of full-adder in sum of products

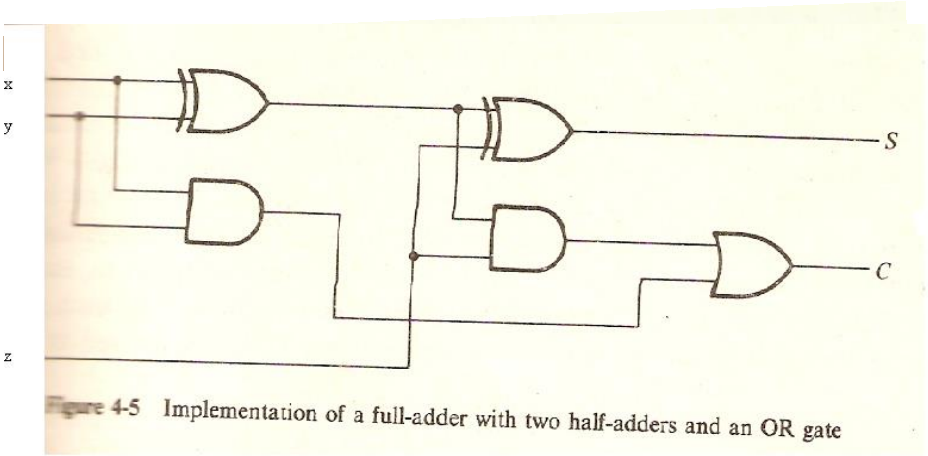


Figure 4-5 Implementation of a full-adder with two half-adders and an OR gate

number of AND and OR gates interchanged. A full-adder can be implemented with two half-adders and one OR gate, as shown in Fig.4-5. The *S* output from the second half-adder is the exclusive-OR of z and the output of the first half-adder, giving:

$$S = z \oplus (x \oplus y)$$
$$= z'(xy' + x'y) + (xy' + x'y)'$$
$$= z'(xy' + x'y) + z(xy + x'y')$$
$$= xy'z' + x'yz' + xyz + x'y'z$$

and the carry output is:

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

4-3 SUBTRACTORS

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend (Section 1-5). By this method, the subtraction operation operation becomes an addition operation requiring full-adders for its machine implementation. It is possible to implement subtraction with logic circuits in a direct manner, as done with paper

and pencil. By this method, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position. The fact that a 1 has been borrowed must be conveyed to the next higher pair of bits by means of a binary signal coming out (output) of a given stage and going into (input) the next higher stage. Just as there are half- and full-adders, there are half- and full-subtractros.

4.3.1 Half-Subtractor

A half-subtractor is a combinational circuit that subtracts two bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Designate the minuend bit by *x* and the subtrahend bit by *y*. To perform *x – y,* we have to check the relative magnitudes of *x* and *y*. If *x > y,* we have three possibilities: 0 - 0 = 0 , 1 - 0 = 1, and 1 – 1 = 0. The result is called the difference bit. If *x < y,* we have 0 - 1, and it is necessary to borrow a 1 from the next higher stage. The 1 borrowed from the next higher stage adds 2 to the minuend bit, just as in the decimal system a borrow adds 10 to a minuend digit. With the minuend equal to 2, the difference becomes 2-1=1. The half-subtractor needs two outputs. One output generates the difference and will be designated by the symbol *D*. The second output, designated *B* for borrow, generates the binary signal that informs the next stage that a 1 has been borrowed. The truth table for the input-output relationship of a half-subtractor can now be derived as follows:

x	***y***	***B***	***D***
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

The output borrow *B* is a 0 as long as $x \geq y$. It is a 1 for $x = 0$ and $y = 1$. The *D* output is the result of the arithmetic operation $2B + x – y$.

The Boolean functions for the two outputs of the half-subtractor are derived directly from the truth table:

$$D = x'y + xy'$$

$$B = x'y$$

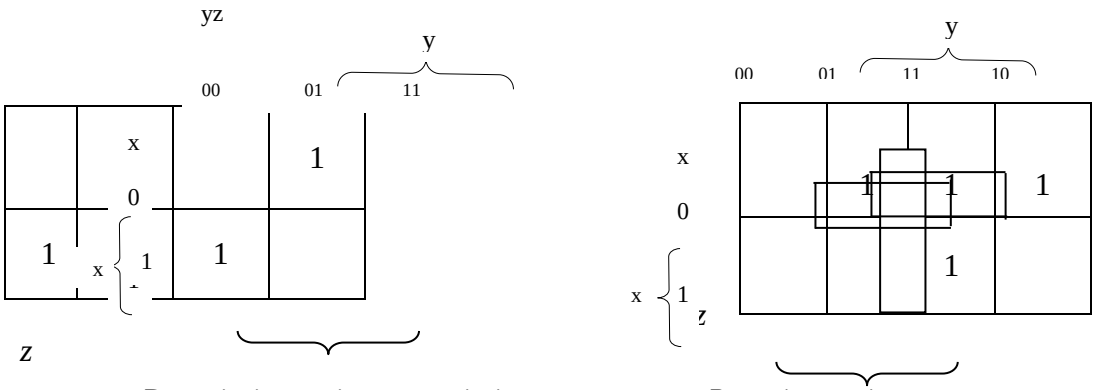
It is interesting to note that the logic for *D* is exactly the same as the logic for output *S* in the half-adder.

4.3.2 Full-Subtractor

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a

lower significant stage. This circuit has three inputs and two outputs. The three inputs, *x, y,* and *z*, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, *D* and *B*, represent the difference and output borrow, respectively. The truth table for the circuit is as follows:

x	*y*	*z*	*B*	*D*
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



$D = x'y'z + x'yz + xy'z' + xyz$ $B = x'y + x'z + yz$

Figure 4-6 Maps for Full-adder

The eight rows under the input variables designate all possible combinations of 1's and 0's that the binary variables may take. The 1's and 0's for the output variables are determined from the subtraction of x – y – z. The combinations having input borrow *z = 0* reduce to the same four conditions of the half-adder. For *x = 0, y = 0,* and *z = 1,* we have to borrow a 1 from the next stage, which makes *B = 1* and adds 2 to *x*. Since 2 – 0 – 1 = 1, *D = 1*. For *x = 0* and *y z = 11,* we need to borrow again, making *B = 1* and *x = 2*. Since 2 – 1 – 1 = 0, *D = 0.* For *x = 1* and *y z =01,* we have *x – y – z = 0,* which makes *B = 0* and *D = 0.* Finally, for *x = 1, y = 1, z = 1,* we have to borrow 1,making *B = 1* and *x = 3,* and 3 – 1 – 1 = 1, making *D = 1*.

The simplified Boolean functions for the two outputs of the full-subtractor are derived in the maps of Fig. 4-5. The simplified sum of products output functions are:

$$D = x'y'z + x'yz' + xyz' + xyz$$

$$B = x'y + x'z + yz$$

Again we note that the logic function for output *D* in the full-subtractor is exactly the same as output *S* in the full-adder. Moreover, the output *B* resembles the function for *C* in the full-adder, except that the input variable *x* is complemented. Because of these similarities, it is possible to convert a full-adder into a full-subtractor by merely complementing input *x* prior to its application to the gates that form the carry output.

4.4 DECODERS

Discrete quantities of information are represented in digital systems with binary codes. A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information. A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit decoded information has unused or don't-care combinations, the decoder output will have less than 2^n outputs.

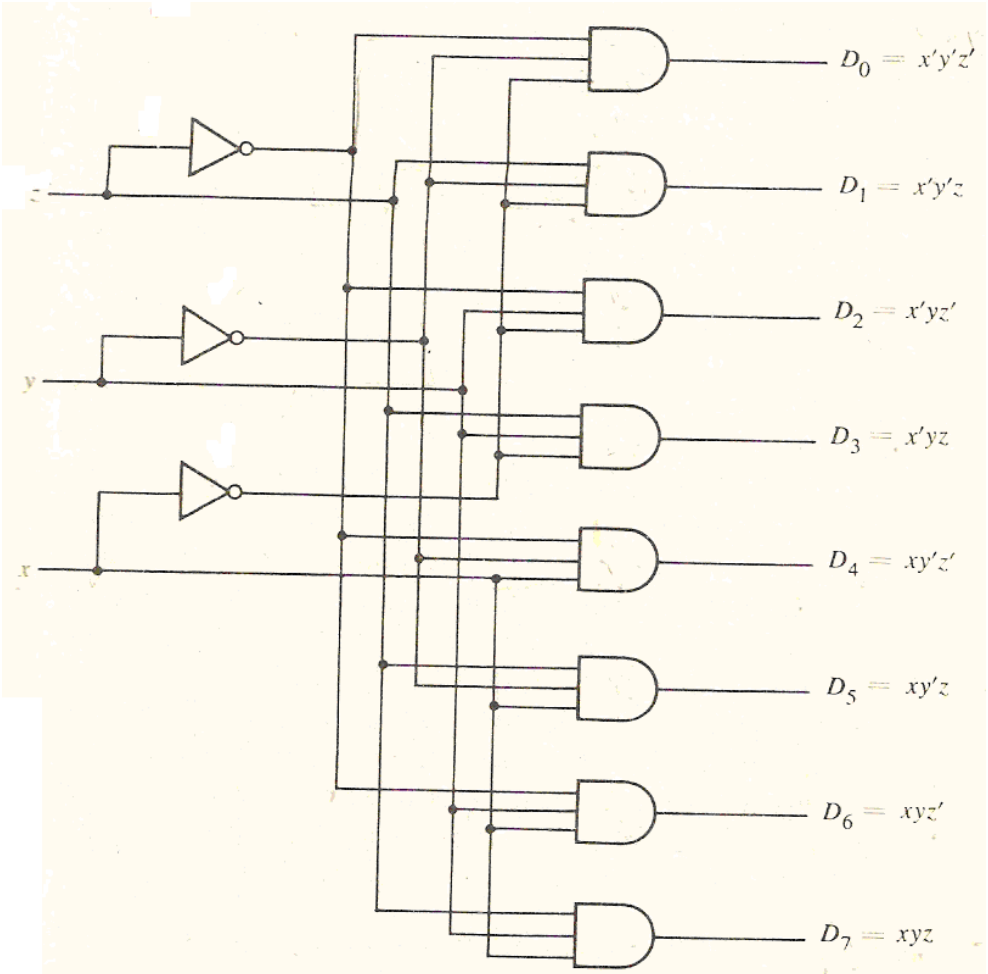


Figure 5-8 A 3-to-8 line decoder

The decoders presented here are called n – to - m line decoders where $m \leq 2^n$. Their purpose is to generate the 2^n (or less) minterms of n input variables. The name decoder is also used in conjunction with some code converters such as a BCD-to-seven-segment decoder.

As an example, consider the 3-to-8 line decoder circuit of Fig.5.8. The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. A particular application of this decoder would be a binary-to-octal conversion. The input variables may represent a binary number, and the outputs will then represent the eight digits in the octal number system. However, a 3-to-8-line decoder can be used for decoding any 3-bit code to provide eight outputs, one for each element of the code.

The operation of the decoder may be further clarified from its input-output relationship, listed in Table 4-8. Observe that the output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.*

Inputs			**Outputs**							
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

4.5 ENCODERS

An encoder is a digital function that produces a reverse operation from that of a decoder. An encoder has 2^n (or less) input lines and n output lines. The output lines generate the binary code for the 2^n input variables. An example of an encoder is shown in Fig.5-15. 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. It is constructed with OR gates whose inputs can be determined from the truth table given in

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Table 5-4. The low-order output bit z is 1 if the input octal digit is odd. Output y is 1 for octal digits 2,3,6 or 7. Output x is a 1 for octal digits 4,5,6, or 7. Note that D_0 is not connected to any Or gate; the binary output must be all 0's in this case. An all 0's output is also obtained when all inputs are all 0's. This discrepancy can be resolved by providing one more output to indicate the fact that all inputs are not 0's.

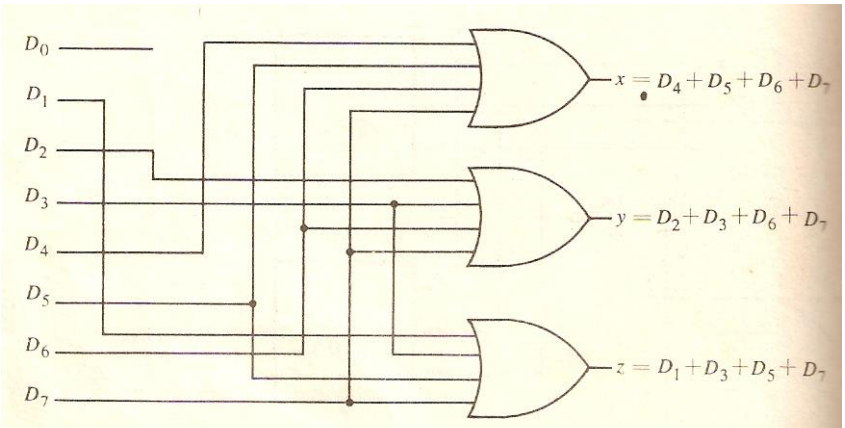


Figure 5-15 Octal-to-binary encoder

The encoder in Fig. 5-15 assumes that only one input line can be equal to 1 at any time; otherwise the circuit has no meaning. Note that the circuit has eight inputs and could have 2^8 =256 possible input combinations. Only eight of these combinations have any meaning. The other input combinations are don't-care conditions.

Encoders of this type (Fig.5-15) are not available in *IC* packages, since they can be easily constructed with OR gates. The type of encoder available in *IC* form is called a priority encoder.* These encoders establish an input priority to ensure that only the highest- priority input line is

encoded. Thus, in Table 5-4, if priority is given to an input with a higher subscript number over one with a lower subscript number, then if both D_2 and D_5 are logic-1 simultaneously, the output will be 101 because D_5 has a higher priority over D_2. Of course, the truth table of a priority encoder is different from the one in Table 5-4.

4.6 MULTIPLEXERS

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2n input lines and n selection lines whose bit combinations determine which input is selected.

A 4-line to 1-line multiplexer is shown in Fig.5-16. Each of the four input lines, I_0 to I_3, is applied to one input of an AND gate. Selection lines s_1 and s_0 are decoded to select a particular AND gate. The function table in the figure lists the input-to-output path for each possible bit combination of the selection lines. When this MSI functions is used in the design of a digital system, it is represented in block diagram form as shown in Fig.5-16(c). To demonstrate the circuit operation, consider the case when $s_1s_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input equal to I_2. The other three AND gates have at least one input equal to 0, which makes their output equal to 0. The OR-gate have at least one input equal to the value of I_2, thus providing a path from the selected input to the output. A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output line.

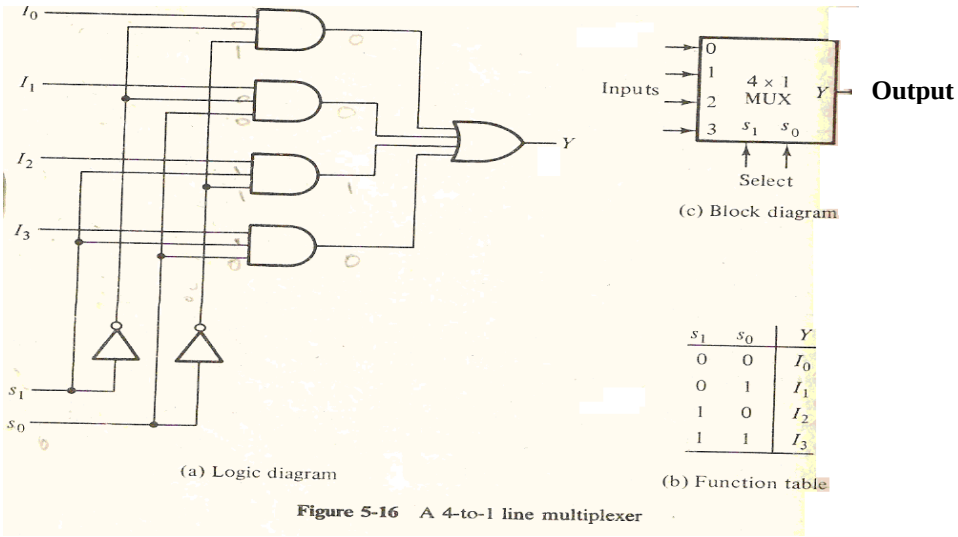


Figure 5-16 A 4-to-1 line multiplexer

The AND gates and inverters in the multiplexer resemble a decoder circuit and, indeed, they decode the input selection lines. In general, a 2n-to-1 line multiplexer is constructed from an n-to-2n decoder by adding to it 2n input lines, one to each AND gate. The outputs of the AND gates are applied to a single OR gate to provide the 1-line output. The size of a multiplexer is

specified by the number 2n of its input lines and the single output line. It is then implied that it also contains n selection lines. A multiplexer is often abbreviated as MUX.

As in decoders, multiplexer *ICs* may have an enable input to control the operation of the unit. When the enable input is in a given binary state, the outputs are disabled, and when it is in the other state (the enable state), the circuit functions as a normal multiplexer. The enable input *ICs* to a digital multiplexer with a larger number of inputs.

In some cases two or more multiplexers are enclosed within one *IC* package. The selection and enable inputs in multiple-unit *ICs* may be common to all multiplexers. As an illustration, a quadruple 2-line to 1-line multiplexer *IC* is shown in Fig. 5-17.* It has four multiplexers, each capable of selecting one of two input lines. Output Y_1 can be selected to be equal to either A_1 or B_1. Similarly, output Y_2 may have the value of A_2 or B_2, and so on. One input selection line, *S*,

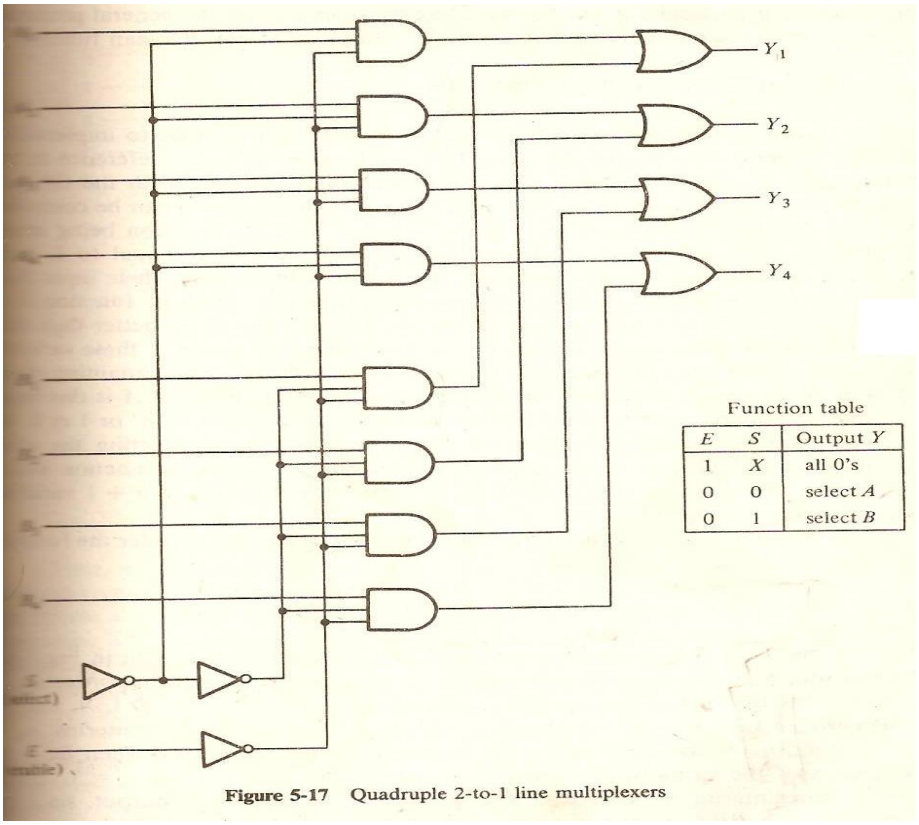


Figure 5-17 Quadruple 2-to-1 line multiplexers

***This is similar to IC type 74157.**

Suffices to select one of two lines in all four multiplexers. The control input *E* enables the multiplexers in the 0 state and disables them in the 1 state. Although the circuit contains four multiplexers, we may think of it as a circuit that selects one in a pair of 4-input lines. As shown in the function table, the unit is selected when *E=0.* Then, if *S=0*, the four A inputs have a path to the

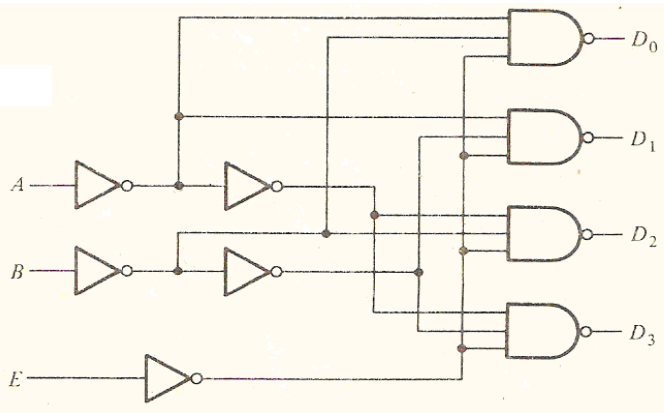
outputs. On the other hand, if *S=1,* the four *B* inputs are selected. The outputs have all 0's when *E=1,* regardless of the value of *S.*

4.7 DEMULTIPLEXERS

Some *IC* decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form. Most, if not all, *IC* decoders include one or more enable inputs to control the circuit operation. A 2-to-4 line decoder with an enable input constructed with NAND gates is shown in Fig. 5-12. All outputs are equal to 1 if enable input *E* is 1, regardless of the values of inputs *A* and *B.* When the enable input is 0, the circuit operates as a decoder with complemented outputs. The truth table lists these conditions. The *X's* under *A* and *B* are don't-care conditions. Normal decoder operation occurs only with *E = 0,* and the outputs are selected when they are in the 0 state.

The block diagram of the decoder is shown in Fig. 5-13(a). The small circle at input *E* indicates that the decoder is enabled when *E = 0.* The small circles at the outputs indicate that all outputs are complemented.

A decoder with an enable input can function as a demultiplexer. A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines



(a) Logic diagram.

E	A	B	D_0	D_1	D_3
1	X	X	1	1	1
0	0	0	0	1	1
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	1	1	0

(b) Truth table

Figure 5-12 A 2-to-4 line decoder with enable (*E*) input

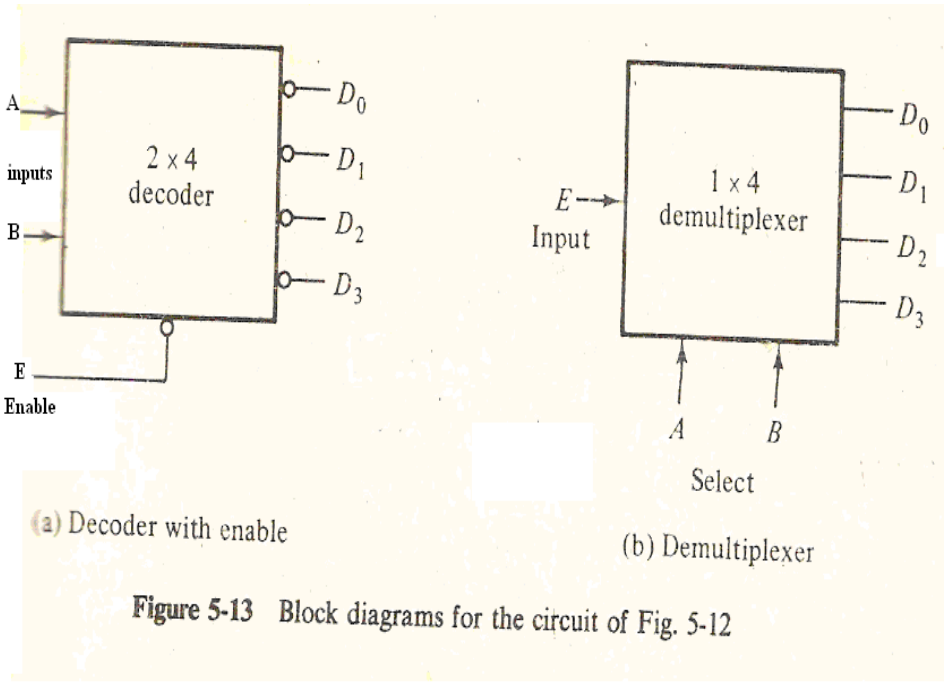


Figure 5-13 Block diagrams for the circuit of Fig. 5-12

The decoder of Fig. 5-12 can function as a demultiplexer if the E line is taken as a data input line and lines *A* and *B* are taken as the selection lines. This is shown in Fig. 5-13(b). The single input variable *E* has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary value of the two selection lines *A* and *B*. This can be verified from the truth table of this circuit, shown in Fig. 5-12(b). For example, if the selection lines *AB=10*, output D_2 will be the same as the input value *E*, while all other outputs are maintained at 1. Because decoder with an enable input that makes the circuit a demultiplexer; the decoder itself can use AND, NAND, or NOR gates.

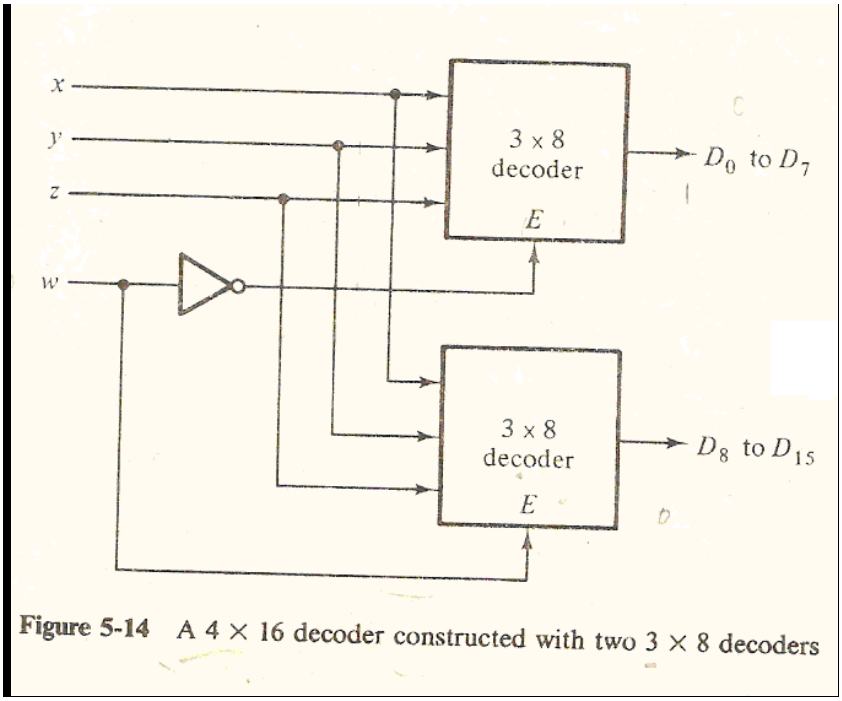


Figure 5-14 A 4 × 16 decoder constructed with two 3 × 8 decoders

Decoder/ Demultiplexer circuits can be connected together to form a larger decoder circuit. Figure. 5-14shows two 3 × 8 decoders with enable inputs connected to form a 4 × 16 decoder. When *w = 0*, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's and the top eight outputs generate minterms 0000 to 0111. When *w = 1*, the enable conditions are reversed; the bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's. This example demonstrates the usefulness of enable inputs in *ICs.* In general, enable lines are a convenient feature for connecting two or more *IC* packages for the purpose of expanding the digital function into a similar function with more inputs and outputs.

4.8 FLIP-FLOPS

A flip-flop circuit can maintain a binary state indefinitely (as long as power is delivered to the circuit) until directed by an input signal to switch states. The major differences among various types of flip-flops are in the number of inputs they posses and in the manner in which the inputs affect the binary state. The most common types of flip-flops are discussed below.

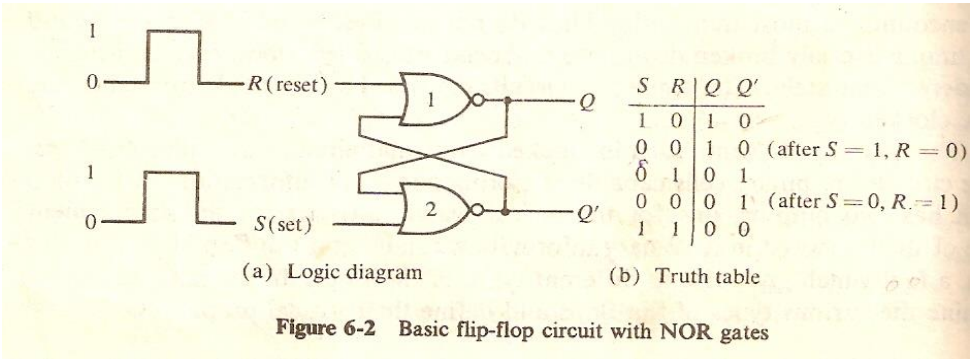
4.8.1 BASIC FLIP-FLOP CIRCUIT

It was mentioned in sections 4-7 and 4-8 that a flip-flop circuit can be constructed from two NAND gates or two NOR gates. These constructions are shown in the logic diagrams of Fig. 6.2 and 6-3. Each circuit forms a basic flip-flop upon which other more complicated types can be built. The cross-coupled connection from the output of one gate to the input of the other gate constitutes a feedback path. For this reason, the circuits are classified as asynchronous sequential circuits. Each flip-flop has two outputs, *Q* and *Q´*

and two inputs, set and reset. This type of flip-flop is sometimes called a direct-coupled *RS* flip-flop or SR latch. The *R* and *S* are the first letters of the two input names.

To analyze the operation of the circuit of Fig. 6-2, we must remember that the output of a NOR gate is 0 if any inputs is 1, and that the output is 1 only when all inputs are 0. As a starting point, assume that the set input is 1 and the reset input is 0. Since gate 2 has an input of 1, its output *Q´* must be 0, which puts both inputs of gate 1 at 0, so that output *Q* is 1. When the set input is returned to 0, the outputs remain the same, because output Q remains a 1, leaving one input of gate 2 at 1. That causes output Q´ to stay at 0, which levels both inputs of gate number 1 at 0, so that output *Q* is a 1. In the same manner it is possible to show that a 1 in the reset input changes output *Q* to 0 and Q´ to 1. When the reset input returns in 0, the outputs do not change.

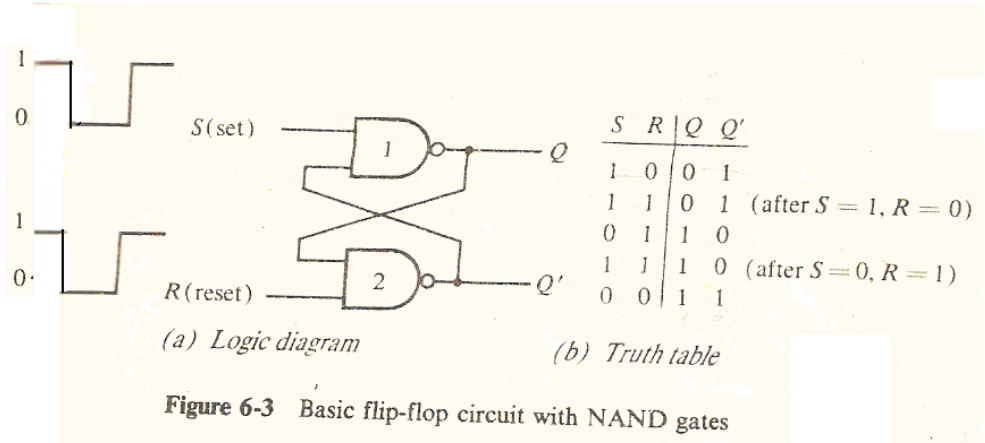
When a 1 is applied to both the set and the reset inputs, both *Q* and *Q´* outputs go to 0. This condition violates the fact that outputs *Q* and *Q´* are the complements of each other. In normal operation this condition must be avoided by making sure that 1's are not applied to both inputs simultaneously.



S	R	Q	Q′	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	

(a) Logic diagram (b) Truth table

Figure 6-2 Basic flip-flop circuit with NOR gates



S	R	Q	Q′	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	

(a) Logic diagram *(b) Truth table*

Figure 6-3 Basic flip-flop circuit with NAND gates

A flip-flop has two useful states. When *Q = 1* and *Q´ = 0*, it is in the set state (or 1-state). When *Q = 0* and Q´ = 1, it is in the clear state (or 0-state). The outputs *Q* and *Q´* are complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output.

Under normal operation, both inputs remain at 0 unless the state of the flip-flop has to be changed. The application of a momentary 1 to the set input causes the flip-flop to go to the set state. The set input must go back to 0 before a 1 is applied to the reset input. A momentary 1 applied to the reset input causes the flip-flop to go the clear state. When both inputs are initially 0, a 1 applied to the set input while the flip-flop is in the set state or a 1 applied to the reset input while the flip-flop is in the clear state leaves the outputs unchanged. When a 1 is applied to both the set and the reset inputs, both outputs go to 0. This state is undefined and is usually avoided. If both inputs now go to 0, the state of the flip-flop is indeterminate and depends on which input remains a 1 longer before the transition to 0.

The NAND basic flip-flop circuit of Fig. 6-3 operates with both inputs normally at 1 unless the state of the flip-flop has to be changed. The application of a momentary 0 to the set input causes output *Q* to go to 1 and *Q´* to go to 0, thus putting the flip-flop into the set state. After the set input returns to 1, a momentary 0 to the reset input causes a transition to the clear state. When both inputs go to 0, both outputs go to 1–a condition avoided in normal flip-flop operation.

4.8.2 Clocked *RS* Flip-Flop

The basic flip-flop as it stands is an asynchronous sequential circuit. By adding gates to the inputs of the basic circuit, the flip-flop can be made to respond to input levels during the occurrence of a clock pulse. The clocked *RS* flip-flop shown in Fig. 6-4(a) consists of a basic NOR flip-flop and two AND gates. The outputs of the two AND gates remain at 0 as long as the clock pulse (abbreviated *CP*) is 0, regardless of the S and R input values. When the clock pulse goes to 1, information from the S and R inputs is allowed to reach the basic flip-flop. The set is reached with *S = 1, R =0,* and *CP = 1.* To change to the clear state, the inputs must be *S = 0, R = 1,* and *CP = 1.* With both *S = 1* and *R = 1,* the occurrence of a clock pulse causes both outputs to momentarily go to 0. When the pulse is removed, the state of the flip-flop is indeterminate, i.e., either state may result, depending on whether the set or the reset input of the basic flip-flop remains a 1 longer before the transition to 0 at the end of the pulse.

The graphic symbol for the clocked RS flip-flop is shown in Fig.6-4(b). It has three inputs: *S, R,* and *CP*. The *CP* input is not written within the box because it is recognized from the marked small triangle. The triangle is a symbol for a *dynamic indicator* and denotes the fact that the flip-flop responds to an input clock transition from a low-level (binary 0) to a high-level (binary 1) signal. The outputs of the flip-flop are marked with *Q* and *Q´* within the box. The flip-flop can be assigned a different variable name even though *Q* is written inside the box. In that case the letter chosen for the flip-flop variable is marked outside the box along the output line. The state of the flip-flop is determined from the value of its normal output *Q.* If one wishes to obtain the complement of the normal output, it is not necessary to insert an inverter, because the complemented value is available directly from output *Q'*.

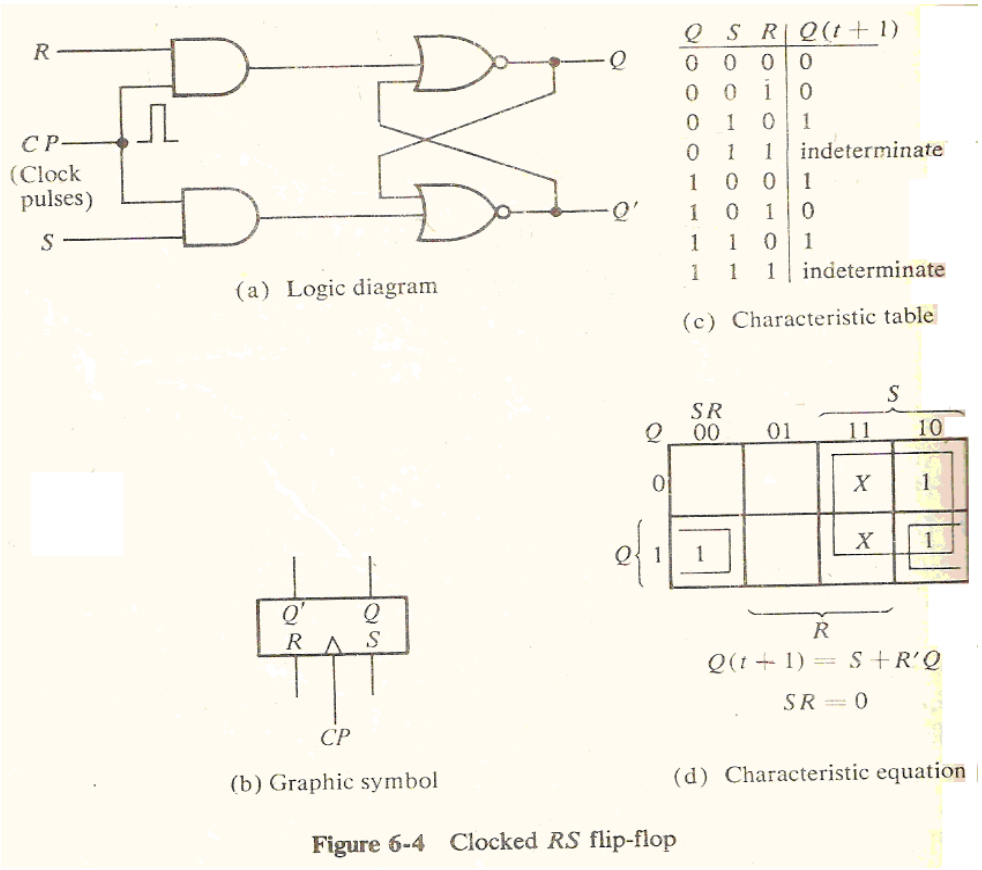


Figure 6-4 Clocked *RS* flip-flop

The characteristic table for the flip-flop for the flip-flop is shown in Fig. 6-4(c). This table summarizes the operation of the flip-flop in a tabular form. *Q* is the binary state of the flip-flop at a given time (referred to as present state), the *S* and *R* columns give the possible values of the inputs, and *Q (t + 1)* is the state of the flip-flop after the occurrence of a clock pulse (referred to as next state).

The characteristic equation of the flip-flop is derived in the map of Fig. 6-4(d). This equation specifies the value of the next state as a function of the present state and the inputs. The characteristic equation is an algebraic expression for the binary information of the characteristic table. The two indeterminate states are marked by *X's* in the map, since the may result in either a 1 or a 0. However, the relation *SR=0* must be included as part of the characteristic equation to specify that both *S* and *R* can not equal 1 simultaneously.

4.8.3 D FLIP- FLOP

The *D* flip-flop shown in Fig. 6-5 is a modification of the clocked *RS* flip-flop. NAND gates 1 and 2 form a basic flip-flop and gates 3 and 4 modify it in to a clocked *RS* flip-flop. The *D* input goes directly to the *S* input, and its complement, through gate 5, is applied to the *R* input. As long as the clock pulse in put is at 0, gates 3 and 4 have a 1 in their outputs, regardless of the value of the other in puts. This conforms to the requirement that the two inputs

of a basic NAND flip-flop (Fig. 6-3) remain initially at the 1 level. The *D* input is sampled during the occurrence of a clock pulse. If it is 1, the output of gate 3 goes to 0, switching the flip-flop to the set state (unless it was already set). If it is 0, the output of gate 4 goes to 0, switching the flip-flop to the clear state.

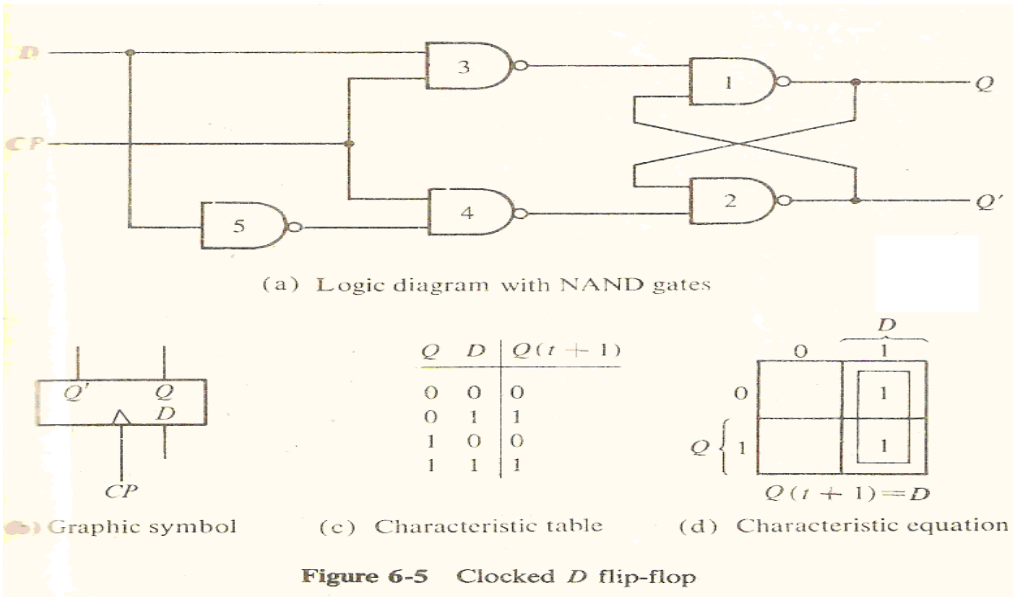


Figure 6-5 Clocked *D* flip-flop

The *D* flip-flop receives the designation from its ability to transfer "data" into a flip-flop. It is basically an *RS* flip-flop with an inverter in the *R* input. The added inverter reduces the number of inputs from two to one. This type of flip-flop is sometimes called a gated D-latch. The *CP* input is often given the variable designation *G* (for gate) to indicate that this input enables the gated latch to make possible the data entry into the flip-flop.

The symbol for a clocked *D* flip-flop is shown in Fig. 6-5(b). The characteristic table is listed in part (c) and the characteristic equation is derived in part (d). The characteristic equation shows that the next state of the flip-flop is the same as the *D* input and is independent of the value of the present state.

4.8.4 JK Flip-flop

A *JK* flip-flop is a refinement of the *RS* flip-flop in that the indeterminate state of the *RS* type is defined in the *JK* type. Inputs *J* and *K* behave like inputs *S* and *R* to set and clear the flip-flop (note that in a *JK* flip-flop, the letter J is for set and the letter *K* is for clear). When inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, that is, if *Q = 1*, it switches to *Q = 0*, and vice versa.

A clocked *JK* flip-flop is shown in Fig. 6-6(a). Output *Q* is ANDed with *K* and *CP* inputs so that the flip-flop is cleared during a clock pulse only if *Q* was previously 1. Similarly, output *Q'* is ANDed with *J* and *CP* inputs so that the flip-flop is set with a clock pulse only if *Q'* was previously 1.

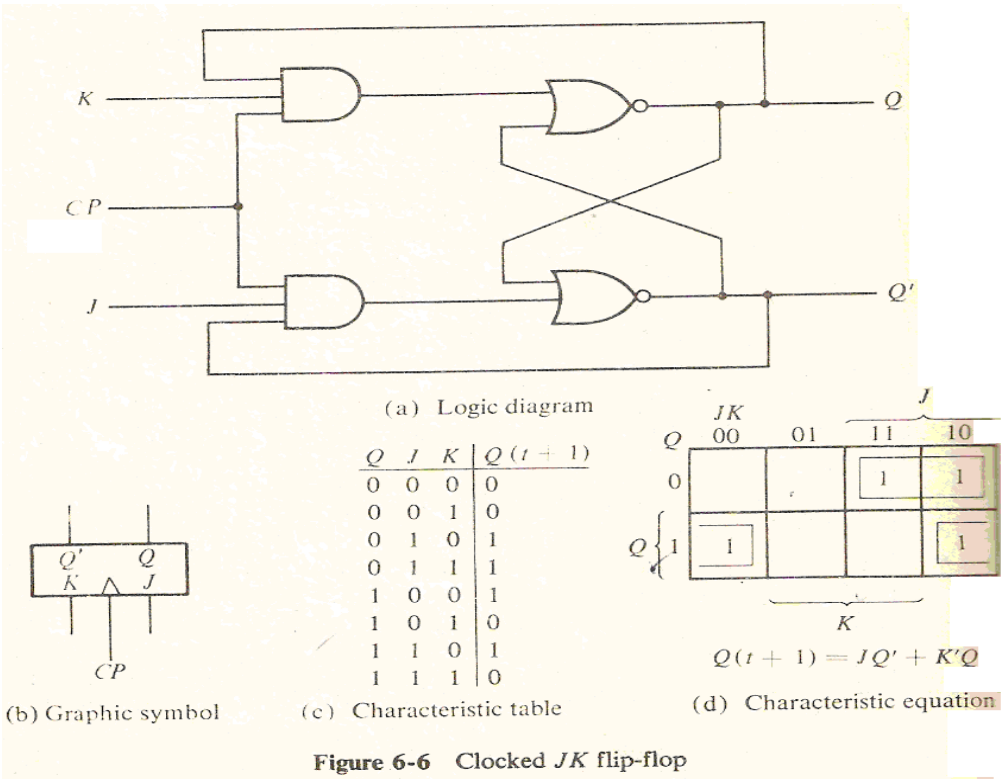
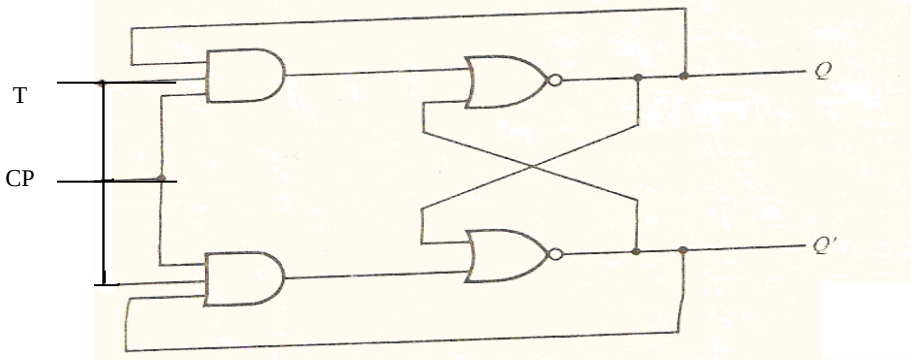


Figure 6-6 Clocked *JK* flip-flop

As shown in the characteristic table in Fig. 6-6(c), the *JK* flip-flop behaves like an *RS* Flip-flop, expect when both *J* and *K* are equal to 1. When both *J* and *K* are 1, the clock pulse is transmitted through one AND gate only – the one whose input is connected to the flip-flop output which is presently equal to 1. Thus, if *Q = 1,* the output of the upper AND gate becomes 1 upon application of a clock pulse, and the flip-flop is cleared. If *Q' = 1,* the output of the lower AND gate becomes a 1 and the flip-flop is set. In either case, the output state of the flip-flop is complemented.

The inputs in the graphic symbol for the *JK* flip-flop must be marked with a *J* (under *Q*) and *K* (under *Q'*). The characteristic equation is given in Fig. 6-4(d) and is derived from the map of the characteristic table.

Note that because of the feedback connection in the *JK* flip-flop, a *CO* signal which remains a 1 (while *J = K = 1*) after the outputs have been complemented once will cause repeated and continuous transitions of the outputs. To avoid this undesirable operation, the clock pulse must have a time duration which is shorter than the propagation delay through the flip-flop. This is a restrictive requirement, since the operation of the circuit depends on the width of the pulses. For this reason, *JK* flip-flops are never constructed as shown in Fig. 6-6(a). The restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction, as discussed in the next section. The same reasoning applies to the T flip-flop presented below.



(a) Logic diagram

Q	T	Q(t + 1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Characteristic table

Q \ T	0	1
0		1
1	1	

$Q(t+1) = TQ' + T'Q$

(d) Characteristic equation

Figure 6-7 Clocked *T* flip-flop

4.8.5 T Flip-flop

The *T* flip-flop is a single-input version of the *JK* flip-flop. As shown in Fig. (a), the *T* flip-flop is obtained from a *JK* type if both inputs are tied together. The designation *T* comes from the ability of the flip-flop to "toggle," or change state. Regardless of the present state of the flip-flop, it assumes the complement state when the clock pulse occurs while input *T* is logic-1. The symbol, characteristic table, and characteristic table, and characteristic equation of the T flip-flop are shown in Fig. 6-7, parts (b), (c), and (d), respectively.

4.8.6 Master-Slave Flip-Flop

A master-slave flip-flop is constructed from two separate flip-flops. One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a *master-slave flip-flop*. The logic diagram of an *RS* master-slave flip-flop is shown in Fig. 6-9. It consists of a master flip-flop, a slave flip-flop, and an inverter. When clock pulse *CP* is 0, the output of the inverter is 1. Since the clock input of the slave is 1, the flip-flop is enabled and output *Q* is equal to *Y*, while *Q'* is equal to *Y'*. The master flip-flop is disabled because *CP = 0.* When the pulse becomes 1, the information then at the external R and S inputs is transmitted to the master flip-flop. The slave flip-flop, however, is isolated as long as the pulse is at its 1 level, because the output of the inverter is 0. When the pulse returns to 0, the master flip-flop is isolated, which prevents the external inputs from affecting it. The slave flip-flop then goes to the same state as the master flip-flop.

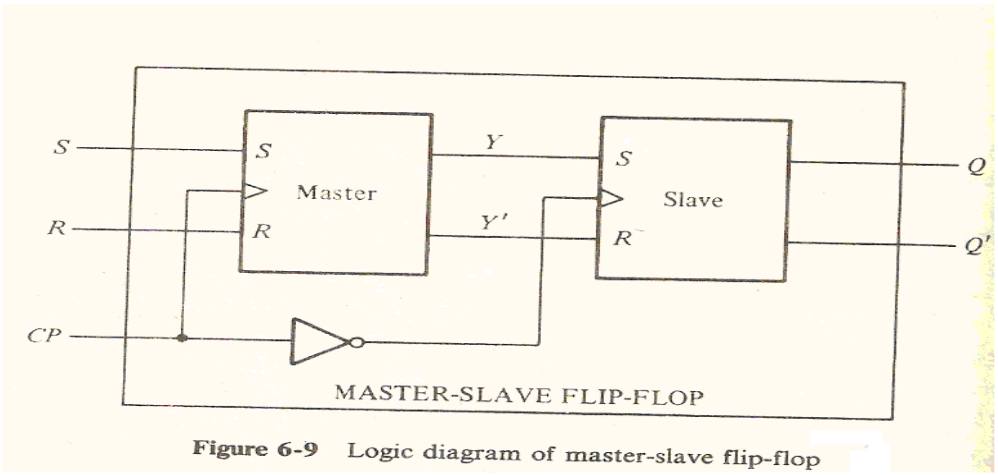


Figure 6-9 Logic diagram of master-slave flip-flop

4.9 REGISTERS

Various types of registers are available in MSI circuits. The simplest possible register is one that consists of only flip-flops without any external gates. Figure 7-1 shows such a register constructed with four *D-type* flip-flops and a common clock pulse input. The clock pulse input, *CP*, enables all flip-flops so that information presently available at the four inputs can be transferred into the 4-bit register. The four outputs can be sampled to obtain the information presently stored in the register.

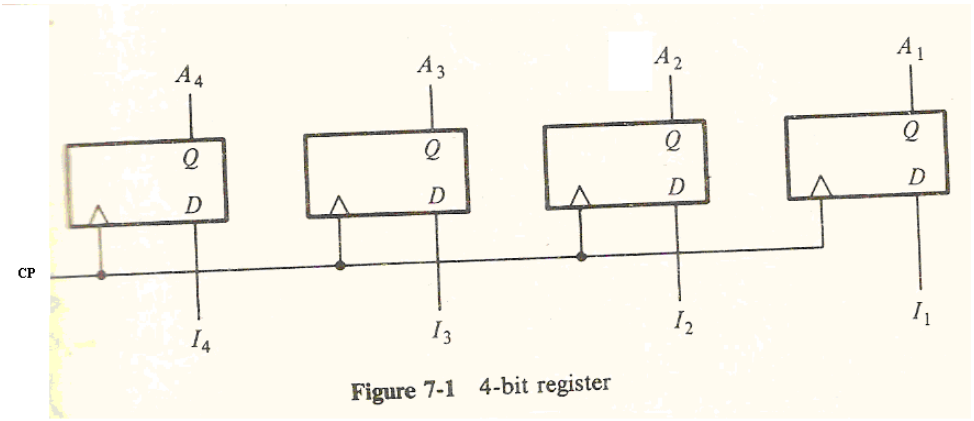


Figure 7-1 4-bit register

The way that the flip-flops in a register are triggered is of primary importance. If the flip-flops are constructed with gated *D -type* latches as in Fig. 6-5, then information present at a data (*D*) input is transferred to the *Q* output when the enable (*CP*) is 1, and the *Q* output follows the input data as long as the *CP* signal remains 1. When *CP* goes to 0, the information that was present at the data input just before the transition is retained at the *Q* output. In other words, the flip-flops are sensitive to the pulse duration, and the register is enabled for as long as *CP = 1*. A register that responds to the pulse duration is commonly called a gated latch, and the *CP* input is frequently labeled with the variable *G* (instead of *CP*). Latches are suitable for use as temporary storage of binary information that is to be transferred to an external destination.

A group of flip-flops sensitive to pulse duration is usually called a latch, whereas a group of flip-flops sensitive to pulse transition is called a register.*

4.9.1 Register with parallel load

The transfer of new information into a register is referred to as loading the register. If all the bits of the register are loaded simultaneously with a single clock pulse, we say that the loading is done in parallel. Clock pulse must be inhibited from the *CP* terminal if the content of the register must be left unchanged. In other words, the *CP* input acts as an enable signal which controls the loading of new information is loaded into the register. If *CP* goes to 1, the input information is loaded into the register. If *CP* remains at 0, the content of the register is not changed. Note that the change of state in the outputs occurs at the positive edge of the pulse. If a flip-flop changes state at the negative edge, there will be a small circle under the triangle symbol in the *CP* input of the flip-flop.

Most digital systems have a master-clock generator that supplies a continuous train of clock pulse. All clock pulses are applied to all flip-flops and registers in the system. The master-clock generator acts like a pump that supplies a constant beat to all parts of the system. A seprate control signal then decides what specific clock pulses will have an effect on a particular register. In such a system, the clock pulses must be ANDed with the control signal, and the output of the AND gate is then applied to the *CP* terminal of the register shown in Fig. 7-1. When the control signal is 0,the output of the AND gate is 0, and the stored information in the register remains unchanged. Only when the control signal is a 1 does the clock pulse pass through the AND gate and into the *CP* terminal for new information to be loaded into the register.

A 4-bit register with a load control input using *RS* flip-flops. The *CP* input of the register receives continuous synchronized pulses which are applied to all flip-flops. The inverter in the *CP* path flip-flops to be triggered by the negative edge of the incoming pulses. The purpose of the inverter is to reduce the loading of the master-clock generator. This is because the CP input is connected to only one gate (the inverter) instead of the four-gate inputs that would have been required if the connections were made directly into the flip-flop clock inputs (marked with small triangles).

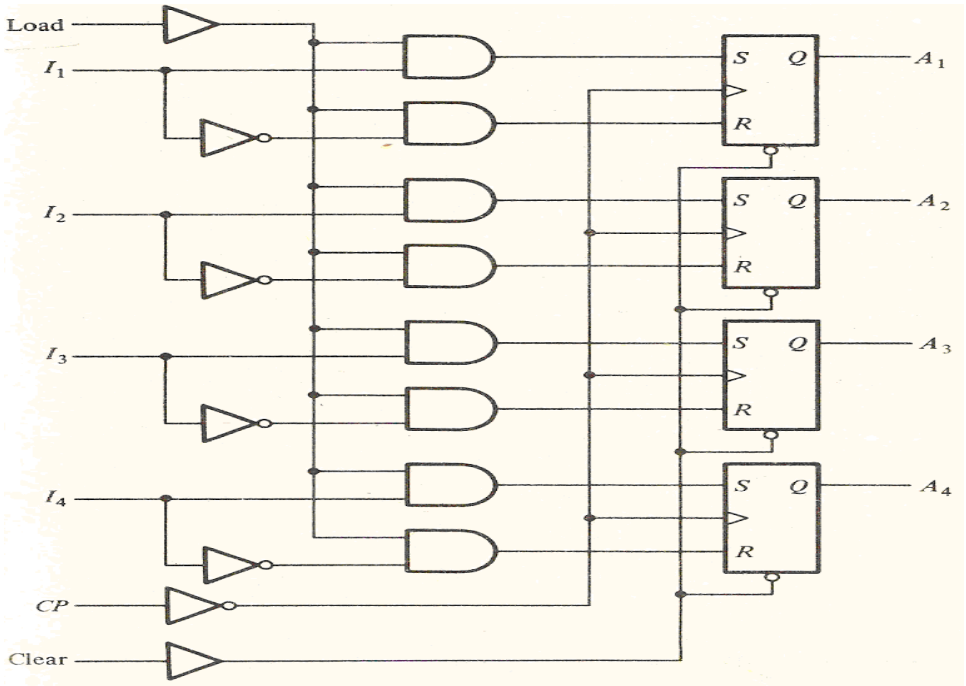


Figure 7-2 4-bit register with parallel load

The clear input goes to a special terminal in each flip-flop through a noninverting buffer gate. When this terminal goes to 0, the flip-flop is cleared asynchronously. The clear input is useful for clearing the register to all 0's prior to its clocked operation. The clear input must be maintained at 1 during normal clocked operations (see Fig. 6-14).

The load input goes through a buffer gate (to reduce loading) and through a series of AND gates to the R and S inputs of each flip-flop.

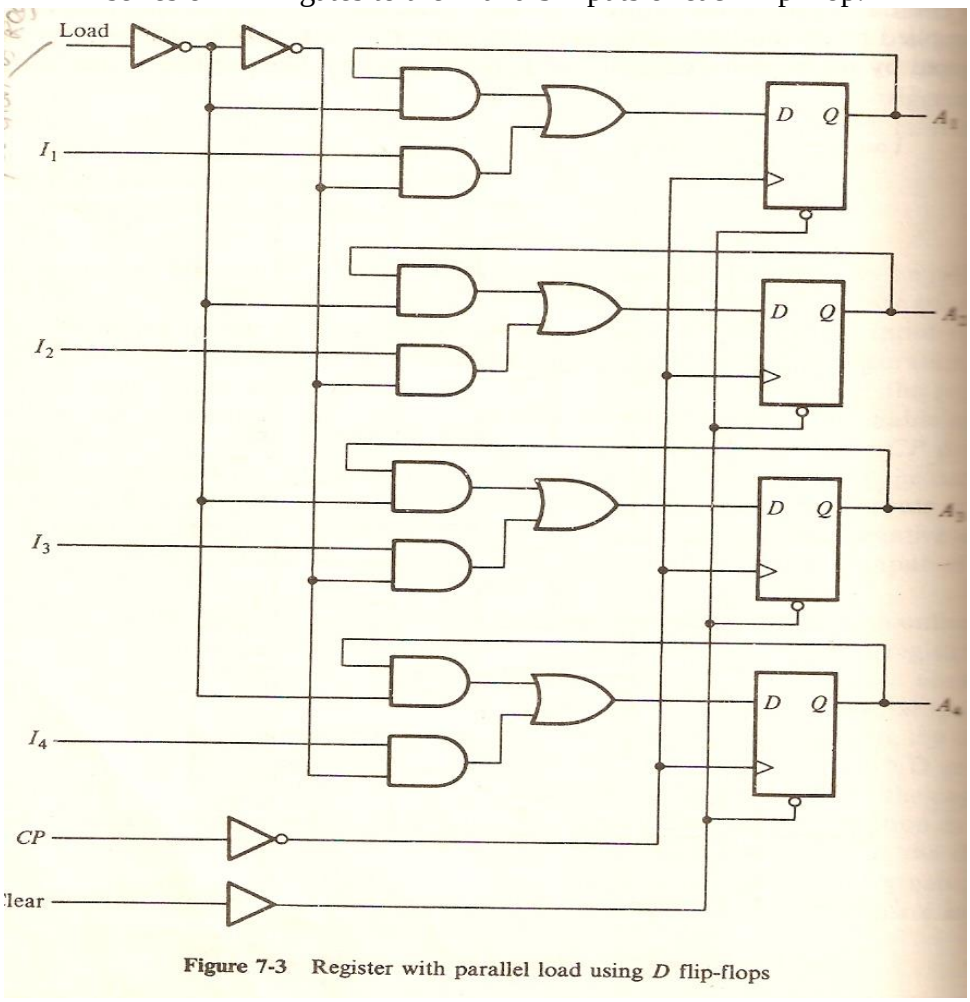


Figure 7-3 Register with parallel load using *D* flip-flops

it is the load input that controls the operation of the register. The two AND gates and the inverter associated with each input *I* determine the values of *R* and *S*. If the load input is 0, both R and S are 0, and no change of state occurs with any clock pulse. Thus, the load input is a control variable which can prevent any information change in the register as long as its input is 0. when the load control goes to 1, inputs I_1 through I_4 specify what binary information is loaded into the register on the next clock pulse. For each *I* that is equal to 1, the corresponding flip-flop inputs are $S = 1$, $R = 0$. For each I that is equal to 0, the corresponding flip-flop inputs are $S = 0$, $R = 1$. Thus, the input value is transferred into the register provided the load input is 1, the clear input is 1, and a clock pulse goes from 1 to 0. This type of transfer is called a parallel-load transfer because all bits of the register are loaded simultaneously. If the buffer gate associated with the load input is changed to an inverter gate, then the register is loaded when the load input is 0 and inhibited when the load input is 1.

A register with parallel load can be constructed with *D* flip-flops as shown in Fig. 7-3. The clock and clear inputs are the same as before. When the load input is 1, the *I* inputs are transferred into the register on the next clock pulse. When the load input is 0, the circuit inputs are inhibited and the *D* flip-flops are reloaded with their present value, thus maintaining the content of the register. The feedback connection in each flip-flop is necessary when *D* type is used because a *D* flip-flop does not have a "no-change" input condition. With each clock pulse, the *D* input determines the next state of the output. To leave the output unchanged, it is necessary to make the *D* input equal to the present *Q* output in each flip-flop.

We saw in chapter 6 that a clocked sequential circuit consists of a group of flip-flops and combinational gates. Since registers are readily available as MSI circuits, it becomes convenient at times to employ a register as part of the sequential

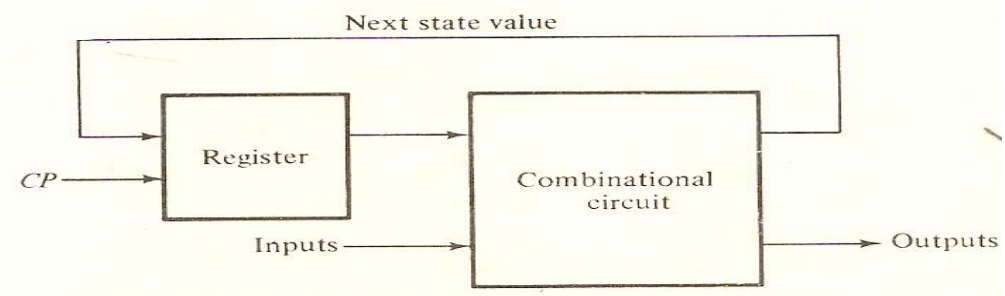


Figure 7-4 Block diagram of a sequential circuit

circuit. A block diagram of a sequential circuit that uses a register is shown in Fig. 7-4. The present state of the register and the external inputs determine the next state of the register and the values of external outputs. Part of the combinational circuit determines the next state and the other part generates the outputs. The next state value from the combinational circuit is loaded into the register with a clock pulse. If the register has a load input, it must be set to 1; otherwise, if the register has no load input (as in Fig. 7-1), the next state value will be transferred automatically every clock pulse.

The combinational circuit part of a sequential circuit can be implemented by any of the methods discussed in chapter 5. It can be constructed with SSI gates, with ROM, or with a programmable logic array (PLA). By using a register, it is possible to reduce the design of a sequential circuit to that of a combinational circuit connected to a register.

EXAMPLE 7-1 Design the sequential circuit whose state table is listed in Fig. 7-5(a).

The state table specifies two flip-flops A_1 and A_2, one input x, and one output y. The next state and output information is obtained directly from the table:

$$A_1(t+1) = \sum(4, 6)$$

$$A_2(t+2) = \sum(1, 2, 5, 6)$$

$$Y(A_1, A_2, x) = \sum(3, 7)$$

The minterm values are for variables A_1, A_2, and x, which are the present state and input variables. The functions for the next state and output can be simplified by means of maps to give:

$$A_1(t+1) = A_1 x'$$

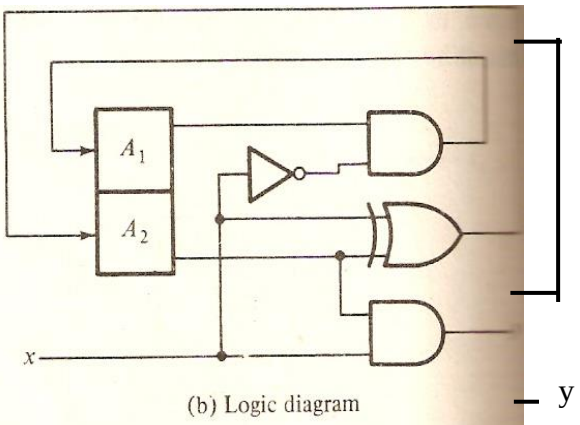
$$A_2(t+1) = A_2 x$$

$$Y = A_2 x$$

The logic diagram is shown in Fig. 7-5 (b).

Present state A_1	A_2	Input x	Next state A_1	A_2	Output y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

(a) State Table



(b) Logic diagram

Figure 7-5 Example of sequential-circuit implementation

ROM truth table

Address 1	2	3	Outputs 1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

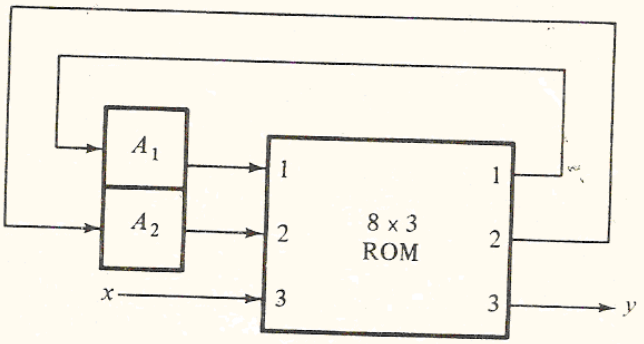


Figure 7-6 Sequential circuit using a register and a ROM

EXAMPLE 7-2 Repeat Example 7-1, but now use a ROM and a register.

The ROM can be used to implement the combinational circuit and the register will provide the flip-flops. The number of inputs to the ROM is equal to the number of flip-flops plus the number of external inputs. The number of outputs

of the ROM is equal to the number of flip-flops plus the number of external outputs. In this case we have three inputs and three outputs for the ROM; so its size must be 8 × 3. The implementation is shown in Fig. 7 – 6. The Rom truth table is identical to the state table with “present state” and “inputs” specifying the address of ROM and “next state” and “outputs” specifying the ROM outputs. The next state values must be connected from the ROM outputs to the register inputs.

4.10 SHIFT REGISTERS

A register capable of shifting its binary information either to the right or to the left is called a shift register. The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive a common clock pulse, which causes the shift from one stage to the next.

The simplest possible shift register is one that uses only flip-flops, as shown in Fig. 7-7. The *Q* output of a given flip-flop is connected to the *D* input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right. The serial input determines what goes into the leftmost flip-flop during the shift. The serial output is taken from the output of the rightmost flip-flop prior to the application of a pulse. Although this register shifts its contents to the right, if we turn the page upside down, we find that the register shifts its contents to the left. Thus a unidirectional shift register can function either as a shift-right or as a shift-left register.

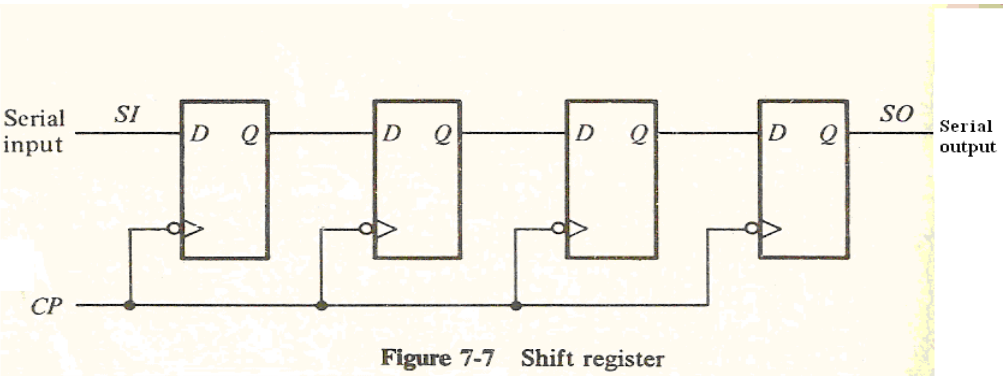


Figure 7-7 Shift register

The register in Fig. 7-7 shifts its contents with every clock pulse during the negative edge of the pulse transition. (This is indicated by the small circle associated with the clock input in all flip-flops.) If we want to control the shift so that it occurs only with certain pulses but not with others, we must control the *CP* input of the register. It will be shown later that the shift operations can be controlled through the *D* inputs of the flip-flops rather than through the *CP* input. If, however, the shift register in Fig. 7-7 is used, the shift can easily be controlled by means of an external AND gate as shown below.

4.10.1 Serial Transfer

A digital system is said to operate in a serial mode when information is transferred and manipulated one bit at a time. The content of one register is transferred to another by shifting the bits from one register to the other. The information is transferred one bit at a time by shifting the bits out of the source register into the destination register.

The serial transfer of information from register *A* to register *B* is done with shift registers, as shown in the block diagram of Fig. 7-8(a). The serial output (*SO*)of register *A* goes to the serial input (*SI*) of register *B*. To prevent the loss of information stored in the source register, the *A* register is made to circulate its information by connecting the serial output to its serial input terminal. The initial content of register *B* is shifted out through its serial output and is lost unless it is transferred to a third shift register. The shift-control input determines when and by how many times the registers are shifted. This is done by the AND gate that allows clock pulses to pass into the *CP* terminals only when the shift-control is 1.

Suppose the shift registers have four bits each. The control unit that supervise the transfer must be designed in such a way that it enables the shift registers, through the shift-control signal, for a fixed time duration equal to four clock pulses. This is shown in the timing diagram of Fig. 7-8(b). The shift-control signal is synchronized with the clock and changes value just after the negative edge of a clock pulse. The next four clock pulses find the shift-control signal in the 1 state, so the output of the AND gate connected to the *CP* terminals produces the four pulses T_1 , T_2 , T_3 , and T_4. The fourth pulse changes the shift control to 0 and the shift registers are disabled.

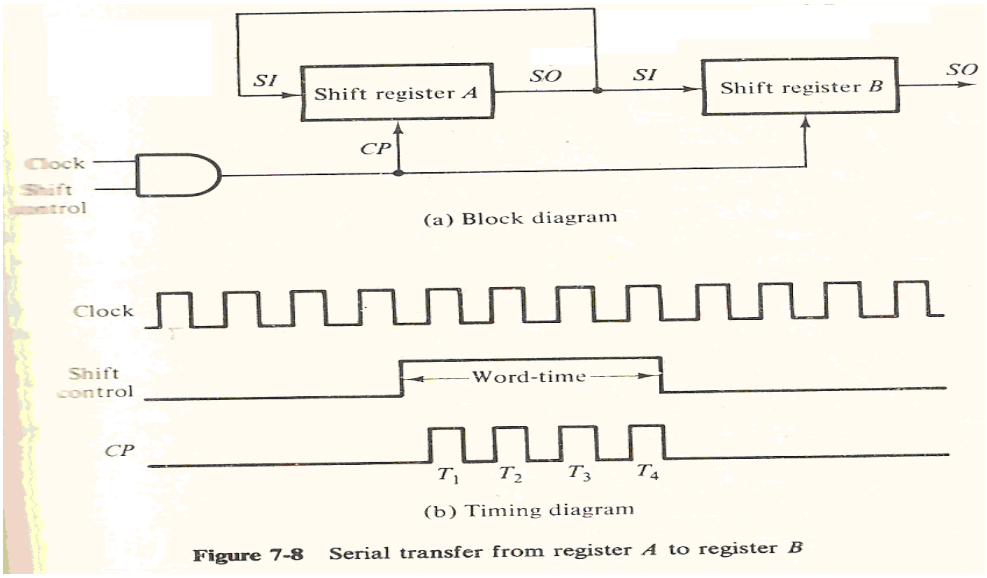


Figure 7-8 Serial transfer from register *A* to register *B*

Assume that the binary content of *A* before the shift is 1011 and that of *B*, 0010. The serial transfer from *A* to *B* will occur in four steps as shown in Table 7-1. After the first pulse T_1, the rightmost bit of *A* is shifted into the leftmost bit of *B* and, at the same time, this bit is circulated into the leftmost

position of *A*. The other bits of *A* and *B* are shifted once the right. The previous serial output from *B* is lost and its value changes from 0 to 1. The next three pulses perform identical operations, shifting the bits of *A* into *B*, one at a time. After the fourth shift, the shift control goes to 0 and both registers *A* and *B* have the value 1011. Thus, the content of *A* is transferred into while the content of *A* remains unchanged.

TABLE 7-1 Serial transfer example

Timing pulse	ft register *A*	Shift register *B*	Serial output of *B*
Initial value	0 1 1	0 0 1 0	0
After T_1	1 0 1	1 0 0 1	1
After T_2	1 1 0	1 1 0 0	0
After T_3	1 1 1	0 1 1 0	0
After T_4	0 1 1	1 0 1 1	1

The difference between serial and parallel modes of operation should be apparent from this example. In the parallel mode, information is available from all bits of a register and all bits can be transferred simultaneously during one clock pulse. In the serial mode, the registers have a single serial input and a single serial output. The information is transferred one bit at a time while the registers are shifted in the same direction.

The time interval between clock pulses is called the *bit time*, and the time required to shift the entire contents of a shift register is called the *word time*.

4.10.2 Bidirectional Shift Register with Parallel Load

Shift registers can be used for converting serial data to parallel data, and vice versa. If we have access to all the flip-flop outputs of a shift register, then information entered serially by shifting can be taken out in parallel from the outputs of the flip-flops. If a parallel load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register.

Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities. The most general shift register has all the capabilities listed below.

1. A clear control to clear the register to 0.
2. A *CP* input for clock pulses to synchronize all operations.
3. A shift-right control to enable the shift-right operation and the serial input and output lines associated with the shift-right.
4. A shift-left control to enable the shift-left operation and the serial input and output lines associated with the shift-left.
5. A parallel-load control to enable a parallel transfer and the n input lines associated with the parallel transfer.

6. Parallel output lines.
7. A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.

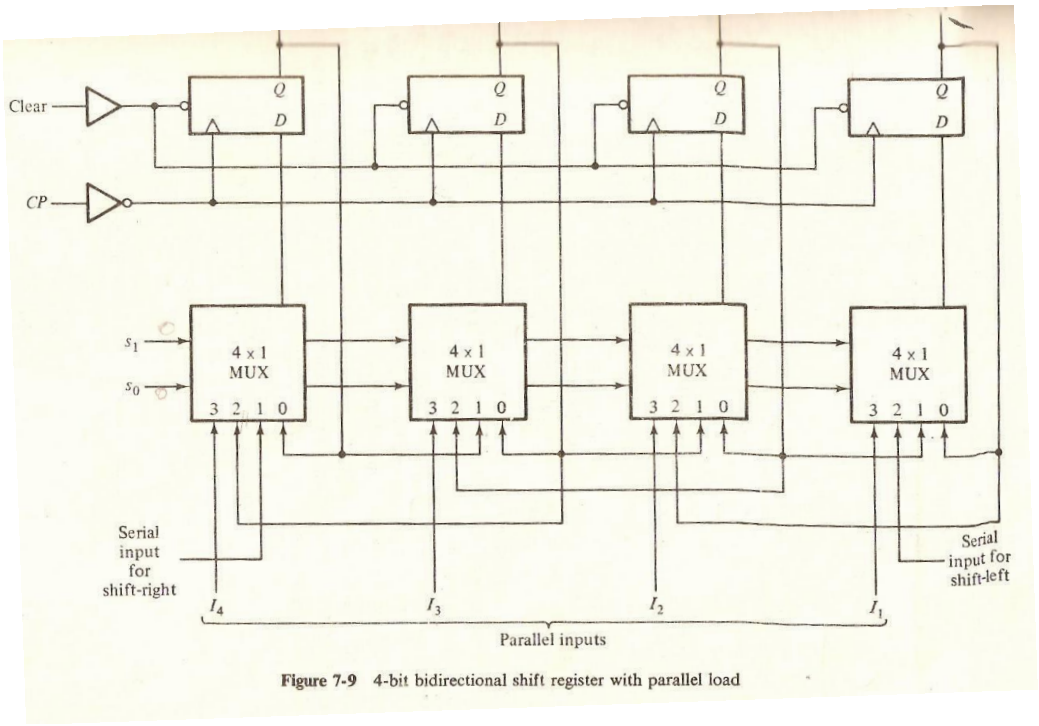


Figure 7-9 4-bit bidirectional shift register with parallel load

A register capable of shifting both right and left is called a *bidirectional shift register*. One that can shift in only one direction is called a *unidirectional shift register*. If the register has both shift and parallel-load capabilities, it is called a *shift register* with *parallel load.*

The diagram of a shift register that has all the capabilities listed above is shown in Fig. 7-9.* It consists of four *D* flip-flops, although *RS* flip-flops could be used provided an inverter is inserted between the *S* and *R* terminals. The four multiplexers (MUX) are part of the register. The four multiplexers have two common selection variables, s_1 and s_0. Input 0 in each MUX is selected when $s_1s_0 = 00$, input 1 is selected when $s_1s_0 = 01$, and similarly for the other two inputs to the multiplexers.

When $s_1s_0 = 00$, the present value of the register is applied to the *D* inputs of the flip-flops. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock pulse transfers into each flip-flop the binary value it held previously, and no change of state occurs.

When $s_1s_0 = 01$, terminals 1 of the multiplexer inputs have a path to the *D* inputs of the flip-flops. This causes a shift-right operation, with the serial input transferred into flip-flop A_4. When $s_1s_0 = 10$, a shift-left operation results, with the other serial input going into flip-flop A_1. Finally, when $s_1s_0 = 11$, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock pulse.

Table 7-2 Function table for the register of Fig. 7-9

Mode control		Register operation
s_1	s_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

*This is similar to IC type 74194.

A bidirectional shift register with parallel load is a general-purpose register capable of performing three operations: shift left, shift right, and parallel load. Not all shift registers available in MSI circuits have all these capabilities. The particular application dictates the choice of one MSI shift register over another.

Serial Addition

Operations in digital computers are mostly done in parallel because this is faster mode of operation. Serial operations are slower but require less equipment. To demonstrate the serial mode of operation.

The two binary numbers to be added serially are stored in two shift registers. Bits are added one pair at a time, sequentially, through a single full-adder (*FA*) circuit. The carry out of the full-adder is transferred to a *D* flip-flop. The output of this flip-flop is then used as an input carry for the next pair of significant bits.

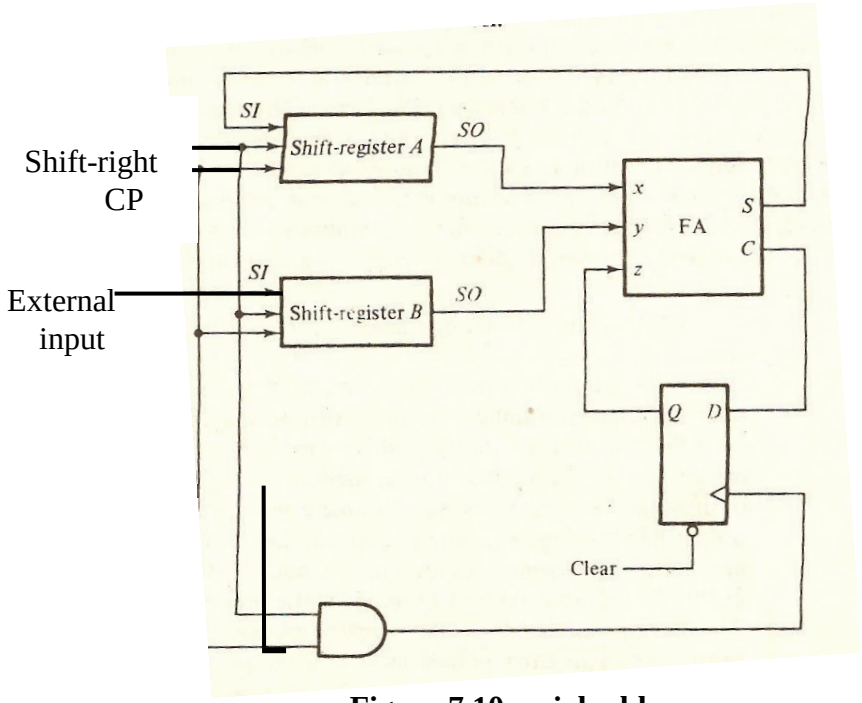


Figure 7.10 serial adder

The operation of the serial adder is as follows. Initially, the A register holds the augend, the B register holds the addend, and the carry flip-flop is cleared to 0. The serial outputs (*SO*) of *A* and *B* provide a pair of significant bits for the full-adder at *x* and *y*. Output *Q* of the flip-flop gives the input carry at *z*. The shift-right control enables both registers and the carry flip-flop; so at the next clock pulse, both registers are shifted once to the right, the sum bit from *S* enters the leftmost flip-flop of *A*, and the output carry is transferred into flip-flop *Q*. The shift-right control enables the registers for a number of clock pulse, a new sum bit is transferred to A, a new carry is transferred to *Q*, and both registers are shifted once to the right. This process continues until the shift-right control is disabled.

4.11 Ripple Counters

MSI counters come in two categories: ripple counters and synchronous counters. In a ripple counter, the flip-flop output transition serves as a source for triggering other flip-flops. In other words, the *CP* inputs of all flip-flops (except the first) are triggered not by the incoming pulses but rather by the transition that occurs in other flip-flops. In a synchronous counter, the input pulses are applied to all *CP* inputs of all flip-flops. The change of state of a particular flip-flop is dependent on the present state of other flip-flops. Synchronous MSI counters are discussed in the next section. Here we present some common MSI ripple counters and explain their operation.

4.11.1 Binary Ripple Counter

A binary ripple counter consists of a series connection of complementing flip-flops (*T* or *JK* type), with the output of each flip-flop connected to the *CP* input of the next higher-order flip-flop. The flip-flop

holding the least significant bit receives the incoming count pulses. The diagram of a 4-bit binary ripple counter is shown in Fig. 7-12. All J and *K* inputs are equal to 1. The small circle in the *CP* input indicates that the flip-flop complements during a negative-going transition or when the output to which it is connected goes from 1 to 0.

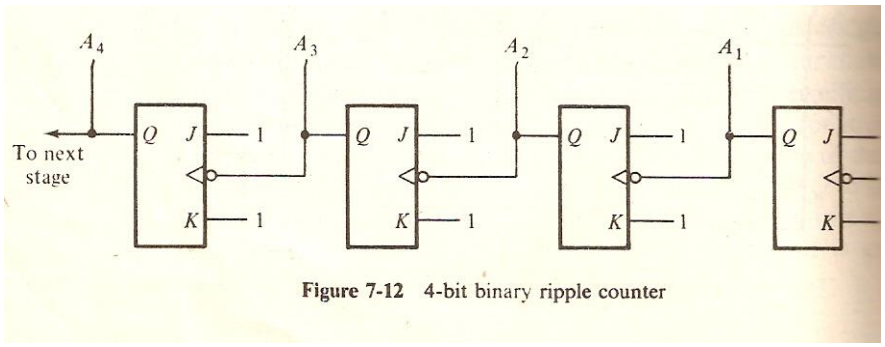


Figure 7-12 4-bit binary ripple counter

Count sequence				Conditions for complementing flip-flops	
A_4	A_3	A_2	A_1		
0	0	0	0	Complement A_1	
0	0	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	0	1	0	Complement A_1	
0	0	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2; A_2 will go from 1 to 0 and complement A_3
0	1	0	0	Complement A_1	
0	1	0	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2
0	1	1	0	Complement A_1	
0	1	1	1	Complement A_1	A_1 will go from 1 to 0 and complement A_2; A_2 will go from 1 to 0 and complement A_3; A_3 will go from 1 to 0 and complement A_4
1	0	0	0	and so on . . .	

To understand the operation of the binary counter, refer to its count sequence given in Table 7-4. It is obvious that the lowest-order bit A_1 must be complemented with each count pulse. Every time A_1 goes from 1 to 0, it complements A_2. Every time A_2 goes from 1 to 0, it complements A_3, and so on. For example, take the transition from count 0111 to 1000. The arrows in the table emphasize the transitions in this case. A_1 goes from 1 to 0, it triggers A_2 and complements it. As a result, A_2 goes from 1 to 0, which in turn complements A_3. A_3 now goes from 1 to 0, which complements A_4. The output transition of A_4, if connected to a next stage, will not trigger the next flip-flop since it goes from 0 to

1. The flip-flops change one at a time in rapid succession, and the signal propagates through the counter in a ripple fashion. Ripple counters are sometimes called *asynchronous counters.*

A binary counter with a reverse count is called a binary down-counter. In a down-counter, the binary count is decremented by 1 with every input count pulse. The count of a 4-bit down-counter starts from binary 15 and continues to binary counts 14 , 13 , 12, . . ., 0 and then back to 15. The circuit of Fig. 7-12

will function as a binary down-counter if the outputs are taken from the complement terminals *Q'* of all flip-flops. If only the normal outputs of flip-flops are available, the circuit must be modified slightly as described below.

A list of the count sequence of a count-down binary counter shows that the lowest-order bit must be complemented with every count pulse. Any other bit in the sequence is complemented if its previous lower-order bit goes from 0 to 1. therefore, the diagram of a binary down-counter looks the same as in Fig. 7-12, provided all flip-flops trigger on the positive edge of the pulse. (The small circles in the *CP* inputs must be absent.) If negative-edge-triggered flip-flops are used, then the *CP* input of each flip-flop must be connected to the *Q'* output of the previous flip-flop. Then when *Q* goes from 0 to 1, *Q'* will go from 1 to 0 and complement the next flip-flop as required.

4.11.2 BCD Ripple Counter

A decimal counter follows a sequence of ten states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits. The sequence of states in a decimal counter is dictated by the binary code used to represent a decimal digit. This is similar to a binary counter, except that the state after 1001 (code for decimal digit 9) is 0000 (code for decimal digit 0).

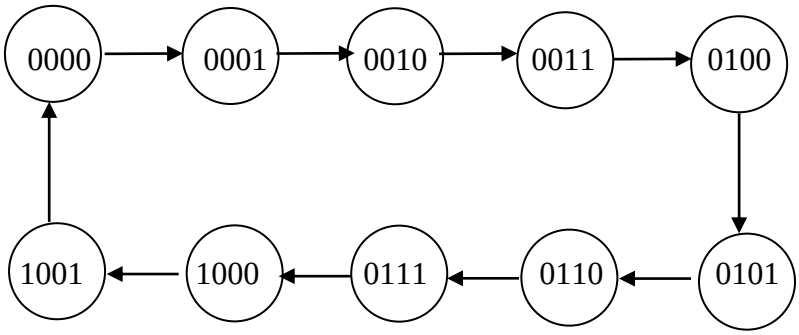


Figure 7-13 State diagram of a decimal BCD counter

The design of a decimal ripple counter or of any ripple counter not following the binary sequence is not a straightforward procedure.

The logic diagram of a BCD ripple counter is shown in Fig. 7-14.* The four outputs are designated by the letter symbol *Q* with a numeric subscript equal to the binary weight of the corresponding bit in the BCD code. The flip-flops trigger on the negative edge, i.e., when the *CP* signal goes from 1 to 0. Note that the output of Q_1 is applied to the *CP* input of Q_4. The J and K inputs are connected either to a permanent 1 signal or to outputs of flip-flops, as shown in the diagram.

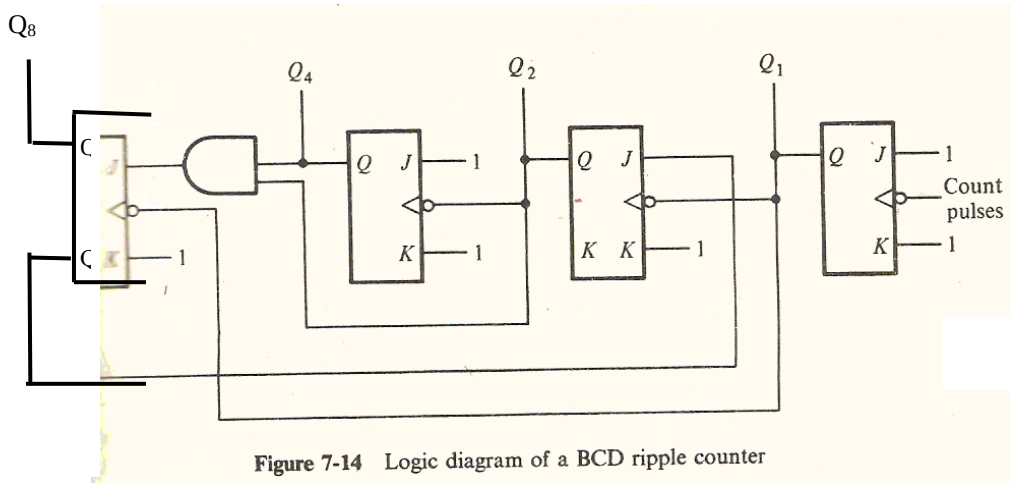
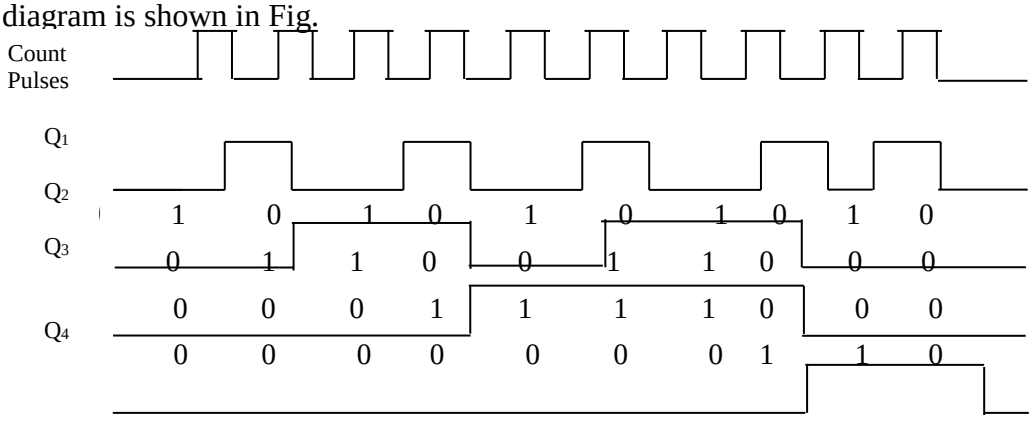


Figure 7-14 Logic diagram of a BCD ripple counter

A ripple counter is an asynchronous sequential circuit and cannot be described by Boolean equations. The *CP* input goes from 1 to 0, the flip-flop is set if $J = 1$, is cleared if $K = 1$, is complemented if $J = K = 1$, and is left unchanged if $J = K = 0$. The following are the conditions for each flip-flop state transition:

1. Q_1 is complemented on the negative edge of every count pulse.
2. Q_2 is complemented if $Q_8 = 0$ and Q_1 goes from 1 to0. Q_2 is cleared if $Q_8 = 1$ and Q_1 goes from 1 to 0.
3. Q_4 is complemented when Q_2 goes from 1 to 0.
4. Q_8 is complemented when $Q_4Q_2 = 11$ and Q_1 goes from 1 to 0. Q_8 is cleared if either Q_4 or Q_2 is 0 and Q_1 goes from 1 to 0.

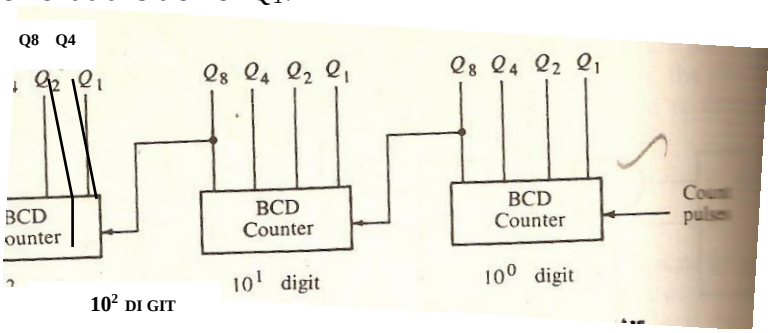
To verify that these conditions result in the sequence required by a BCD ripple counter. Another way to verify the operation of the counter is to derive the timing diagram for each flip-flop from the conditions listed above. This diagram is shown in Fig.



7-15 Timing diagram for the decimal counter of Fig. 7-14

7-15 with the binary states listed after each clock pulse. Q_1 changes state after clock pulse. Q_2 complements every time Q_1 goes from 1 to 0 as long as $Q_8 = 0$. When Q_8 becomes 1, Q_2 remains cleared at 0. Q_4 complements every time

Q_2 and Q_4 become 1's, Q_8 complements when Q_1 goes from 1 to 0. Q_8 is cleared on the next transition of Q_1.



The BCD counter of Fig. 7-14 is a decade counter, since it counts from 0 to 9. To count in decimal from 0 to 99, we need a two-decade counter. To count from 0 to 999, we need a three-decade counter. Multiple-decade counters can be constructed by connecting BCD counters in cascade, one for each decade. A three-decade counter is shown in Fig. 7-16. The inputs to the second and third decades come from Q_8 of the previous decade. When Q_8 in one decade goes from 1 to 0, it triggers the count for the next higher-order decade while its own decade goes from 9 to 0. For instance, the count after 399 will be 400.

4.11.3 Synchronous Counters

Synchronous counters are distinguished from ripple counters in that clock pulses are applied to the *CP* inputs of all flip-flops. The common pulse triggers all the flip-flops simultaneously, rather than one at a time in succession as in a ripple counter. The decision whether a flip-flop is to be complemented or not is determined from the values of the *J* and *K* inputs at the time of the pulse. If $J = K = 0$, the flip-flop remains unchanged. If $J = k = 1$, the flip-flop complements.

A design procedure for any type of synchronous counter was presented in Section 6-8. The design of a 3-bit binary counter was carried out in detail and is illustrated in Fig. 6-30. In this section, we present some typical MSI synchronous counters and explain their operation. It must be realized that there is no need to design a counter if it is already available commercially in *IC* form.

4.11.4 Binary Counter

The design of synchronous binary counters is so simple. In a synchronous binary counter, the flip-flop in the lowest-order position is complemented with every pulse. This means that its *J* and *K* inputs must be maintained at logic-1. A flip-flop in any other position is complemented with a pulse provided all the bits in the lower-order positions are equal to 1, because the lower-order bits (when all 1's) will change to 0's on the next count pulse. The binary count dictates that the next higher-order bit be complemented. For example, if the present state of a 4-bit counter is $A_4A_3A_2A_1 = 0011$, the next count will be 0100. A1 is always complemented. A_2 is complemented because the present state of $A_1 = 1$. A_3 is complemented because the present state of $A_3A_2A_1 = 011$, which does not give an all-1's condition.

Synchronous binary counters have a regular pattern and can easily be constructed with complementing flip-flops and gates. The regular pattern can be clearly seen from the 4-bit counter depicted in Fig.7-17. The *CP* terminals of all flip-flops are connected to a common clock-pulse source. The first stage A1 has its *J* and *K* equal to 1 if the counter is enabled. The other *J* and *K* inputs are equal to 1 if all previous low-order bits are equal to 1 and the count is enabled. The chain of AND gates generates the required logic for the *J* and *K* inputs in each stage. The counter can be extended to any number of stages, with each stage having an additional flip-flop and an AND gate that gives an output of 1 if all previous flip-flop outputs are 1's.

Note that the flip-flops trigger on the negative edge of the pulse. This is not essential here as it was with the ripple counter. The counter could also be triggered on the positive edge of the pulse.

4.11.5 Binary Up-Down Counter

In a synchronous count-down binary counter, the flip-flop in the lowest-order position in complemented with every pulse. A flip-flop in any other position is complemented with a pulse provided all the lower-order bits are equal to 0. For example, if the present state of a 4-bit count-down binary counter is $A_4A_3A_2A_1 = 1100$, the next count will be 1011. $A_1 = 0$. A_3 is complemented because the present state of $A_2A_1 = 00$.

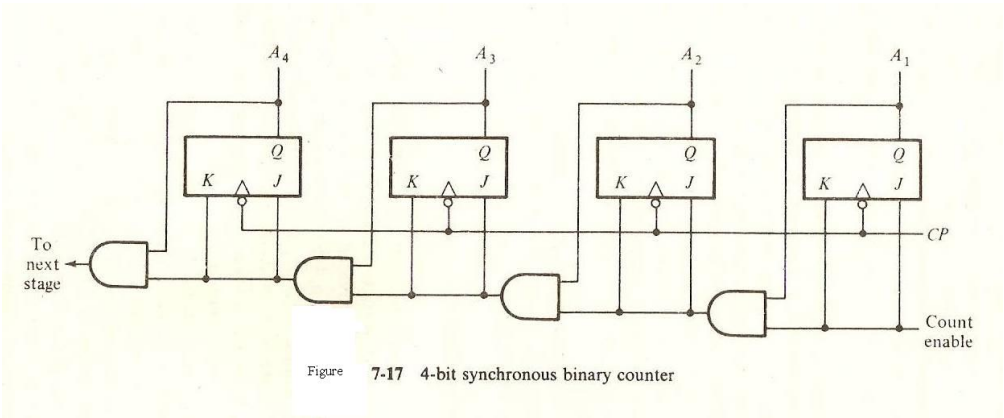


Figure 7-17 4-bit synchronous binary counter

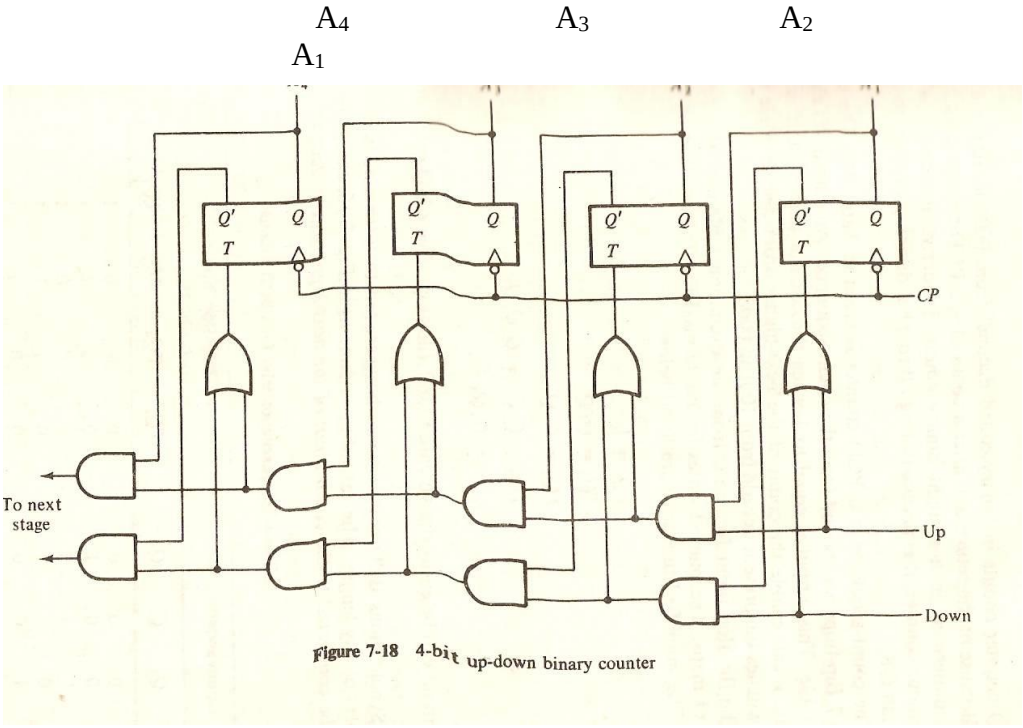


Figure 7-18 4-bit up-down binary counter

But A_4 is not complemented because the present of $A_3A_2A_1 = 100$, which is not an all-0's condition.

A count-down binary counter can be constructed as shown in Fig. 7-17, except that the inputs to the *AND* gates must come from the complement outputs *Q'* and not from the normal outputs *Q* of the previous flip-flops. The two operations can be combined in one circuit. A binary counter capable of counting either up or down is shown in Fig. 7-18. The *T* flip-flops employed in this circuit may be considered as *JK* flip-flops with the *J* and *K* terminals tied together. When the up input control is 1, the circuit counts up, since the *T* inputs are determined from the previous values of the normal outputs in *Q*. when the down input control is 1, the circuit counts down, since the complement outputs *Q'* determine the states of the *T* inputs. When both the up and down signals are 0's, the register does not change state but remains in the same count.

4.11.6 BCD Counter

A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000. Because of the return to 0 after a count of 9, a BCD counter does not have a regular pattern as in a straight binary count. To derive the circuit of a BCD synchronous counter, it is necessary to go through a design procedure as discussed in Section 6-8.

The count sequence of a BCD counter is given in Table 7-5. The excitation for the *T* flip-flops is obtained from the count sequence. An output *y* is also shown in the table. This output is equal to 1 when the counter present state is 1001. In this way, *y* can enable the count of the next-higher-order decade while the same pulse switches the present decade from 1001 to 0000.

The flip-flop input functions from the excitation table can be simplified by means of maps. The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are listed below:

$$TQ_1 = 1$$

$$TQ_2 = Q'_8Q_1$$

$$TQ_4 = Q_2Q_1$$

$$TQ_8 = Q_8Q_1 + Q_4Q_2Q_1$$

$$y = Q_8Q_1$$

The circuit can be easily drawn with four *T* flip-flops, five AND gates, and one OR gate.

Synchronous BCD counters can be cascaded to form a counter for decimal numbers of any length. The cascading is done as in Fig. 7-16, except that output y must be connected to the count input of the next-higher-order decade.

Table 7-5 Excitation table for a BCD counter

Count sequence				Flip-flop inputs				Output carry
Q8	***Q4***	***Q2***	***Q1***	***TQ8***	***TQ4***	***TQ2***	***TQ1***	***Y***
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	1	1

Unit-IV

Self-Assessment Questions

Fill in the blanks:

1. A combinational circuit that performs the addition of 2 bits is called a
2. A _____ is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
3. IC type 74138 is a _____line decoders.
4. An _____has 2^n input lines and n output lines.

True / False:

1. A combinational circuit that performs the addition of two bits is called as half-adder and the addition of three bits is called as full-adder.
2. An encoder is a digital function that produces a reverse operation from that of a decoder.
3. A register is not a group of binary storage cells suitable for holding binary information.
4. The flip-flops hold binary information and the gates control when and how new information is transferred into the register.

Multiple Choice:

1. The type of encoder available in the IC form is called a _____
   a) Property encoder b) priority encoder
   c) Proper encoder d) none of the above

2. A register capable of shifting both right and left is called a _____
   a) Bidirectional shift register b) unidirectional shift register
   c) Tridirectional shift register c) None of the above
3. The time interval between clock pulses is called the _____
   a) Bit time b) Baud time
   c) Word time d) All of the above
4. Shift registers can be used for converting _____
   a) Serial data to parallel data b) Parallel data to serial data
   c) All the above d) None of the above

Questions

1. Explain about Flip-Flops with neat diagram?

2. What is the use of Shift Registers?
3. Explain in detail about encoder & decoder with neat circuit and truth table?
4. What is the function of demultiplexer & multiplexer?
5. Explain in detail about address with neat diagram?
6. Explain in detail about subtractors with truth tables?

Self Assessment Answers:

Fill in the blanks

1. Half adder
2. Decoder
3. 3 to 8
4. Encoder

True OR False

1. True
2. True
3. False
4. True

Multiple Choice

1. (b)
2. (a)
3. (a)
4. (c)

UNIT-V

Computer Design

UNIT-V

Computer Design

5.1 Introduction

The hardware design of a digital computer may be divided into three interrelated phases: system design, logic design, and circuit design. System design is concerned with the specifications and general properties of the system. This task includes the establishment of design objective and design philosophy, the formulation of computer instructions, and the investigation of its economic feasibility. The specifications of the computer structure are translated by the logic designer to provide the hardware implementation of the system. The circuit design specifies the components for the various logic circuits, memory circuits, electromechanical equipment, and power supplies. The computer hardware design is greatly influenced by the software system, which is normally developed concurrently and which constitutes an integral part of the total computer system.

The design process is divided into six phases:

1. The decomposition of the digital computer into registers which specify the general configuration of the system.
2. The specification of computer instructions.
3. The formulation of a timing and control network.
4. The listing of the register-transfer operations needed to execute all computer instructions.
5. The design of the processor section.
6. The design of the control section.

5.2 SYSTEM CONFIGRATION

The configuration of the computer is shown in Fig.11.1. Each block represents a register, expect for the memory unit, the master-clock generator, and the control logic. This configuration is assumed to satisfy the final system structure. In a practical situation, the designer starts with a tentative system configuration and constantly modifies it during the design process. The name of each register is written inside the block, together with a symbolic designation in parentheses.

The master-clock generator is a common clock-pulse source, usually an oscillator, which generates a periodic train of pulses. These pulses are fanned out by means of amplifiers and distributed over the entire system. Each pulse must reach every flip-flop and register at the same time. Phasing delays may be needed intermittently so that the difference in transmission delays may be needed intermittently so that the difference in transmission delays is uniform throughout. The frequency of the pulses is a function of the speed with which the system operates. We shall assume a frequency of 1 megahertz, which gives

one pulse every microsecond. This pulse frequency is chosen for the sake of having a round number and to avoid problems of circuit propagation delays.

The memory unit has a capacity of 4096 words of 16 bits each. This capacity is large enough for meaningful processing. A smaller size may be used if the computer is to be constructed in the laboratory under economic restrictions. Twelve bits of an instruction are needed to specify the address of an operand,

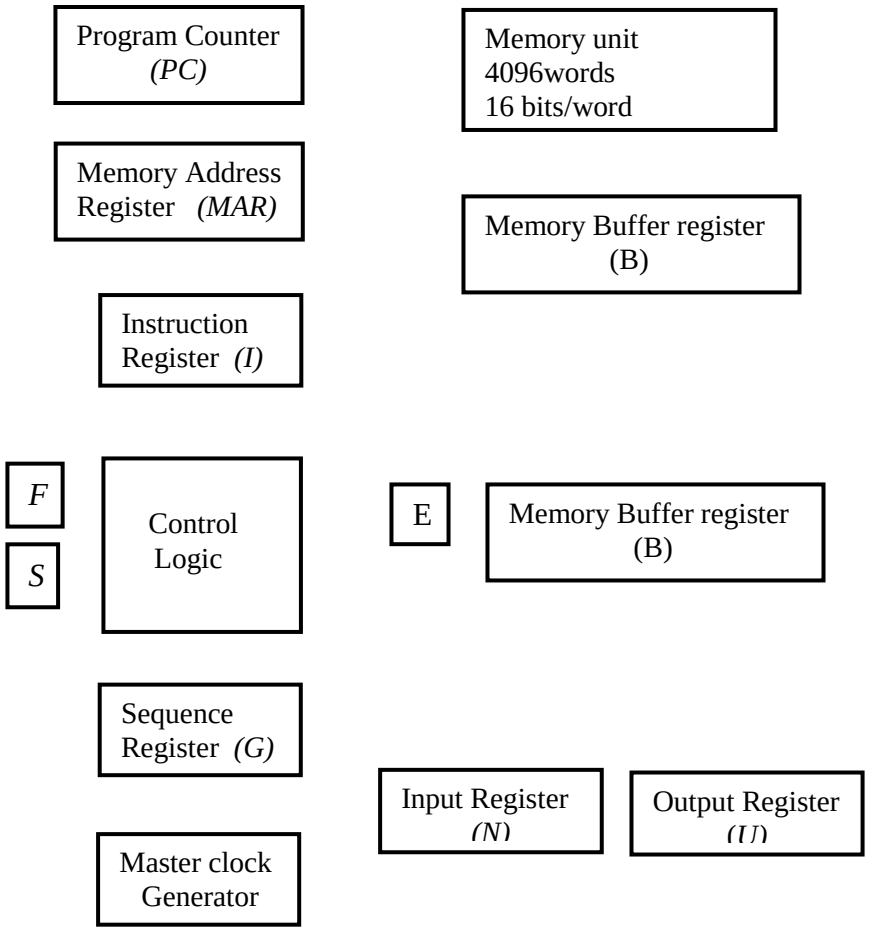


Figure 11.1 Block diagram of digital computer

Table 11-1 List of registers for computer

Symbolic designation	Name	Number of Bits	Function
A	Accumulator register	16	Processor register
B	Memory buffer register	16	Holds contents of memory word
PC	Program counter	12	Holds address of next instruction
MAR	Memory address register	12	Holds address of memory word
I	Instruction register	4	Holds current operation-code
E	Extension register	1	Accumulator extension
F	Fetch flip-flop	1	Controls fetch and execute cycles
S	Start-stop flip-flop	1	Starts and stops computer
G	Sequence register	2	Provides timing signals
N	Input register	9	Holds information from input device
U	Output register	9	Holds information from output device

Which leaves four bits for the operation part of the instruction. The access time of the memory is assumed to be less than 1 microsecond so that a word can be read or written during the interval between two clock pulses.

The part of the digital computer to be designed is decomposed into register subunits. The following paragraphs explain why each register is needed and what function it performs. A list of the registers and a brief description of their functions is presented in Table 11-1. Registers that hold memory words are 16 bits long. Those that hold an address are 12 bits long. Other registers have different numbers of bits, depending on their function.

5.2.1 Memory Address and Memory Buffer Registers

The memory address register, MAR, is used to address specific memory locations. MAR is loaded from *PC* when an instruction is to be read from memory, and from the 12 least significant bits of the *B* register when an operand is to be read from memory. Memory buffer register *B* holds the word read from or written into memory. The operation part of an instruction word placed in *B* is transferred into the *I* register, and the address part is left in the *B* register for transfer to MAR. An operand word placed in the *B* register is accessible for operation with the *A* register. A word to be stored in memory must be loaded into the *B* register before a write operation is initiated.

5.2.2 Program Counter

Program counter *PC* holds the address of the next instruction to be read from memory. This register goes through a step-by-step counting sequence and causes the computer to read successive instructions previously stored in memory. When the program calls for a transfer to another location or for skipping the next instruction in sequence, the *PC* is modified accordingly, causing the program to continue from a memory location out of the counting sequence. To read an instruction, the contents of *PC* are transferred to MAR and a read operation is initiated. The program counter is always incremented by 1 while a memory write operation reads the present instruction. Therefore, the address of the next instruction, one higher than the one presently being executed in the processor, is always available in *PC*.

5.2.3 Accumulator Register

Accumulator register *A* is a processor register that operates on data previously stored in memory. This register is used to execute most instructions and for accepting data from the input device or transferring data to the output device. The *A* register, together with the *B* register, makes up the bulk of the processor unit for the computer. Although most data processing systems include more registers for the computer. Although most data processing systems include more registers for the processor unit, we have chosen to include only one accumulator here in order most to complicate the design. With a single accumulator as the arithmetic element, it is possible to implement only the add operation. Other arithmetic operations such as subtraction, multiplication, and division must be implemented with a sequence of instructions that form a subroutine.

5.2.4 Instruction Register

Instruction register *I* holds the operation-code bits of the current instruction. This register has only four bits since the operation-code of instructions is four bits long. The operation-code bits are transferred to the *I* register from the *B* register, while the address part of the instruction is left in *B*. The operation-code part must be taken out of the *B* register because an operand read from memory into the *B* register will destroy the previously held instruction. The operation part of the instruction is needed by the control to determine what is to be done to the operand just read.

5.2.5 Sequence Register

Sequence register *G* is a counter that produces the timing signals for the computer. The *G* register is decoded to supply four timing variables for the control unit. The timing variables, together with other control variables, produce the control functions that initiate all the micro operations for the computer.

5.2.6 E,F and *S* Flip-flops

Each of these flip-flops is considered a one-bit register. The *E* flip-flop is an extension of the *A* register. It is used during shifting operations, receives

the end carry during addition, and otherwise is a useful flip-flop that can simplify the data processing capabilities of the computer. The *F* flip-flop distinguishes between the fetch and execute cycles. When *F* is 0, the word read from memory is treated as an instruction. When *F* is 1, the word is treated as an operand. *S* is a start-stop flip-flop that can be cleared by program control and manipulated manually. When *S* is 1, the computer runs according to a sequence determined by the program stored in memory. When *S* is 0, the computer stops its operation.

5.2.7 Input and Output Registers

The input-output (I/O) device is not shown in the block diagram of Fig. 5.1. It is assumed to be a teletypewriter unit with a keyboard and a printer. The teletypewriter sends and receives serial information. Each quantity of information has 8 bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register. The serial information for printer is stored in the output register. These two registers communicate with the teletypewriter serially and with the accumulator register in parallel.

Input register N consists of nine bits. Bits 1 through 8 hold alphanumeric input information; bit 9 is a control bit called an input flag. The flag bit is set when a new character is available from the input device and cleared when the character is accepted by the computer. The flag bit is needed to synchronize the slow rate by which the input device operates compared to the high-speed circuits in the computer. The process of information transfer is as follows. Initially, the flag bit in N_9 is cleared. When a key is struck on the keyboard, an 8-bit code is shifted into the input register $(N_1\text{-}N_8)$. As soon as the shift operation is completed, the flag bit in N_9 is set to 1. The computer checks the flag bit; if it is 1, the character code from the *N* register is transferred in parallel into the A register and the flag bit is cleared. Once the flag is cleared, a new character can be shifted into the *N* register by striking another key.

Output register *U* works in a similar fashion, but the direction of information flow is reserved. Initially, the output flag in U_9 is set to 1. The computer checks the flag bit; if it is 1,a character code from the *A* register is transferred in parallel to the output register $(U_1\text{-}U_8)$ and the flag bit U_9 is cleared to 0. The output device accepts the coded information and prints the corresponding character; when the operation is completed, it sets the flag bit to 1. The computer does not load a new character into the output register when the flag is 0, because this condition indicates that the output device is in the process of printing the previous character.

5.3 Computer Instructions

The number of instructions available in a computer and their efficiency in solving the problem at hand are a good indication of how well the system designer foresaw the intended application of the machine. Medium- to large-scale computing systems may have hundreds of instructions, while most small computers limit the list to less than 100. The instructions must be chosen carefully to supply sufficient capabilities to the system for solving a wide range

of data processing problems. The minimum requirements of such a list should include a capability for storing and loading words from memory, a sufficient set of arithmetic and logic operations, some address-modification capabilities, unconditional branching and branching under test conditions, register manipulation capabilities and I/O instructions. The instruction list chosen for our computer is believed to be close to the absolute minimum required for a restricted but practical data processor.

The formulation of a set of instructions for the computer goes hand in hand with the formulation of the formats for data and instruction words. A memory word consists of 16 bits. A word may represent either a unit of data or an instruction. The formats of data words are shown in Fig. 11-2. Data for arithmetic operations are represented by a 15-bit binary number, with the sign in the 16^{th} bit position. Negative numbers are assumed to be in their 2's-complement equivalent. Logical operations are performed on individual bits of the word, with bit 16 treated as any other bit. When the computer communicates with the I/O device, the

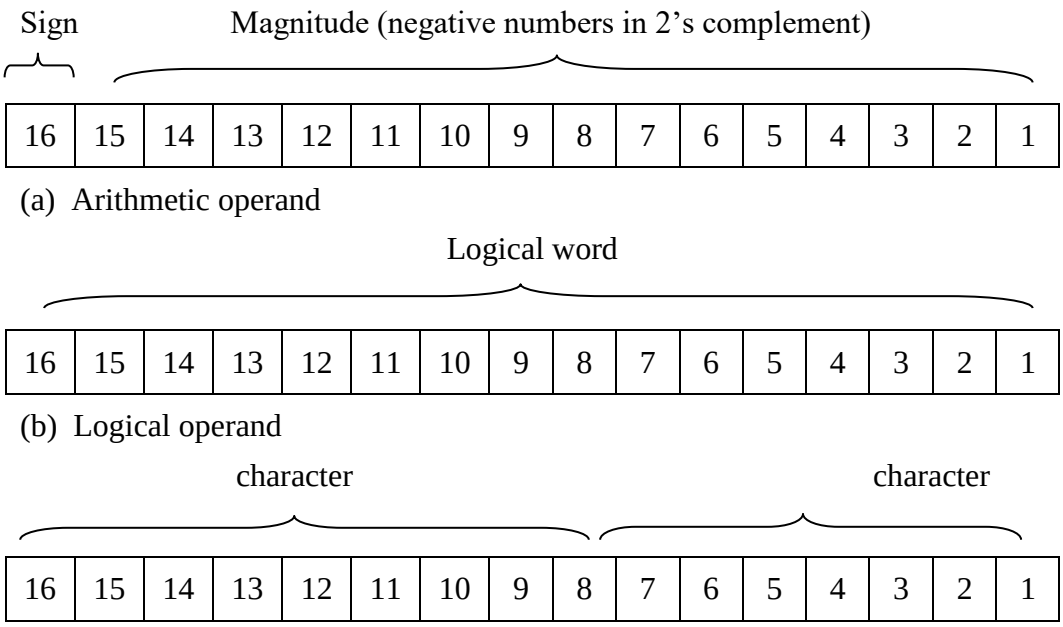
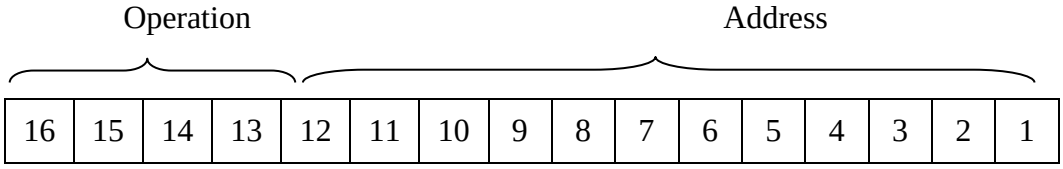


Figure 11-2 Data formats

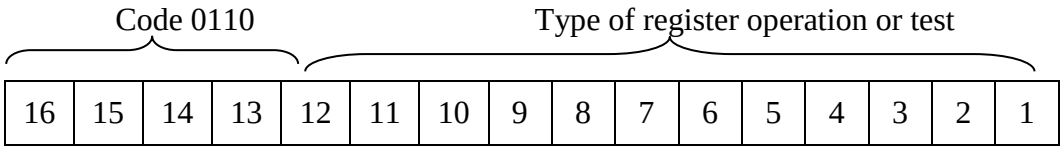
Information transferred is considered to be 8-bit alphanumeric characters. Two such characters can be accommodated in one computer word.

The formats of instruction words are shown in Fig. 11-3. The operation part of the instruction contains four bits; the meaning of the remaining 12 bits depends on the operation-code encountered. A memory-reference instruction

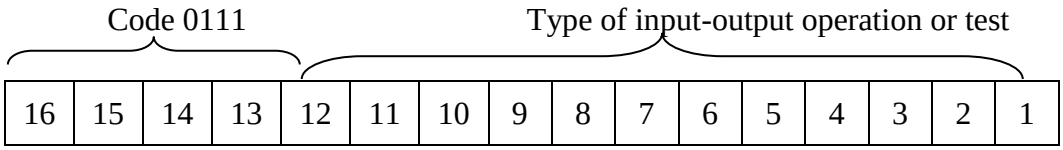
uses the remaining 12 bits to specify an address. A register-reference instruction implies an



(a) Memory-reference instruction



(b) Register-reference instruction



(c) Input/output instruction

Figure 11-3 Instruction formats

Operation on, or a test of, the *A* or *E* register. An operand from memory is not needed; therefore, the 12 least significant bits are used to specify the operation or test to be executed. A register-reference instruction is recognized by the code 0110 in the operation part. Similarly, an *input-output* instruction does not need a reference to memory and is recognized by the operation code 0111. The remaining 12 bits are used to specify the particular device and the type of operation or test performed.

Only four bits of the instruction are available for the operation code. It would seem, then, that the computer is restricted to a maximum of 16 distinct operations. However, since register-reference and input-output instructions use the remaining 12 bits as part of the operation-code, the total number of instructions can exceed 16. In fact, the total number of instructions chosen for the computer is 22.

Of the 16 distinct operations that can be formulated with four bits, only eight have been utilized by the computer because the leftmost bit of all instructions (bit 16) is always a 0. This leaves open the possibility of adding new instructions and extending the computer capabilities if desired.

The six memory-reference instructions for the computer are listed in Table 11-2. The symbolic designation is a three-letter word and represents a abbreviation intended for use by programmers and users when writing symbolic programs for the computer. The hexadecimal code listed is an equivalent

hexadecimal number of the binary code adopted for the operation-code. A memory-reference instruction uses one hexadecimal digit (4 bits) for the operation-code; the remaining three hexadecimal digits (12 bits) of the instruction represent an address designated by the letter *m.* Each instruction has a brief word description and is specified more precisely in the function column with a macrooperation statement. A further clarification of each instruction is given below, together with an explanation of its use.

Table 11-2 Memory-reference instruction

Symbol	Hexadecimal code	Description	Function
AND	0 *m* *	AND to *A*	$A \leftarrow A \wedge M^*$
ADD	1 *m*	Add to *A*	$A \leftarrow A + M,\ E \leftarrow Carry$
STO	2 *m*	Store in *A*	$M \leftarrow A$
ISZ	3 *m*	Increment and skip if zero	$M \leftarrow M+1,\ if\ (M+1=0)\ then\ (PC \leftarrow PC+1)$
BSB	4 *m*	Branch to subroutine	$M \leftarrow PC+5000,\ PC \leftarrow m.+1$
BUN	5 *m*	Branch unconditionally	$PC \leftarrow m$

**m* is the address part of the instruction. *M* is the memory word addressed by *m.*

5.3.1 AND to A

This is a logic operation that performs the AND operation on corresponding pairs of bits in A, with the memory word M specified by the address part of the instruction. The result of the operation is left in register A, replacing its previous contents. Any computer must have a basic set of logic operations for manipulating nonnumerical data. The most common logic operations found in computer instructions are AND,OR, exclusive-OR, and complement. Here we use only the AND and complement. The latter is included as a register-reference instruction. These two logic operations constitute a minimal set from which all other logic operations can be derived, because together the AND and the complement perform a NAND operation. In section 4-7 we saw that this is a universal operation from which any other logic operation can be obtained.

5.3.2 ADD to A

This instruction adds the contents of the memory word M, specified by the address part of the instruction, to the present contents of register A. The addition is done assuming that negative numbers are in their 2’s-complement form. This requires that the sign bit be added I the same way as all other bits are added. The end-carry out of the sign-bit position is transferred to the E flip-flop. This instruction, together with the register-reference instructions, is sufficient for writing programs to implement all other arithmetic operations. Subtraction is achieved by complementing and incrementing the subtrahend. Multiplication is achieved by adding and shifting. The increment and shift are register-reference instructions.

The ADD instruction must be used for loading a word from memory into the A register. This is done by first clearing the *A* register with the register-reference instruction CLA (defined in Table 11-3). The required word is then loaded from memory by adding it to the cleared *A* register.

5.3.3 STORE in A

This instruction stores the contents of the *A* register into the memory word specified by the instruction address. The first three memory-reference instructions are used to manipulate data between memory words and the *A* register. The next three instructions are control instructions that cause in normal program sequence.

5.3.4 Increment and Skip if Zero(ISZ)

The increment-and-skip instruction is useful for address modification and for counting the number of times a program loop is executed. A negative number previously stored in memory at address m is read by the ISZ instruction. This number is incremented by 1 and stored back into memory. If, after it is incremented, the number reaches 0, the next instruction is skipped. Thus, at the end of a program loop, one inserts an ISZ instruction followed by a branch unconditionally (BUN) instruction to the beginning of the program loop. If the stored number does not reach 0, the program returns to execute the loop again. If it reaches 0, the next instruction (BUN) is skipped and the program continues to execute instructions after the program loop.

5.3.5 Branch Unconditionally (BUN)

This instruction transfers control unconditionally to the instruction at the location specified by the address part *m*. Remember that the program counter holds the address of the next instruction to be read and executed. Normally, the *PC* is incremented to give the address of the next instruction in sequence. The programmer has the prerogative of specifying any other instruction out of sequence by using the BUN instruction. This instruction tells the computer to take the address part m and transfer it into *PC*. The address of the next instruction to be executed is now in *PC* and is the one which was previously the address part of the BUN instruction.

The BUN instruction is listed with the memory-reference instructions because it needs an address part m. However, it does not need a reference to memory to access a memory word (designated by the symbol *M*), as is required by the other memory-reference instructions.

5.3.6 Branch to Subroutine

This instruction is useful for branching to a subroutine portion of a program. When executed, the instruction stores the address of the next instruction in sequence which is presently held in *PC* (called the return address) into the memory word specified by the address part of the instruction. It also stores the operation code of BUN (hexadecimal 5) in the same memory location. The contents of the address part m plus 1 are transferred into *PC* to start executing the subroutine program at this location. After the subroutine is

executed, control is transferred back to the calling program by means of a BUN instruction placed at the end of the subroutine.

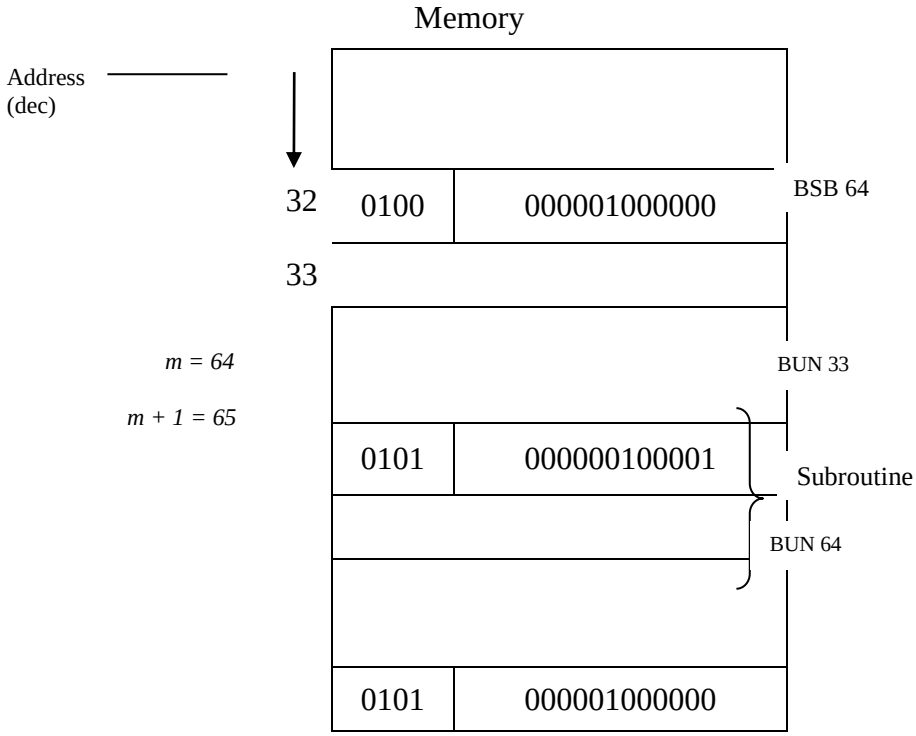


Figure 11-4 Demonstration of branch-to-subroutine instruction

The process of branching to a subroutine and the return to the calling program is demonstrated in Fig. 11-4 by means of a specific numerical example. The calling program is now in location 32. The subroutine program starts at location 65. The BSB instruction causes a transfer to the subroutine, and the last instruction in the subroutine causes a branch back to location 33 in the calling program. The numerical example in Fig. 11-4 shows a BSB instruction in location 32 with an address part m equal to binary 64. While this instruction is being executed, PC holds the address of the next instruction in sequence, which is 33.

The BSB instruction performs the macrooperation (see Table 11-2):

$$M \quad \leftarrow PC + 5000, PC \quad \leftarrow m + 1$$

The contents of *PC* plus hexadecimal 5000 (code for BUN) are transferred into location 64. This transfer produces an instruction BUN 33. The address part of the instruction is incremented and placed in *PC*. *PC* now holds the binary equivalent of 65, so the computer starts executing the subroutine at this location. The last instruction in the subroutine is BUN 64. When this instruction is executed, control is transferred to the instruction in location 64. But in address 64, there is now an instruction that branches back to address 33. The address stored in location 64 by the BSB instruction will always have the proper return address no matter where the BSB instruction is located. In this way, the subroutine return is always to a location one higher than the location

of the BSB instruction. Note that the address number of BUN instruction placed at the end of the subroutine must always be equal to the address number where the return address is temporarily stored, which is 64 in this case.

5.3.7 Register-reference Instructions

The 12 register-reference instructions for the computer are listed in Table 11-3. Each register-reference instruction has an operation code 0110 (hexadecimal 6) and contains a single 1 in one of the remaining 12 bits of the instruction. These

Table 11-3 Register-reference instructions

Symbol	Hexadecimal code	Description	Function
CLA	6800	Clear *A*	*A? 0*
CLE	6400	Clear *E*	*E* ← *0*
CMA	6200	Complement *A*	*A* ← *A*
CME	6100	Complement *E*	*E* ← *E)*
SHR	6080	Shift-right *A* and *E*	*A* ← *shr A*, A_{16} ← *E*, *E* ← A_1
SHL	6040	Shift-left *A* and *E*	*A* ← *shl A*, A_1 ← *E*, *E* ← A_{16}
INC	6020	Increment *A*	*A* ← *A* + *1*
SPA	6010	Skip on positive *A*	*If*(A_{16}=*0*)*then*(*PC* ← *PC* + *1*)
SNA	6008	Skip on negative *A*	*If*(A_{16}=*1*)*then*(*PC* ← *PC* + *1*)
SZA	6004	Skip on zero *A*	*If*(*A*=*0*)*then*(*PC* ← *PC* + *1*)
SZE	6002	Skip on zero *E*	*If*(*E*=*0*)*then*(*PC* ← *PC* + *1*)
HLT	6001	Halt computer	*S* ← *0*

instructions are specified with four hexadecimal digits which represent all 16 bits of an instruction word. The first seven instructions perform an operation on the *A* or *E* register and are self-explanatory. The next four are skip instructions used for program control, conditioned on certain status bits. To skip the next instruction, the *PC* is incremented by 1 once again. The first increment occurs when the present instruction is read. In this way, the next instruction read from memory is two locations up from the location of the present (skip) instruction.

The status bits for the skip instructions are the sign bit in *A*, which is in flip-flop A_{16}, and a zero condition for *A* or *E*. If the designated status condition is present, the next instruction in sequence is skipped; otherwise, the computer continues from the next instruction in sequence because *PC* is not incremented.

The halt instruction is usually placed at the end of a program if one wishes to stop the computer. Its execution clears the start-stop flip-flop, which prevents further operations.

5.3.8 Input-Output Instructions

The computer has four input-output instructions and they are listed in Table 11-4. These instructions have an operation code 0111 (hexadecimal 7), and each contains a 1 in only one of the remaining 12 bits of the instruction word. The input-output instructions are specified with four hexadecimal digits starting with 7.

The INP instruction transfers the input character from *N* to *A* and also clears the input flag in N_9. The OUT instruction transfers an 8-bit character code from *A* into the output register and also clears the output flag in U_9. The two skip instructions check the corresponding status flags and cause a skip of the next

Table11-4 Input-Output instructions

Symbol	Hexadecimal code	Description	Function
SKI	7800	Skip on input flag	$If(N_9=1)then(PC \leftarrow PC+1)$
INP	7400	Input to *A*	$A_{1-8} \leftarrow N_{1-8}, N_9 \leftarrow 0$
SKO	7200	Skip on output flag	$If(U_9=1)then(PC \leftarrow PC+1)$
OUT	7100	Output from *A*	$U_{1-8} \leftarrow A_{1-8}\ U_9 \leftarrow 0$

Instruction if the flag bit is 1. The instruction that is skipped is normally a BUN instruction. The BUN instruction is not skipped if the flag bit is 0; this causes a branch back to the skip instruction to check the flag again. If the flag bit is 1, the BUN instruction is skipped and an input or output operation is executed. Thus, the computer stays in a two-instruction loop (skip on flag and branch back to previous instruction) until the flag bit is set by the external device. The next instruction in sequence must be an input or output instruction.

5.4 Design Of Computer Registers

The design of a synchronous digital system follows a prescribed procedure. From a knowledge of the system requirements, one formulates a control network and obtains a list of register-transfer operations for the system. Once this list is derived, the rest of the design is straightforward. Some installations utilize computer design automation techniques for translating the register-transfer statements to a circuit diagram composed of integrated circuits.

Section 11-5 specified the register-transfer statements for the computer in five separate tables. The entries in the tables consist of control functions and microoperations. The list of control functions provides the Boolean functions for the gates in the control logic network. The list of microoperations gives an indication of the types of registers that must be chosen for the computer. Although these tables are sufficient to complete the logic design of the system, it may be convenient to rearrange the information in the tables in a more convenient way during the actual implementation process.

5.4.1 Register Operation

To determine the type of control input that must be provided in each register, we must obtain the list of microoperations that affect each register separately. This can be done by scanning the tables in Section 11-5 and retrieving all those statements that change the contents of a particular register. This also applies to the read and write operations in the memory unit. For example, a memory-read operation is symbolized with the microoperation:

$$B \quad M \leftarrow$$

The statement also indicates that the contents of register B will change in value. This statement is found twice in the list of microoperations. In Table 11-5, we find it with control function $F't_1$, and in Table 11-6, with control function $F(q_0 + q_1 + q_3)t_1$. Since both control functions produce the same operation, they can be combined with an OR into one statement:

$$R = F't_1 + F(q_0 + q_1 + q_3)t_1: \; B \leftarrow M$$

The symbol *R* is used for convenience to designate the read operation with a single Boolean control variable. The equal sign after *R* designates its equality with the control function listed.

This procedure is repeated for the memory-write operation and for all the registers in the computer. The result is as shown in Table 11-10. Each control function listed in the table is assigned a control-variable name. The single-letter variable names are not necessary, but they help shorten the algebraic expressions of input control for the registers. In most cases, the control variable is assigned a lowercase letter identical to the capital letter reserved to symbolize the corresponding register. The control variables common to the same register are distinguished by different numerical subscripts.

Table 11-10 is derived directly from Tables 11-5 through 11-9. The register to which a microoperation belongs is recognized by the presence of its symbol on the left side of the arrow. To recognize the microoperations belonging to register A, we scan the operations listed in Tables 11-5 through 11-9 and retrieve all those that have an A as a destination register. The microoperation occurs more than once, the corresponding control functions are ORed to form a composite control function.

The operations for the *E* flip-flop must be separated from the operations for the *A* register, even though they were listed together in the previous tables. The circular shift-right operation, for example, is stated in Table 11-8 as:

$$rB_8 : A \leftarrow shr\ A, A_{16} \leftarrow E, E \leftarrow A_1$$

Note that r is a variable equal to q_6t_3, and rB_8 is assigned a control variable a_5. In Table 11-10 under the *A* register, we have:

$$a_5 = rB_8: A \leftarrow shr\ A, \quad A_{16} \leftarrow E$$

which is the part of the shift operation that changes the contents of A. Under the E flip-flop, we have:

a5 = rB8: $E \leftarrow A_1$

which shows the part of the shift operation that changes the *E* flip-flop. Thus, the shift-right control variable *a5* shifts the contents of *A* to the right and inserts the value of *E* into the leftmost bit of *A*. it also transfers the rightmost bit of *A* into *E*. The sequence register *G* does not have any listed microoperations in the previous tables. This register is shown in Fig. 11-6 to be a counter whose clock pulses are enabled by the start-stop flip-flop *S*. This is included in Table 11-10 with the statement: ~~$S:$~~ $G \quad G + 1$

Table 11-10 Microoperations for register

Memory control		
$R = F't_1 + F(q_0+q_1+q_3)t_1$:	$B \leftarrow M$	Memory read
$W = F(q_2 +q_3+q_4)t_3$:	$M \leftarrow B$	Memory write
A register		
$a_1 = Fq_0t_3$:	$A \quad A \quad B$	AND
$a_2 = Fq_1t_3$:	$A \leftarrow A + B$	Add
$a_3 = rB_{12}$:	$A \quad 0$	Clear
$a_4 = rB_{10}$:	$A \leftarrow A$	Complement
$a_5 = rB_8$:	$A \leftarrow shr\ A, A_{16} \quad E$	Shift-right
$a_6 = rB_7$:	$A \quad shl\ A, A_1 \quad E$	Shift-left
$a_7 = rB_6$:	$A \leftarrow A + 1$	Increment
$a_8 = pB_{11}$:	$A_{1-8} \quad N_{1-8}$	Transfer
B register		
$b_1 = Fq_2t_2$:	$B \leftarrow A$	Transfer
$b_2 = Fq_3t_2$:	$B \quad B + 1$	Increment
$b_3 = Fq_4t_2$:	$B(AD) \quad PC, B(OP)$	Transfer
PC register		
$c_1 = F't_1$		
$+(q_3B_z + q_4)Ft_3$		
$+(B_5A'_{16} + B_4A_{16})$		
$+B_3A_z + B_2E')r$		
$+(B_{12}N_9 + B_{10}U_9)P$:	$PC \leftarrow PC + 1$	Increment
$c_2 = q_5t_3$::	$PC \quad B(AD)$	Transfer
$b_3 = Fq_4t_2$:	$PC \quad MAR$	Transfer
MAR Register		
$d_1 = F't_0$:	$MAR \quad PC$	Transfer
$d_2 = Ft_0$:	$MAR \quad B(AD)$	Transfer
I Register		
$i_1=F't_2$:		
E Flip-Flop		
$e_1 = rB_{11}$:	$E \quad 0$	Clear
$e_2 = rB_9$:	$E \leftarrow E$	Complement
$a_2 = Fq_1t_3$:	$E \quad carry$	Transfer
$a_5 = rB_8$:	$E \leftarrow A_1$	Shift-right
$a_6 = rB_7$:	$E \leftarrow A_{16}$	Shift-left

F Flip-Flop $f_1 = F'(q_0 + q_1$ $+q_2+q_3+q_4)t_3$: $f_2 = Ft_3$:	*F 1* *F 0*	Set Clear
S Flip-Flop $s_1 = rB_1$:	*S 0*	Clear
G register *S:*	*G G + 1*	Count
U Register $u_1 = p\ B_9$:	U_{1-8} A_{1-8}, U_9 *0*	Transfer
N Register $a_8 = pB_{11}$:	*N9 0*	Clear

5.5 Design of Computer

The list of microoperations given in Table 11-10 provides the information needed to design the registers of the computer. The operations to be performed on each register are clearly demonstrated by the listed statements. For example, program counter *PC* has three microoperations:

$$c1: \quad PC \leftarrow PC + 1$$

$$c2: \quad PC \leftarrow B(AD)$$

$$b3: \quad PC \leftarrow MAR$$

This register must have increment and transfer capabilities. It can be implemented by means of a counter with parallel load of the type shown in Fig. 7-19. Since *PC* receives input information from two sources, it requires a multiplexer to select between the two inputs, as explained in conjunction with Fig. 8-3. The other registers are designed in a similar manner.

A block diagram showing the types of registers needed for the computer is given in Fig. 11-7. The memory unit is also included to show its connection to the processor. The control logic provides all the control variables for the registers. The design of the control logic is discussed in the next section. The control variables that are generated in the control unit are applied to the registers as indicated in the diagram. In addition to the registers, the processor uses four multiplexers to select from two or more sources. All the registers and multiplexers are MSI functions available in standard integrated circuits. The three flip-flops, *E*, *F*, and *S*, and their corresponding combinational logic must be designed with SSI gates and flip-flops.

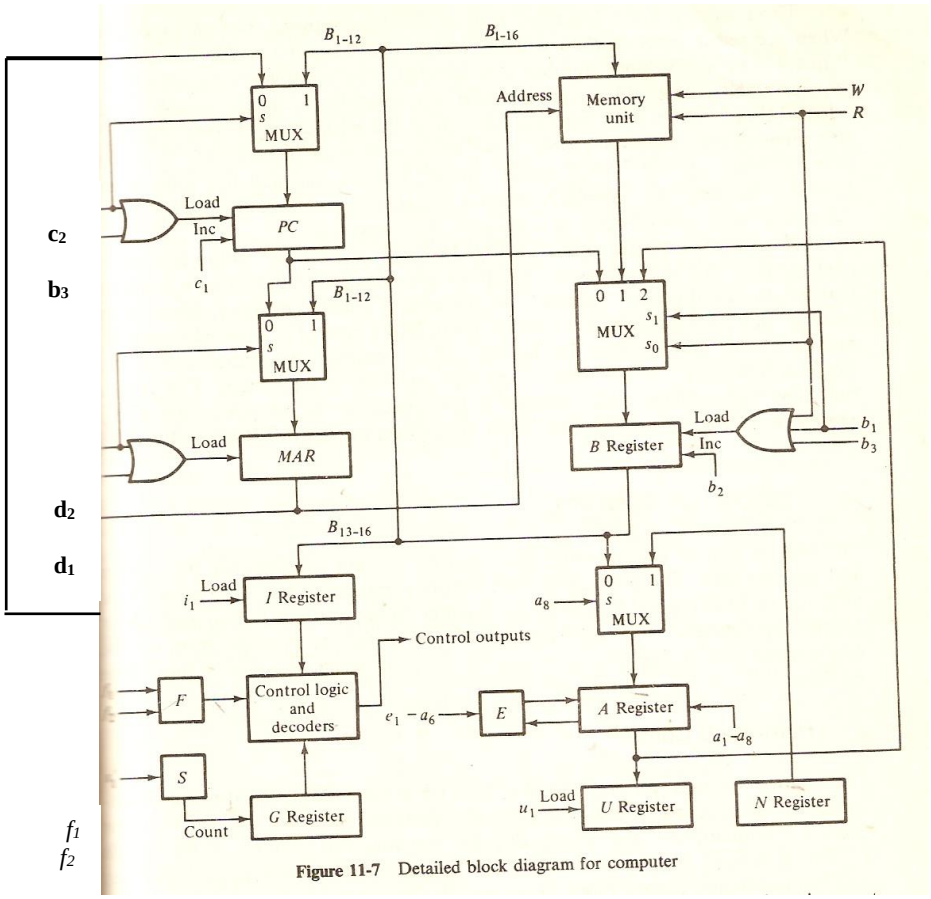


Figure 11-7 Detailed block diagram for computer

All of the registers in the computer, expect register A, require a load, increment, or both load and increment control inputs. One can choose to employ an MSI counter with parallel load for all registers. In this manner, it would be possible to have an inventory of just one standard type of IC component for the registers. A possible commercial component is IC type 74161. This MSI circuit contains a 4-bit counter with parallel load and an asynchronous clear input. The clear inputs of the registers can be connected to a master reset switch in the computer to clear all registers asynchronously prior to the clocked operations. The 12-bit registers, *PC* and MAR, will need three such ICs, and the 16-bit register, *B*, will require four ICs. The I and G registers can be implemented with one IC each. The 4-bit counter, IC, can be converted to a 2-bit counter for *G* by the method outlined in Section 7-5, in conjunction Fig 7-20.

The *A* register is the most complicated register because it performs all the processing tasks for the computer. This register is an accumulator register of the type designed in section 9-10 and can use the configuration shown in Fig.9-22. It can also be implemented with a bidirectional shift register with parallel load, as shown in Fig. 7-9, together with an ALU of the type discussed

in Section 9-6. A better choice would be to use an accumulator MSI circuit such as type 74S281 IC.

When implemented with an ALU or accumulator IC, the control unit must generate the corresponding control variables to select the required microoperations in the ALU. These will be different from the single control functions defined for the control unit in this design.

Input register *N* and output register *U* can be part of a standard teletypewriter interface. Integrated circuits that interface with a teletypewriter unit are available commercially and are usually called universal asynchronous receiver-transmitters (abbreviated UART). Such an IC includes an input register and an output register within the unit, together with the two flags required for synchronizing the transfer.

Three of the multiplexers in Fig.11-7 select between two input sources. When the select input marked with an *S* is 1, MUX input number 1 is selected. When $s = 0$, MUX input number 0 is selected. The multiplexer associated with register *B* has three input sources. Selection variables s_1 and s_0 determine the selected input. When both selection lines are 0, the selected input comes from *PC*. The memory-read signal *R* makes $s_0 = 1$ while s_1 remains 0 (because $b_1 = 0$ when $R = 1$). With $s_1s_0 = 01$, MUX input number 1 is selected and this input comes from the memory unit. Similarly, control variable b_1 produces a selection $s_1s_0 = 10$, which causes the contents of register *A* to be selected.

The entire computer shown in Fig.11-7 can be enclosed within a single IC package to form a microcomputer. A typical microcomputer IC normally has added features in the processor section, but includes a smaller memory. Most of the memory in a microcomputer is usually of the ROM type. The internal design of a microcomputer chip requires that the logic of the computer be derived with a set of Boolean functions that specify all gates and flip-flops in the system. The Boolean functions that implement each register in the system can be derived by the method presented in Section 9-10 for the design of registers in terms of Boolean functions.

5.6 Design Of Control

The control unit of the computer generates the control variables for the registers and memory unit. There are 24 distinct control variables and all of them are listed in Table 11-10 as control functions. In Chapter 10, we presented three methods for control logic design: hard-wired control, PLA control, and microprogram control. The control unit of the computer can be designed using any one of these three methods.

5.6.1 Hard-wired Control

The control organization presented in Fig.11-6 is essentially a hard-wired organization of the sequence register and decoder method. Sequence register *G* in this case is a counter, and the timing decoder provides four control states for the system. A second decoder is used for the operation code stored in

the *I* register. The control-logic-network block generates all the control functions for the computer.

The implementation of the control logic network in Fig. 11-6 completes the design of the hard-wired control. This implementation consists of combinational gates that generate the 24 control functions listed in Table 11-10. The Boolean functions listed as control functions specify the Boolean equations from which the combinational circuit can be derived. This circuit will not be drawn here but can be easily obtained from the 24 Boolean functions that define the control variables R,W,a_1 through $a_8,b_1,b_2,b_3,c_1,c_2,d_1,d_2,e_1,e_2,f_1,f_2,s_1,$ and u_1.

5.6.2 PLA Control

The PLA control is similar to the sequence register and decoder method, expect that all combinational circuits are implemented within the PLA. The two decoders are included inside the PLA implementation, since they are combinational circuits. The total number of control outputs is 24. The total number of PLA inputs is also 24. A 24-input, 24-output PLA may not be available in one commercial IC package. For this reason, the control unit should be partitioned in such a way so it can be implemented with a minimum number of PLA ICs.

One way to partition the control is according to the function tables presented in section 11-5. The register-transfer statements in this section are listed in Tables 11-5 through 11-9.The PLA control partitioned according to these tables is shown in Fig. 11-8. This implementation replaces the hard-wired control of Fig. 11-3.

Figure 11-8 shows three PLAs and two registers for the control unit. The two decoders are not needed here, since they are implemented inside the PLA. Note that there are no connections from the outputs of any PLA to the inputs of sequence register *G*. A feedback connection in not necessary because the G register is a counter and the next state is predetermined from the continuous count sequence. PLA 1 implements the control variables listed in Table 11-5 (fetch cycle) and Table 11-3 (common operations for execute cycle). These control variables depend on the timing variables from *G*, the operation code from I, and cycle control in *F*. PLA 2 implements the control functions listed in Table 11-4 (execution of memory-reference instructions). These control functions have the same input variables as PLA 1, with the addition of binary variable B_z. Remember that B_z is a binary variable equal to 1 when the *B* register contains all 0's.

The third PLA generates the register-reference and input-output control functions listed in Tables 11-8 and 11-9. These control functions have two common variables:

$r = q_6 t_3$ *for the register-reference operations*

$p = q_7 t_3$ *for the input-output operations*

These two common variables are generated in PLA 1 and applied as inputs to PLA 3. The other inputs to third PLA come from register *B* (bits 1-12) and other status-bit conditions.

Control variable c_1 increments the program counter. This control variable is generated an all three PLAs. The three outputs must be combined with an external OR gate to provide a single output. This output is applied to the increment input of *PC*.

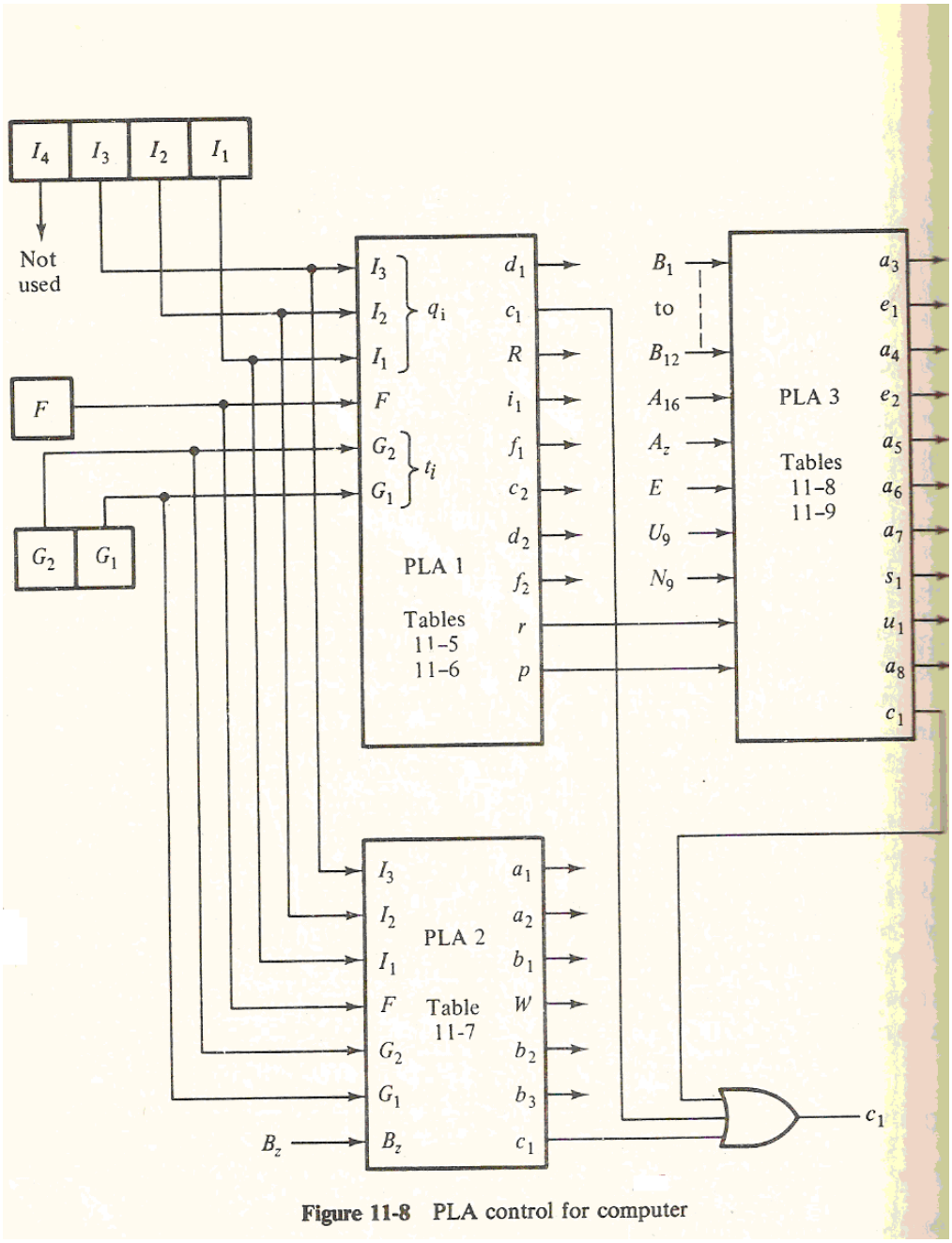


Figure 11-8 PLA control for computer

The derivation of the program tables for the three PLAs completes the control design. The PLA 1 program table can be obtained from the control functions listed in Tables 11-5 and 11-6. These functions are repeated again in Table 11-11 for convenience. Some of the functions have been simplified for entry in the program table. For example, the read control variable *R* was originally listed as:

$$R = F't_1 + F(q_0 + q_1 + q_3)t_3$$

The decoded output variables *q0,q1*, and *q3* are a function of the variables in the *I* register and can be simplified as follows:

$$q_0 + q_1 + q_3 = I'_3I'_2I'_1 + I'_3I_2I_1 = I'_3I_1 + I'_3I'_2$$

Table 11-11 Control functions for PLA 1

$d_1 = F't_0$:	$MAR \leftarrow PC$
$c_1 = F't_1$:	$PC \leftarrow PC + 1$
$R = F't_1 + F(I'_3I_1 + I'_3I'_2)t_1$:	$B \leftarrow M$
$i_1 = F't_2$:	$I \leftarrow B(OP)$
$f_1 = F'(I'_3 + I'_2I'_1)t_3$:	$F \leftarrow 1$
$c_2 = q_5t_3$:	$PC \leftarrow B(AD)$
$d_2 = Ft_0$:	$MAR \leftarrow B(AD)$
$f_2 = Ft_3$:	$F \leftarrow 0$
$r = q_6t_3$:	*Register reference*
$p = q_7t_3$:	*Input-output*

Since the PLA accepts the *I* variables rather than the *q* variables, it is more convenient to use the two-term function rather than the three-term function. Control variable f_1 is simplified in a similar manner. The other Boolean variables need a translation from the *t* designation to a state in the sequence register *G* and from the *q* designation to the corresponding operation code in the *I* register.

The program table for PLA 1 is given in Table 11-12. The PLA has 6 inputs, 12 product terms, and 10 outputs. The entries for G_2 and G_1 are 00, 01, 10, and 11 and correspond to timing variables t_0 , t_1 , t_2 , and t_3 , respectively. The entry for I_3,I_2,and I_1 is a binary number equal to the value of subscript *i* in q_i, unless the function is simplified. Note that register I has four bits, but I_4 is not used since it is always 0. The procedure for obtaining a PLA program table from a set of Boolean functions is explained in Section 5-8.

Table 11-12 Program table for PLA 1

Product term	Inputs						Outputs										
	I_3	I_2	I_1	F	G_2	G_1	d_1	c_1	R	i_1	f_1	c_2	d_2	f_2	r	p
1	-	-	-	0	0	0	1	-	-	-	-	-	-	-	-	-	$F't_0$
2	-	-	-	0	0	1	-	1	1	-	-	-	-	-	-	-	$F't_1$
3	0	-	1	1	0	1	-	-	1	-	-	-	-	-	-	-	$FI'_3I_1t_1$
4	0	0	-	1	0	1	-	-	1	-	-	-	-	-	-	-	$FI'_3I'_2t_1$
5	-	-	-	0	1	0	-	-	-	1	-	-	-	-	-	-	$F't_2$
6	0	-	-	0	1	1	-	-	-	-	1	-	-	-	-	-	$F'I'_3t_3$
7	-	0	0	0	1	1	-	-	-	-	1	-	-	-	-	-	$F'I'_2I'_1t_3$
8	1	0	1	-	0	0	-	-	-	-	-	1	-	-	-	-	q_5t_3
9	-	-	-	1	0	0	-	-	-	-	-	-	1	-	-	-	Ft_0
10	-	-	-	1	1	1	-	-	-	-	-	-	-	1	-	-	Ft_3
11	1	1	0	-	1	1	-	-	-	-	-	-	-	-	1	-	q_6t_3
12	1	1	1	-	1	1	-	-	-	-	-	-	-	-	-	1	q_7t_3

The program table for PLA 2 can be derived in a similar manner, although it is not listed here. The third PLA requires 12 AND terms and a 6-input OR gate (to generate control variable c_1). This part of the control may be implemented more economically with SSI gates or with a field-programmable gate array (FPGA). The FPGA' is similar to the FPLA has 9 AND (or NAND) gates sharing 16 common inputs.* Two such FPGA integrated circuits are required to replace PLA 3 in Fig. 11-8. The external OR gate can be combined with the other lines that generate variable c_1.

5.6.3 Microprogram Control

The organization of the control unit for the computer is more suitable for a PLA control than for a microprogram control, mostly because of the way the register-reference instructions were originally formulated. The microprogram control configuration to be developed here implements the control functions for the fetch cycle and the memory-reference instructions. The register-reference and input-output operations can be implemented more efficiently with a hard-wired or PLA control.

The microprogram control does not need the *I, G,* and *F* registers. The operation code is in *B(OP)* at the end of the fetch cycle, and this code can be used to specify a macrooperation address for control memory without the need for an *I* register. The timing variables generated in the sequence register *G* can be replaced by a sequence of clock pulses that read consecutive microinstructions from control memory. The transfer from the fetch cycle to the execute cycle can be done in control memory by a branch microinstruction that

transfers control to the next cycle without the use of the *F* flip-flop. The microprogram control configuration to be developed here replaces the entire hard-wired control of Fig. 11-6 (except the *B* register).

Going over Tables 11-5,11-6, and 11-7, we note that all microinstructions can be sequenced by incrementing the control memory address, except for going to execute a particular memory-reference instruction or for returning to the fetch cycle. A particular memory-reference instruction routine can be accessed with an external macrooperation address. If we start the fetch cycle from address 0, it would be possible to branch to the fetch cycle by clearing the control memory address register CAR. Therefore, the address-sequencing part of the microprogram control needs only three operations:

1. Increment CAR to read the next microinstruction in sequence.
2. Clear CAR to start the fetch cycle.
3. Provide a bit transformation from B(OP) to an external address for CAR.

*IC type 82S103 from Signetics.

A possible microprogram control for the computer is shown in Fig. 11-9. The control memory ROM has 32 words of 7 bit each. The first four bits are encoded computer has 24 control functions, 16 are sufficient to generate those control functions associated with the fetch cycle and the execution of the memory-reference instructions. Instead of using 16 bits of ROM to specify 16 outputs, we chose to employ only 4 bits and decode them through a 4-to-16 line decoder to provide up to 16 distinguishable output variables. This scheme saves ROM bits but requires an external decoder. It also limits the capability of the microinstructions because only one control function can be specified in any given microinstruction.

The address-sequencing part of the microprogram unit does not require a multiplexer to select status-bit conditions. There is only status bit to be considered and we will show later how this can be included with an external circuit. There is no need for an address field in the microinstruction because no branching capabilities are provided except to return to the beginning of the fetch cycle or to transfer an external address. The last three bits of a microinstruction determine the next address. Bit 7 increments the control address register. Bit 6 clears CAR, which causes a return to the fetch cycle. Bit 5 loads an external address into CAR. The input address must contain 5 bits because the ROM has $32=2^5$ words. Three of these bits come from the *B*-register part that holds the operation-code. The last two bits are always equal to11. This is a code transformation from the operation-code bits of the instruction to an external address for control memory. This transformation causes the AND instruction whose operation code is 000 to be changed into an address for CAR equal to 00011. The ADD instruction transforms from 001 to 00111; and so on, up to an input-output instruction whose operation code is 111

and whose address transformation is 11111. The most significant bit in *B(OP)* is not used because it is always 0.

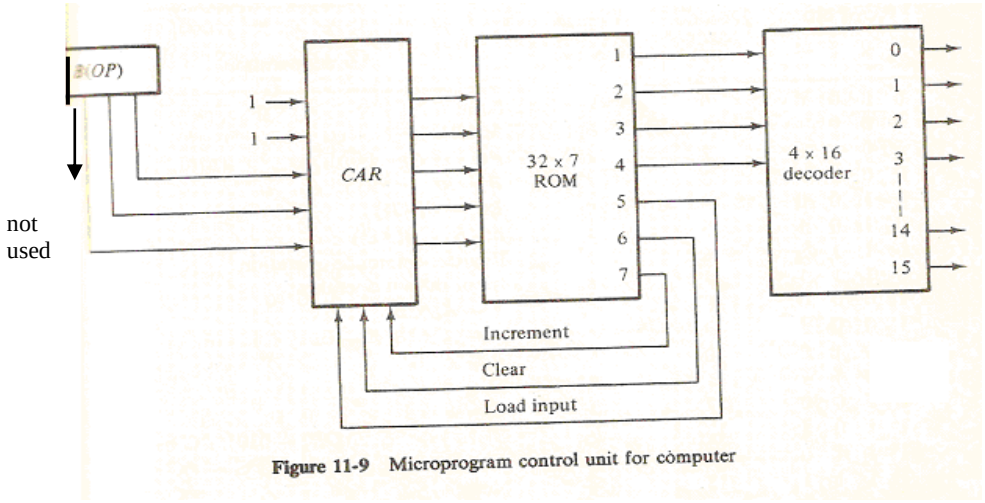


Figure 11-9 Microprogram control unit for computer

The microprogram control unit shown in Fig. 11-9 is very simple and requires only three MSI circuits. Because of its simplicity, it is not very flexible and, as sown subsequently, requires additional circuits for a complete control-unit implementation.

The microoperations for the fetch cycle and the execution of memory-reference instructions are listed in Tables 11-5, 11-6, and 11-7. The microoperations for the *I* and *F* registers are not needed, since these registers are not used. The remaining microoperations and their encoded control functions are listed in Table 11-13. The first four bits of a ROM word in control memory provide 16 combinations, and each combination specifies a microoperation. The other 14 combinations are decoded to provide control variables for the listed microoperations. Decoder output 14 initiates the memory-write operation, M B, and also specifies a conditional control for incrementing *PC* dependent on variable B_z. The reason for repeating these two microoperations in one microinstruction will be clarified later. Note that the memory-write microoperation is also initiated with decoder output 11, and the control variable that increments *PC* is also available from decoder output 2.

TABLE 11-13 Encoding of ROM bits for microoperations

ROM bits				Decoder output	Control function	Microoperation
1	2	3	4			
0	0	0	0	0	—	None
0	0	0	1	1	d_1	$MAR \leftarrow PC$
0	0	1	0	2	c_1	$PC \leftarrow PC + 1$
0	0	1	1	3	R	$B \leftarrow M$
0	1	0	0	4	c_2	$PC \leftarrow B(AD)$
0	1	0	1	5	d_2	$MAR \leftarrow B(AD)$
0	1	1	0	6	r	Register-reference operation
0	1	1	1	7	p	Input-output operation
1	0	0	0	8	a_1	$A \leftarrow A \wedge B$
1	0	0	1	9	a_2	$A \leftarrow A + B, E \leftarrow$ carry
1	0	1	0	10	b_1	$B \leftarrow A$
1	0	1	1	11	W	$M \leftarrow B$
1	1	0	0	12	b_2	$B \leftarrow B + 1$
1	1	0	1	13	b_3	$B(AD) \leftarrow PC, B(OP) \leftarrow 0101, PC \leftarrow MAR$
1	1	1	0	14	W, c_1	$M \leftarrow B$, if $(B_z = 1)$ then $(PC \leftarrow PC + 1)$
1	1	1	1	15	—	None

The microprogram for control memory is given in Table 11-14. This is also the truth table for programming the ROM. There are 32 words of ROM, and the address and content of each word are specified in the table. The table is subdivided into nine routines showing the microinstructions that belong to the fetch cycle and the microinstructions for executing each of the computer instructions. The symbolic designation column gives the microprogram in symbolic form and the address sequencing for *CAR*.

TABLE 11-14 ROM truth table for microprogram control

Instruction	ROM address	ROM outputs							Symbolic designation	
		1	2	3	4	5	6	7	Microoperations	Next address
FETCH	00000	0	0	0	1	0	0	1	$MAR \leftarrow PC$	$CAR \leftarrow CAR + 1$
	00001	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	00010	0	0	1	0	1	0	0	$PC \leftarrow PC + 1$	$CAR \leftarrow 2^2B(OP) + 3$
AND	00011	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	00100	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	00101	1	0	0	0	0	1	0	$A \leftarrow A \wedge B$	$CAR \leftarrow 0$
	00110	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
ADD	00111	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	01000	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	01001	1	0	0	1	0	1	0	$A \leftarrow A + B, E \leftarrow$ carry	$CAR \leftarrow 0$
	01010	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
STO	01011	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	01100	1	0	1	0	0	0	1	$B \leftarrow A$	$CAR \leftarrow CAR + 1$
	01101	1	0	1	1	0	1	0	$M \leftarrow B$	$CAR \leftarrow 0$
	01110	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
ISZ	01111	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	10000	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	10001	1	1	0	0	0	0	1	$B \leftarrow B + 1$	$CAR \leftarrow CAR + 1$
	10010	1	1	1	0	0	1	0	$M \leftarrow B$, if $(B_z = 1)$ then $(PC \leftarrow PC + 1)$	$CAR \leftarrow 0$
BSB	10011	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	10100	1	1	0	1	0	0	1	$B(AD) \leftarrow PC, PC \leftarrow MAR$	$CAR \leftarrow CAR + 1$
	10101	1	0	1	1	0	0	1	$M \leftarrow B$	$CAR \leftarrow CAR + 1$
	10110	0	0	1	0	0	1	0	$PC \leftarrow PC + 1$	$CAR \leftarrow 0$
BUN	10111	0	1	0	0	0	1	0	$PC \leftarrow B(AD)$	$CAR \leftarrow 0$
	11000	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
	11001	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
	11010	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
REGISTER	11011	0	1	1	0	0	1	0	Register operation	$CAR \leftarrow 0$
	11100	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
	11101	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
	11110	0	0	0	0	0	1	0	None	$CAR \leftarrow 0$
I/O	11111	0	1	1	1	0	1	0	Input-output operation	$CAR \leftarrow 0$

The fetch cycle starts from address 0. The three consecutive microoperations in the fetch routine transfer the contents of *PC* to MAR, read the instruction into the *B* register, and increment *PC*. At address 2 (0010), bit 5 of the microinstruction is equal to 1. The same clock pulse that increments *PC* also performs the microoperation:

$$\text{CAR} \qquad 2^2\text{B(OP)} + 3$$

B(OP) contains the three bits of the operation code. These bits are shifted twice to the left (multiplied by 2^2) and binary 3 (11) is added to form an address for *CAR*. The address received in *CAR* transfers control to one of the routines listed in the table, and control continues to execute the specified instruction. The implementation of this code transformation is depicted in Fig.11-9.

This configuration assigns four words of ROM for each instruction, except for the I/O instruction. For example, the ISZ instruction has operation code 011. The beginning of the routine that executes this instruction is at address *4 3 + 3 = 15*, which is binary 01111. The four ROM words for this routine are at addresses 15, 16, 17 and 18. We cannot use the word at address 19 because this address contains the first microinstruction for the BSB routine. Since there are no branching capabilities in this microprogram unit, we cannot branch to an unused ROM word; therefore, each routine must be completed with no more than four microinstructions.

The AND routine can be implemented with three microinstructions. The address of the instruction is transferred into MAR, the operand is read from memory into *B*, and the AND microoperation is performed between the *A* and *B* registers. The last microinstruction at address 5 (00101) has bit 6 equal to 1. This causes *CAR* to be cleared, and control returns to address 0 to start the fetch cycle again. The first two microinstructions of the AND routine have bit 7 equal to 1, which causes *CAR* to be incremented. The last word in this routine at address 6 is not used. This word cannot be left empty because we must specify something for the ROM truth table. The best way to fill in this word is to specify no microoperation in bits 1 through 4 and to clear *CAR* with bit 6. In this way, if a malfunction occurs and control memory finds itself in address 6, no operation will be executed and control will return to the fetch cycle.

The ADD and STO routines need three microinstructions. The BSB instruction uses all four words available for the routine. The BUN instruction needs only one microinstruction. A register-reference instruction initiates a control variable r, which must be used in conjunction with a bit in the *B* register to initiate one of the specified operations. The same applies to an input-output (I/O) instruction.

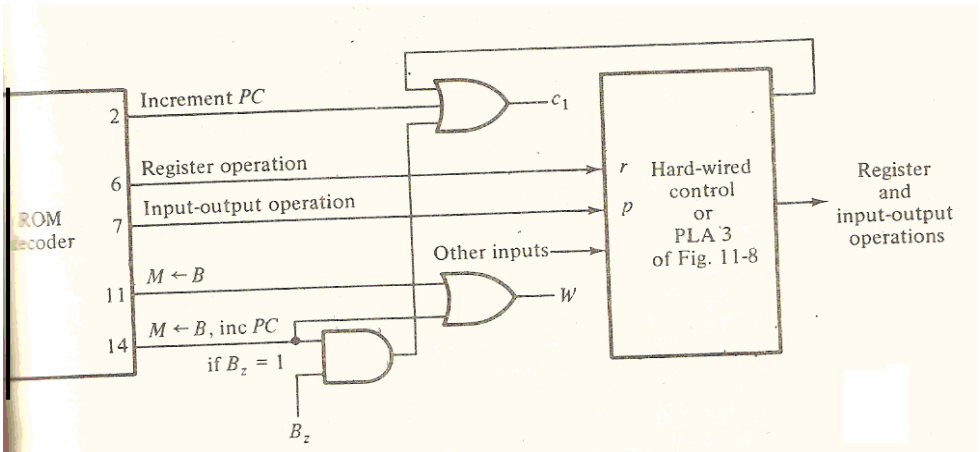


Figure 11-10 Additional circuits for microprogram control

The ISZ routine needs four microoperations and a conditional operation dependent on the value of B_z. This imposes a problem, because there are only four ROM words available for this routine and the microprogram configuration has no facility for checking status-bit conditions. This problem can be solved by including two microoperations in one microinstruction and checking the status bit with an external AND gate. To compensate for this unorthodox configuration, we insert an external circuit as shown in Fig. 11-10. The ROM decoder has two outputs for a memory-write operation M B : one in output 11 and the other in output 14. These two outputs is ANDed externally with status bit B_z to provide the increment-*PC* control function. Decoder output 2 also specifies an increment-*PC*. Some of the operations in the register-reference and input-output instructions specify this operation as well. The three outputs must be ORed together to form a single output for incrementing *PC*. Variables *r* and *p* from the ROM decoder are used in conjunction with other status-bit conditions to generate the remaining control variables can be generated with an external hard-wired configuration or with a PLA as indicated in the diagram.

5.7 Computer Console

Any computer has a control panel or console with switches and lamps to allow manual and visual communication between an operator and the computer. This communication is needed for starting the operation of the computer (bootstrapping) and for maintenance purposes. For the sake of completeness, we shall enumerate a set of useful console functions for the computer, although the circuits required to implement these functions will not be shown.

Lamps indicate to the operator the status of registers in the computer. The normal output of a flip-flop connected to an indicator lamp to light when the flip-flop is set and to turn off when the flip-flop I cleared. The registers whose outputs are to be observed in the computer console are: *A* , *B* ,*PC* , *MAR* , *I* , *E* , *F* and *S*. When a count is made of the total number of the flip-flops involved, we find that 63 indicator lamps are needed.

A set of switches and their functions for the console may include the following:

1. Sixteen “word” switches to set manually the bits of one word.
2. A “start” switch to set the S flip-flop. The signal from this switch also clears flip-flop F , N_9 ,U_9 and register G.
3. A “stop” switch to clear the S flip-flop. To ensure the completion of the current instruction, the signal from the switch is ANDed with the Boolean function $(F + q5 + q6 + q7)t3$ before it is applied to clear S.
4. A “load address” switch to transfer an address to the *PC* register. When this switch is activated, the contents of 12 “word” switches are transferred to *PC.*
5. A “deposit” switch to manually store words into memory. When this switch is activated, the content of *PC* is transferred to MAR and a memory cycle is initiated. After 1 s, the contents of the 16 “word” switches are transferred into the B register and *PC* is incremented by 1.
6. A “display” switch to examine the content of a word in memory. When this switch is activated, the content of *PC* is transferred to MAR, a memory cycle is initiated, and *PC* is incremented by 1. The contents of the memory word specified by the address in *PC* are in register B and can be seen in the corresponding indicator lamps.

To ensure that the computer is not running when the power is turned on, the *S* flip-flop must have a special circuit that forces it to always turn to the clear position right after the application of power to the machine.

Unit-V

Self Assessment Questions

Fill in the blanks :

1. A _____switch to manually store words into memory.
2. Program counter PC holds the address of the _____instruction to be read from memory.
3. _____ is a counter that produces the timing.

True/ False

1. Start stop flip-flop which is doing stars and stops computer.
2. The symbolic designation E stands for electrical flip-flop.
3. The memory unit has a capacity of 4096 words of 16 bits each

Multiple Choice :

1. MAR stands for _____
   a) Memory address register b) Main address register
   c) Memory accumulator register c) None
2. Integrated circuits that interface with a teletypewriter unit are available commercially and are abbreviated as _____
   a) VBRT b) UART
   c) UARP d) ABRP
3. A "start" switch to set the _____Flip-flop
   a) JK b) RS
   c) S d) None
4. BSB stands for _____
   a) Branch to subprogram b) Branch to store
   c) Branch to subroutine d) None of the above

Questions:

1. Explain in detail about system configuration?
2. Describe detail about computer instructions?
3. Explain about the Design of computer registers?
4. What is computer console discussed in detail?
5. Explain about the design of control?
6. What are the six phases of design process?

Self Assessment Answers:

Fill in the blanks

1. Deposit
2. Next
3. Sequence register G

True OR False

1. True
2. False
3. True

Multiple Choice

1. (a)
2. (b)
3. (c)
4. (c)

NOTES

……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..

NOTES

……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..
……..