Uttar Pradesh Rajarshi Tandon Open University

Master of Computer Application

MCA-105/MCS-106 /PGDCA-105

Computer Organization

Uttar Pradesh Rajarshi Tandon Open University

Master of Computer Application

MCA-105/MCS-106 /PGDCA-105

Computer Organization

BLOCK

1

INTRODUCTION TO DIGITAL ELECTRONICS

UNIT-1

Introduction to Number System

UNIT-2

Boolean Algebra and Logic Gates

UNIT-3

Reduction Techniques

UNIT-4

Binary Arithmetic

UNIT-5

Sequential Circuit

Course Design Committee

Prof. Ashutosh Gupta **Chairman**
Director (In-charge)
School of Computer and Information Science, UPRTOU Allahabad
Prof. Suneeta Agarwal **Member**
Department of CSE
MNNIT Allahabad, Prayagraj
Dr. Upendra Nath Tripathi **Member**
Associate Professor, Department of Computer Science
Deen Dayal Upadhyaya Gorakhpur University, Gorakhpur
Dr. Ashish Khare **Member**
Associate Professor, Department of Computer Science
University of Allahabad, Prayagraj
Dr. Marisha **Member**
Assistant Professor (Computer Science),
School of Science, UPRTOU Allahabad
Mr. Manoj Kumar Balwant **Member**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad

Course Preparation Committee

Mr. Manoj Kumar Balwant **Author Block 1 (Unit 1,2,3,4,5)**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad.
Dr. JitendraPande **AuthorBlock 2, 3 (Unit 6,7,8,9,10,11)**
Associate Professor
School of Computer Sciences & Information Technology
Haldwani, Uttarakhand 263139
Prof. Ashutosh Gupta **Editor Block 1 (Unit 1, 2, 3, 4, 5)**
Director (In-Charge)
School of Computer & Information Sciences, UPRTOU Allahabad
Prof. Abhay Saxena **Editor Block 2, 3 (Unit 6, 7, 8, 9, 10, 11)**
Professor and Head, Department of Computer Science
Dev SanskritiVishwavidyalya, Hardwar, Uttrakhand
Mr. Manoj Kumar Balwant **Coordinator**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad.

ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.

MCA-105/4

Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

BLOCK INTRODUCTION

In a digital system, discrete elements of information are represented by signals. The signals in the digital systems have only two discrete values 0 and 1 (also called binary signal) because these binary signals are more reliable than multi-valued signals. The first unit presents you binary, octal and hexadecimal numbers which are commonly used for understanding and designing digital systems such as computers and electronic calculator. The second unit introduces you various identities essential for understanding Boolean algebra. It illustrates the relationship between a Boolean function and its logic circuit consisting of basic logic gates. It briefly explains you the basic building blocks of integrated circuits. The third unit presents Karnaugh Map which is a powerful technique for simplification of any Boolean function. A minimized Boolean function requires minimum number of gates for its implementation which is more reliable and also decreases manufacturing cost. The fourth unit explains some frequently used logic functions in combinational circuits and their circuit implementations. It includes adder, substractor, encoder, decoder and multiplexer which are most common in ICs. This unit also presents a technique to implement any Boolean functions using a universal gate NAND or NOR. The last unit describes basic concept of a sequential circuit and flip flops which are basic building block of memory element to stores a binary information. It explains some basic sequential circuit components such as Register, shift registers and counters upon which a complex digital system is based.

UNIT-1 INTRODUCTION TO NUMBER SYSTEM

Structure

1.1 INTRODUCTION

We all use mathematical digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 in our daily life to represent any number for counting and measuring. This is decimal number system which is most common for every day usage. Since, this number system uses 10 distinct digits, the base of this number system is 10. But in digital systems (such as computers and electronic calculators) there are other number systems such as binary, octal and hexadecimal are commonly used for understanding and designing digital systems. For example a computer uses binary number system to represent any data and manipulate them. It performs arithmetic operations where each operand is represented in binary number system. Data and information are stored in the computer in binary numbers and data processing is carried out with binary numbers. In this unit, we will discuss commonly used number system in digital systems.

1.2 OBJECTIVES

After the study of this unit, you should able to

- Understand binary, octal and hexadecimal numbers.
- Convert a number in one number system to different number systems.
- Know how negative numbers are represented in a computer.

1.3 BINARY NUMBER SYSTEM

A binary number system expresses any number using only two symbols: 0 and 1. So, the base or radix of this number system is 2. Each binary symbol is called

a bit (**Bi**nary Digi**t**). A group of 8 bits is called as one byte and a group of 4 bits is called as nibble. A binary number is written with subscript 2 after the number. Otherwise, the number is considered as decimal number. For example, $(101101)_2$ is a binary number. The counting in binary number system is illustrated in Table 1.1 along with their corresponding decimal numbers for comparison. The leftmost digit in any binary number is called MSB (most significant bit), while the rightmost digit is called as LSB (least significant bit).

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table 1.1- Numbers with different bases.

1.4 OCTAL NUMBER SYSTEM

A octal number system represents any number using 8 symbols: 0,1,2,3,4,5,6,7. So, the base or radix of this number system is 8. The relationship between some octal numbers and their equivalent decimal and binary numbers are shown in Table 1.1.Each digit in an octal number requires 3 bits (2^3=8) to represent in binary number. The smallest 2 digit number in octal is $(10)_8$ whose decimal equivalent number is 8. Any number in this number system is written with subscript 8 after parentheses otherwise, the number is considered as a decimal number.

1.5 HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system expresses any number by using 16 distinct symbols. These symbols include numeric digits 1-9 and alphabets A-F (for numbers 10 to 15). So, the base or radix of this number system is 16. This number system is also called as alphanumeric number system because it uses both numeric digits and alphabets. For example, $(D5CF)_{16}$is a hexadecimal number. Table 1.1 shows similarities of this number system with decimal and binary number system. Since,

this number system has 16 different symbols, it requires 4 bits (2^4=16) to represent each digit of any hexadecimal number to equivalent binary number. Any hexadecimal number is represented with the superscript 16 after the number.

1.6 INTERCONVERSION TO DIFFERENT NUMBER SYSTEM

1.6.1 BINARY,OCTAL AND HEXADECIMAL NUMBER TO DECIMAL NUMBER

A number $a_n a_{(n-1)}...a_1 a_0.a_{-1}a_{-2} \ldots a_{-m}$ written in base r number system can be converted into corresponding decimal number by summing each digit multiplied to their powers of base r as shown below.

$a_n.r^n+a_{(n-1)}.r^{(n-1)}+...+a_1.r^1+a_0.r^0+a_{-1}.r^{-1}+a_{-2}.r^{-2}+ ... +a_{-m}.r^{-m}$.

For example, the decimal equivalent of a Binary number $(1011.01)_2$:

= 1*2^3+0*2^2+1*2^1+1*2^0+0*2^-1+1*2^-2

=8+0+2+1+0+0.25

= $(11.25)_{10}$

Similarly, an Octal Number $(365)_8$is equivalent to decimal number:

= (3 * 8^2) + (6 * 8^1) + (5 * 8^0)

= (3 * 64) + (6 * 8) + (5 * 1)

= $(245)_{10}$

Check Your Progress

1. Find the decimal equivalent of a Binary number $(101.1)_2$
2. Find the decimal equivalent of a octal number $(535)_8$.

1.6.2 DECIMAL TO BINARY, OCTAL, HEXADECIMAL NUMBER

For Integer Part : Any integer in decimal number system can be converted into any other number system with base r (including binary or octal or hexadecimal number) by repeated division of the number by base r to get quotient and remainder each time until the final quotient becomes 0. This conversion procedure can be more precisely explained with following steps.

1. Divide the integer in decimal number by base r. Consider the division as an integer division.

2. Write down the quotient and remainder. The remainder value should be between 0 to r-1.

3. Perform integer division of the quotient by base r.

4. Repeat step 2 and 3 until the final quotient becomes 0.

5. The number in base r number system is the sequence of the remainder starting from the last remainder to first.

For example, the number 256 in DECIMAL can be converted to HEXADECIMAL number as follow:

We know that base of Hexadecimal Number system is 16, so perform division by 16.

DIVISION	QUOTIENT	REMAINDER (in HEX)
256 / 16	16	0
16 / 16	1	0
1 / 16	0	1

Equivalent hexadecimal number = $(100)_{16}$

Illustrative Example : A decimal number $(35)_{10}$ can be converted into binary number by first dividing 35 with base 2 to give quotient 17 and a remainder 1. The quotient 17 is again divided by 2 to give quotient 8 and remainder 1. This process is repeated till the quotient becomes 0.

DIVISION	QUOTIENT	REMAINDER (in HEX)
35 / 2	17	1
17 / 2	8	1
8 / 2	4	0
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

Answer= $(100011)_2$

For Fractional Part : Any fractional number in decimal number system can be converted to base r number system by multiplying the decimal fractional part with the base r and take its integer part. Again multiply the remaining decimal fractional part by base r and take out the integer part. Repeat this process until it became 0. The integer part of the results of every step is the equivalent fraction number in base r number system. This procedure can be performed by following these steps:

1. Multiple the decimal fractional part by base r.

2. Take the integer part of the result in an array. Note that, the integer part of the result will be between 0 to r-1.

3. Multiply the remaining decimal fractional part by base r.

4. Repeat steps 2 and 3 until the number became zero.

5. The number in base r is the sequence of the integer part taken each time starting from first to last.

For example, fractional number $(0.06640625)_{10}$ can be converted into the equivalent hexadecimal number by above procedure:

Multiplication	Resultant integer part
0.06640625 * 16=1.0625	1
0.0625 * 16 =1.0	1
0 * 16=0.0	0

Equivalent hexadecimal fractional number is $(0.110)_{16}$.

Illustrative Example: Convert the decimal number $(0.513)_{10}$into octal number.

Multiplication	Resultant integer part
0.513*8= 4.104	4
0.104*8= 0.832	0
0.832*8=6.656	6
6.656*8=5.248	5
5.248*8=1.984	1
1.984*8=7.872	7

Equivalent octal fractional number is $(0.406517)_{8}$.

Check Your Progress

1. Find the octal equivalent of the decimal number $(0.83)_{10}$.

2. Convert the decimal number $(512)_{10}$ to octal number.

1.6.3 BINARY NUMBER TO OCTAL, HEXADECIMAL NUMBER AND VICE-VERSA

Binary to octal and hexadecimal conversion :

In octal number system, there are only 8 digits (0-7) while the hexadecimal number system has 16 digits (0-9, A-F). We can represent any octal digit by using only 3 binary bits because 2^3 is equal to 8. While, any hexadecimal digit can be represented by only 4 bits (2^4=16). So, we can convert any binary number to equivalent hexadecimal number by grouping every 4 binary bits starting from left to right for integer part and every 4 bits of the fractional part is grouped starting from right to left. Each group of 4 bits is then assigned corresponding octal digit. This procedure can be summarized with the following steps :

1. Take binary number and starting from the binary point, groups every 4 bits while moving both ends. The groups formation take place starting from right to left for integer part and from left for fraction part.

2. Additional zeros can be added to the leftmost and rightmost bit, if requires in groups formation.

3. Convert each group of 4 bits to corresponding hexadecimal digit.

The conversion from binary to octal number is same as above procedure except the binary bits are divided into groups of 3 bits.

Illustrative Example : Convert the following binary numbers to octal and hexadecimal numbers.

i) Binary to octal number

$$(110011001.110101)_2 = \frac{110}{6}\frac{011}{3}\frac{001}{1} . \frac{110}{6}\frac{101}{5}$$

$= (631.65)_8$

Binary to hexadecimal number

$$(110011001.110101)_2 = \frac{0001}{1}\frac{1001}{9}\frac{1001}{9} . \frac{1101}{D}\frac{0100}{4}$$

$=(199.D4)_{16}$

ii) Binary to octal number

$$(10011001.11011)_2 = \frac{010}{2}\frac{011}{3}\frac{001}{1} . \frac{110}{6}\frac{110}{6}$$

$= (231.66)_8$

Binary to hexadecimal number

$$(10011001.11011)_2 = \underset{9}{1001}\ \underset{9}{1001} . \underset{D}{1101}\ \underset{8}{1000}$$

$= (99.D8)_{16}$

Octal and hexadecimal to Binary conversion: An octal number can be converted into equivalent binary number by converting each octal digit to corresponding 3 bits binary number. In case of hexadecimal to binary number conversion, each hexadecimal digit is replaced with its 4 bits binary number.

This conversion procedure is illustrated with the following examples.

Example 1 : Convert the following octal numbers to binary number.

$$(6357)_8 = \underset{110}{6}\ \underset{011}{3}\ \underset{101}{5}\ \underset{111}{7}$$

$=(110011101111)_2$

Example 2: Convert the following hexadecimal numbers to binary number.

$$(2DF5)_{16} = \underset{010}{2}\ \underset{1101}{D}\ \underset{1111}{F}\ \underset{0101}{5}$$

$=(10110111110101)_2$

Check Your Progress

1. Convert the hexadecimal number $(8BF5)_{16}$ to binary number.
2. Express the binary number $(10011001.11011)_2$ in its equivalent octal number.

1.7 SIGNED BINARY NUMBERS

1's Complement : In a binary number, if 0 is replaced with 1 and 1 is replaced with 0, then the number thus formed is the 1's complement of a given binary number. Both, the binary number and the number formed by 1's complement are complement of each other i.e. if one number is positive, the other number is negative with the same magnitude. For example, $(0101)_2$ represents +5, while its 1's component $(1010)_2$ represents -5.

2's Complement : In the 1's complement of a binary number, if 1 is added then the resulting number thus formed is 2's complement of the given binary number. For example, 2's complement of a binary number $(0101)_2$ is $(1011)_2$. Here, $(0101)_2$ represents +5 and $(1011)_2$ represent -5 in 2's complement.

Signed Binary Number : Generally, positive or negative integers are represented and written with plus sign (+) or minus (-) sign before the number. Positive integers including zero can be represented in binary form as unsigned binary numbers. But, due to the limitations of computer, it has only two digits 0 and 1 to represent any positive or negative number. So, a signed bit is placed at the leftmost position of a

binary number to indicate whether the number is positive or negative. The convention is to place sign bit 0 for positive number and sign bit 1 for negative number. In a signed binary number, the leftmost bit represents sign of the number and the rest bits represent the number. For example, 10111 is the signed binary number representation of -7, while it represents 23 in unsigned binary number. This representation of positive and negative numbersis known as signed magnitude system.

Signed Complement System : The signed magnitude number representation is generally used for general arithmetic operations. When these arithmetic operations are implemented in computers, a different system is used for representing positive and negative integers called as signed complement representation. In this system, a negative number written in signed magnitude system is represented by its complement. There is only one way to represent any positive integer. For example, +7 can be represented with 8 bits in signed magnitude system by a signed bit 0 followed by the binary equivalent of 7 i.e. 00000111. However, there are three different ways to represent -7 :

1. In signed magnitude representation: 10000111.
2. In signed 1's complement representation:11111000 (1's complement of signed magnitude representation of +7 i.e 00000111).
3. In signed 2's complement representation: 11111001 (2's complement of signed magnitude representation of +7 i.e 00000111).

Table 1.2 shows signed magnitude, 1's complement and 2's complement representation of different integer using 4 bits.

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Table 1.2 – Signed Binary Number Representation.

1.8 SUMMARY

In summary, you understand

- A binary number system expresses any number using only two symbols: 0 and 1.
- The octal number system represents any number using 8 only symbols: 0,1,2,3,4,5,6,7.
- The hexadecimal number system expresses any number by using 16 distinct symbols. These symbols include numeric digits 1-9 and alphabets A-F (for numbers 10 to 15).
- Any number in one number system can be converted to its equivalent number in another number system.
- In a digital system, any positive integer is represented in only one way which is signed magnitude system.
- In digital systems, a negative number can be represented in three different ways: signed magnitude, 1's complement and 2's complement representation.

1.9 TERMINAL QUESTIONS

1. Convert the following binary numbers to octal and hexadecimal numbers.
   a. 10111011
   b. 010110.10101
   c. 110010.011
   d. 100011.101
2. Convert the following decimal numbers to indicated number system:
   a. 175.175 to binary
   b. 98.25 to hexadecimal
   c. 0.6875 to octal
3. Find the 1's and 2's complement of following 8 bits binary numbers:
   a. 10111001
   b. 00111010

c. 01011011

d. 11011110

4. Find the equivalent decimal number of the following numbers:

a. $(647.12)_8$

b. $(365.75)_8$

c. $(4B.2C)_{16}$

d. $(57D.23F)_{16}$

UNIT-2 BOOLEAN ALGEBRA AND LOGIC GATES

Structure

2.1 INTRODUCTION

We all are familiar with digital systems such as computers, calculators and smart watches which have become an important part of our daily life. These devices works on the principle of digital electronics. In a digital system, inputs given to it are converted into digital signals which has two discrete values: High and Low. The circuits present inside the calculator which process these signals are known as digital circuit. The introduction of microprocessor by Intel Corporation gives tremendous power to digital devices which results in significant progress in digital electronics theory. Now, the digital electronic theory has become a significant part of modern digital systems whether it is computers, cars, scientific and medical instruments, domestic or defence equipments. Some of the important reasons for the widespread applications of digital electronic theory are:

- It requires Boolean algebra which is easy to understand.
- Devices used in a digital circuit operate on a digital signal which has only two values: High and Low.
- There are a number of ICs available for performing various operations. These ICs are fast, reliable and small in size.
- Digital circuits have the capability of memory which makes it suitable for digital systems such as calculators and watches.

2.2 OBJECTIVES

After studying this unit:

1. You familiar with Boolean algebra is related to designing computer logic.

2. You understand basic logic gates.

2.3 BOOLEAN ALGEBRA

A digital system operates on signalswith two discrete values Low and High which are also represented by the binary digits 0 and 1 respectively. A binary digit 0 or 1 is called as bit. The binary number system is used for design and analysis of digital systems. George Boole introduced the concept of binary number system and developed an algebra for it which is known as Boolean algebra. The logic concepts described in Boolean algebra are used in design of digital systems. Any digital system requires only a few basic operations which includes NOT, AND, OR and FLIP FLOP. These operations are performed a number of times based on the complexity of the digital system.

A Boolean Algebra is defined on a set of two elements, B={0,1} which support two binary operators and one unary operator: + (OR) , . (AND) ,'(NOT). A binary variable is represented with an alphabet symbol such as A,B,X,Y,a,b,c…..and it can take either 0 or 1 value. The operations on binary variables with these operators are shown in Figure 2.1.

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 2.1- Boolean Operators

These operators are exactly same aslogical OR, AND and NOT which can be describe as:

AND: It produces output 1 if **all** inputs are 1 otherwise 0.

OR: It produces output 1 if **any** input is 1 otherwise 0.

NOT: It gives **inverse** of a variable.

2.3.1 BASIC THEOREM AND PROPERTIES OF BOOLEAN ALGEBRA

Duality : The principle of duality is an important property inBoolean algebra which state that any Boolean identity will remain valid even if we interchange operators OR and AND and replace 0 by 1 and 1 by 0. Table 2.1 shows various identities of

Boolean algebra in pairs as (a) and (b). Each part of a pair can be obtained from other by using duality principle i.e. interchanging OR with AND, AND with OR, 0 with 1 and 1 with 0.

Basic Theorems : There are 10 basic identities which governs Boolean algebra. These basic identities are shown in Table 2.1.You should remember these identities as soon as possible in order to understand Boolean algebra.

1.	A + 0 = A	A · 1 = A	**identity**
2.	**A + A' = 1**	**A ·A' = 0**	complement
3.	**A + A = A**	**A · A = A**	
4.	**A + 1 = 1**	**A · 0 = 0**	
5.	**(A')'= A**		involution
6.	**(A + B) ' = A' · B'**	**(A· B) ' = A' + B'**	de Morgan's theorem
7.	**A + (A · B) = A**	**A · (A + B) = A**	absorption
8.	**A + B = B + A**	**A · B = B · A**	commutative law
9.	**A + (B + C) = (A + B) + C**	**A · (B · C) = (A · B) · C**	associative law
10.	**A + (B · C) = (A + B) · (A + C)**	**A · (B + C) = (A · B) + (A · C)**	distributive law

Table 2.1- Postulates and theorems of Boolean algebra.

2.3.2 BOOLEAN FUNCTIONS

We have already seen that a Boolean variable can take either 0 or 1 value. A Boolean function is an algebraic expression which is formed by binary variables (such as x,y and z), operators (NOT, AND, OR) and parentheses. For example, consider the following algebraic expression:

F1= x'yz

The output of this algebraic function will be 1 if the values of Boolean variables x'=1,y=1,z=1 otherwise, its output will be zero.

The output of an algebraic function can also be represented witha truth table as shown in Table 2.2. Consider another Boolean functions F1=xyz' and F2=x+y'z.

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

Table 2.2- Truth tables for F1=xyz', F2=x+y'z.

To represent these Boolean functions with truth table, we need to list all possible combinations of the Boolean variables involved in the expression. Each Boolean function involves three Boolean variables which gives 2^3=8 different combinations of 0 and 1. These combinations are listed in columns of x,y and z. In general, any Boolean function which involves n Boolean variables gives 2^n different combinations of 0 and 1. For each combination of Boolean variables x,y,z the column F1 can take either 0 or 1 value. This way, the truth table helps in analysing output of a Boolean function for each possible input values of the Boolean variables.

2.3.3 OPERATOR PRECEDENCE

Any Boolean function or expression expressed in terms of Boolean variables and operators can be simplified with following operator precedence.

1. Parentheses
2. NOT
3. AND
4. OR

This means that for any Boolean expression, an expression inside parentheses must be evaluated first. Then we follow complement operator and after that AND operator is evaluated. Finally, the OR operator is evaluated in the end.

2.3.4 COMPLEMENT OF A BOOLEAN FUNCTION

The complement of a Boolean function can be obtained by using DeMorgan's theorem which we have seen in previous section in Table 1.1. The pair

of identities of DeMogans listed in Table 1.1 are for only two variables. We can extend these identities for more than two Boolean variables and can find complement of any Boolean function. For example, consider a Boolean function (A+B+C)' as given below.

(A+B+C)'

= (A+X)' assume X= B+C

= A'.X' by using DeMorgan's theorem 5a

= A'.(B+C)' substitute the value of X = B+C

= A'.B'.C' by using DeMorgan's theorem 5a

Similarly, we can show

(A.B.C)'=A' + B '+C'

In general, DeMorgan's theorem states that the complement of any Boolean expression can be obtained by interchanging + with .and .with + and complementing each Boolean variables.

2.3.5 OTHER LOGIC OPERATIONS

We have seen that, a Boolean function which involves only two Boolean variables has $2^n=2^2=4$ different combinations of inputs. This results in $2^4=16$ different functions with two binary variables which are shown in Table 2.3.

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 2.3 - Truth table for 16 different functions possible with two Boolean variables.

The Boolean expression for each of the 16 functions are shown in table 2.4 in columns F0,F1,F2,.....F15. There are 16 different functions with Boolean two variables because we can form 16 different combinations with 4 different input combinations. These 16 functions can be divided into three categories as follows:

1. Two functions which produce either all 0 (null operation) or all 1 (identity operation).

2. Four functions with unary operations: two complement (x' and y') and two transfer operations (x and y).

3. Ten functions with binary operations which are AND, OR, NAND, NOR, exclusive-AND, exclusive-OR, 2 inhibition and 2 implications.

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Table 2.4 - Boolean expressions for 16 different Boolean functions

Check Your Progress

1. Find complement of following Boolean functions.

   i) (b'+d')ac+e ii) (x'y+z)u+v'

2. Simplify the following Boolean expression to a minimum number of variables:

   a) xy + xy′

   b) (x + y)(x + y′)

   c) xyz + x′y + xyz′

2.4 LOGIC GATES

Electronic digital circuits are also known as switching circuits because they behave like switch with the active element such as transistor either conducting or not conducting. An electronic digital circuits use binary signals to control conduction and non-conduction of an active element. Throughout a digital system, electrical signals such as voltages and currents exist in two voltage levels. For example, a voltage operated circuit responds to either of two different voltage levels: logic 0 for 0 volt (with allowable tolerance of -0.5 to 0.5) and logic 1 for 3 volts (with allowable tolerance of 2 to 4 volts). The input terminals of a digital circuit accept binary signals with allowable tolerance and produce binary signal as output at the output terminals with the specified tolerances. An electronic digital circuits are also called as logic circuits because any desired computation can be performed by passing binary signals through different combinations of logic circuits. For example, multiple combinations of logic circuits can be used to perform storage of binary number (storage circuit) or manipulation (computing circuit). Each signal represents a Boolean variable which carries 1 bit information. Logic circuits which perform logical AND, OR and NOT along with some other operations are shown in Figure 2.3.These circuits are also called as logic gates or digital circuits or switching circuits. These logic gates are blocks of hardware which produce 0 or 1 binary signal based upon their input logic signals. For two input logic gates, their input signals x and y at any time can have one of the four combinations: 00,01,10,11. For example, Figure 2.2 shows input signals x,y and output response of AND, OR, NOT logic gates for different combinations of input at various times.

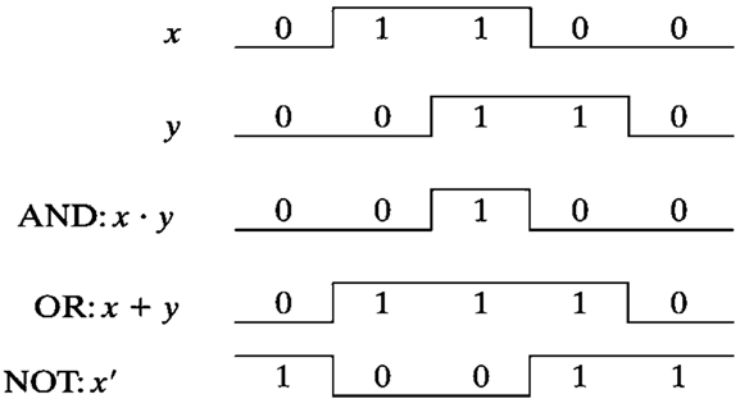


Figure 2.2 – Input/output signals for AND, OR and NOT gates.

These logic gates are used to implement Boolean functions in digital systems. Out of 16 Boolean functions, only 8 functions are used as standard logic gates for digital circuit design. This is because 2 functions are constant and 4 functions are repeated twice. Now, out of 10 functions, two functions (implication and in habilation) are impractical for circuit design because these two functions do not follow either commutative or associative property. So, we are now left with only 8 functions suitable for logic gates which are: AND, OR, NAND, NOR, exclusive-AND, exclusive-OR, Complement and Transfer.

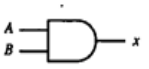
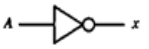
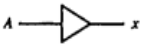
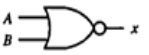
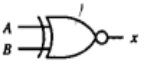
Name	Logic Circuit	Algebraic function	Truth table	Output
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	It produces output 1 if **all** inputs are 1 otherwise 0.
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	It produces output 1 if **any** input is 1 otherwise 0.
Inverter		$x = A'$	A | x 0 | 1 1 | 0	It gives **inverse** of a variable.
Buffer		$x = A$	A | x 0 | 0 1 | 1	It gives same output as the input.
NAND		$x = (AB)'$	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	It gives output 0 if all of its inputs are 1, otherwise it gives output 1.
NOR		$x = (A + B)'$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	It produces output 1 If none of its inputs are 1 otherwise 0.
Exclusive-OR (XOR)		$x = A \oplus B$ or $x = A'B + AB'$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	It gives output 1 if the number of 1 inputs is odd, otherwise 0.
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1	It gives output 1 if all of its inputs are 1 or if all of its inputs are 0 otherwise it gives 0 output.

Figure 2.3- Various logic gates along with their truth tables.

The various logic gates along with their truth tables are shown in Figure 2.3. Each logic gate has one or two input variable and produce one binary output x.

2.5 IMPLEMENTING CIRCUIT FROM BOOLEAN FUNCTION

A Boolean function may be implemented with logic gates such as AND,OR and NOT. For example, consider four Boolean functions:

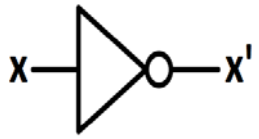
F2=x+y'z

F3=x'y'z+x'yz + xy'

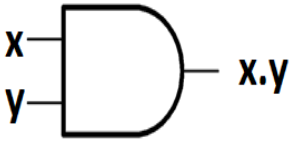
F4= xy' + x'z

The implementation of these four functions F2, F3 and F4 with logic gates are shown in Figure 2.4.

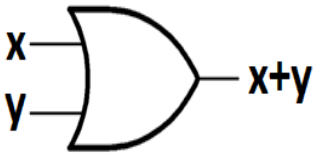
- For every variable which appear in complement form, a NOT gate is used. For example,



- For each product term in function, an AND gate is used. For example,



- To combine two or more terms an OR gate is used. For example,



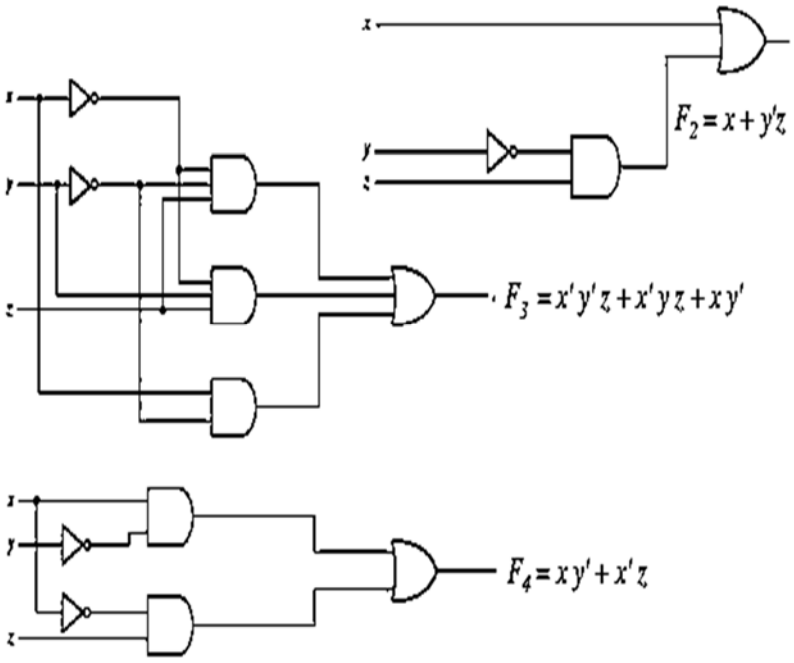


Figure 2.4 –Implementation of Boolean functions F2, F3 and F4 with logic gates.

Check Your Progress

1. Implement the following Boolean functions to circuit using logic gates.

   i) (a+b)'

   ii) abc+abc'+a'b'c

2.6 INTEGRATED CIRCUIT

A Digital circuit is made with Integrated Circuit which is abbreviated IC. An integrated circuit is a small semiconductor crystal called as chip which contains electronic components for logic gates. The logic gates inside the chip are interconnected to form required circuit. The chip is mounted on plastic or ceramic platform and its connections are welded to external pins to form the integrated circuit.

The number of pins vary from 14 pins in small IC package to 64 or even more in large IC package. The size of IC packages are very small. For example, four AND gates can be embedded inside 14 pins IC package with dimension of 20*8*3 millimeters. An entire microprocessor can be embedded inside 64 pins IC package with dimension 50*15*4 millimeters. Some SSI ICs are shown in Figure 2.5. The gates in each IC are embedded within 14 or 16 pins package. A small cut on the left side of each IC package is used to reference pin numbers. The pin numbering starts from the cut mark and proceeds further in anticlockwise direction. The inputs and outputs of gates are directly connected to package pins as shown in Figure 2.5.

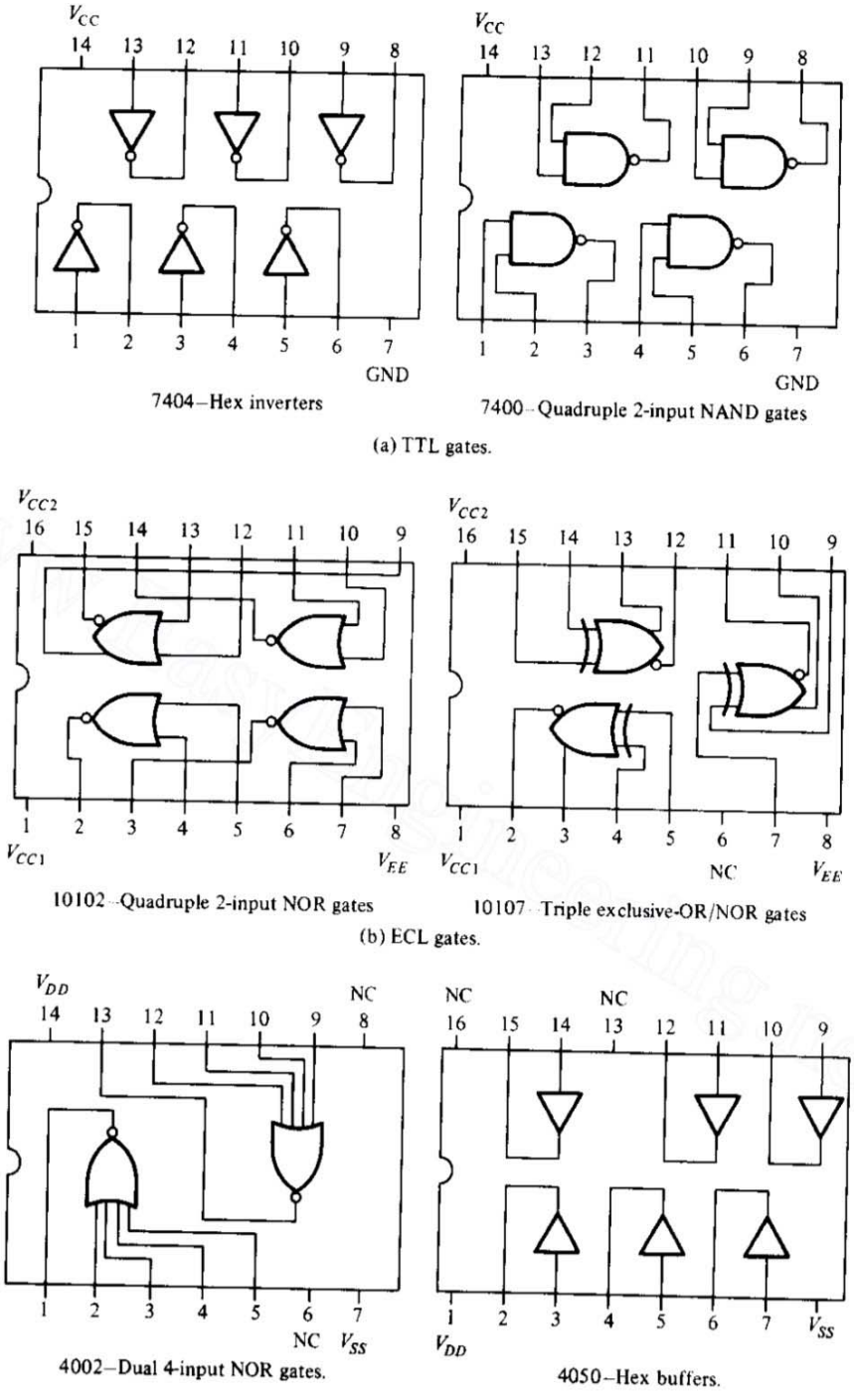


Figure 2.5 Integrated Circuits with 14 or 16 pins.

Integrated Circuits are commonly classified on the basis of their circuit complexity which is measured by number of logic gates enclosed in a chip.

Small Scale Integration (SSI) : It contains several independent gates in a single package as a chip. The inputs and outputs of gates are directly connected to pins in the package. The SSI contains number of gates fewer than 10. The number of gates are limited by the number of pins available in the IC.

Medium Scale Integration (MSI) : It contains 10 to 100 gates within a single package. It is basically used for performing elementary digital operations such as adder, decoder and multiplier.

Large Scale Integration (LSI) : It contains 100 to a few thousand gates inside a single package. It is used for processors, memory chips etc.

Very Large Scams Integration (VLSI) : It contains thousands of gates in a single package. It is used for complex microprocessor chips and large memory arrays.

2.7 POSITIVE AND NEGATIVE LOGIC

Positive and Negative logic: A binary signal at input and output of a logic gate can be either logic 0 or logic 1. These two signal values are also represented by two signal levels: H for higher signal level and L for lower signal level. When we use higher signal level H to represent logic 1, then it is called positive logic system. On other hand, when we use lower signal level to represent logic 1, it is called as negative logic system. These two logic systems are shown in Figure 2.6.

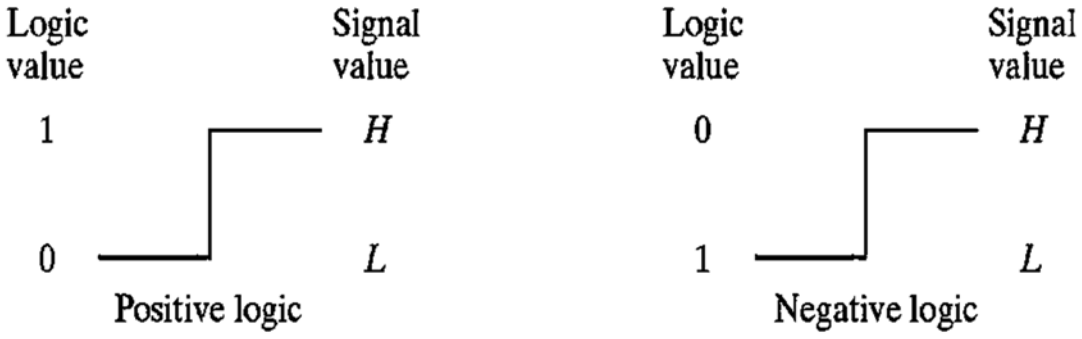


Figure 2.6 – Assignment of positive and negative logic system.

The conversion from positive logic system to negative logic system and vice-versa, can be done by changing 0 to 1 and 1 to 0 in both inputs and output of a logic gate. This operation results in dual of a function. So, the AND operation is converted to OR operation and the OR operation is converted to AND operation. For example, let us see how a logic gate behave in positive logic system and negative logic system. Consider a logic gate which produce high signal level only if all its inputs are high signal levels, otherwise it produces low signal level. The truth table the gate in positive logic system is shown below in Figure 2.7 :

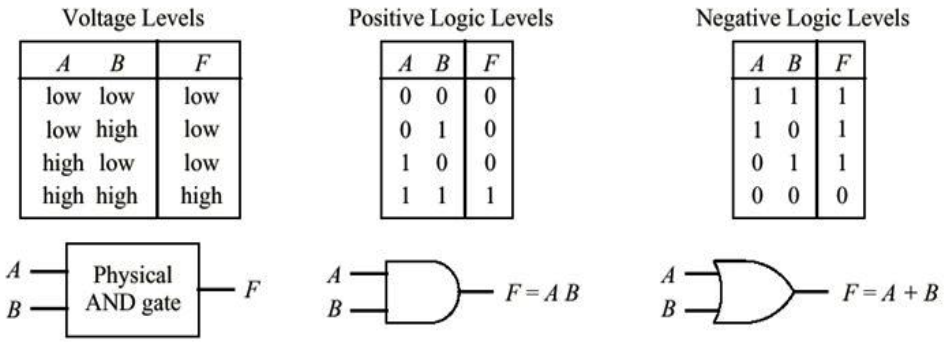


Figure 2.7-Illustration of positive and negative logic system.

From the truth table, it is clear that the gate behaves as AND gate in positive logic system. Now, let us see the truth table of the gate in negative logic system which is shown in Figure 2.7. The truth table clearly illustrates that the gate behaves as OR gate in negative logic system.

2.8 SUMMARY

- We discussed Boolean algebra which supports operations + (OR), . (AND), ' (NOT) on Boolean variables with either 0 or 1 as their value.
- We have seen the principle of duality where any Boolean identity remains valid even if we interchange operators OR and AND and replace 0 by 1 and 1 by 0.
- We get familiar with various basic theorems to analyse and manipulate any algebraic expression.
- We learned about Boolean functions and demonstrate how to construct truth table from any Boolean function.
- We discussed various logic gates which implement different types of operations on Boolean variables.
- We illustrated how to implement circuit from any Boolean function.
- We saw that a positive logic system uses higher signal level H to represent logic 1 and a negative logic system uses lower signal level L to represent logic 1.

2.9 TERMINAL QUESTIONS

1. What are various applications of digital electron theory?
2. Explain different functions possible with two binary variables.
3. Describe how you implement Boolean functions.
4. How do you find complement of a Boolean function?
5. Define Boolean function with a suitable example.

6. What do you mean by positive and negative logic system?
7. Implement the following Boolean functions to circuit using logic gates.
   i) ab+ab'
   ii) (a+b).(a+b')
8. Find dual of following expressions.
   i) x+0=x
   ii) x+x'=1
   iii) x+x=x
   iv) x+1=1
9. Find complement of following Boolean functions.
   i) b'd'+bd
   ii) (c+b'd)(c+b+d')(c'b'+d)
10. Write the truth tables of following logic gates.
   i) AND
   ii) NAND
   iii) EX-OR
   iv) NOR
   v) Ex-NOR
11. Draw the truth table of the following functions:
   a) (xy + z′)
   b) (A′B′C)

UNIT-3 REDUCTION TECHNIQUES

Structure

3.1 INTRODUCTION

The complexity of a digital logic circuit is directly dependent on the complexity of the Boolean function which implements it. A minimized Boolean function requires minimum number of gates, decreases the cost for its implementation into logic circuits and is more reliable. Boolean functions can be simplified algebraically using Boolean identities as discussed in previous unit. But, this process is lengthy and error prone when the number of Boolean variables increases. Also, this process does not give consistent result always. In this unit, we will discuss a simple and powerful technique for simplification of any Boolean function which is called as Karnaugh Map or simply K-map. It is a graphical way of minimizing a Boolean function which represent its truth table in 2 dimension.

3.2 OBJECTIVES

After studying this unit you should able to

- Understand minterms and maxterms and express any function in standard form: SOP and POS form.
- Find complement of any function expressed in SOP or POS form and interconversion between SOP and POS form of a function.
- Simplify any Boolean function expressed in SOP or POS form using k-map.

3.3 MINTERMS AND MAXTERMS

A function expressed in a standard form can be minimized by K-Map. We can directly apply the procedure of K- Map on the standard form of a function. A standard form of a function contains either minterms or maxterms.

Minterms : Any Boolean variable can be in two forms: normal form (x) and complement form (x'). Consider, two Boolean variables x and y which are combined by an AND gate. Since each Boolean variable may appear as normal form and complement form, there are four combinations possible with these two Boolean variables: x'y',x'y,xy' and xy. Each of these combinations formed with AND operation is called a minterm. In general, with n Boolean variables, there are 2^n different combinations possible which results in 2^nminterms. The process for obtaining these 2^nminterms is illustrated in Table 2.3. The columns of Boolean variables x,y,z correspond to all possible inputs with three Boolean variables which can be seen as binary equivalents of decimal numbers from 0 to 2^3-1. Each minterm in the column term can be obtained by combining the inputs x,y,z such that each Boolean variable appears in complement form if its value is zero otherwise it appears in normal form if its value is 1. Each minterm is denoted with symbol m_j, where j corresponds to the decimal equivalent of the binary number formed from inputs x,y,z.

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Table 3.1 - Minterms and maxterms for three binary variables.

Maxterms : Similarly, n Boolean variables can be combined with OR operation with each variable may appear in complement or normal form and this will result in 2^n maxterms. Each maxterm can be obtained by combining the input Boolean variables x,y,z with OR operation such that each variable appears in complement form if its value is 1 otherwise it appears in normal form if its value is 0. For

example, Table 3.1 shows 8 maxterms possible with 3 Boolean variables x,y,z in column Term. Thesymbolic representation ofeach maxterm is shownwith Mj (where j is decimal equivalent of binary numberformed from x,y,z) in Designation column. From the Table 3.1, we can notice that, each minterm is complement of its corresponding maxterm and vice-versa.

$$m_j' = M_j$$

3.4 STANDARD FORM OF A BOOLEAN FUNCTION

Any Boolean function can be expressed in a standard form. The standard form of a function contains either all minterms or all maxterms. There are two standard forms for representation of a Boolean function: Sum of Product and Product of Sum.

Sum of Products (or Sum of minterms) : Any Boolean function can be expressed using minterms by combining each minterms with OR operation which corresponds to output logic 1 of the function. This way of a function representation is called Sum of Product (or SOP). For example, consider two Boolean functions f1 and f2 whose truth table is shown in Table 3.2.

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 3.2 : Functions of three variables.

The function f1 shown in Table 3.2 can be expressed with sum of minterms x'y'z,xy'z' and xyz because each of these minterms corresponds to output logic 1. This can be written as follow:

f1=x'y'z + xy'z' + xyz= m1+ m4 + m7

$=\sum(1,4,7)$

Similarly, the function f2 shown in Table 3.2, can be expressed as:

f2= x'yz + xy'z + xyz= m3+m5+m6+m7

=∑(3,5,6,7)

Product of Sums (or Sum of maxterms) : The product of sum representation of a Boolean function is a combination of each maxterm which corresponds to output logic 0 with AND operation. For example, the function f1 shown in Table 3.2 can be expressed in POS form as follows.

f1=(x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z)

=M0.M2.M3.M5.M6

=∏(0,2,3,5,6)

Similarly, the POS expression for the function f2 is:

f2=(x+y+z)(x+y+z')(x+y'+z)(x'+y+z)

=M0.M1.M2.M4

=∏(0,1,2,4)

3.5 OTHER USES OF STANDARD FORMS

Complement of a function : If a Boolean function is expressed as Sum of Products (SOP) form, then the complement of this function in SOP form is sum of all minterms missing in the original function. In other words, complement of the function contains all the missing minterms in its original uncomplemented function. Similarly, if a function is expressed in POS form, then complement of the function in POS form is sum of all missing maxterms in its original uncomplemented function.For example, consider the function Fshown in Table 3.3.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.3 Truth table for F=A+B'C

It can be expressed in POS form as:

$F(A,B,C)=\sum(1,4,5,6,7)$

Then, complement of the function F can be written in POS form as:

$F'(A,B,C)= \sum(0,2,3)$

Conversion between SOP and POS : In the previous example, we have seen that:

$F(A,B,C)=\sum(1,4,5,6,7)$

$F'(A,B,C)=\sum(0,2,3)=m_0+m_2+m_3$

Now, if we again take complement of F', we will get back the original function F in a different form:

$F= (F')'=(m_0+m_2+m_3)'$

$=m'_0.m'_2.m'_3$(using DeMorgan's law)

$=M_0.M_2.M_3$ (each minterm is complement of its corresponding maxterm and vice-versa i.e. $mj'=M_j$)

$=\prod (0,2,3)$

A general conversion procedure is interchange symbol $\sum$ and $\prod$ and write only those numbers that are missing in the original form. Please note that, while writing missing numbers, if there are n input Boolean variables, we should consider all 2^nminterms or maxterms. For example, if a function is expressed in SOP form as:

$f(x,y,z)=\sum(1,4,6)$

Then, its POS form can be written as:

$F(x,y,z)=\prod(0,2,3,5,7)$ (since the function contains three input variables x,y,x, there are $2^3=8$ minterms or maxterms possible.)

Conversion to standard forms : A Boolean function expressed in a standard form i.e. SOP or POS has many uses such as for circuit simplification using k-map. But, if a Boolean function is expressed in non-standard form, it must be converted into standard form by using Boolean identities. For example, a Boolean function f5 as shown below is neither SOP nor POS.

f5=(AB+CD) (A'B'+C'D')

It can be converted into SOP form by removing parentheses using distributive law.

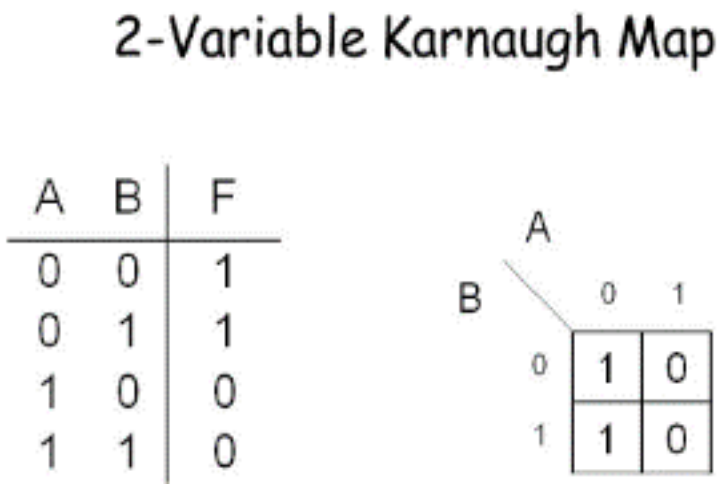
f5=A'B'CD+ABC'D'

Check Your Progress

1. Express the Boolean function $F(x,y,z) = \prod (0,5,7)$in SOP form.
2. Convert the Boolean function $F(a,b,c) = \sum (2,3, 6, 5)$to POS form.
3. Find Complement of following Boolean functions:
   i. $F=\prod (1, 4, 5, 6, 7)$
   ii. $F=\sum (1, 4, 5, 6, 7,15)$

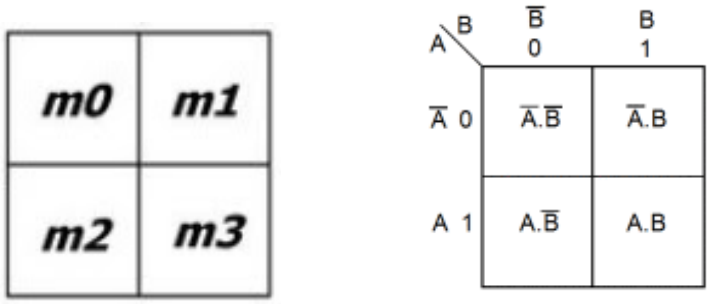
3.6 K-MAP

K- Map is a straight-forward method of minimization of a function by a special arrangement of truth table. For example, the similarity between truth table and k-map for a general case of a function with two Boolean variables is shown below :



Each square (or cell) corresponds to one of the 2^2 outputs in the truth table. Each output which is written in the square corresponds to variable values when seen in its respective row and column in four grids. For example, the square with value 0 at the top right hand corner with column, A=1 and row, B=0 corresponds to third row of the truth table.

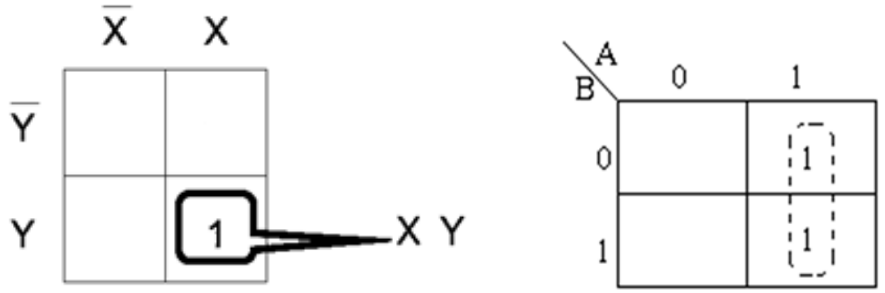
Two Variable Maps:A two variable k-map consists of 2^2 =4 minterms and each minterm corresponds to one square in k-map. Figure 3.1 (a) and (b) show the relationship between each square and its minterm.



a) b)

Figure 3.1- A two variable k-map

If a function is expressed in sum of products (SOP) form, then its k-map contains 1 marked in squares which corresponds to its minterms. For example, the function f= XY is represented by the k-map (as shown in Figure 3.2-a) with the square which belong to mintermxy is marked with 1. As another example, f=AB' + AB is represented by the k-map (shown in Figure 3.2-b) where the squares corresponding to minterms AB' and AB are marked with 1.



(a) (b)

Figure 3.2- (a) K-Map for f=XY (b) K-Map for f=AB' + AB

Three Variable k-maps : Since, three variables consist of 2^3=8 minterms, its k-map consists of 8 squares. Figure 3.3 (a) and (b) show relationship between each square and its minterm. For example, the square which is assigned minterm m_6 corresponds to the row marked x and column marked yz'. When these two terms are concatenated, they give minterm m_6 = xyz' for the square.

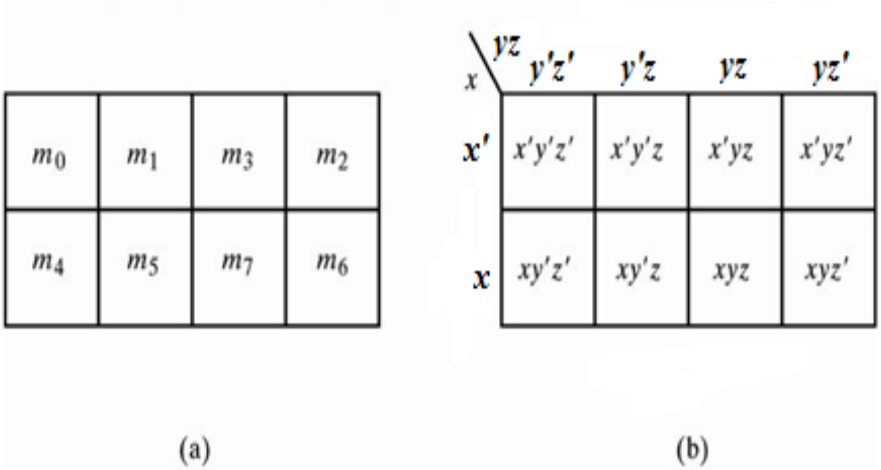
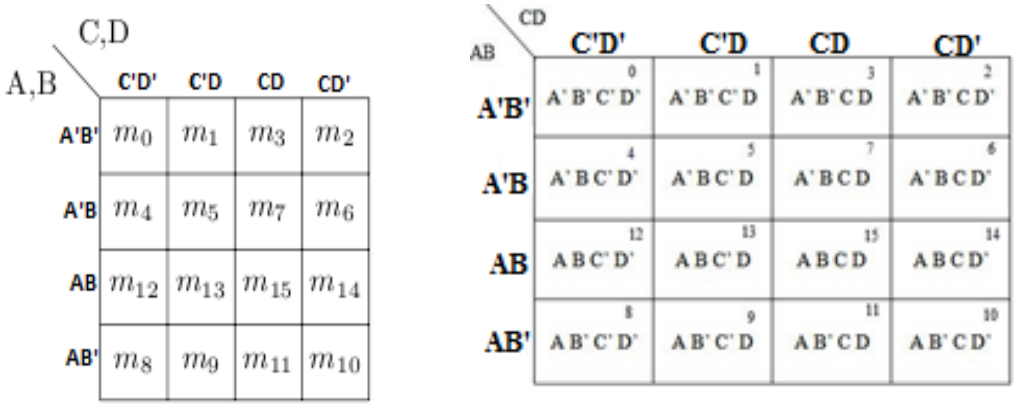


Figure 3.3 – (a) A three variable k-map with minterms notations (b) A three variable k-map containing minterms expression with variables.

Four variable k-map : A four variables k map consists 2^4 =16 minterms. Figure 3.4(a) and (b) show 4 variables k map and relationship of each minterm to its cell. For example, the cell which is assigned m_{10}corresponds to its row marked AB' and column marked CD'. When these two terms are concatenated, they give minterm m_{10} =AB'CD'.



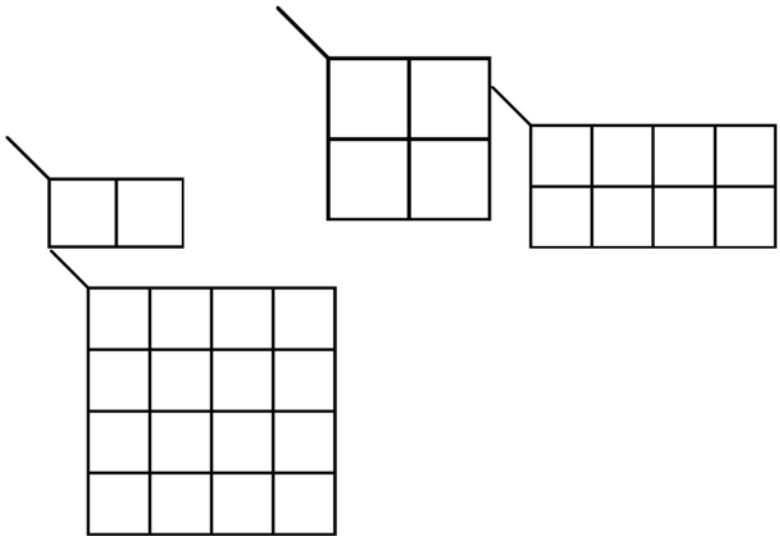
(a) (b)

Figure 3.4 - (a) A four variable k-map with minterms notations (b) A three variable k-map containing minterms expression with variables.

Simplification using k-map :

Now, we will see how k-map can be used for minimization of any Boolean function. Any Boolean function expressed in SOP form can be minimized by using k-map as follows:

1. Choose a k-map according to number of variables in the Boolean function.

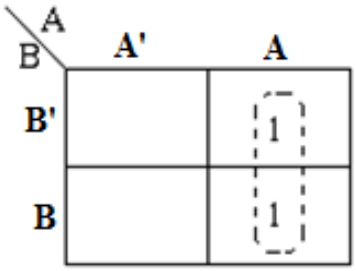


1-variable k-map2-variable k-map3-variable k-map4-variable k-map.

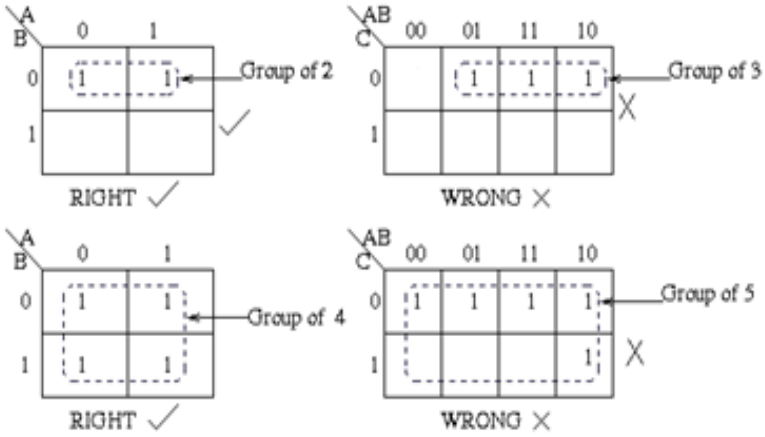
2. Identify minterms from SOP form of the function and put 1 in cells correspond to each minterm.

   For example:

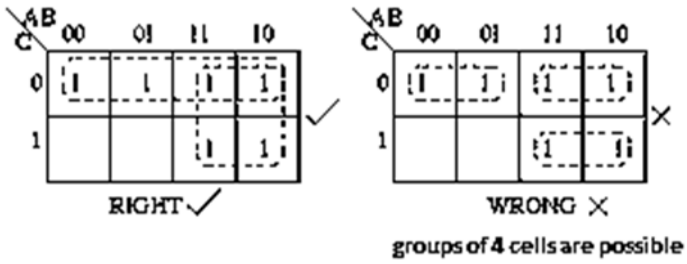
   F(A,B)=AB'+AB

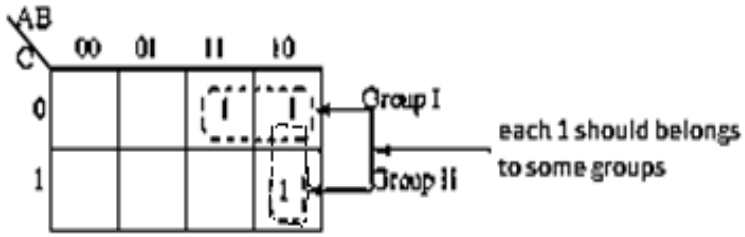


3. Form Groups in power of two i.e 2^n squares or cells such as 1, 2, 4 and 8.

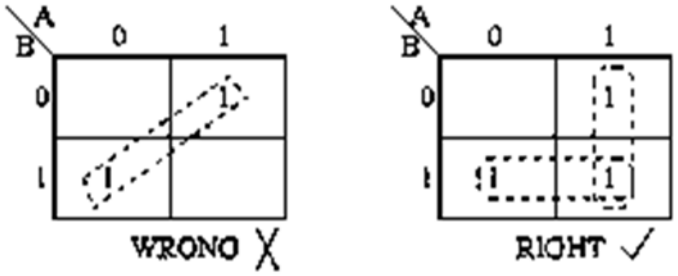


Groups formation should take place from larger groups first till each cell containing 1 belongs to some group. In other words, we should first try to make large groups and if it is not possible then look for small groups as shown below.

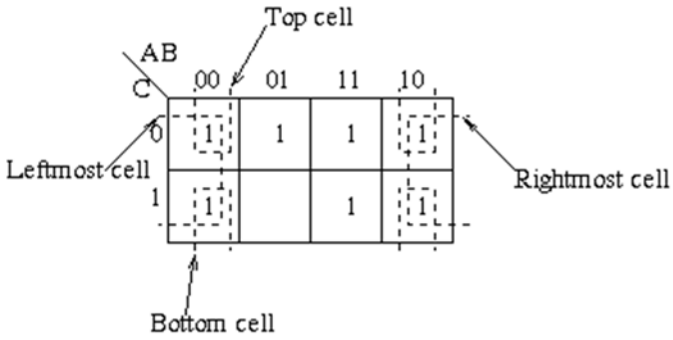




A group must be either horizontal or vertical. A square with 1 may belong to more than one group as shown below.

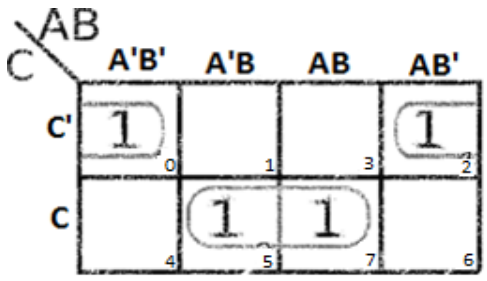


A Group may form from leftmost cells with right most cells containing 1s. A group may also be formed from top cell in a column with the bottom cell in the same column.



4. For each group, find the common variables (that do not changes) along the cells rows and along the cells columns and then concatenate the common variables to form products terms. The product terms thus obtained for each group are summed together to get minimized function in SOP form.

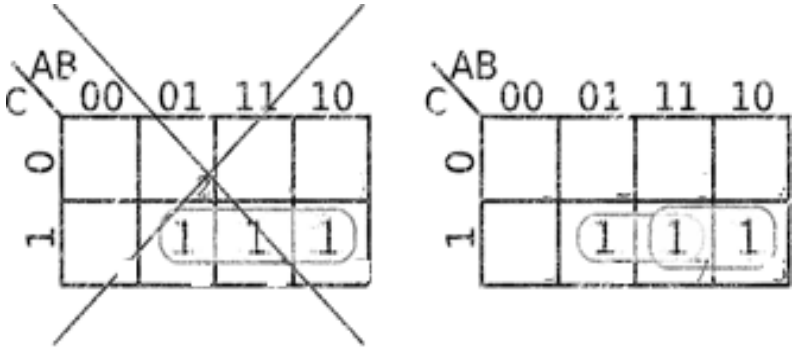
   For example, consider the k-map below.



In this k map, two groups are formed one with cells 0 and 2, while other with cells 5 and 7. For the group with cells 0 and 2, if we see along its row with heading C', there is only C' which is common and does not change. If we look at these cells along their columns headings A'B' and AB', we find that B' is common in them. So, the product term for this group is B'C'. Now, consider the second group formed with cells 5 and 7. These cells correspond to only one row with heading C. Obviously, it is common which does not change. The cells 5 and 7 correspond to two columns with heading A'B and AB. Here, B is common in them which does not change. So the product term for this group is BC. The final minimized function for this k map is: B'C'+BC.

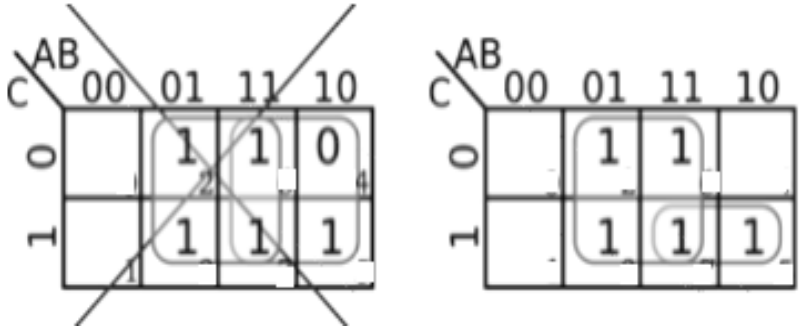
Illustrative example 1: Identify the valid k-map in the following pairs of k- maps:

a.



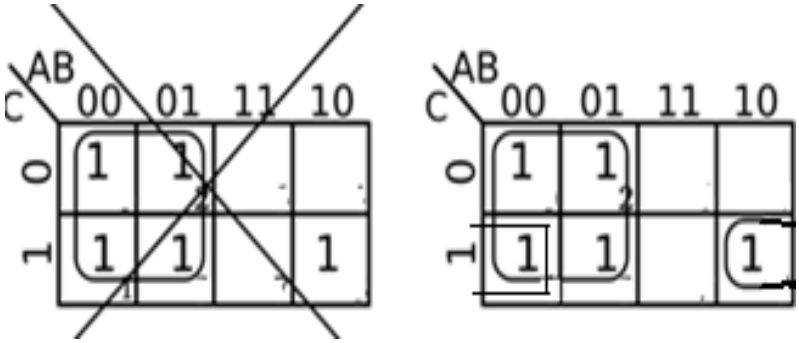
Groups should contain number of cells equal to power of 2. (e.g. 1, 2, 4, 8, 16)

b.



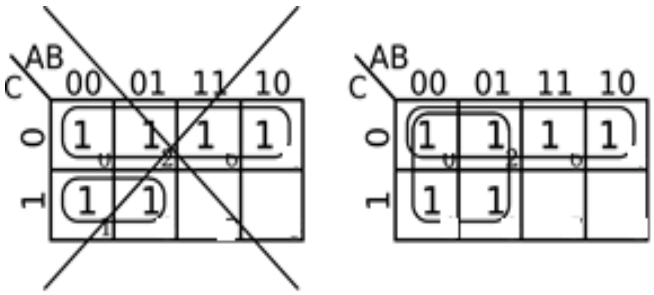
Each group should contain cells containing 1s only - i.e. no cells without 1s

c.



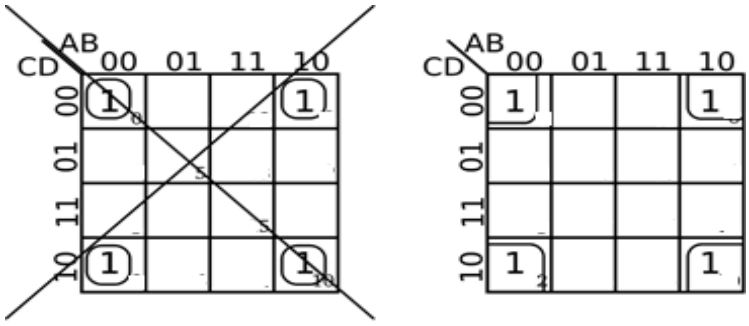
Each cell of 1 must be covered in some group

d.



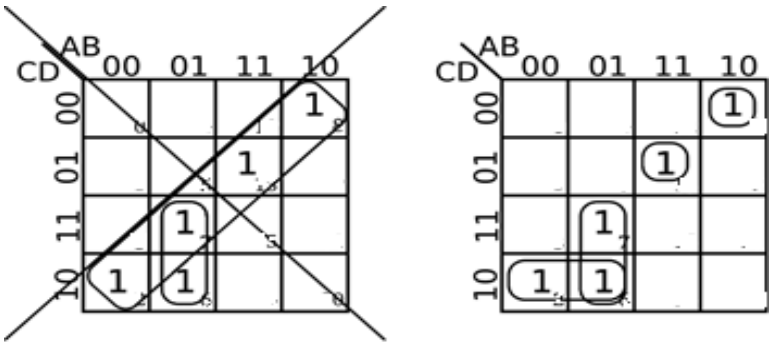
A Group can be overlapped to form larger group if possible.

e.



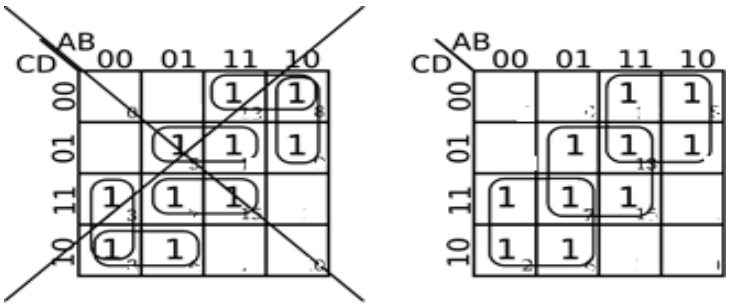
A Group can be formed with corner cells of 4 variable k-map. This wrapping around gives us a large group.

f.



Groups must be formed by covering cells of 1s in horizontal or vertical direction. Groups cannot be made from diagonal cells of 1s.

g.



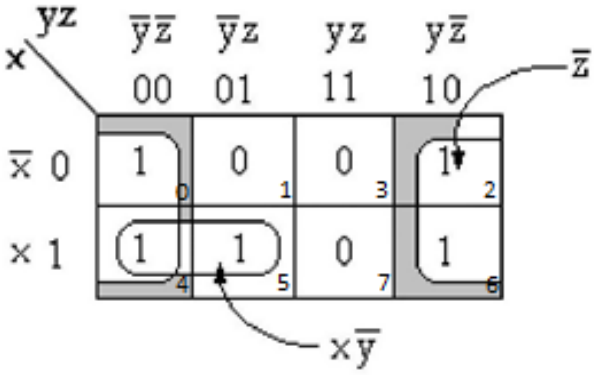
We should always first look to form large groups and if it is not possible then consider other alternatives.

Illustrative example : Simplify the Boolean function expressed in SOP form

F(x,y,z)= ∑(0,2,4,5,6).

Solution:

1. Since the above Boolean function contains 3 variables, draw a k-map with these three variables.



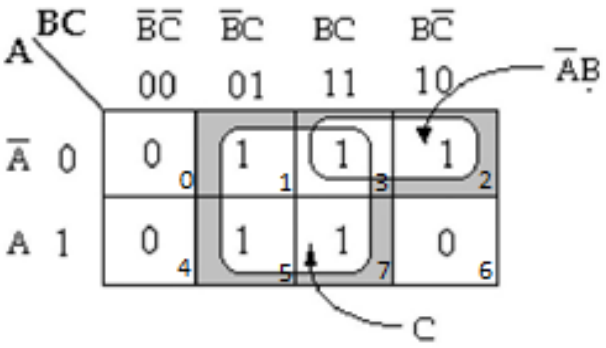
2. The minterms of the function are 0,2,4,5 and 6. Put 1 in cells corresponds to each minterm.
3. The first group can be formed with four cells:0,2,4,6. In this group, no variable is common in its rows x and x'. But, z' is common along its columns y'z' and yz'. So the product term for this group is z'.
4. The second group is formed with two cells: 4,5. In this group, x is common along its only row x, and y' is common along its columns y'z' and y'z. The product term for this group is xy'.

   The POS expression for the simplified function is z'+xy'.

Illustrative example : Minimize the sum of product expression for the following Boolean function:

f(A,B,C)=A'C+B'C+BC+A'BC'

Solution: The Boolean function consists of 3 variables, draw a k-map of 3 variables as given below.



We are given the function in SOP form with variable instead of individual minterms as in previous example. For each term in the SOP expression, identify the cells in the k map that are covered by it and put 1 in corresponding cells. For example, the term A'B covers the cells which corresponds to row A' and all columns containing C i.e. cells 1, 3, 5, 7, so put 1 in these positions. Similarly, the term B'C covers all cells under column name B'C i.e. 1, 5, so put 1 in these positions. And so on, the A'BC' term covers cells corresponding to row A' and column BC' which gives a cell 2, so put 1 in this cell.

1. The first group forms with four cells: 1,3,5,7. Here, no variable is common along this group's row headings A', A and C is common along its columns headings B'C and BC. So the product term for this group is C.

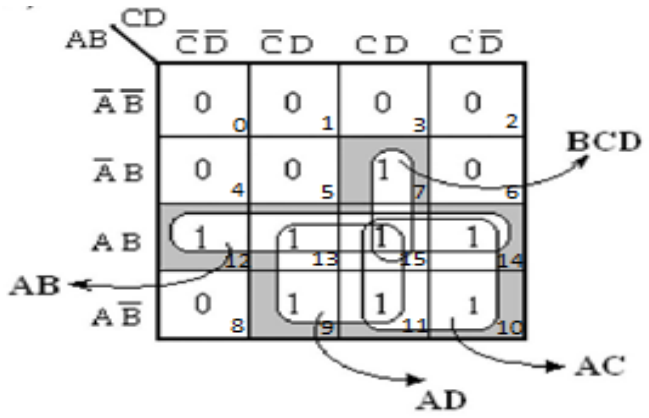
2. The second group is formed with cells 2 and 3. Here, A' is common along its only row A' correspond to this group and B is common along the columns BC and BC' corresponds to this group. So the product term for this group is A'B.

The SOP expression of minimized function is sum of all product terms: C+A'B

Illustrative example: Simplify the following function given in sum of products (SOP) form.

f(A,B,C,D)=A'BCD + ABC' + ABC + AB'D + AB'C

Solution: The above Boolean function consists of 4 variables, so draw a k-map of 4 variables as given below.



The function is given in SOP form with variables instead of individual minterms. For each term in the SOP expression, identify the cells in the k map that are covered by it and put 1 in corresponding cells. The term A'BCD covers all cells which corresponds to row A'B and column CD i.e. cell 7 so put 1 in this position. Similarly, the term ABC' covers all cells under row AB and columns contains C' i.e. 12 and 13, so put 1 in these positions. The term ABC covers all cells corresponding to row AB and columns containing C which are 14,15, so put 1 in

all these positions. And so on, the AB'C term covers cells corresponding to row AB' and columns containing C which gives cells 10, 11, so put 1 in these cells.

1. The first group can be formed with four cells: 12,13,14,15. In this group, AB is common along this group's row heading AB and no variable is common along its columns headings C'D',C'D,CD and CD'. So the product term for this group is AB.

2. The second group can be formed with cells 9,11, 13, 15. Here, A is common along the rows AB, AB' and D is common along the columns C'D and CD. So the product term for this group is AD.

3. The third group can be formed with cells 10,11,14,15. In this group, A is common along its rows AB,AB', and C is common along its columns CD,CD'. So the product term for this group is AC.

4. The last group can be formed with cells: 7 and 15. In this group, B is common along its rows A'B, AB and CD is common along its only column CD. So the product term for this group is BCD.

The SOP expression for minimized function is sum of all product terms: AB+AD+AC+BCD.

Check Your Progress

1. Simplify the following Boolean function expressed in SOP form: f(w,x,y,z)=∑(1,3,7,11,15).
2. Simplify the following Boolean function expressed in SOP form:f(A,B,C)=∑ (0,2,3,4,5).

3.7 DON'T CARE CONDITIONS

The minterms in the SOP form of a Boolean function specify the conditions which make the function value 1 or 0. In other words, a minterm specifies a combination of input Boolean variables which make the function either 0 or 1. This means all combinations of input variables of a function are valid. But for some applications, we don't want the function outputs for certain combinations of input variables. The function with unspecified outputs for some combinations of input variables is called as incompletely specified function and the unspecified combinations of variables or minterms are called as don't care conditions. In a k map, a don't care condition is specified with × sign which means we do not care whether the output of a function is 0 or 1 for this minterm or combination of inputs. In the k map, the don't care values provide opportunity for further simplification of

a Boolean function. The don't care values can be treated as either 0 or 1, whichever results in formation of larger groups.

For example, Figure 3.5 shows valid combination of don't care values × with 1 to form larger groups.

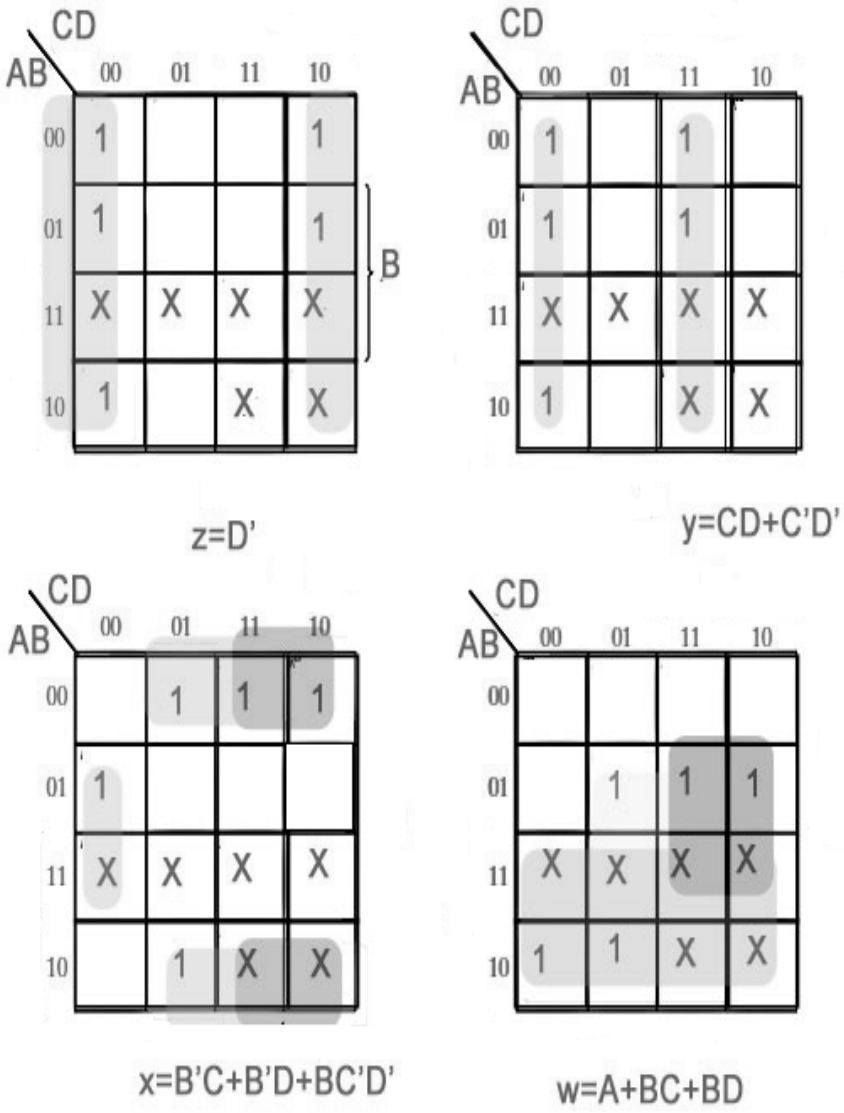


Figure 3.5 - Valid combinations of don't care values × with 1 to form larger groups.

Check Your Progress

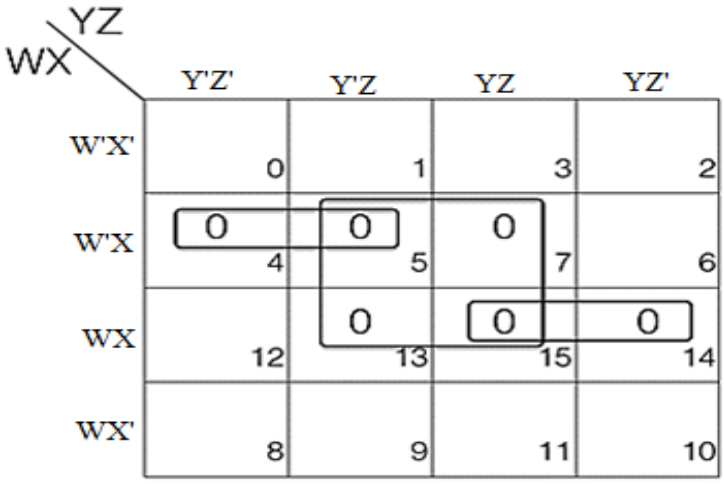
1. Simplify the following Boolean function expressed in SOP form: f(A,B,C,D)=∑(0,2,4, 6,8,10,11,12,13,14,15).
2. Simplify the following Boolean function expressed in SOP form: f(A,B,C,D)=∑(3,7,11,12,13,14,15).

3.8 PRODUCT OF SUM SIMPLIFICATION

Generally, we use k map with SOP form of a function to obtain minimized function in SOP form. The k map can also be used with POS form of a function to obtain minimized function in POS form. This requires the same rules as discussed earlier for SOP simplification. But, here we put 0s in respective cells correspond to maxterms in k map. For each group, the common variables obtained along the row and column are complemented individually and then summed to get sum terms. Finally, the sum terms of each group are combined with AND operation to obtain minimized POS function.

For example, consider a Boolean function expressed in POS form as.

F(A,B,C,D)=∏(4,5,7,13,14,15)



1. First form quad group with cells 5,7,13 and 15. Here, X is not changing along its rows headings and Z is not changing along its columns heading. So, the sum term for this group after complementing common variables individually is X'+Z'.

2. Form second group pair with cells 4 and 5. Here, W' and X is common along its row heading and Y' is common along its columns heading. So the sum term for this group after complementing common variables individually is (W+X'+Y).

3. Form the last group which is a pair with cells 14 and 15. In this group, W and X is common along its row and Y is common along its columns. This group forms the sum term after complementing common variables individually (W'+X'+Y').

The POS expression for minimized function product of all sum terms: (W+X'+Y) (W'+X'+Y') (X'+Z').

Check your progress

1. Simplify the following Boolean function expressed in POS form: $f(X,Y,Z)=\prod(0,1,2,4)$

3.9 SUMMARY

- We understand how to write minterms and maxterms of a function.
- We learned the standard form of a Boolean function and demonstrated to express any function in SOP and POS form.
- We discussed how to find complement of any function expressed in SOP or POS form.
- We learned about interconversion between SOP and POS form of a function.
- We illustrated how to minimize any boolean function with the help of K-Map.
- We understand the need of don't care conditions X, where we don't want the function outputs for certain combinations of input variables. This unspecified combinations of variables or minterms are called as don't care conditions.
- We demonstrated the process of a function simplification with Product of Sum form using k-map.

3.10 TERMINAL QUESTIONS

1. What do you mean by minterms and maxterms?
2. Explain Sum of Product (SOP) and Product of Sum (POS) representation of a Boolean function with suitable example.
3. What do you mean by complement of a Boolean variable?
4. What is standard form?
5. How many input combinations are possible for a function with 5 input Boolean variables?
6. Simplify the following Boolean functions with k maps.

   i) $F(A,B,C)=\sum(1,3,6,7)$

ii) $F(P,Q,R,S)=\sum(0,2,5,7,8,10,13,15)$

7. Express the Boolean function F = xy + x'z in product of sums form.

8. Express the Boolean function F = A + B'C in standard sum of products form.

UNIT-4 COMBINATIONAL CIRCUITS

Structure

4.1 INTRODUCTION

A combinational circuit consists of logic gates which accept signals from n input variables to produce m output variables. Figure 4.1 shows block diagram of a general combination circuit. It consists of n inputs which results in 2^n different input combinations and each input combination corresponds to one of 2^m outputs. In other words, a combinational circuit implement m Boolean functions and each Boolean function corresponds to one m output variables. The inputs to each Boolean function comes from n input variables.

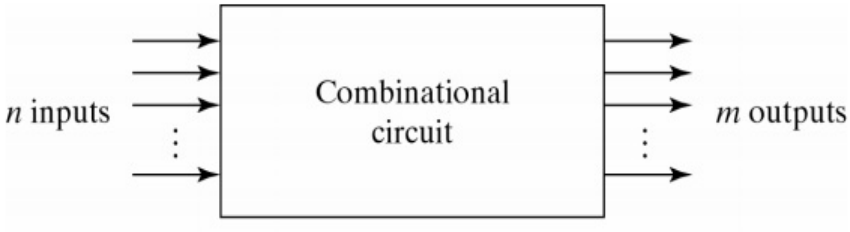


Figure 4.1 – A block Diagram of Combinational Circuit

In unit 1, we discussed about binary numbers which are used to represent and manipulate binary signals in digital systems. In the second unit, we learned Boolean algebra to represent Boolean functions. The Boolean functions are ultimately used to implement logic circuits. The third unit illustrated how to minimize a Boolean function so that it gives minimum numbers of gates. This achieves economically feasible implementation of logic circuits. The purpose of this unit is to demonstrate the knowledge acquired in all previous unit in designing and analysing combination circuits. This will give us some picture of how computers work.

4.2 OBJECTIVES

After studying this unit you should able to:

- Design combinational circuit for various circuits such as half adder, full adder, half Substractor and full Substractor.
- Understand working of 4 bits magnitude comparator, demultiplexer, encoder, decoder and multiplexer.
- Design binary to grey code convertor using combinational circuit.
- Design any combinational circuit with multiplexer or decoder.
- Implement any Boolean function with NAND and NOR gates which are considered as universal gates.

4.3 DESIGN PROCEDURE

The design procedure for any combinational circuit can be described by following steps:

1. An outline of the problem is stated.
2. The number of input variables and output variables are determined.
3. The truth table is derived to show the required relationships between input and output variables.
4. Each Boolean function that represent each output of the truth table is simplified by the k-map.
5. The logic diagram is drawn for simplified Boolean functions.

The truth table for a combination circuit consists of input columns and output columns. The input columns represents 2^n input combination possible with n input variables. The output values 0s or 1s for each input combinations are determined from the stated problem. However, if the problem specification indicates some input combinations do not occur. The don't care conditions are assigned to outputs of these input combinations. Since the output functions are derived from the truth table, the problem specifications must be interpreted correctly into the truth table. Any wrong interpretation results in incorrect combinational circuit. The output functions may be simplified by either k-map or algebric manipulation. Some practical design goals of a combinational circuit are:

1. Minimum number of gates.
2. Minimum number of inputs to gate.
3. Minimum propagation time of signal through the circuit.
4. Minimum number of interconnections.

4.4 ADDER

Modern Computers perform a variety of tasks. Some of the basic tasks include arithmetic operations. Addition of binary digits is one of the arithmetic operations. In this section, we will design a combinational circuit called as half adder which will perform addition of two 1 bit numbers. The combinational circuit which performs addition of 3 one bit numbers is known as full adder.

4.4.1 HALF ADDER

A half adder takes two binary numbers each of 1 bit and produce a binary number of 2 bit. In other words, it requires two 1 bit numbers and produces 2 bit numbers i.e. 0+0=00, 0+1=01,1+0=01, and 1+1= 10. The truth table for the half adder is shown in Figure 4.2.

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

Figure 4.2 - Truth table of half adder.

The truth table shows all possible inputs in addition of two 1 bit numbers in columns x and y. Since the addition produces a binary number with 2 digits, there are two output variables S (sum) and C (carry) in the truth table. The Boolean functions for S and C can be obtained from the truth table in sum of products (SOP). Remember, the SOP form includes minterms that correspond to output logic 1. Each minterm can be obtained by combining the inputs x,y,z such that each Boolean variable appears in complement form if it is 0 otherwise it appears in normal form if it is 1.

S=xy' + x'y

= x⊕y

C=xy

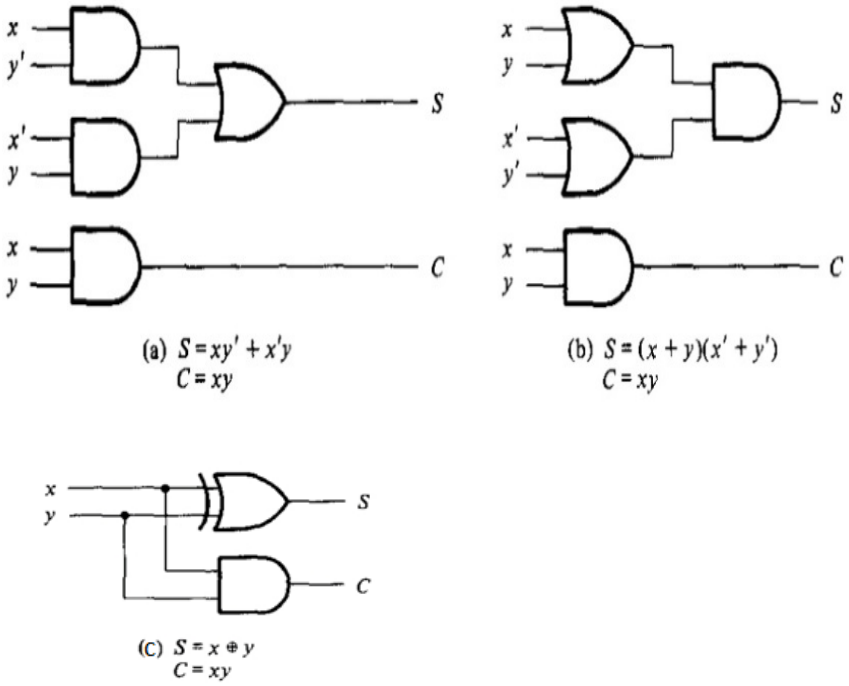


Figure 4.3 – Various implementation of a half adder.

The Figure 4.3-(a) shows implementation of half adder in SOP form. While, Figure 4.3(b) shows its implementation in POS form. Since the expression for S in SOP form in equivalent to Ex-OR gate, so the half adder can also be implemented with Ex-OR andAND gate as shown in Figure 4.3(c).

4.4.2 FULL ADDER

A full adder is a combinational logic circuit that performs arithmetic sum of 3 one bit numbers. It takes two 1 bit numbers and one 1 bit number as carry and produces 2 output bits. Two bits output is necessary because the arithmetic sum of 3 input bits produces sum which ranges from 0 to 3 and 2 bits are necessary to represent 2 and 3. Truth table for the full adder is shown in Figure 4.4.

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

Figure 4.4 - Truth table for the full adder

In the truth table, the two inputs x and y of the full adder represent two significant bits to be added and the third bit z is considered as carry from previous lower significant position. The least significant bit of the sum is given by S and higher significant bit of the sum by C which is considered as output carry.

The Boolean functions of S and C are expressed in SOP form which are minimized through k-map (as shown in Figure 4.4).

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

C=xy+xz+yz

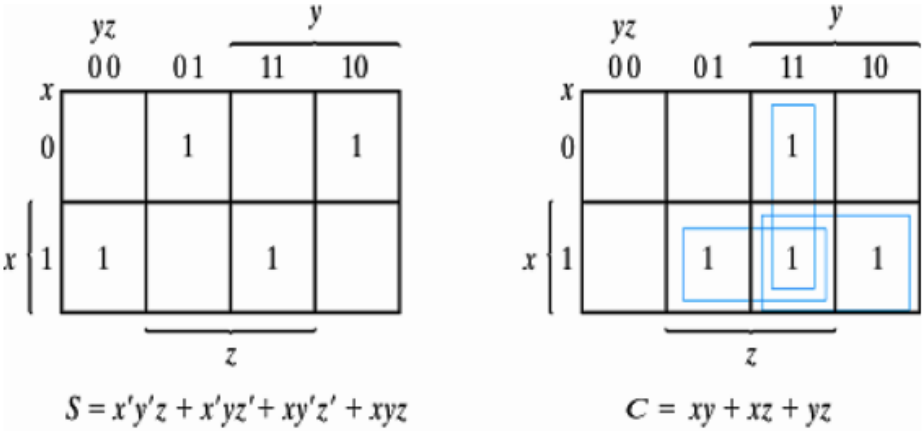


Figure 4.4 - k-map simplification of full adder.

The logic circuit for above simplified functions is shown in Figure 4.5.The implementation of full adder for POS form requires same number of gates but number of AND and OR gates are replaced.

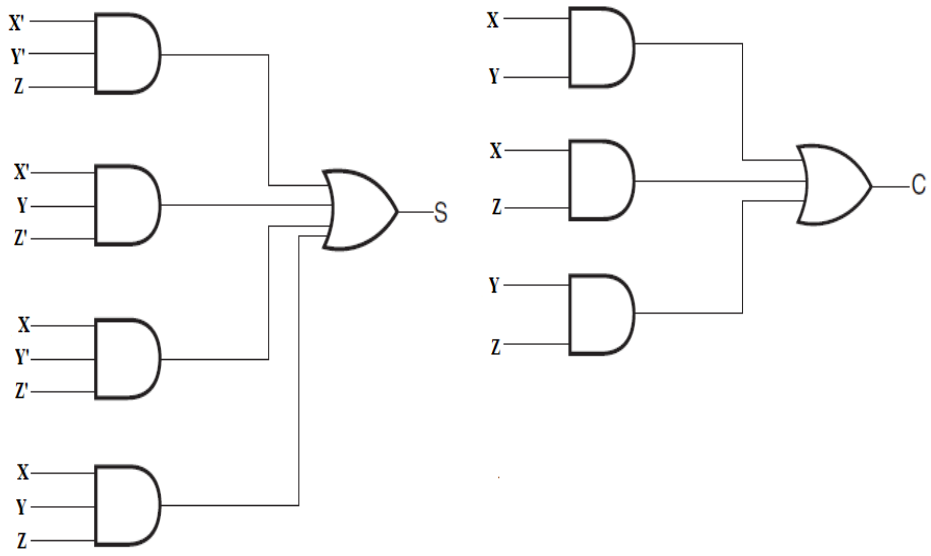


Figure 4.5- Full adder implementation with simplified SOP form.

A full adder can also be implemented with EX-OR,AND and OR gates. This requires algebraic manipulation of Boolean functions S and C as follows:

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

=x'y'z+z'(x'y+xy')+xyz

=z(x'y'+xy)+z'(x'y+xy')

=z(x'y+xy')'+z'(x'y+xy')

=z⊕(x'y+xy')

= z⊕(x⊕y)

C=xy'z+x'yz+xyz+xyz' (minterms of SOP form)

=z(xy'+x'y)+xy(z+z')

=z(x⊕y)+xy

The manipulated Boolean functions S and C can now be implemented using EX-OR,AND and OR gates as shown in Figure 4.6.

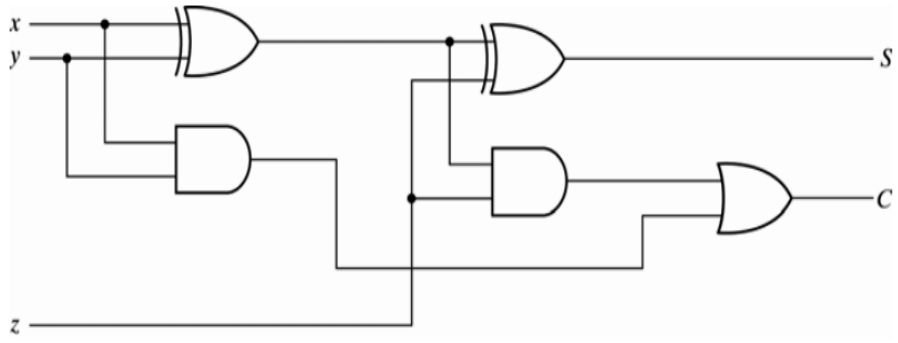


Figure 4.6 – Implementation of full- adder with EX-OR, AND and OR gates.

This implementation of full adder can be thought as two half after cascaded side by side. In a computer, to add two 8 bits numbers, we require 8 full adder which are cascaded together such that each full adder allows addition of two 1 bit numbers and one bit carry from previous lower significant position and sends one bit carry to next higher significant position.

4.5 SUBSTRACTOR

4.5.1 HALF SUBTRACTOR

A half subtractor is a combinational circuit which performs arithmetic subtraction of two 1 bits numbers. To perform subtraction of two binary bits represented with variables x and y, we need to check the relative magnitude of x

and y. Consider x as minuend and y as substrahend. If x>= y, it results in three possibilities: 1-0=0,1-1=0 and 0-0=0. The result of the operations are called as difference bit. But if x<y i.e. 0-1, this requires borrow from next higher significant bit. In decimal number system a borrow adds 10 to the minuend digit. Similarly, in the case of binary number system, a borrow adds 2 to the minuend bit. Now the difference 0-1 becomes 2-1 which gives 1. This subtraction generates two outputs, one is the difference D=1 and other is borrow B= 1. The borrow B in case of x>y is 0. The truth table for the half subtractor which shows the input-output relationships is shown in Figure 4.6.

Inputs		Outputs	
X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Figure 4.6- Truth table of half substractor.

The Boolean functions for two outputs D and B can be expressed in POS form as follows:

D = x'y + xy'

= x ⊕ y

B =x'y

The combinational circuit for above functions which represents half substractor is shown in Figure 4.7.

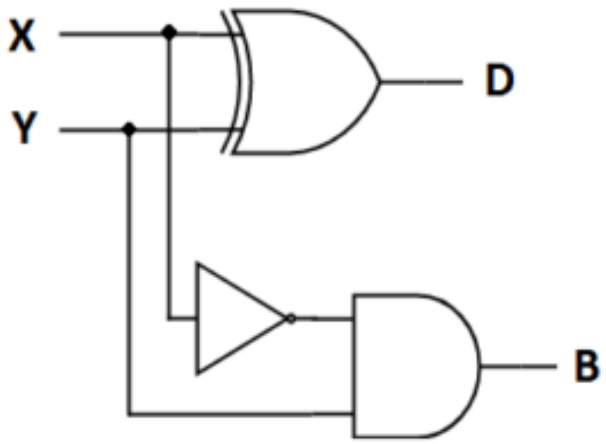


Figure 4.7- Combinational circuit of half substractor.

4.5.2 FULL SUBSTRACTOR

A full substractor is a combinational circuit which performs arithmetic substaction of two 1 bit numbers by considering that 1 may be borrowed by previous lower significant bit. For example, consider three 1 bits numbers x,y and z as minuend, subtrahend and previous borrow respectively. The full subtraction can be treated as x-y-z. If the previous borrow, z=0 then the subtraction of x and y is the same as the half adder. But, when the previous borrow, z=1, x=0 and y=0, we need to borrow 1 from next higher significant bit which makes borrow B= 1. This results in addition of 2 to x and now subtraction of x-y-z becomes 2-0-1. This gives D=1. For x=0,y=1 and z=1, again there is a need to borrow 1 from higher significant bit which adds 2 to x and makes borrow B=1. This results in 2-1-1=0 which outputs D=0. Similarly, when x=1,y=1 and z=1, we need to borrow 1 which makes B=1 and x= 3. Now the subtraction becomes 3-1-1=1, so the output D= 1. The truth table for the full subtractor shown in Figure 4.8 shows all these results.

Inputs			Outputs	
X	Y	Z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figure 4.8- Truth table of full substractor.

The Boolean functions for outputs D and B of the above truth table can be expressed in SOP form as follows:

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

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

The above Boolean functions D and B can be simplified using algebraic manipulation as follows:

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

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

= z(x'y' + xy) + z'(x'y + xy')

= z (x ⊙ y) + z' (x ⊕ y)

= z (x ⊕y)' + z' (x ⊕ y)

= z ⊕ (x ⊕ y)

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

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

= z(x'y' + xy) + x'y(z' + z)

= z (x ⊙ y) + x'y

(z' + z)=1

= z (x ⊕y)' + x'y

The combinational circuit for simplified functions D and B is shown in Figure 4.9.

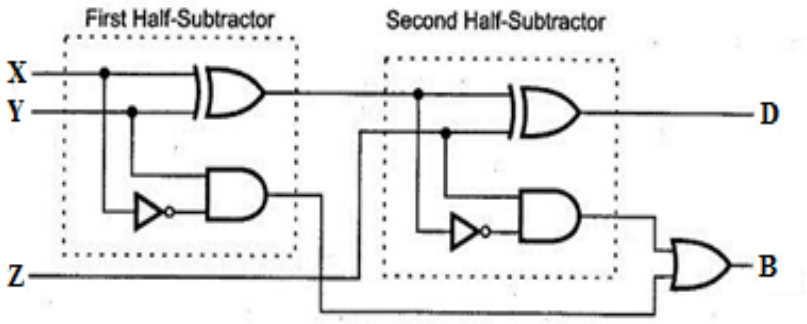
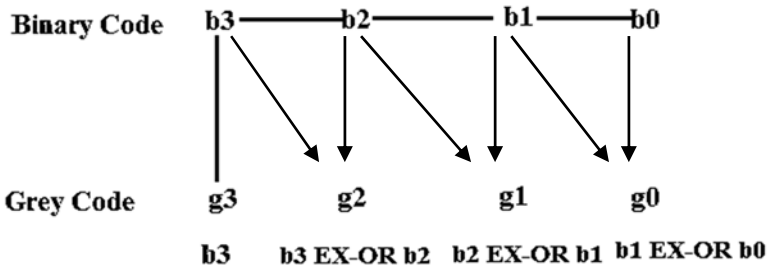


Figure 4.9 - Combinational circuit of full substractor.

4.6 CODE CONVERSIONS

There are a large variety of code available to represent the same information by different digital systems. For example, there are different types of binary code such as BCD, grey and excess 3 code. Sometimes, we need output of one system as input to another system. The two system may use different coding scheme, so there is a need of conversion circuit that makes both system compactable to each other. The conversion circuit is a combinational circuit that convert one type of binary code to another type of binary code. In this section, we will discuss the design procedure for Binary to Graycode conversion circuit.

Binary to Gray: A Binary coded digits b3b2b1b0 can be converted to its equivalent Gray Code with following procedure:



From the above procedure, each digit of the corresponding grey code thus obtained are as follows:

g3 = b3

g2 = b3 XOR b2

g1= b2 XOR b1

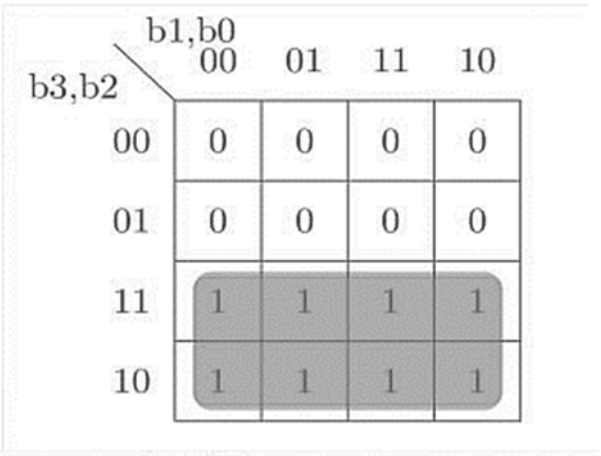
g0 = b1 XOR b0.

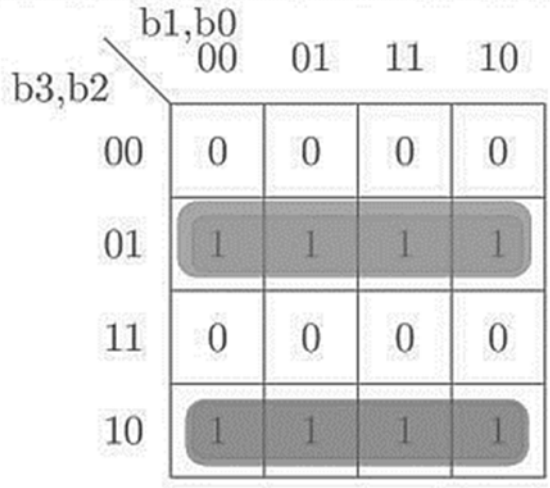
Binary				Gray Code			
b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Table 4.1- Binary to Gray Code Converter Table

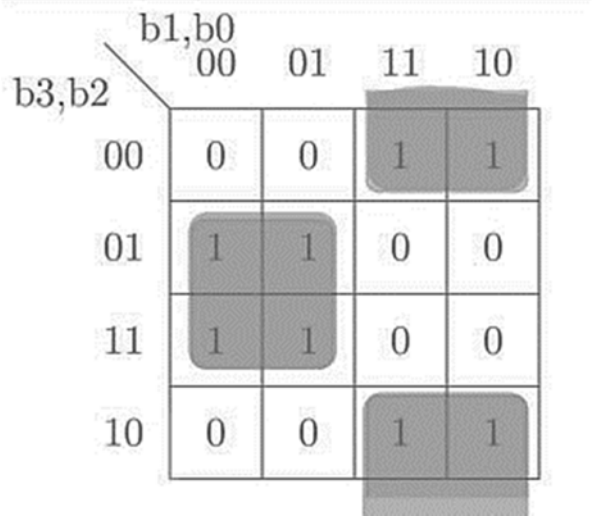
The bit combinations of BCD code and their equivalent grey codes are shown in truth table in Table 4.1. Each BCD code requires 4 bits for its representation and its corresponding grey code also requires 4 bits. So, there are 4 input binary variable with symbols: b3,b2,b1,b0 and 4 output variables: g3,g2,g1,g0 in the truth table.

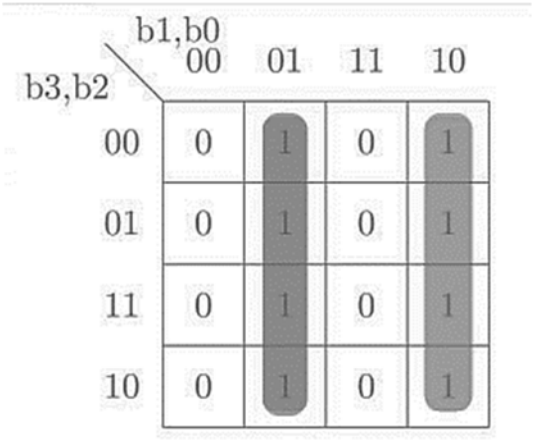
The output variables g3,g2, g1 and g0 are Boolean functions which can be simplified with k-maps as shown in Figure 4.10.





g3=b3g2=b3b2' + b3'b2





g1=b1b2' + b1'b2 g0=b0b1' + b0'b1

Figure 4.10 - k maps of Binary to Gray Code simplification

Each of the four k - maps represents minimization process of each of 4 output variables. The 1s marked in squares is obtained from the minterms of its output variables which corresponds to 1 in truth table. The expressions obtained from the Figure 4.10 can be manipulated further to obtain more compact representation using Ex-OR gates.

$$g3=b3$$

$$g2=b3b2' + b3'b2=b3 \oplus b2$$

$$g1=b1b2' + b1'b2= b1 \oplus b2$$

$$g0=b0b1' + b0'b1= b0 \oplus b1$$

After the manipulation of each Boolean function which represents one output variable, the logic diagram with these functions is shown in Figure 4.11

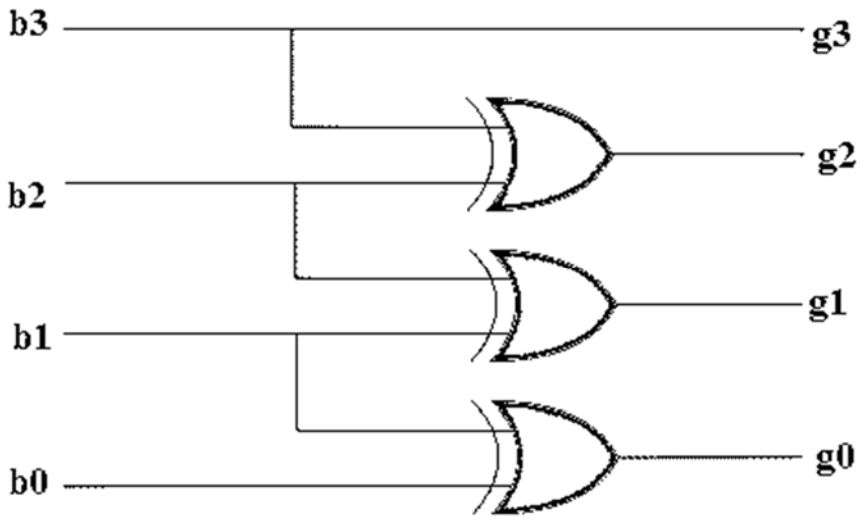


Figure 4.11 – Logic circuit for Binary to Gray Code.

Check Your Progress

Find the grey code of following BCD code.

a) 0101

b) 1101

4.7 DECODER

In a digital system, the discrete quantities of information are represented with binary codes. An n bits binary code is capable of representing 2^n discrete quantities of information. A decoder is a combinational circuit which converts an n bits binary information coming from n input lines to one of 2^n outputs. A decoder is represented with n to m line decoder such that m<=n. Each output of the decoder corresponds to one of the 2^n minterms of n variables. For example Figure 4.12 shows a 3*8 decoder.

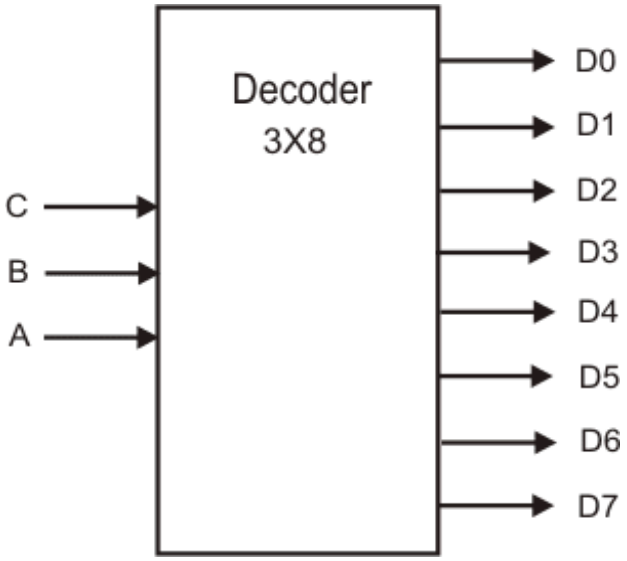


Figure 4.12 - A 3*8 decoder.

The above decoder takes a binary code of 3 bits and converts one of the 8 outputs. The operation of the decoder canfurther be illustrated with its truth table (as shown in Table 4.2) which shows relationships the inputs and corresponding outputs.

A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
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

Table 4.2 - Truth table of a 3*8 decoder.

Implementation of any combination circuit with decoder: We know that any Boolean function can be expressed as Sum of Products (SOP) form. Since a decoder generates all 2^n minterms containing n variables. We can generate any combination circuit containing n inputs and m outputs with the help of n to 2^n decoder and m OR gates. Please note that, the implementation of any combination circuit using decoder requires its Boolean function to be expressed in SOP form. We can obtain SOP form of a combinational circuit from its truth table or from algebraic manipulation if the function is directly given. For example, implement full adder with help of a decoder requires two OR gates. From the truth table of the full adder (as shown in Table 4.4), we can express the outputs S,C in SOP form as given below:

$S(x,y,z)=\sum(1,2,4,7)$

$C(x,y,z)=\sum(3,5,6,7)$

As there are 3 input variables which results in 2^3= 8 minterms, we need a 3 to 8 decoder as shown in Figure 4.13.

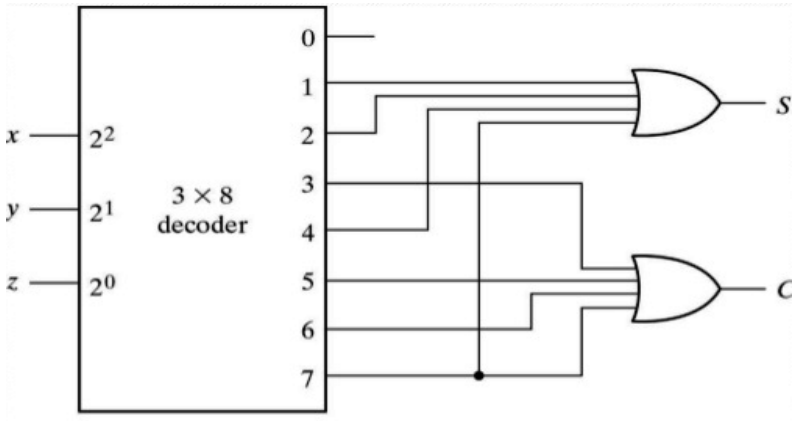


Figure-4.13- A full adder implementation with decoder and OR gate.

The decoder generates 8 minterms contains three variables x,y,z. The output S is obtained from the OR gate whose inputs are minterms: 1,2,4,7. The output C is obtained from the OR gate whose inputs are: 3,5,6,7.

Check your progress

1. Implement the following boolean functions with decoder.

   f(x,y,z)=(y' + x)z

4.8 DEMULTIPLEXER

A decoder with enable input works as demultiplexer. In the demultiplexer the input lines of the decoder functions as selection lines and contains one enable line as input. The process of getting information on an input line and send this information to one of the multiple output lines is called as demultiplexing. A demultiplxer is a combinational circuit that performs such operation. The demultiplexer is also called as data distributor because it transfers the data present at input line to one of destinations as shown in Figure 4.14.

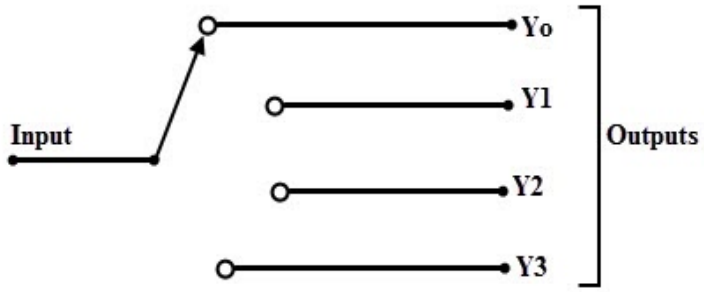


Figure 4.14 - Demultiplexer as a data distributor.

There are selection lines which select specific output line for data transfer. A 1 to n demultiplexer consists of 1 input line, n output lines and m selection lines. The m selection lines are used to select one of 2^m=n output lines. For example, a 1 to 4 demultiplexer consists of 2 selection lines (because 2^2=4) as shown in Figure 4.15.

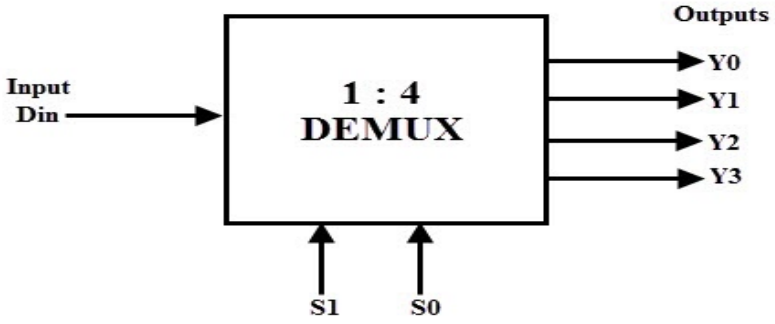


Figure 4.15- A 1 to 4 demultiplexer.

1 to 8 Demultiplexer: A 1 to 8 demultiplexer (as shown in Figure 4.16) consists of one input line D whose data is transmitted to one of the 8 output lines (Y0 to Y7) depending on the values of selection lines S0,S1,S2.

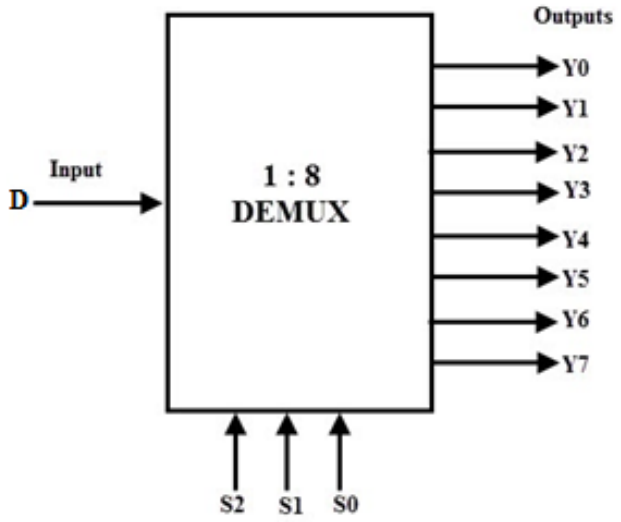


Figure 4.16- A block diagram of 1 to 8 demultiplexer.

The truth table for the demultiplexer is shown in Figure 4.17. It shows the relation between the values of selection lines S0,S1,S2 and output lines Y0 to Y7 for transmission of the input data D.

Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

Figure 4.17- Truth table of 1 to 8 de multiplexer.

4.9 ENCODERS

An encoder is a combination circuit that performs the inverse operation of a decoder. The encoder consists of 2^n input lines and n output lines. The output lines generate the binary code of the input value. For example an octal to binary encoder (as shown in Figure 4.18) consists of 8 input lines, one for each octal digit and 3 output lines which generate corresponding binary number.

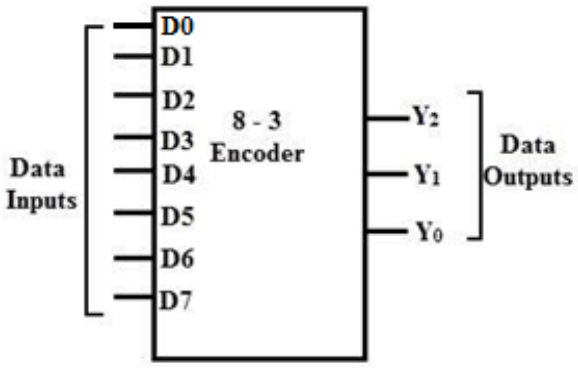


Figure 4.18- An octal to binary encoder.

The truth table for the octal to binary encoder is shown in Figure 4.19.

No	Inputs								Outputs		
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
2	0	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	0	0	0	0	1	1
4	0	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	0	0	0	0	1	0	1
6	0	1	0	0	0	0	0	0	1	1	0
7	1	0	0	0	0	0	0	0	1	1	1

Figure 4.19-Truth table of an octal to binary encoder.

4.10 MULTIPLEXER

A multiplexer is also called as data selector because it selects binary information from one of several inputs and sends to one output line as shown in Figure 4.20. A particular input line is selected by the selection lines. A multiplexer is often abbreviated as MUX.

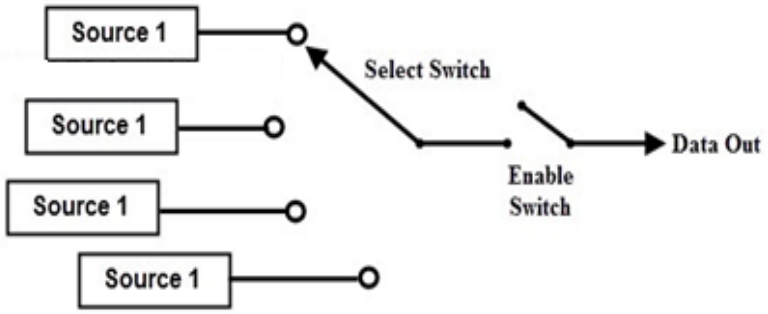


Figure 4.20 - Multiplexer as a data selector.

The multiplexer is a Combinational circuit which consists of 2^n input lines from which one input line is selected and directed to the output line with the help of m (2^m=n) selection lines. For example, a 4 to 1 multiplexer is shown in Figure 4.21 which has 4 input lines I0, I1,I2,I3 and selection of one of the input lines as output Y is controlled by selection lines S0,S1.

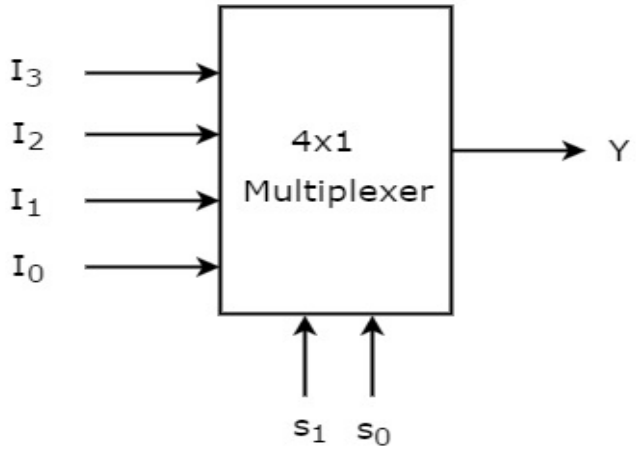


Figure 4.21- Block diagram of a 4 to 1 multiplexer.

Implementation of any Boolean function with multiplexer : Any Boolean function of n variables can be implemented with a $2^{(n-1)}$ to 1 multiplexer. The n-1 variables of the Boolean function are used for selection lines and the remaining 1 variable is used for input lines of the multiplexer. If A is the remaining variable, then each input to the multiplexer is either A or A' or 0 or 1. With suitable choice of these four values as inputs of the multiplexer, any Boolean function can be implemented. For example, consider the following Boolean function to illustrate this procedure.

F(A,B,C)= ∑(1,3,5,6)

The truth table for above function is shown in Figure 4.22.

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Figure 4.22- Truth table for the function.

Procedure-

1. Express the given Boolean function in sum of products form (SOP).
2. Since there are 3 variables, we require 4 to 1 multiplexer ($2^{(3-1)}$=4).
3. Consider the minterms of the function with ordered sequence of variables ABC where A is the most significant bit (MSB) and C is the least significant bit (LSB). Connect the n-1 variables to selection lines such that B connected to higher order selection line S1 and C connected to the next higher order selection line S0 and so on till the last variable. This is shown in Figure 4.23

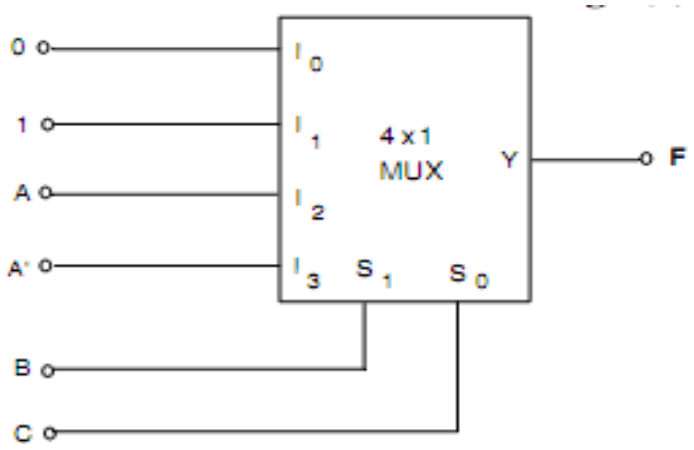


Figure 4.23- Function implementation with 4 to 1 multiplexer.

4. The remaining variable A is in MSB position and left most variable in the truth table. This variable is 0 for the first half of the truth table and 1 for the remaining half. Consider the table shown in Table 4.4. The variable A will appear in complement form as A' for minterms 0 to 3 (because A is 0 for these minterms) and it will appear in uncomplemented form as A for minterms 4 to 7 (because A is 1 for these minterms).

	I_0	I_1	I_2	I_3
A'	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	A'

Table 4.4- Implementation table.

5. Write the inputs of the multiplexer I0,I1,I2,I3 in columns and list the minterms of the function in two rows such that the first row contains all minterms where A is in complemented form and the second row contains all the minterms where A is in uncomplemented form. This is shown in Table 4.4.

6. Circle all the minterms of the functions.

7. While looking the table 4.4:

   a) If a column contains both mintermsuncircled, apply 0 to the multiplexer input corresponding to the column heading.

   b) If the column contains both minterms circled, apply 1 to the multiplexer input corresponding to the column heading.

   c) If only one minterm is circled, look at its row and column heading and apply row heading as input to the column heading of the multiplexer.

Illustrative example: Implement following Boolean function with multiplexer.

$F(A,B,C,D)= \sum(0,1,3,4,8,9,15)$

The truth table can be drawn as:

A	**B**	**C**	**D**	**minterms**	**F**
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	2	0
0	0	1	1	3	1
0	1	0	0	4	1
0	1	0	1	5	0
0	1	1	0	6	0
0	1	1	1	7	0
1	0	0	0	8	1
1	0	0	1	9	1

1	0	1	0	10	0
1	0	1	1	11	0
1	1	0	0	12	0
1	1	0	1	13	0
1	1	1	0	14	0
1	1	1	1	15	1

This is a 4 variable function, so it requires 8 to 1 multiplexer ($2^{(4-1)}$=8). The n-1 variables i.e. BCD after leaving the MSB bit are connected to selection lines of the 8 to 1 multiplexer starting from higher order to lower order of selection lines.

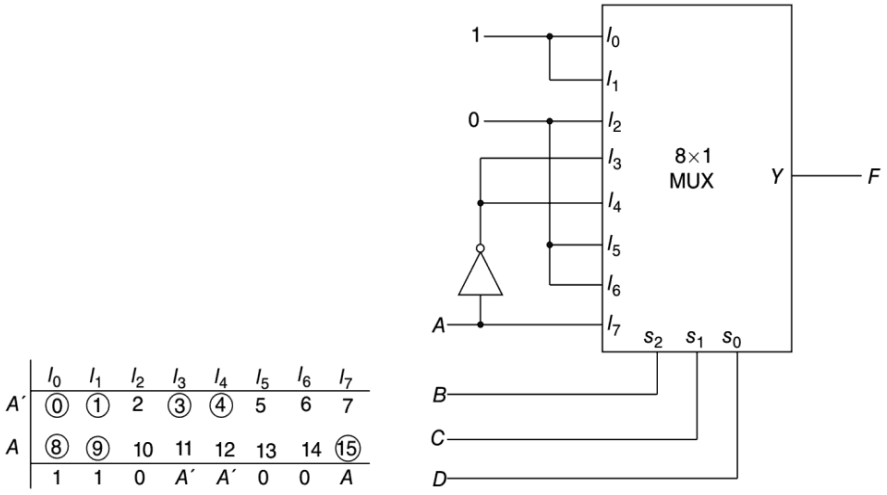


Figure 4.24- Function implementation with 8 to 1 multiplexer.

The first half of the minterms have variable A as 0s (as shown in truth table) and so A will appear in complement form (as shown in Figure 4.24)for theseminterms. The second half of the minterms have variable A as 1s (as shown in truth table) and so it will appear in uncomplemented form (as shown in Figure 4.24) for these minterms. The implementation table is shown in Figure 4.24. The minterms of the functions are circled and the values for the multiplexer inputs are obtained as follows:

The I0 and I1 column have both minterms circled, so input 1 will be applied to I0 and I1. The I2 column has none of its minterms circled, so input 0 will be applied

to I2. The column I3 has only one minterm circled, so input A' will be applied to I3. Similarly we can find out input values for the rest of the columns.

Comparison of multiplexer method with decoder method: We have seen that, both multiplexer and decoder can implement any Boolean function. The decoder method requires one OR gate for each Boolean function but the same decoder can be used to implement any number of Boolean functions. The multiplexer method requires one multiplexer for each Boolean function but it is smaller size unit. If there are small numbers of output functions, multiplexer method is a good choice for their implementation. But, if the number of output functions are large, the decoder method requires fewer ICs.

Check your progress

Implement the following functions with multiplexer.

$F(A,B,C)= \sum (1,2,6,7)$

4.11 MAGNITUDE COMPARATORS

A magnitude comparator is a combinational circuit that compares two binary numbers and determines whether one binary number is greater than (>) or equal to (=) or less than (<) the second binary number. Consider two numbers each containing 4 digits as follows:

$A=A_3A_2A_1A_0$

$B=B_3B_2B_1B_0$

The numbers are written with letter followed by subscript according to their significant positions.

- Two binary numbers are equal if and only if each pair of significant bits is equal i.e. $A_3=B_3$, $A_2=B_2$, $A_1=B_1$, $A_0=B_0$. This equality relation between a pair of bits X_i can be shown with logical function as:

  $X_i=A_iB_i+A'_iB'_i$ where i = 0,1,2,3

  In the above function, X_i will be 1 only if both bits in position i are either 0 or 1. For equality of two numbers each pair of variables at position X_i should be equal to 1. In other words, the AND operation of each $X_i$pair should be equal to 1. This can be expressed with Boolean function (A=B) as:

  $(A=B)=X_3X_2X_1X_0$

- A binary number A is greater than another binary number B in the following four cases:

  1. If A3 = 1 and B3 = 0i.e. the output of A_3B_3' should be 1.

2. If A3 = B3 and A2 = 1 and B2 = 0. This can be expressed this with Boolean expression $X_3A_2B_2'$ and its output should be 1.

3. If A3 = B3, A2 = B2 and A1 = 1 and B1 = 0. This can be expressed with Boolean expression $X_3X_2A_1B_1'$ and its output should be 1.

4. If A3 = B3, A2 = B2, A1 = B1 and A0 = 1 and B0 = 0. This can be written with Boolean expression $X_3X_2X_1A_0B_0'$ and its output should be 1.

The above comparisons performed in steps 1,2,3,4 can be expressed with Boolean function (A>B) as:

$(A>B)=A_3B_3' + X_3A_2B_2' + X_3X_2A_1B_1' + X_3X_2X_1A_0B_0'$

- Similarly, the Boolean function (A<B) for a binary number A is less than another binary number B can be expressed as follows:

$(A<B)=A_3'B_3 + X_3A_2'B_2 + X_3X_2A_1'B_1 + X_3X_2X_1A_0'B_0$

The combinational circuit for 4 bit magnitude comparator can be implemented from these four Boolean functions: (A<B),(A>B) and (A=B) as shown in Figure 4.25.

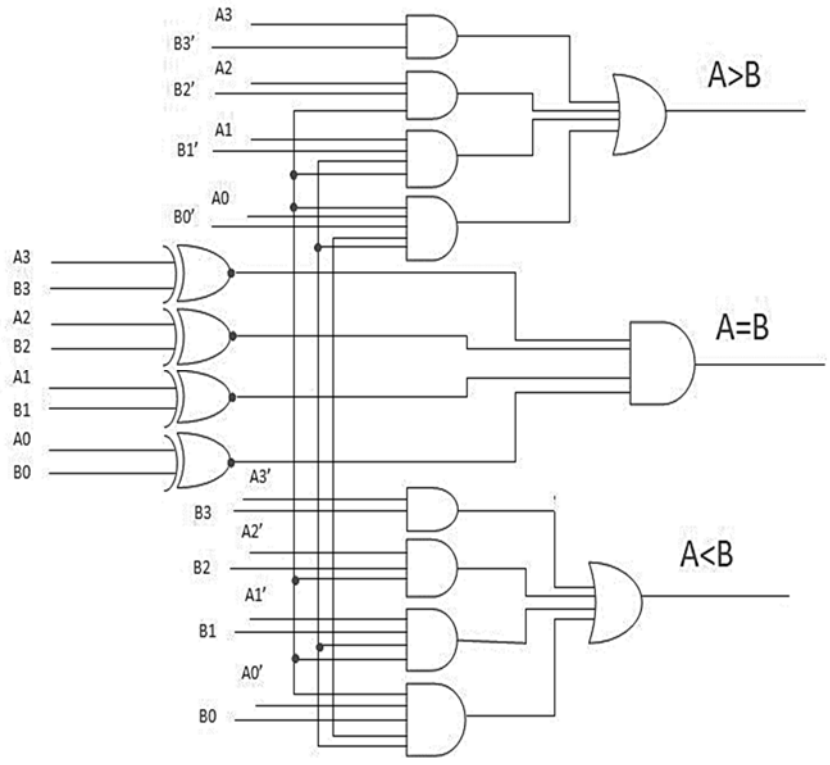


Figure 4.25- 4 bit magnitude comparator.

4.12 NAND AND NOR IMPLEMENTATION

The NAND and NOR gates are very convenient to fabricate with electronic components and so they are mostly used in all IC logic families. This is why digital circuits are usually constructed with NAND and NOR gates. Any Boolean function

implemented with AND,OR and NOT gates can be converted into another logic circuit which contains only NAND or NOR gates. To understand this conversion process, we need to familiar with alternate logic diagrams of NAND and NOR gates which are shown in Figure 4.26.

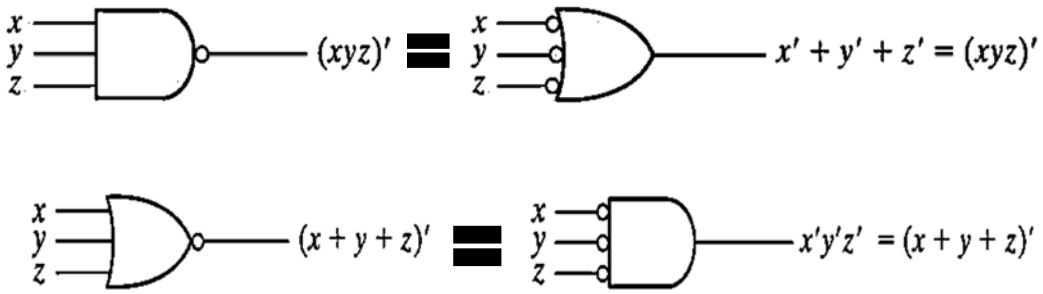
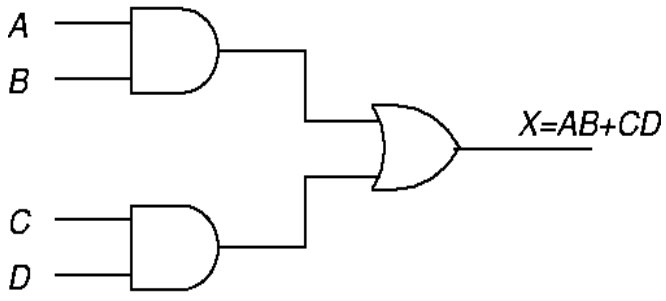


Figure 4.26 – Alternate symbols for NAND and NOR gates using Demorgan's theorem.

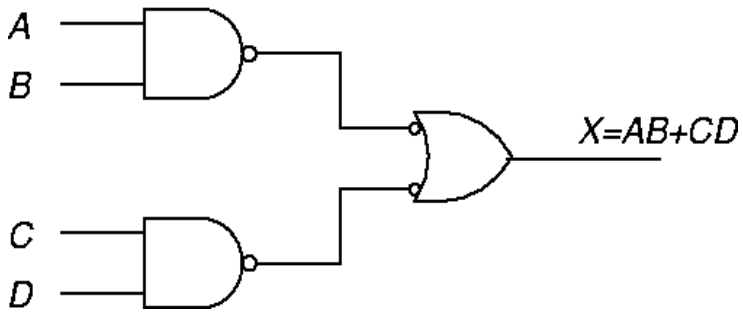
The NAND logic diagram is equivalent to OR gate preceded by small circles in all its inputs. The circles at inputs represent complemented inputs. These alternate logic diagrams are obtained using DeMorgan's law. Similarly, the NOR gate is equivalent to AND gate with small circles in all its inputs. The NAND and NOR gates with only one input behave as NOT gate.

NAND implementation : NAND gate is considered as universal gate because any digital circuit can be implemented with only NAND gates. The implementation of any function with NAND gates requires following steps:

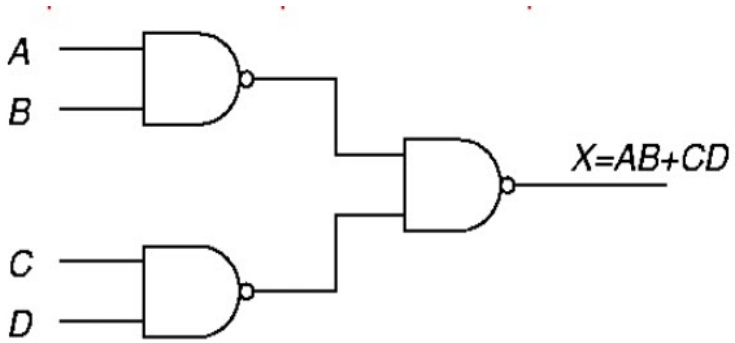
1. Implement the simplified function with AND and OR gate. For example, the circuit implementation of a function f=AB+CD with AND and OR gate is shown below:



2. Introduce two bubbles (complements) at the outputs of each AND gate. The first bubble presents just after each AND gate and the other presents at input to the gate receiving the output of bubbled AND gate. This is illustrated below for previous circuit diagram:

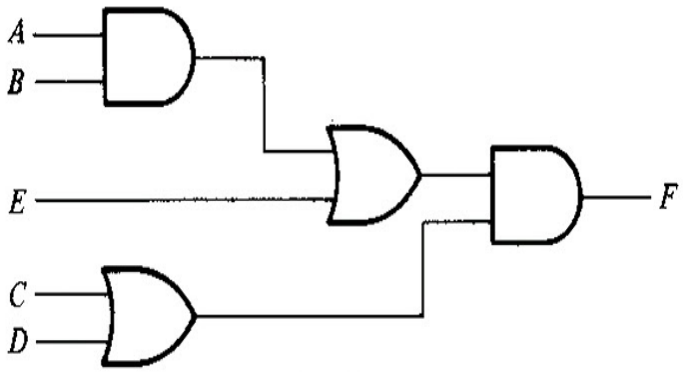


3. If any input of a gate is bubbled, make all its inputs bubbled by introducing two bubbles, one acts as input to gate and other makes its input variable as the complement.
4. Replace any OR gate containing all bubbled inputs with NAND gate.
5. Replace any complemented variable with single input NAND gate (it acts as NOT gate). This is illustrated below for previous circuit diagram:

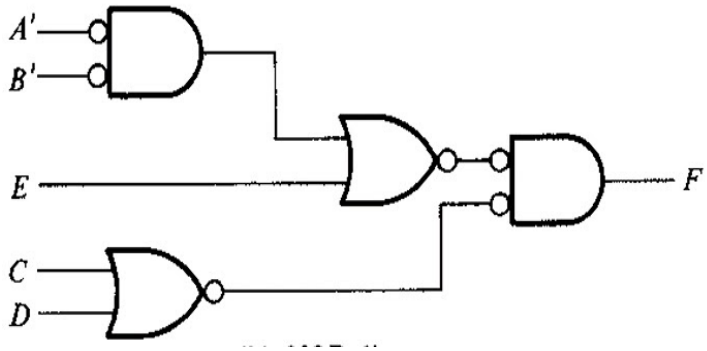


NOR implementation: The NOR gate is also considered as universal gate because any digital circuit can be implemented using only NOR gates. The implementation of any function with NOR gates requires following steps:

1. Implement the function with AND and OR gates. For example, consider a function f=(AB+E)(C+D) which is implemented with AND and OR gates as shown below.

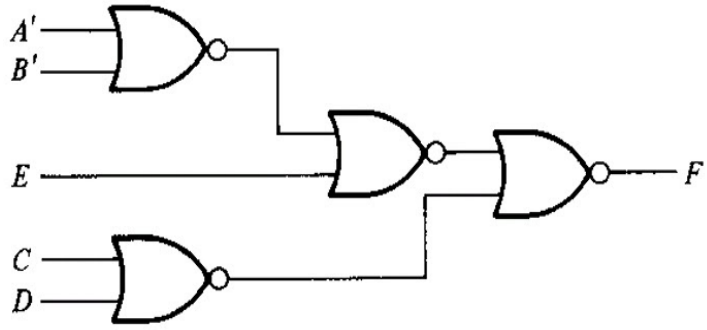


2. Introduce two bubbles (complements) at the outputs of each OR gate. The first bubble presents just after each OR gate and the other at input to the gates receiving the output of bubbled OR gate. For example, the bubbles are introduced in above circuit as given below.



3. If any input of a gate is bubbled, make all its inputs bubbled by introducing two bubbles, one acts as input to gate and other makes its input variables as the complement.

4. Replace any AND gate containing all bubbled inputs with NOR gate. For example, bubbled AND gates are replaced with NOR gates in the above circuit diagram as given below.



5. Replace any complemented variable with single input NOR gate (it acts as NOT gate).

4.13 SUMMARY

- We understood about half adder, full adder, half Substractor, full Substractor and their circuit implementations.
- We understood working of demultiplexer and encoder.
- We learned to design binary to grey code convertor using combinational circuit.

- We demonstrated that how multiplexer can be used to implement any boolean function.
- We learned that a decoder along with OR gate can be used to implement any number of Boolean functions.
- We saw how the 4 bits magnitude comparator is implemented to perform comparison of two 4 bits numbers.
- We are illustrated to implement any Boolean function with NAND and NOR gates which are considered as universal gates.

4.14 TERMINAL QUESTIONS

1. Explain combination circuit with suitable block diagram.
2. How Boolean functions of a combinational circuit are derived?
3. What are the design goals of combination circuit?
4. What are the differences between half and full adder?
5. Design truth table and logic circuit of full adder.
6. Design truth table and logic circuit of half adder.
7. Find the grey code of following BCD code.
    1. 0101
    2. 1101
    3. 1001
    4. 1110
8. What is conversion circuit? Design a Combinational circuit for BCD to Grey code conversion.
9. What is decoder? Explain decoder with its block diagram.
10. What is demultiplexer? How it is different from decoder?
11. Explain 3 to 8 demultiplexer with its block diagram and its truth table.
12. Implement the following Boolean expression with only NAND gates.
    i) (AB' + CD')E + BC(A + B)
    ii) w(x + y + z) + xy
13. Implement the following boolean functions with decoder.
    i) half adder
    ii) full adder
14. Implement the following functions with multiplexer.

i) $C=\sum (3,5,6,7)$

ii) $F(p,q,r)= pq + pq's + q'r's'$

15. What is the use of magnitude comparator? Draw the combination circuit for 4 bit binary comparator.

16. How half Substractor is different from full Substractor? Explain with their circuit diagrams.

17. Implement the following function with NAND gates only:

    i. $A + (B' + C)(D' + BE ')$

    ii. $(CD + E)(A + B ')$

    iii. $AB' + A'B$

18. Implement the following functions with NOR gates only:

    i. $(AB + E)(C +D)$

    ii. $AB+A'B'$

    iii. $AB' + A'B$

UNIT-5 SEQUENTIAL CIRCUIT

Structure

5.1 INTRODUCTION

Digital circuits that we have seen so far are combinational circuits in which their outputs are entirely dependent on their current inputs at any time. However, most of the digital circuits also include sequential circuits in which their outputs depend on their current inputs as well as previous inputs. A sequential circuit is a kind of combinational circuit which consists of a memory element to store previous inputs. Figure 5.1 illustrates a sequential circuit which consists of a combinational circuit and a memory element. Itshows a combinational circuit which is connected to a memory element and the memory element is connected back to the combinational circuit as other input.

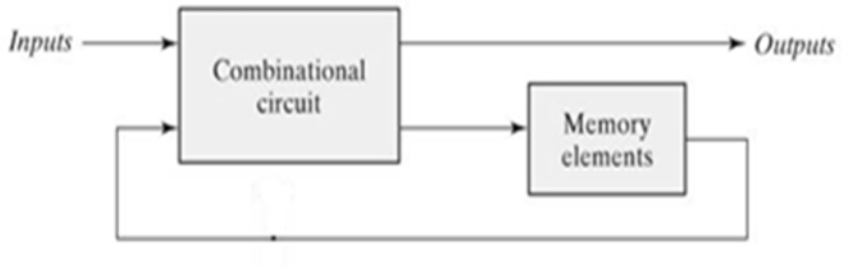


Figure 5.1- A block diagram of sequential circuit.

The memory element stores binary information which represents the state of the memory element. The sequential circuit uses the state of the memory element along with otherinputs to produce output. This way, the output of a sequential circuit is determined not only by its external inputs but also by the current state of the memory element.

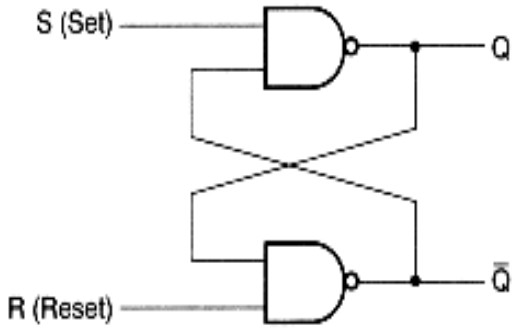
5.2 OBJECTIVES

After studying this unit, you should able to

- Describe how sequential circuits differ from combinations circuits.
- Explain the various types of flip flops with their truth tables.
- Understand key differences among flip flops, registers and counters.

5.3 FLIP FLOP

The memory element which we have seen in a sequential circuit are flip flops. A flip flop is a circuit made of logic gates and it is capable of storing 1 bit of information. It retains the binary information until it is directed through input signals to change its state. It is also called a bi-stable device because its output is in one of the two states 0 and 1. The output state remains indefinitely until some other inputs are applied to it. A flip flop circuit can be designed from either two NAND gates or two NOR gates. The flip flop construction using NAND gate is shown in Figure 5.2.



(a) Logic diagram

S	R	Q	$\overline{Q}$	
0	1	1	0	Set state
1	1	1	0	
1	0	0	1	Reset state
1	1	0	1	
0	0	1	1	Undefined

(b) Truth table

Figure 5.2 - A basic flip flop with NAND gate.

The circuit form cross connections such that output of one gate connected to the input of other gate. This forms a basic flip flop upon which different types of flip-flops will be built later. The basic flip flop has two inputs set and reset and two outputs Q and Q'. The working of the basic flip flop (as shown in Figure 5.2) can be explained as follows:

1. When the input R = 0 and input S = 1, the NAND gate 2 has one of its inputs at logic 0, so its output Q' becomes 1. The output Q' is fed as input to gate 1. This makes both inputs of NAND gate 1 to 1 and therefore it produces output Q = 0.

2. Consider, the input R changes to 1, while input S remains at 1. The NAND gate 2 still gives its output Q'=1 because one of its input coming from Q is 0. Therefore in this case the state of the flip-flop circuit remains unchanged with Q = "0" and Q' = "1".

3. Now, assume the input R remains 1, while input S changes to 0. The NAND gate 2 now gives the output Q'=0 because one of its input coming from Q is now 1. Therefore in this case the state of the flip-flop circuit changed with Q = "1" and Q' = "0".

4. Now consider the input S remains at 0 and input R changes to 0, the output Q now becomes 1 because one of the inputs of the NAND gate 1 is 0. On the other hand, the NAND gate 2 also produces output Q' as 1 because one of its input R=0. This input combination S = "0" and R = "0" must be avoided because this causes both outputs Q and Q' to be 1. But, we actually want Q' to be the inverse of Q.

The relationship between inputs and corresponding outputs as discussed above in shown with the truth table in Figure 5.2 (b).

5.4 SR FLIP FLOP

The basic flip flop can be modified by adding two additional NAND gates and a clock pulse as shown in Figure 5.3.

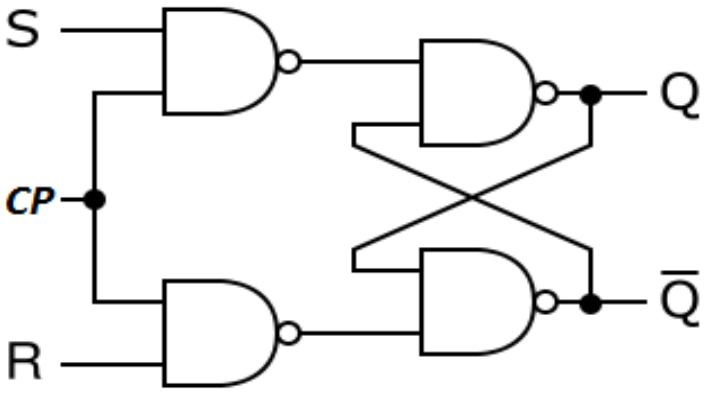


Figure 5.3- Logic circuit for SR flip flop

This provides controlled inputs to basic flip flop and this flip flop is known as SR flip flop. The truth table for the SRflip flop is shown in Figure 5.4. In the truth table, Q(t) is present state of the flip flop before applying clock pulse. The columns S and R contain possible combinations of inputs that can be applied to flip flop. The Q(t+1) refers to the next state of the flip flop.

S	R	Q(t+1)
0	0	Q(t) (No change)
0	1	0 (reset)
1	0	1 (set)
1	1	Undefined

Figure 5.4- Truth Table for SR flip flop.

SR flip flop changes state from present state Q(t) to next state Q(t+1) when inputs S and R is applied to SR flip flop along with single clock pulse. When S=0 and R=1, the flip flop is in 0 state or reset state. The flip flop goes to 1 state or set state on inputs S=1, and R=0. When inputs S=0 and R=0, the flip flop remains at its present state Q(t). The inputs S=1 and R= 1 is indeterminate condition because the state of the flip flop is not defined for this input combination. Figure 5.6 shows the block diagram of SR flip flop which is generally used in sequential circuits for design and analysis.

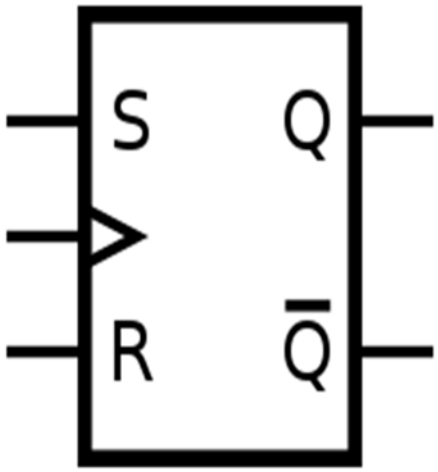
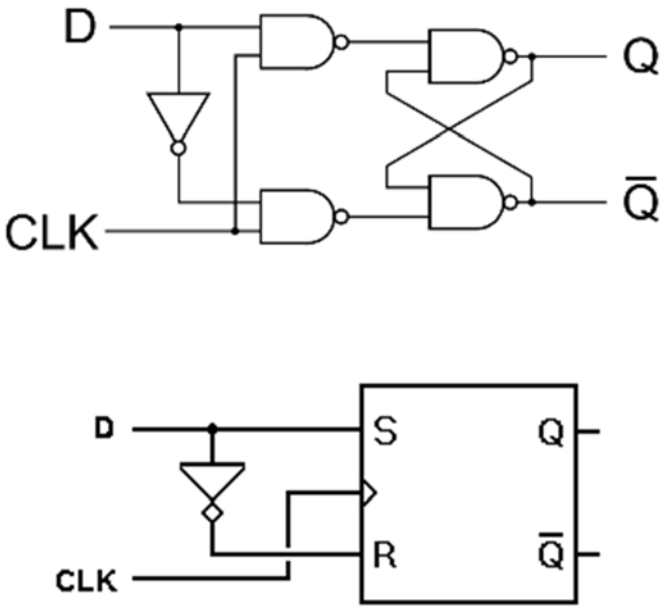


Figure 5.5- A block diagram of SR flip flop.

5.5 D FLIP FLOP

The D flip flop is a modified version of SR flip flop which eliminates the undesirable condition where flop flop state is not defined. This can be achieved by short circuiting inputs S and R of SR flip flop and introducing not gate between them as shown in Figure 5.6 (a). The block diagram of the D flip flop is shown in

Figure 5.6 (b)which is mostly used for design and analysis of sequential circuits.



(a) **(b)**

Figure 5.6- Logic circuit and block diagram of D flip flop.

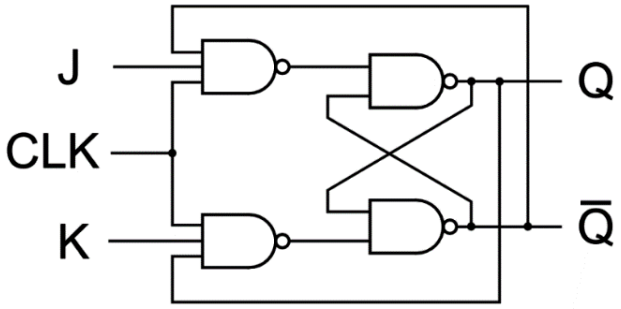
This makes SR flip flop to receive only two input combinations S=0,R=1 and S=1,R=0. The truth table of D flip flop contains only one input D as shown in Figure 5.7. If input D=0, the flip flop goes to 0 state or reset state. But when input D=1, the flip flop goes to 1 state or set state.

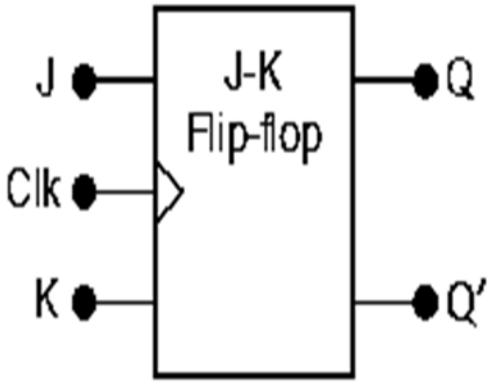
D	Q(t+1)
0	0
1	1

Figure 5.7- Truth table for D flip flop

5.6 JK FLIP FLOP

The SR flip flop discussed earlier suffers from the undesirable condition (when both S=1 and R=1) which causes undefined state. A J-K flip flop is a refinement of SR flip flop which defines state transition for even invalid inputs S=1 and R=1. This can be achieved by replacing previous inputs S and R with J and K respectively and introducing one more input to each NAND gate from outputs Q and Q'. The logic circuit and the block diagram of JK flip flop is shown in Figure 5.10 (a) and (b) respectively.





(a) **(b)**

Figure 5.10- logic circuit and block diagram of JK flip flop.

The operation of JK flip flop are defined for all 4 possible input combinations. The operation of JK flip flop is same as SR flip flop with the addition of a new transition of state for inputs J=1, and K=1. The operation of JK flip flop is shown with truth table in Figure 5.11.

J	K	$Q_{(t+1)}$
0	0	$Q_{(t)}$
0	1	0
1	0	1
1	1	$\overline{Q}_{(t)}$

Figure 5.11- Truth table of JK flip flop.

An important thing to note that when the inputs J=1 and K=1, during the duration of active clock pulse, the flip-flop keep on producing output as complement of present state until the clock pulse goes back to 0. So the flip flop changes its state multiple times in a single clock pulse. For this reason, this version of JK flip flop never used in real time. This problem can be eliminated by a modified version of JK flip flop known as master slave flip flop which will be discussed next.

5.7 MASTER-SLAVE OR EDGE-TRIGGERED

The J K flip flop is a clocked flip-flop which is triggered by pulses. A pulse is the entire duration starting from an initial value of 0 and suddenly goes to 1, and after a short time, it returns back to its initial value of 0. A positive transition i.e. transition from 0 to 1 is called as positive edge and the negative transition from 1 to 0 is called as negative edge. A positive pulse (as shown in Figure 5.13) is the duration from positive edge to negative edge of a clock pulse. On the other hand, a negative pulse (as shown in Figure 5.13) is the duration from negative edge to positive edge of a clock pulse.

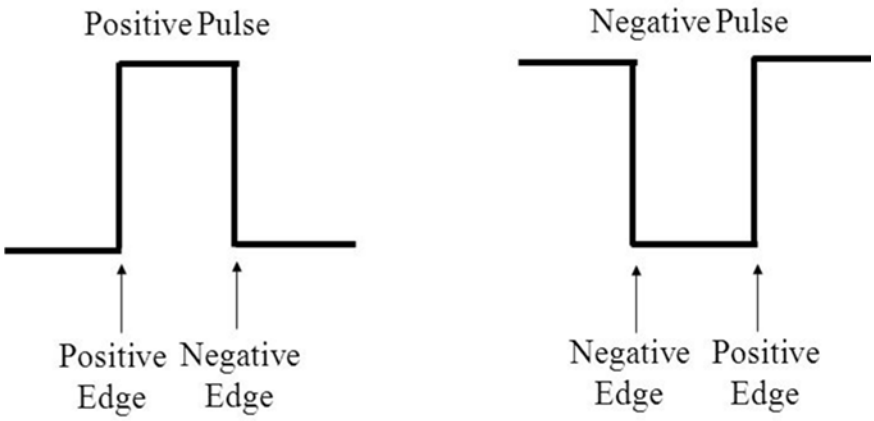


Figure 5.13- A pulse is duration between positive and negative edge of a clock pulse.

Assume that a clock pulse is applied to the J K flip flop and both its inputs are at logic 1. At the positive edge of the clock pulse, it changes its state. Since, the inputs are at logic 1 for the entire duration of positive pulse, the output of the flip flop continue to change from 0 to 1 and vice versa. This continuous toggling of JK flip flop when both its inputs are at logic 1 is called as race around condition. But, we want the JK flip flop to change its state only once throughout the clock pulse. If we can design the JK flip flop in such a way that, the flip flop responds to its inputs only at either positive edge or negative edge, then this race around condition can be eliminated. A master slave JK flip flop is such a design which works on this principle.

A master slave JK flip flop consists of two SR flip flops which are cascaded in such a way that the outputs of the second flip flop are connected back to inputs of first flip flop. The circuit for master slave flip flop is shown in Figure 5.14.

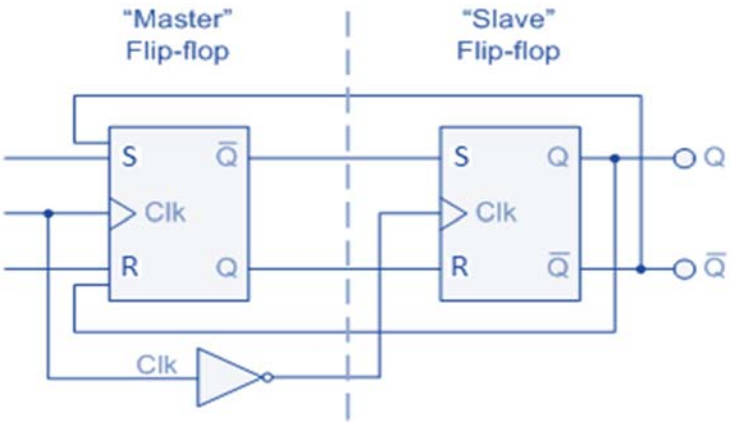


Figure 5.14- Master slave JK flip flop.

The first flip flop is called as master flip flop which works when a clock pulse CP=1. The second flip flop is called as slave flip flop which works when the clock pulse CP=0. Due to the presence of NOT gate, at any time either master or slave flip flop will be active. When a clock pulse CP=1, at the positive edge of clock pulse, the J and K inputs are passed to the master flip flop and are held there till the negative edge of the clock pulse. During this clock pulse the slave flip flop is disabled because the clock pulse CP=1. When the clock pulse CP becomes 0, at the negative edge of the clock pulse, the slave flip flop is active and the output of master flip flop is passed to inputs of the slave flip flop. The slave flip flop is simply SR flip flop which passes the outputs of the master flip flop so that outputs Q=y and Q'=y'. The outputs Q and Q' are available as feedback inputs to the master flip flop which is currently disabled because the clock pulse CP=0. When the clock pulse CP again becomes 1, at the positive edge, the master flip flop becomes active and the slave flip flop outputs are available as the inputs to master flip flop. This way, forJ and K inputs, the output of the master flip flop appears at Q and Q' always at the negative edge of the clock pulse CP=1. This makes the JK flip flop to change its state only once for entire clock pulse CP=1.

5.8 T FLIP FLOP

A T flip flop is a modified version of JK flip flop with single input. The T flip flop can be obtained from the JK flip flop when both of its inputs J and K are combined together to get a single input as shown in Figure 5.15.

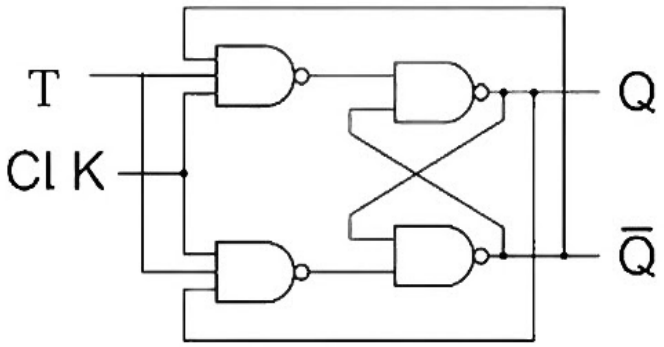


Figure 5.15- logic circuit of T flip flop.

The operation of the T flip flop is shown with truth table in Figure 5.16. The T flip flop remains in the same state as previous output state when input T=0. But when the input T=1, the flip flop complements or toggles its previous output state.

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

Figure 5.16 - Truth table of T flip flop.

5.9 REGISTER

In previous section, we have seen that a flip flop holds 1 bit of binary information. A register is a group of flip flops connected in a certain manner. An n-bit register consists of n flip flops which is capable of storing n bits of binary information. Beside flip flops, a register may also contains logic gates which control the flow of binary information into the register. There are various types of registers on MSI circuits. A simple register consists of only flip flops without any logic gates. For example, Figure 5.17 shows the simplest possible register with D flip flops only.

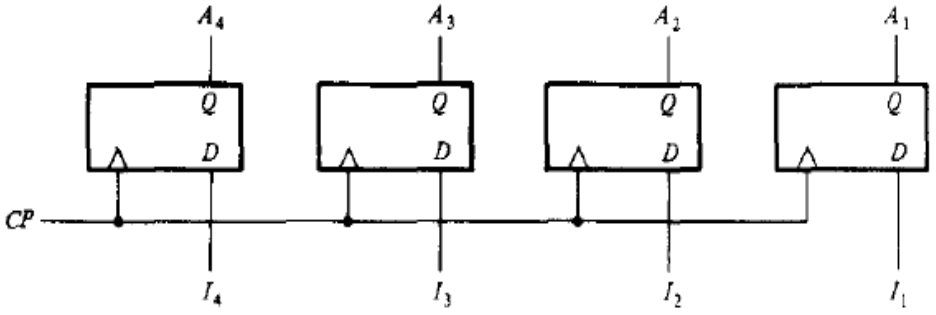


Figure 5.17 - 4 bit register using D flip flops.

When a clock pulse CP is applied to the register, it enables all 4 flip flops simultaneously. The binary information present at all four D inputs of flip flops transfer and appear at output Q of each flip flop. When the clock pulse CP goes to 0, the binary information retained at Q outputs of flip flops. Since flop flops are sensitive to clock pulse duration, the register thus obtained works as long as the CP=1. A register which is sensitive to pulse duration is also called as gated latch. They are suitable for temporary storage of data. In the subsequent discussion, we discuss registers which consist of groups of flop flops sensitive to clock pulse transitions or edge triggered.

Shift Register : If a register is capable of shifting its bits either right or left side, it is called as shift register. An n bits shift register contains n flip flops connected in a series with a common clock pulse such that the output of one flip flop becomes

input to next flip flop. An n bits shift register is capable of storing n bits of information. For example, Figure 5.18 shows a 4 bits shift register with 4 D flip flops connected in series with a common clock pulse.

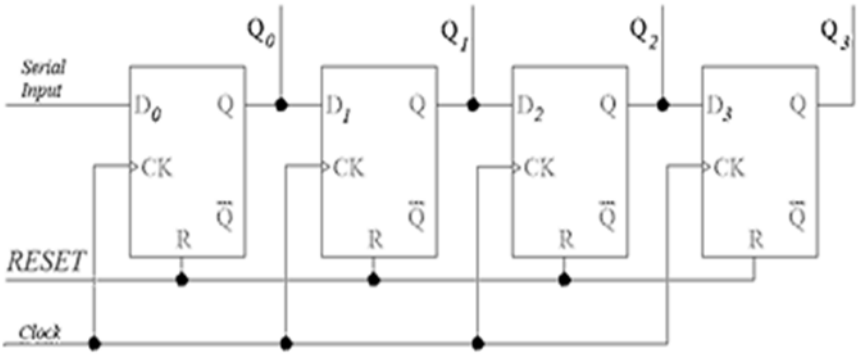


Figure 5.18- 4 bits shift register

If a shift register shifts its bits to left, it is called left shift register. If a shift register shifts its bits to right, it is called right shift register. In each clock pulse the content of the right shift register shifts one bit to the right. The data bits are fed through serial input from leftmost register and they are collected from rightmost register through serial output. Shift registers are commonly used in calculators and computers for data storage and movement. For example, the two binary numbers are stored in registers before applying addition operation. A data bits in a shift register can be fed in and out of the register serially one bit at a time or all together parallel at the same time. Based on the basic movement of data through the register, they are classified into 4 categories.

i. **Serial-in to Serial-out (SISO):** It is a shift register which stores input data serially with one bit after the other bit through a single data line and produces a serial output with one bit at a time through a single output line. An n bit data requires n clock pulses to store the input data and (n-1) clock pulse to shift the inputs out of the n bit Serial-in-Serial-out-shift-register. It requires total n-1+n =2n-1 clock pulse for storing input data and taking out the stored input from the SISO register.

   For example, a 3 bits serial-in serial-out shift register is shown in Figure 5.19. It consists of 3 D flip-flops connected in series with a common clock.

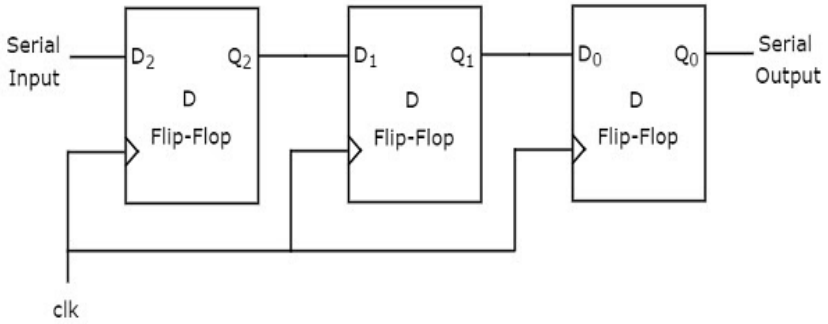


Figure 5.19 - A 3 bits serial-in serial-out shift register

The above circuit is a right shift register which takes serial data input from the left side of the flip flop.

ii. **Serial-in to Parallel-out (SIPO):** The shift register which stores input data with one bit at a time through a single data line and the stored data is taken out of the register in parallel all at a time through multiple output lines is called Serial-in to Parallel-out shift register. An n bit data requires n clock pulses to store the input data and 0 clock pulse to take the stored data out of the n bit Serial-in-Parallel-out shift register. It requires total n+0=nclock pulse for storing input data and taking out stored input from the SIPO register. For example, a 3 bits serial-in parallel-out shift register is shown in Figure 5.20. It consists of 3 D flip-flops connected in series with a common clock.

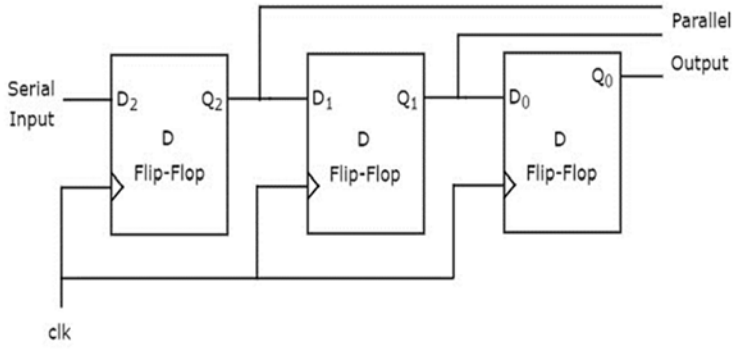


Figure 5.20 - A 3 bits serial-in parallel-out shift register.

The above circuit is a right shift register which takes data input serially from the leftmost flip flop and stored input are available parallel all at once at each flip flop output.

iii. **Parallel-in to Serial-out (PISO):** The shift register which stores input data parallel all at once from each flip flop input and the stored data taken out of the register serially with 1 bit at a time is known as Parallel-In Serial-Out shift register. An n bit data requires 1 clock pulse to store the input data into the register and n-1 clock pulse to take the stored data out of the n bit Parallel-in-serial-out shift register. It requires total 1+n-1=n clock pulses for storing and taking out the stored input from the PISO register. For example, a 4 bits Parallel-in to Serial-out shift register is shown in Figure 5.21. It consists of 4 D flip-flops connected in series with a common clock.

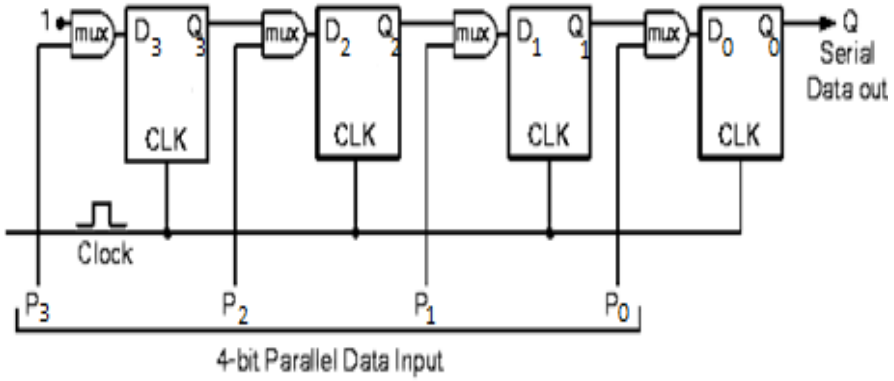


Figure 5.21- 4 bits Parallel-in to Serial-out shift register

In the above circuit, the input data is directly connected with each flip-flop through multiplexer to stored data parallel. The output is obtained from single output line from the rightmost flip flop.

iv. **Parallel-in to Parallel-out (PIPO) :** The shift register which stores input data parallel all at once and the stored data also taken out of the register parallel with all bits simultaneously at once is known as Parallel-In Parallel-Out shift register. An n bit data requires 1 clock pulses to store the input data into the register and 0 clock pulse to take the stored data out of the n bit Parallel-in-Parallel-out shift register. It requires total 0+1=1 clock pulse for giving input data and taking out stored input from the PIPO register. For example, a 4 bits Parallel-in to Parallel-out shift register is shown in Figure 5.22. It consists of 4 D flip-flops connected in series with a common clock.

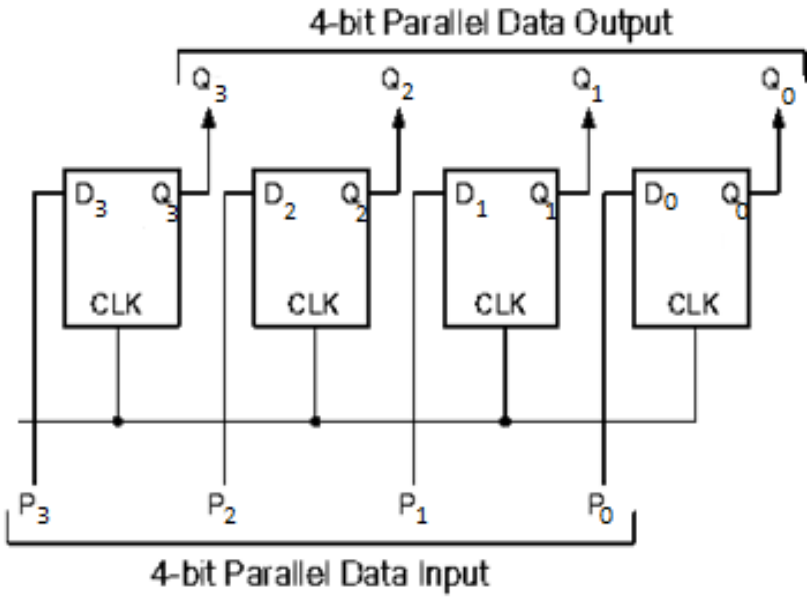


Figure 5.22- A 4 bits Parallel-in to Parallel-out shift register

In the above circuit, there are no interconnections between the individual flip-flops. Input data is given separately with each flip flop and the outputs are also collected individually from each flip flop.

The shift register which can perform all the above four operations is called as Universal shift register.

5.10 COUNTER

A counter is a special type of register which goes through a sequence of states when clock pulses are applied. For example, Figure 5.23 shows state transitions of a 3-bit binary counter. The binary states are indicated inside the circles, the counter repeats the binary count sequence as shown in Figure 5.23.

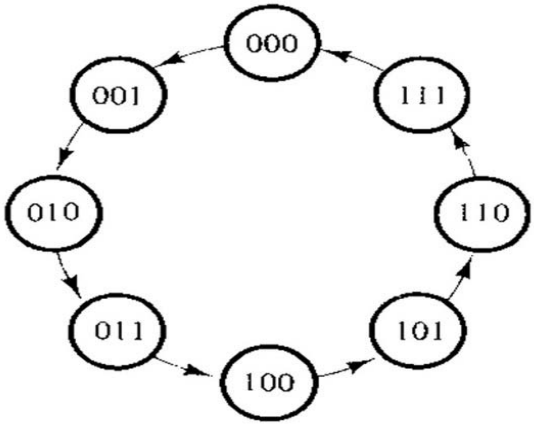


Figure 5.23- State diagram of 3 bit binary counter.

A counter may contain local gates which are connected in a certain way to produce the required sequence of states. A counter is basically used to count the number of clock pulses applied to it. For any counter, we are more interested in the total number of possible states because it tells us two things: mod value of the counter and the number of clock pulses it can count. Broadly, the counters present in MSI circuits are divided into three categories: ring counter, ripple counters and synchronous counters. The ripple counters are also called asynchronous counters.

Ring counter :

1. **Ordinary Ring Counter :** A k bits ring counter consists of k flip flops cascaded together such that the output of one flip flop acts as input to its next flip flop and the uncomplemented output of the last flip flop is given as feedback input to the first flip flop. All the flip flops are driven by a common clock pulse. The number of possible states in an ordinary ring counter is given by the number of flip flops used in it.

   n number of flip flops → n possible states

   → n clock pulse it can count

   For example a 4 bit ordinary ring counter is shown in Figure 5.24. It consists of 4 flip flops, so there are 4 possible states in it. Therefore it is mod 4 counter which can count 4 clock pluses.

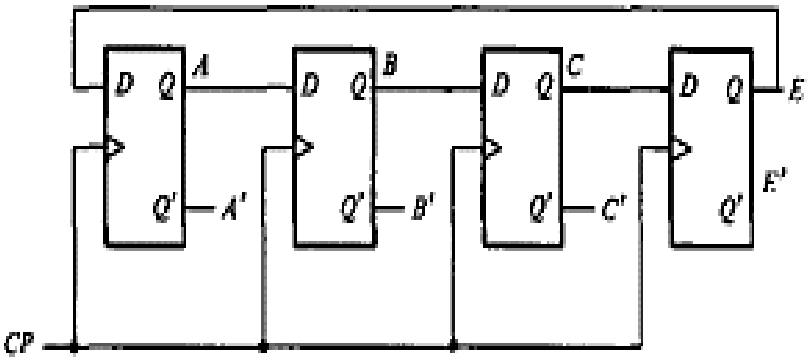


Figure 5.24- 4 bit ordinary ring counter.

2. **Johnson or twisted ring counter :** We have seen that a k bits ordinary ring counter provides k distinguishable states. The number of states can be doubled in an ordinary ring counter if the feedback input to first flip flop is given from the complemented output of last flip flop. The counter thus obtained is called as Johnson counter or twisted ring counter.

   n number of flip flops------> 2n possible states

   ------>2n clock pulse it can count

   For example a 4 bit Johnson counter consists of 4 flip flops as shown in Figure 5.25. It can provides 2*4 distinguishable states and so it can count 8 clock pulses.

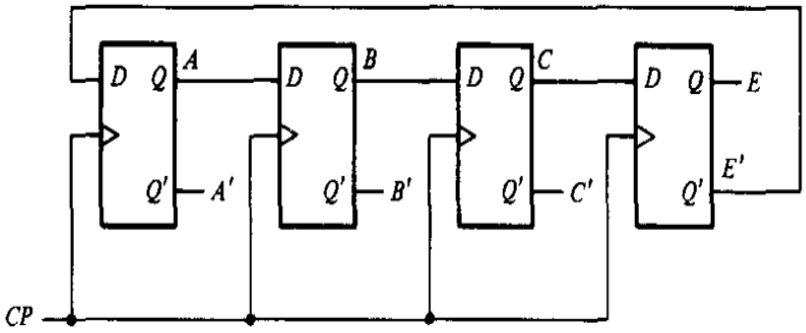


Figure 5.25- A 4 bit Johnson counter

Ripple counter : In a ripple counter or asynchronous counter, the output of one flip flop serves as a source of triggering the other flip flops in the counter. This is shown in Figure 5.26. In this counter, the clock pulse input of all flip flops are not a common clock pulse, rather each flip flop is triggered based on the output of its previous flip flop in the counter.

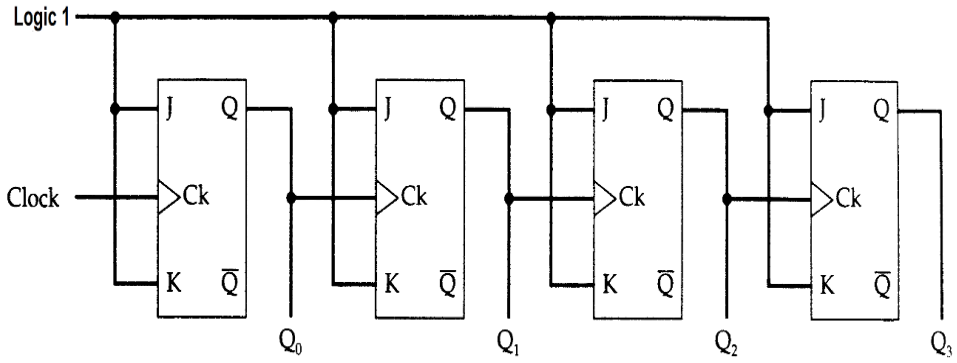


Figure 5.26- A ripple counter.

When there are no logic gates present, a ripple counter with n flip flops results in 2^n states.

n number of flip flops------> 2^n possible states

------>2^n clock pulse it can count

Synchronous counters : In a synchronous counter, all flip flops have a common clock pulse so that all flip flops are triggered simultaneously. The transition of a flip flop state in the synchronous counter is dependent on the present state of other flip flops. For example, Figure 5.17 shows 4 bit synchronous binary counter which gives 16 different states.

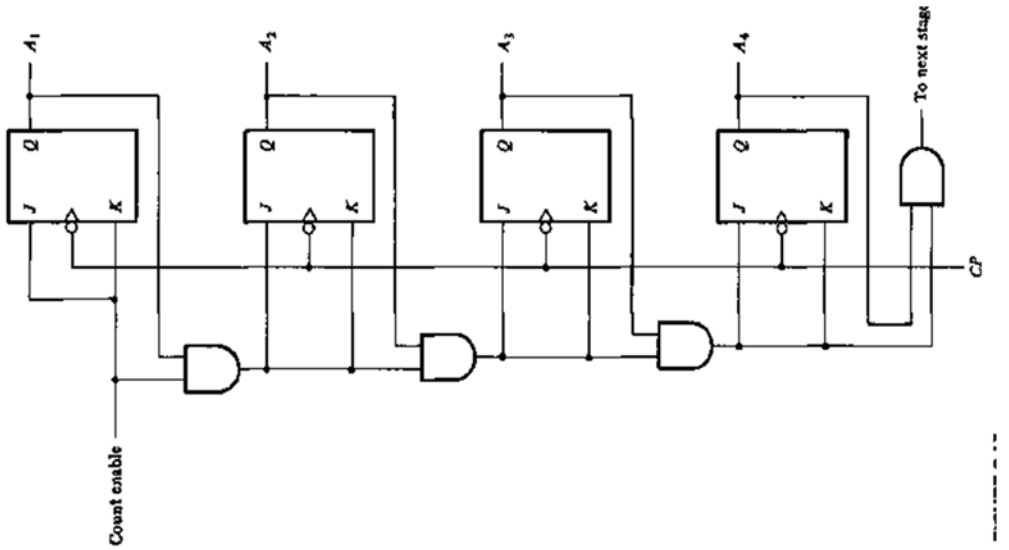


Figure 5.27- A 4 bit synchronous binary counter.

When there are no logic gates present, a synchronous counter with n flip flops results in 2^n different states.

n number of flip flops------> 2^n possible states

------>2^n clock pulse it can count

5.11 SUMMARY

- We saw how sequential circuits differs from combinational circuits.
- We discussed the working of various flip flops with their truth table.
- We illustrated the race around condition in JK flip flop and seen that how this problem is overcame with master slave JK flip flop.
- We saw the purpose of registers and counters.
- We discussed various types of counters: ring counters, ripple counters and asynchronous counters.
- We explained various types of shift registers: Serial-in to Serial-out, Serial-in to Parallel-out, Parallel-in to Serial-out, Parallel-in to Parallel-out.

5.12 TERMINAL QUESTIONS

1. How sequential circuits are different from combinational circuits?
2. Explain sequential circuits with suitable block diagram.
3. Discuss truth table of SR flip flop with its block diagram.
4. Explain the working of JK flip flop with its truth table.
5. What is race around condition? How it is avoided?
6. How does the D flip flop different from SR flip flop?
7. Discuss T flip flop with its truth table.
8. What is register?
9. What is counter?
10. Differentiate between register and counter?
11. Explain various types of shift registers with suitable example.
12. What is universal shift register?

REFERENCES

1. Mano, M. Morris. *Digital logic and computer design*. Pearson Education India, 2017.
2. Jain, Rajendra Prasad. *Modern digital electronics*. Tata McGraw-Hill Education, 2003.

Uttar Pradesh Rajarshi Tandon Open University

Master of Computer Application

MCA-105/MCS-106 /PGDCA-105

Computer Organization

BLOCK

2

BASIC BUILDING

UNIT-6

Building Blocks

UNIT-7

Instruction

UNIT-8

Addressing Techniques

Course Design Committee

Course Preparation Committee

Mr. Manoj Kumar Balwant **Author Block 1 (Unit 1,2,3,4,5)**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad.
Dr. JitendraPande **AuthorBlock 2, 3 (Unit 6,7,8,9,10,11)**
Associate Professor
School of Computer Sciences & Information Technology
Haldwani, Uttarakhand 263139
Prof. Ashutosh Gupta **Editor Block 1 (Unit 1, 2, 3, 4, 5)**
Director (In-Charge)
School of Computer & Information Sciences, UPRTOU Allahabad
Prof. Abhay Saxena **Editor Block 2, 3 (Unit 6, 7, 8, 9, 10, 11)**
Professor and Head, Department of Computer Science
Dev SanskritiVishwavidyalya, Hardwar, Uttrakhand
Mr. Manoj Kumar Balwant **Coordinator**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad.

©UPRTOU, Prayagraj - 2020
ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.
MCA-105/96
Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

BLOCK INTRODUCTION

This block is designed to give learners a clear understanding of components of a computer system. Each major component is further described by decomposing into its subcomponents and describing their structure and function. The course comprises of three units which are as follows:

Unit-6 deals with the basic components of the computer system. It also discusses how these components work together to perform the different functions a computer. It also discusses memory and its various types.

Unit-7 This unit deals with Instruction Formats, Representation of Three Address Instructions, Two Address Instructions, One Address Instructions, Zero Address Instructions and RISC Instructions are described in this unit. In this unit various operations associated with Register Reference Instruction, Memory Reference Instructions and Input-Output Reference Instructions are discussed.

Unit- 8 introduces you with some addressing modes like Implied Mode, Immediate Mode, Register Mode, Register Indirect Mode, Auto increment or Auto decrement Mode, Direct Address Mode, Indirect Address Mode, Relative addressing Mode, Indexed Addressing Mode, Base Addressing Mode and Processor Registers.

UNIT-6 BASIC BUILDING BLOCKS OF A COMPUTER

Structure

6.1 LEARNING OBJECTIVES

After reading this unit, you will be able to:

- Understand the basic building blocks of a Computer.
- Know the functional units of a computer.
- Explain the functioning of a CPU.
- Understand the functioning of Hardwired Control Unit.
- Understand the functioning of Programmed Control Unit.

- Understanding the functioning of various input and output devices.
- Differentiate primary and secondary memories.

6.2 INTRODUCTION

Computer Technology is now part of our everyday life, and almost every task we encounter involves the use of computer technology. This unit presents a brief discussion on computer and their components and other basic concepts you need to familiarize yourself with before discussing the details of the internal architecture of the computer.

6.2. BASIC BUILDING BLOCKS OF A COMPUTER

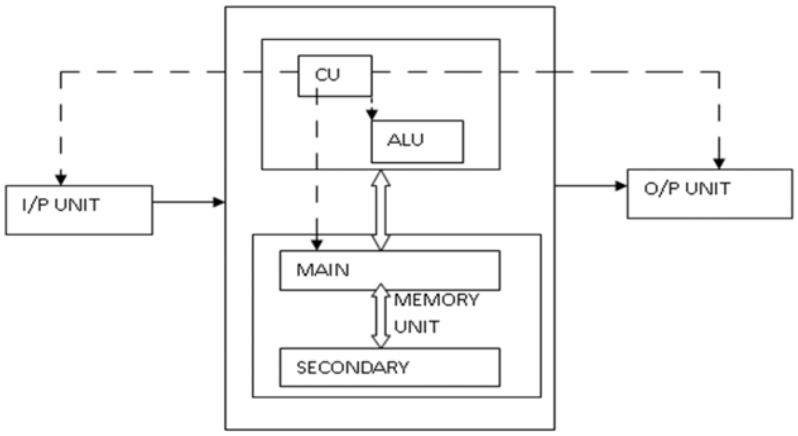


Figure-1 : Basic Building Block of a Computer

----► Dotted lines indicate the control signals issued by Control unit.

——► Represent data or instructions.

A computer is an electronic device that takes input such as numbers, text, sound, image, animations, video, etc., processes it, and converts it into meaningful information that could be understood, presenting the changed input (processed input) as output. All numbers, text, sound, images, animations, and video used as input are called data, and all numbers, text, sound, images, animations, and video returned as output are called information. Input is the raw data entered into the computer by using input devices. It is an electronic machine/device which can input data, process them according to the instruction given and then give out the meaningful information.

- The data consists of numbers, text, sound, images, animations, and video.
- The process converts numbers, text, sound, images, animations, and video (data) into usable data, which is called information.
- The information consists of numbers, text, sound, images, animations, and video that has been converted by the process.

- The data is inserted using an input device.
- The central processing unit (CPU) converts data to information.
- The information is put on an output device.

A storage device is an apparatus for storing data and information. A basic computer consists of 4 components: an input device, a CPU, output devices, and memory. All the units of a digital computer are connected through a conducting path called Bus.

The task of the digital computing unit is to accept the data to be processed via its input unit. The CPU of the computer process the data based on the issued to the computer by the user through program. After processing, the result may be stored in the memory of the computer or may be displayed to the user via output unit.

The four functions are carried out by basic **functional units** namely:

1. Input Unit.
2. Output Unit.
3. Central Processing Unit.
4. Memory Unit.

6.3 INPUT UNIT

We use input devices to provide information to a computer, such as typing a letter or giving instructions to a computer to perform a task. Some examples of input devices are described in the following list.

1. **Mouse :** A device that you use to interact with items displayed on the computer screen. A standard mouse has a left and a right button.

Figure-1: Mouse

2. **Trackball:** This is an alternative to the traditional mouse and is favoured by graphic designers. It gives a much finer control over the movement of items on the screen. Other screen pointing devices are pointing stick, touch pad, joystick, light pen, digitizing table.

Figure-2: Trackball

3. **Keyboard:** A set of keys that resembles a typewriter keyboard. You use the keyboard to type text, such as letters or numbers into the computer.

Figure 3 : Keyboard

4. **Scanner:** A device that is similar to a photocopy machine. You can use this device to transfer an exact copy of a photograph or document into a computer. A scanner reads the page and translates it into a digital format, which a computer can read. For example, you can scan photographs of your family using a scanner.

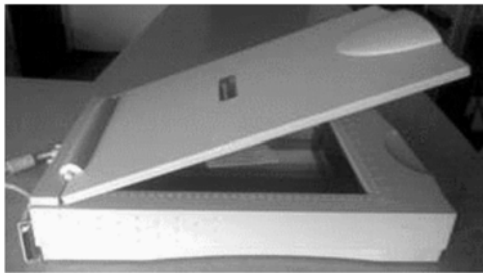


Figure 4: Scanner

5. **Barcode Readers:** When used in a business barcodes provide a lot of information. Made up of columns of thick and thin lines, at the bottom of which a string of numbers is printed.

Figure 5: Barcode reader

6. **Multimedia devices:** This is the combination of sound and images with text and graphics. To capture sound and image data, special input devices are required.

   a. Microphone: Voice input, for instance, can be recorded via a microphone. A device that you can use to talk to people in different parts of the world. You can record sound into the computer by using a microphone. You can also use a microphone to record your speech and let the computer convert it into text.

Figure 6: Microphone

   b. Webcam: A device that is similar to a video camera. It allows you to capture and send the live pictures to the other user. For example, a webcam allows your friends and family to see you when communicating with them.

Figure 7 : Webcam

c. Digital cameras: record photographs in the form of digital data that can be stored on a computer. These are often used to record photographs on identity cards.

Figure 8 : Digital camera

6.4 OUTPUT DEVICES

Output devices in the computer system are the equipment whereby the result of a computer operation can be viewed, heard or printed. You use output devices to get feedback from a computer after it performs a task.

1. Monitor : A device that is similar to a television. It is used to display information, such as text and graphics, on the computer.

Figure 9: Monitor

2. Printer: A device that you use to transfer text and images from a computer to a paper or to another medium, such as a transparency film. You can use a printer to create a paper copy of whatever you see on your monitor.

   a. Impact printers: Dot matrix printers are an example of impact printers. They form characters from patterns of dots. They are inexpensive, but the output can be difficult to read.

Figure 10 : Impact printer

b. Non impact printers: Inkjet printers work by shooting a jet of ink in the shape of the character required, they provide good, low-cost color printing.

Figure 2 : Non-impact printer

c. Laser printer: a laser beam is directed at an electro-statically charged surface, creating a template of the page to be printed. This template is then used to transfer the ink to the page. Toner sticks to the light images and is transferred to paper.

Figure 3: Laser Printer

3. Plotter: A plotter is an output device similar to a printer, but normally allows you to print larger images. It is used for printing house plans and maps.

Figure 4: Plotter

4. Multimedia Output Device: The most common multimedia output is sound, including music. The audio output device on a computer is a speaker. Headphones can also be used to receive audio output.

Figure 5: Multimedia Output Device

6.5 CENTRAL PROCESSING UNIT

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions. The term has been used in the computer industry at least since the early 1960s. Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and I/O circuitry.

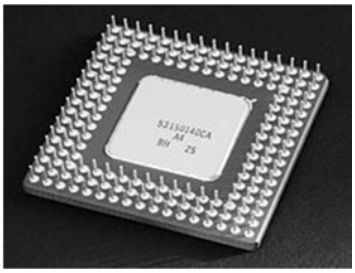


Figure 6: Bottom side of an Intel 80486DX2[1]

The form, design and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that fetches instructions from memory and "executes" them by directing the coordinated operations of the ALU, registers and other components.

Figure 7: An Intel 80486DX2 CPU, as seen from above

Central Processing Unit is the brain of the computer. Based on the input provided to the CPU via one of its input device, the CPU process the data and converts it into meaningful information. It is the place where all the computing takes place. The CPU mainly consists of three parts:

- Arithmetic Logic Unit(ALU)
- Control Unit (CU)
- Register Set (Memory)

6.5.1 ARITHMETIC LOGIC UNIT

It is the part of a computer that performs all arithmetic computations, such as addition and multiplication, and all comparison operations. Typically, the ALU has direct input and output access to the processor controller, main memory (random access memory or RAM in a personal computer), and input/output devices. The data is transferred between the ALU and the Input/Output devices & memory through an electronic conducting path called bus. The input consists of an instruction word (sometimes called a machine instruction word) that contains an operation code (sometimes called an "op code"), one or more operands, and sometimes a format code. The operation code tells the ALU what operation to perform and the operands are used in the operation. (For example, two operands might be added together or compared logically.) The format may be combined with the op code and tells, for example, whether this is a fixed-point or a floating-point instruction. The output consists of a result that is placed in a storage register and settings that indicate whether the operation was performed successfully. (If it isn't, some sort of status will be stored in a permanent place that is sometimes called the machine status word.)

In general, the ALU includes storage places for input operands, operands that are being added, the accumulated result (stored in an accumulator), and shifted results. The flow of bits and the operations performed on them in the subunits of the ALU is controlled by gated circuits. The gates in these circuits are controlled by a sequential logic unit that uses a particular algorithm or sequence for each operation code. In the arithmetic unit, multiplication and division are done by a series of adding or subtracting and shifting operations

---

[1] Adopted from: https://courses.lumenlearning.com/zeliite115/chapter/reading-the-central- processing-unit/

There are several ways to represent negative numbers. In the logic unit, one of 16 possible logic operations can be performed - such as comparing two operands and identifying where bits don't match.

6.5.2 CONTROL UNIT

A control unit is a part of CPU which directs operations within the computer's processor by directing and controlling the input and output of a computer system.

The processor then controls how the rest of the computer operates (giving directions to the other parts and systems). A control unit works by gathering input through a series of commands it receives from instructions in a running program and then outputs those commands into control signals that the computer and other hardware attached to the computer carry out.

Control Unit perform various functions in a computer such as: it controls the movement of data between various units of a computer. It is responsible for the deciding the sequence in which the various instructions are to be executed. It is also responsible for handles multiple tasks, such as fetching, decoding, execution handling and storing results.

CUs are designed in two ways :

- *Hardwired control:* CU is made up of sequential and combinational circuits to generate the control signals The CU is made up of flip-flops, logic gates, digital circuits and encoder and decoder circuits that are wired in a specific and fixed way. When instruction set changes are required, wiring and circuit changes must be made. This is preferred in a reduced instruction set computing (RISC) architecture, which only has a small number of instructions.

- *Microprogram control*: Microprograms are stored in a special control memory and are based on flowcharts. The operation of all the hardware of the computer is control unit. It monitors and controls the input devices, output devices, memory and the ALU of the computer. In case of microprogrammed implementation of a control unit, it some addition/modification is required at the later stage of implementation, one need not to redesign the whole circuit, as in the case of hardwired unit, but control memory is updated with new microprogramme.

6.5.3 REGISTER SET

The operand (data on which the operation is to be performed) and the operation is supplied to the CPU via. memory. Now the CPU requires some location where the data on which the operation is to be performed can be manipulated. For this purpose, a very fast memory element, called registers, are used. These registers along with CU and ALU are the part of CPU. These registers hold the data and directly attached to the electronic circuitry which is required to perform various operations.

Check Your Progress

1. A ……. device is an apparatus for storing data and information.
2. Dot matrix printers are an example of …….. printers.
3. A …….. is an output device similar to a printer, but normally allows you to print larger images.
4. A …….. is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.
5. …….. is the part of a computer that performs all arithmetic computations, such as addition and multiplication, and all comparison operations.
6. The data is transferred between the ALU and the Input/Output devices & memory through an electronic conducting path called ……..
7. Op-code stands for ……
8. A …….. unit is a part of CPU which directs operations within the computer's processor by directing and controlling the input and output of a computer system.

6.6 MEMORY

All computers need to store and retrieve data for processing. The CPU is constantly using memory from the time that it is switched on until the time you shut it down. There are two types of storage devices as illustrated in the flow chart below.

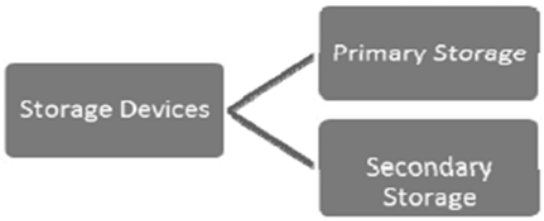


Figure 8: Types of storage devices

Primary Storage is also called main memory or immediate access store (IMAS). This is necessary since the processing unit can only act on data and instructions that are held in primary storage. Primary storage consists of two types of memory chips:

- Random Access Memory (RAM)
- Read Only Memory (ROM)

6.6.1 RANDOM ACCESS MEMORY (RAM)

Random Access Memory (RAM) is the main working memory. RAM is only filled after computer has been turned on and is given something to do. It holds data and instructions temporarily while processing takes place. RAM is volatile – this means that if the power is turned off or the computer reboots (start up again) all the information held in RAM will be lost. RAM is measured in MB (megabytes) and most entry level computers will have 1024 MBRAM but you also find some computers having up to 3 GB RAM. RAM chips are expensive and the price of a computer is determined by the amount of RAM space in the chip.

Figure 9: RAM

6.6.2 READ ONLY MEMORY (ROM)

Read Only Memory (ROM) holds data and instructions necessary for starting up the computer when it is switched on. These instructions are hard-wired at the time of manufacture. ROM is permanent and cannot be deleted but can only be accessed or read, hence the name Read Only Memory. Data stored in ROM is non-volatile – meaning that memory will not be lost when power is turned off.

Figure 19: ROM

6.6.3 UNITS OF STORAGE

The memory of all digital computers is two-state (bi-stable) devices. Computers operate using a **binary number system**– and therefore use *binary digits* (**bits**).

Bits have only two values by 0 and 1. **A bit** is the smallest unit of storage in a computer. The amount of data and instructions that can be stored in the memory of a computer or secondary storage is measured in bytes.

A byte is made up of a combination of eight (8) **bits** and has the storage power to represent one character (a character is a letter or symbol or punctuation mark or blank space).

Table 1: Unit of Storage

Units of Storage	
1 byte	8 bits
1 kilobyte (K)	1024 bytes
1 megabyte (MB)	1000 kilobytes (approx. 1 million bytes)
1 gigabyte (GB)	1000 megabytes (approx. 1 billion bytes)
1 terabyte (TB)	1000 gigabytes (approx. 1 trillion bytes)

6.7 SECONDARY STORAGE DEVICES

PCs use a simple method of designating disk drives to store data. These drives are assigned letters of the alphabet.

A Drive	Floppy drive. Still found in older computers
C Drive	Internal hard drive (hard disk drive) situated inside the system case.
D Drive	Usually the CD-ROM/DVD-ROM drive although can also be used for another virtual or physical hardware if a second one is deployed.
E Drive or Higher	Usually use for any other disks, such as CD-writer, USB flash drive, external hard drive, etc.

Data and information stored on a permanent basis for later use. Secondary storage is cheaper to purchase and access. Hard disks, Zip drives, Optical disks (CD's and DVD's)are all examples of secondary storage.

1. **Internal Hard Disks -** are rigid inflexible disks made of highly polished metal. Data is stored magnetically. They can contain a single disk or two or disks stacked on a single spindle. They come in a variety of sizes but all have a very high storage capacity compared to floppy disks. An average computer has a hard disk of about 80 -250GB. It provides direct access to information.

Figure 10: Internal hard disc

2. **External hard Disks/Drive** - same features as the internal hard disks, but are external to the system unit and therefore can be carried around.

Figure 11: External hard Disc

3. **USB flash drive** consists of a flash memory data storage device integrated with a USB (Universal Serial Bus) interface. USB flash drives are typically removable and rewritable. They come in a variety of sizes to include 128MB, 256MB,512MB, 1G, 2G, 8G etc.

Figure 12: USB flash drive

4. **Memory Card -** Use mainly with digital cameras, cellular phones and music players (MP3, MP4andiPods). They offer high-re-record ability and fast and power-free storage. Data can be access by linking the card to a computer using a USB cable or a memory card reader.

Figure 13: Memory card

5. **Optical Disks** are disks that are read by laser beams of lights. The three main types are CD-R, CD-RW and DVD.

Figure 14: Optical disc

a. **CD-R** or CD-ROM (Compact Disk – Read Only Memory) are so called because you can only red the information on the CD-ROM. They are particularly useful for storing multimedia (texts, graphics, sound and videos), application software packages.

b. **CD-R** or Compact Disk Recordable allows you to write information onto the disk only once using a CD recordable burner.

c. **CD-RW** or Compact Disk Rewriteable, allows you to write and erase information from the disk many times. They are used to store large volumes of information such as texts, graphics, sound and video.

d. **DVD** disks or Digital Versatile Disks are specifically created to store movies. A typical DVD disk can hold between 4.7GB and 17GBof information.

Check Your Progress

9. …….. is the main working memory.
10. ……holds data and instructions necessary for starting up the computer when it is.
11. A …… compute is the smallest unit of storage.

6.8 SUMMARY

1. Since the 1940s when computer technology was used to support the creation of firing tables for the artillery and to the introduction of the World Wide Web network of computers in the 1980s computer technologies have become a large part of our everyday life. The use of computers is accelerating. They are now in our cars, our phones, our refrigerators. Almost every type of electronic device has a computer chip in it. Each chip relays on commands. Commands must be input using different devices. The next topic will examine some of these input and

output devices.

2. Computer hardware consists of input, output, process and storage devices.
3. You use input devices such as keyboard, mouse, scanner and multimedia devices to provide information to a computer.
4. Output devices are used to get feedback from a computer after it performs a task.
5. Examples include monitor, printer and multimedia devices.
6. CPU takes raw data and turns it into information. The CPU is made up of Control Unit, Arithmetic Logic Unit (ALU) and the main memory.
7. Storage devices are divided in primary and secondary storage devices.
8. Primary/main memory is subdivided into ROM and RAM.
9. Random Access Memory (RAM) is the main memory and allows you to temporarily store commands and data.
10. Read Only Memory (ROM) retains its contents even after the computer is turned off.
11. A bit is the smallest unit of storage in a computer.
12. The amount of data and instructions that can be stored in the memory of a computer or secondary storage is measured in bytes. A byte is made up of a combination of eight bits and has the storage power to represent one written character.
13. Hard disks, CD-R, CD-RW and DVD are secondary storage devices.

6.9 ANSWERS TO CHECK YOUR PROGRESS

1. Storage
2. Impact
3. Plotter
4. Central Processing Unit(CPU)
5. ALU
6. Bus
7. Operation Code
8. Control
9. Random Access Memory (RAM
10. Read Only Memory (ROM)
11. Bit

6.10 TERMINAL QUESTIONS

1. What is the role of Control Unit of a CPU?
2. What are the two possible configurations of a Control Unit?
3. Draw and explain the basic building block of a computer.
4. What is volatile memory? Give examples.
5. Give examples of some screen pointing devices.
6. What is an input device? Give some examples.
7. What is an output device? Give some examples.
8. What is a bus?
9. What is the difference between RAM and ROM?

UNIT-7 INSTRUCTION

Structure

7.1 LEARNING OBJECTIVES

After reading this chapter, you will be able to:

- Define an Instruction cycle.
- Explain different steps involved in an instruction cycle.
- Understand the execution of register reference instructions.
- Understand the execution of memory reference instructions.
- Understand the execution of Input/Output reference instructions.

7.2 INTRODUCTION

The instruction code format of a computer varies from platform to platform. A single computer can also have variety of instruction formats. The instructions are given to the computer via program. Once the program is executes, the instructions is decoded and the CPU generates all the necessary control signals to initiate the operation mentioned in the instruction.

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

The Operands residing in processor registers are specified with a **register address**. Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

1. Single accumulator organization.
2. General register organization.
3. Stack organization.

7.3 THREE ADDRESS INSTRUCTIONS

This address instruction has the following parts

- Operation code
- Address of two operands called Address 1 and Address 2
- Address of the memory location where the result of the operation is to be stored i.e. Address of the destination.

Operation Code	**Destination**	**Source 1**	**Source 2**

← Address 1 → ←Address 2 → ←Address 3→

Figure 15 : Three Address Instruction Format

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B)* (C + D) is shown below, together with

comments that explain the register transfer operation of eachinstruction.

*ADD Rl, A, B R1←M[A]+M[B] ADD R2, C, D R2←M[C]+M[D] MUL X, Rl, R2 M[X]←R1*R2*

We will assume that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X. It is assumed that the computer has two processor registers, Rl and R2. The symbol M[A] denotes the operand at memory address symbolized by A. The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary coded instructions require too many bits to specify three addresses.

An example of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

7.4 TWO ADDRESS INSTRUCTIONS

Two-address instructions are the most common in commercial computers.

Operation	**Destination**	**Source**

← Address 1 → ← Address 2 →

Figure 16: Two Address Instruction Format

Here again each address field can specify either a processor register or a memory word. The program to evaluate

X = (A + B) * (C + D) is as follows:

MOV Rl, A R1←M[A] ADD Rl, B Rl←Rl + M[B] MOV R2, C R2←M[C]

*ADD R2,D R2←R2 + M[D] MUL Rl ,R2 R1←R1*R2 MOV X, R1 M[X]←R1*

7.5 ONE ADDRESS INSTRUCTIONS

One-address instructions use an implied accumulator (AC) register for all data manipulation.

Op-code	Address 1

Figure 17 : One Address Instruction Format

For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate

X = (A + B)*(C + D)

is as follows:

LOAD	**A**	**AC ←M[A]**
ADD	B	AC ←AC+M[B]
STORE	T	M[T] ← AC
LOAD	C	AC← M[C]
ADD	D	AC ←AC + M[D]
MUL	T	AC ←AC+M[T]
STORE	X	M[X] ←AC

7.6 ZERO ADDRESS INSTRUCTIONS

A stack-organized computer does not use an address held for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address held to specify the operand that communicates with the stack. The following program shows how **X = (A + B) * (C + D)** will be written for a stackorganized computer. (TOS stands for top of stack.)

PUSH	**A**	**TOS←A**
PUSH	B	TOS←B
ADD		TOS←(A+B)
PUSH	C	TOS←C
PUSH	D	TOS←D
ADD		TIS←(C+D)
MUL		TOS←(C+D)*(A+B)
POP	X	M[X] ←TOS

7.7 RISC INSTRUCTIONS

The instruction set of a typical RISC processor is restricted to the use of load and store instructions when communicating between memory and CPU. All other instructions are executed within the registers of the CPU without referring to

memory. The following is a program to evaluate

X = (A + B)*(C + D)

LOAD	**R1,A**	**R1←M[A]**
LOAD	R2,B	R2←M[B]
LOAD	R3,C	R3←M[C]
LOAD	R4,D	R4←M[D]
ADD	R1,R1,R2	R1←R1+R2
ADD	R3,R3,R4	R3←R3+R4
MUL	R1,R1,R3	R1←R1*R3
STORE	X,R1	M[X] ←R1

Check Your Progress

1. The Operands residing in processor registers are specified with a address.
2. TOS stands for.
3. In case of, Zero-address instruction method the operands are stored in

7.8 BASIC COMPUTER INSTRUCTIONS

The basic instruction code has three fields, as shown in Figure 18**.**

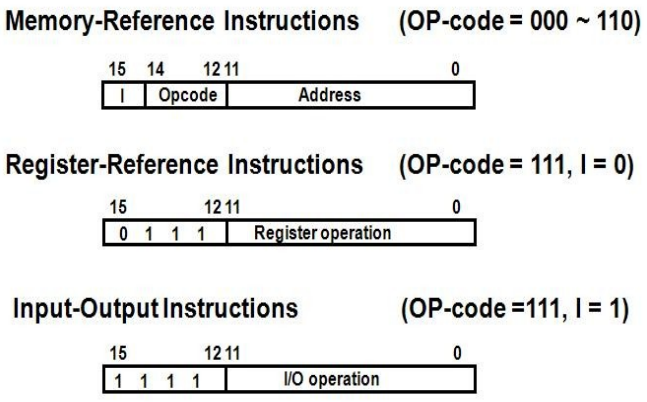


Figure 18 : Basic Computer Instruction Format

The most significant bit, i.e. bit 15, is marked as *I,* which is a mode selection bit. If it is set, it is indirect addressing mode else direct addressing mode. This bit has a direct effect on the address part (bit 0-11) of the instruction code and it is used in conjunction with other three opcode bits (bit 12-14) to specify one of the various addressing modes. In direct addressing mode, the address part of the instruction code gives the address of the operand in the main memory. In indirect mode, the address part of the instruction code specifies that address of the main memory, which contains the address of the operand in the main memory. There are some situations, when the operand is stored in one of the CPU registers or the operand is specified by the Input/Output devices itself. In these cases, no memory reference is required. Hence, the address part of the instruction code is no longer required to specify the address of the operand in the main memory. Therefore, the other 12 bits(0-11) can be used to specify test or other conditions. The above situation can be summarized as follows:

- When *I*=0 and bit 12-14 are all 1's(0111),i.e. the hexadecimal equivalent of the digit 7, it is register mode. In register mode, the other 12 bits of the address part of the instruction code are used to specify one of the various register mode instruction (please refer to table 6). The 16 bits of the instruction code is equivalent to four hexadecimal bits, with the first most significant bit of the hexadecimal equal to four most significant bits (12-15) of the instruction code. In register mode, the most significant hexadecimal bit is always 7. Rest of the 12 bits are used to specify different register related operation. For eg. 7800, which is equivalent to 0111 1000 0000 0000 in binary, specifies Clear AC (clear accumulator) operation.

- When *I*=1 and bit 12-14 are also all 1's (1111), i.e. the hexadecimal equivalent of digit F, it specifies Input/Output mode. In this case also the other 12 bits are used to specify one of the various Input/Output operations. For eg. F800 is an Input/Output operation which is used to transfer input characters to AC.

- When I=0 and bit(12-14) have any combinations except all 1's. i.e. hexadecimal equivalent values of 0-6, it is a direct addressing mode. In this case the other 12 bits are used to specify the address of the operand in the main memory. For eg. 1xxx(i.e. binary equivalent 0001 xxxx xxxx xxxx xxxx(x's denotes don't care conditions) specifies ADD which performs addition operation on the operand and the content of AC.

- When I=1 and bit (12-14) have any combinations except all 1's, i.e. hexadecimal equivalent values of 8-E, it is a indirect addressing mode. In this case the other 12 bits are used to specify the address of that location in the memory, which contains the address of the operand in main memory. If you see

- Table **2** you will observe that all the operation of direct and indirect memory mode are same, the only difference is number of main memory access to fetch operand to the CPU registers. For eg. 1xxx and 9xxx, both are used to add memory word to AC, but in case of 1xxx it is direct mode,

where only one memory reference is required to fetch the operand. In case of 9xxx, two memory references are required.

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable. The instruction set of a computer is said to be complete if it contains the following category of instructions to manipulate the data.

- Functional Instructions
  - Arithmetic, logic, and shift instructions
  - ADD, CMA, INC, CIR, CIL, AND, CLA
- Transfer Instructions
  - Data transfers between the main memory and the processor registers.
  - LDA, STA

Table 2 : Computer Instructions

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

- Control Instructions
  - Program sequencing and control

- BUN, BSA, ISZ

➢ Input/Output Instructions

- Input and output
- INP, OUT

The basic instruction set disused above and summarized in table 2 consists of all the instructions required by a basic computer. Hence, we can claim that the instruction set is complete.

Check Your Progress

1. The sequence of instructions is known is ..... .
2. The CPU reads the next instruction from memory. It is placed in an……. .
3. When ..... an operand is found, using either direct or indirect addressing, it is placed in the.

7.9 INSTRUCTION CYCLES AND SUBCYCLES

Before looking at *how* a computer does what it does, let us look at *what* it can do. The definition of a computer outlines its capabilities; a computer is an electronic device that can store, retrieve, and process data. Therefore, all of the instructions that we give to the computer relate to storing, retrieving, and processing data.

The underlying principle of the von Neumann machine is that data and instructions are stored in memory and treated alike. This means that instructions and data are both addressable. Instructions are stored in contiguous memory locations; data to be manipulated are stored together in a different part of memory.

In this unit, we will discuss the phases involved in the execution of an instruction by the CPU. We ill also discuss how a register reference, memory reference and input/output reference instructions are executed by CPU.

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycle. Each instruction cycle consists of

1. Instruction Fetch from Memory

2. Instruction Decode

3. Read Effective Address(if indirect addressing mode)

4. Instruction Execution

5. Go to step 1) : Next Instruction[PC + 1]

After execution, the control goes back to step number 1 to again fetch the new instruction. This cycle goes on till the end of the program and stops as soon as a HALT instruction is encountered. Now let us discuss each of the above steps in detail.

7.8.1 INSTRUCTION FETCH FROM MEMORY

In register transfer language, the instruction fetch can be represented as follows:

$$T_0 : AR \leftarrow PC$$

$$T_1 : IR \leftarrow M[AR],\ PC \leftarrow PC + 1$$

This could be explained as follows: At T0 = 1

The content of PC are placed onto the bus by making the bus selection inputs S2S1S0=010 and these contents of the common bus are transferred to AR by enabling the LD input of AR. This is denoted by the following register transfer statement:

$$T_0 : AR \leftarrow PC$$

When T1 = 1

The READ signal line of the main memory is enabled to facilitate the memory read operation. This is followed by placing the content of the desired location of the main memory to the common bus by making S2S1S0= 111. The choice of the location of memory is specified by the AR. This is followed by transfer of the content of the common bus to the Instruction Register (IR) by enabling the LD input of IR. Now the instruction fetch is complete. At the end the Program Counter (PC) is incremented by enabling the INR input of PC. This can be denoted by the following register transfer statement:

$$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$$

7.8.2 INSTRUCTION DECODE

In this sub-phase, the instruction which is fetched from the main memory and now residing in the Instruction Register (IR) is decoded.

At T2=1

The opcode bits are decoded to enable one of the eight outputs i.e. D0 –D7. Simultaneously, the address of the operand, i.e. bit (0-11) of the instruction register are transferred to Address Register. To check whether it is a direct addressing mode or an indirect addressing mode, value of bit(15) of Instruction Resister(IR) is transferred to *I*. This can be denoted by the following register transfer statement:

$$T_2 : D_0,....,D_7 \leftarrow Decode\ IR(12-14),\ \ AR \leftarrow IR(0-11),\ \ I \leftarrow IR(15)$$

7.8.3 INSTRUCTION EXECUTION

In this sub-phase, the value of *I* is tested to decide between register, input/output, direct memory or indirect memory mode.

At T3=1

To make this decision, the value of D7 is tested. The bit D7 is set when bit(12-14) are all 1's. If D7 is 1, then it means that it is either a register mode or input/output mode. To resolve this, the status of I is checked. If I=0, then it is register mode i.e. when D7=1 ; Register(I=0) then register mode. In this case no memory reference is required. Hence the instruction is executed. This can be denoted by the following register transfer statement:

D7I'T3(Execute)

If I=1, then it is input/output mode i.e. when D7=1 ; Register(I=1) then input/output mode. i.e., when D7=1 and Register (I=1) then input/output mode. In this case also, no memory reference is required. Hence, the instruction is executed. This can be denoted by the following register transfer statement:

D7IT3 (Execute) At T4=1,

In case, D7 is 0, it means that it is a memory reference instruction. Now to select between direct and indirect memory instruction, again the status of I is checked.

If I=0, then it is direct memory instruction. In this case, the operand is fetch from memory and transferred to Data Register for processing.

$$DR \leftarrow M[AR]$$

After this, the instruction is executed. Only one access to the main memory is required in case of direct memory mode.

If I=1, then this is an indirect memory instruction. In the first step, the address is fetched i.e.

T_5=1

$$AR \leftarrow M[AR]$$

$T_6=1$

$$DR \leftarrow M[AR]$$

After this, the instruction is executed. Two accesses to the main memory is required in case of indirect memory mode. The above process could be summarized with the help of a flowchart as shown in Figure.

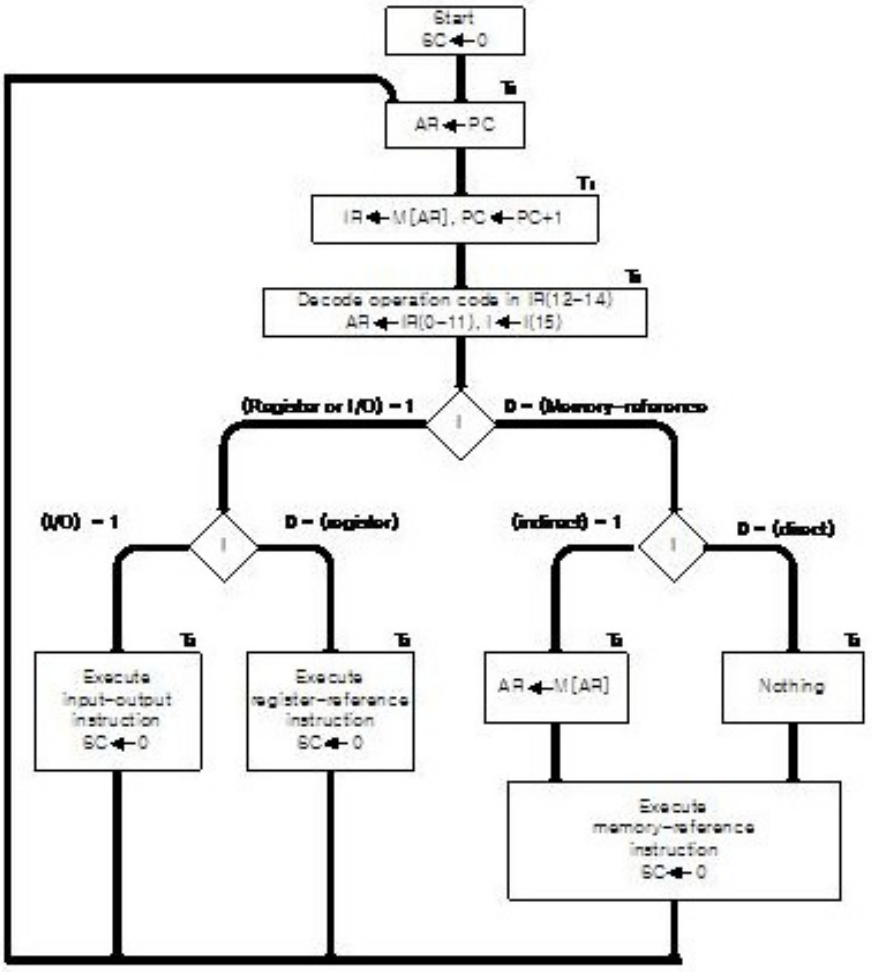


Figure 29: Instruction Cycle

Check Your Progress

1. A residing in the memory unit of the computer consists of a sequence of instructions.
2. The increments one by one to point to the next instruction to be executed,

7.9 REGISTER REFERENCE INSTRUCTIONS

As discussed in the above section, we can recognize register reference instructions when: D7=1 and the mode selection bit I=0

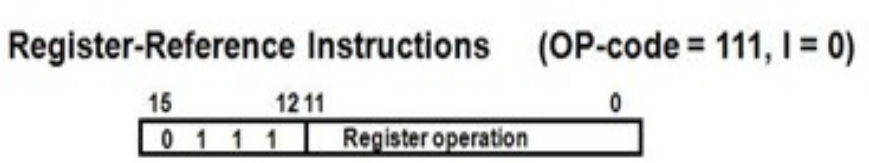


Figure 19 : Register Reference Instruction

In register reference instructions, memory reference is not required since the operand is present in the register. Hence, the 12 bits of the address part of the instruction code can be used to specify one of the various register operations. Various register reference instructions are shown in the Table 3 below.

r = D7 I′T3 => Register Reference Instruction

For convenience, D7 I′T3 is represented as r. It represent that register reference instruction occurs when D7 is 1, I is 0 during clock transition T3. B denotes i^{th} bit of address part of the instruction code. For instance,

B0 represents 0000 0000 0001

B1 represents 0000 0000 0010

.

.

.

B11 represents 1000 0000 0000

The first element in the table 9 is rB11, which can be represented in binary as rB11

=>D7 I′T3B11 => T3: D7 I′B11 => 0111 1000 0000 0000 => CLA

CLA is clear accumulator, which clears all the bits of the accumulator to 0. Likewise there is different register reference instructions operation explained in Table 3 below.

Table 3 : Register Reference Instructions

Symbol	Operational Decoder	Symbolic Instruction
	r:	SC←0
CLA	rB11:	AC←0
CLE	rB10:	E←0
CMA	rB9:	AC←AC’
CME	rB8:	E←E
CIR	rB7:	AC←shr AC, AC(15) ←E, E←AC(0)
CIL	rB6:	AC←shr AC, AC(0) ←E, E←AC(15)
INC	rB5:	AC←AC+1
SPA	rB4:	If(AC(15)=0) then (PC←PC+1)
SNA	rB3:	If(AC(15)=1) then (PC←PC+1)
SZA	rB2:	If(AC = 0) then (PC←PC+1)
SZE	rB1:	If(E = 0) then (PC←PC+1)
HLT	rB0:	S←0(S is a start-stop flip-flop)

7.10 MEMORY REFERENCE INSTRUCTIONS

Memory Reference Instruction can be recognized by either of D0 to D6=1. If I=0 then direct memory reference and if I=1 then indirect memory instruction. Various memory reference instructions can be selected by varying the values of opcode bits, which ultimately set one of seven outputs from D0 to D6 equal to1. This could be summarized by Table 4 below:

Table 4 : Memory Reference Instructions

Symbol	Operational Decoder	Symbolic Instruction
AND	D0	AC←AC^M[AR]
ADD	D1	AC←AC+M[AR]. E←Cout
LDA	D2	AC←M[AR]
STA	D3	M[AR] ←AC
BUN	D4	PC←AR
BSA	D5	M[AR] ←PC, PC←AR+1
ISZ	D6	M[AR] ←M[AR]+1, if M[AR]+1=0 then PC←PC+1

The effective address of the instruction is in AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1. The execution of memory reference instruction starts with T4.

Now let us discuss the various memory reference instructions one by one.

(i) AND to AC

D0T4: DR ← M[AR] Read operand D0T5:
AC ← AC ← DR, SC ← 0 AND with AC

During T4, when D0 is set, the content of main memory, located at the address specified by Address Register (AR) are transferred to Data Register(DR). Now at T5, the content of DR and ANDED with content of AC and the final result is stored in AC. Once the operation is complete, SC is cleared so as to generate T0 again, which will fetch next instruction to CPU for processing.

(ii) ADD to AC

D1T4: DR ← M[AR] Read operand

D1T5: AC ← AC + DR, E ← Cout, SC ← 0 Add to AC and store carry in E

During T4, when D1 is set, the content of main memory, located at the address specified by Address Register (AR) are transferred to Data

Register(DR). Now at T5, the content of DR and added with content of AC and the final result is stored in AC. If a carry is generated, E is set. Once the operation is complete, SC is cleared so as to generate T0 again, which will fetch next instruction to CPU for processing.

(iii) LDA: Load to AC D2T4: DR ← M[AR] D2T5: AC ← DR, SC ← 0

In the previous operations, the operand is added to the content of the accumulator. One may think how the accumulator is loaded with data. The above instruction LDA is used to load the accumulator. During T4, when D2 is set, the content of main memory, located at the address specified by Address Register(AR) are transferred to Data Register(DR). Now at T5, the content of DR are transferred to Accumulator(AC) and the SC is cleared.

(iv) STA: Store AC

D3T4: M[AR] ←AC, SC ← 0

When a new operand is to be loaded in the accumulator and AC already contains some data, which can be further required. This data is stored in the memory for future use. During T4, when D3 is set, the content of AC are transferred to main memory at the address specified by Address Register(AR Once the operation is complete, SC is cleared so as to generate T0 again, which will fetch next instruction to CPU for processing.

(v) BUN: Branch Unconditionally D4T4: PC ← AR, SC ← 0

During execution of an instruction, if a branching instruction is encountered, the program sequence branch to the address specified by the address register. Since it is an unconditional branching, therefore no condition is to be checked before branching. During T4, when D4 is set, the content of AR are transferred to PC and SC is cleared so as to generate T0 again, which will fetch next instruction to CPU for processing from the address that is specified by PC i.e. the branching address that was loaded to PC from AR.

(vi) BSA: Branch and Save Return Address

There are often situations in programming when a subroutine is called. The process of calling a subroutine is as follows: As soon as subroutine is called, the address of the starting location of the subroutine is written into the PC. But after the subroutine is called and executed, the control needs to be transferred to the location where the subroutine was called. Therefore the address of the location where a subroutine was called need to be remembered. To facilitate this, the first location of the subroutine is always kept empty so that the returning address can be saved at that location. This can be explained with the help of a register transfer statement:

D5T4: M[AR] ← PC, PC ← AR + 1

At T5, when D5 is active, the content of the Program Counter(PC), which holds the returning address, are saved in the memory location specified by AR(it is the starting address of the subroutine which is always kept empty). Simultaneously, The PC is updated with the starting address of the subroutine(remember, the starting address of the subroutine is specified by AR. Since, the first location is empty and used for storing the starting address, therefore content of AR are incremented before updating the PC).

This can be explained with the help of an example. The CPU is currently processing the instruction location at 20. Therefore, the current value of PC will be 21. While decoding the instruction, it was found that it was a Branch and Save instruction. The branching address is given by the address part of the Instruction Code i.e. 135. The AR currently stores this address. The starting location of the subroutine at 135 is empty. This is used to store the returning address, i.e. 21, which is stored in PC. The current content of the PC are written into the first location, i.e. 135 of the subroutine. It is followed by updating the PC with incremented content of AR i.e. 135+1=136. This could be explained as follows:

D5T4: M[135] ← 21, PC ← 135 + 1

Now the instruction located at location 136 i.e. subroutine starting address is loaded into the memory. The subroutine is executed. The last line of the subroutine contains an instruction which unconditionally branch to the starting location of the subroutine, i.e. 135. Please note that he mode selection bit I is 1. Therefore, it is an indirect addressing mode. Hence, the control returns to location number 21.

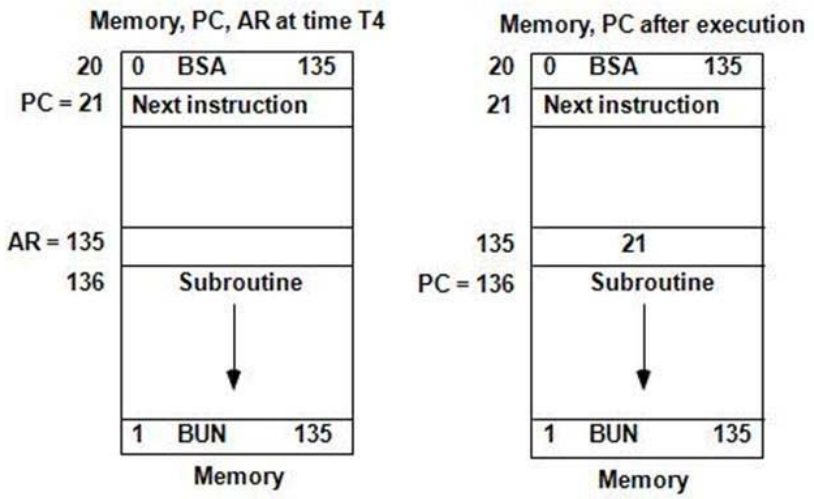


Figure 20 : Branch and Save Address

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and

the bus properly, the BSA instruction must be executed with a sequence of two microoperations. This is expressed in RTL as follows:

$$D5T4: \; M[AR] \leftarrow PC, \; AR \leftarrow AR + 1 \quad D5T5: \; PC \leftarrow AR, SC \leftarrow 0$$

Or

$$D_5T_4 : M[135] \leftarrow 21(PC), 136(AR) \leftarrow 135 \leftarrow 1$$

$$D_5T_5 : 136(PC) \leftarrow 136(AR), SC \leftarrow 0$$

Timing signal T4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR input of AR. The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T5 to transfer the content of AR to PC.

(vii) ISZ: Increment and Skip-if-Zero

There are some situations when before branching some condition is to be checked. If the condition is true then proceed with branching else fetch the next instruction from the sequence. Once such instructions is ISZ, increment and skip if zero. At T4, when D6 is set, the data is fetched from the memory from the location specified by the AR. The data from memory is transferred to DR. At T5, the content of DR is incremented. Now at T6, if after incrementing the content of DR, the incremented content of DR becomes zero, then skip the next instruction.

$$D6T4: \; DR \leftarrow M[AR] \quad D6T5: \; DR \leftarrow DR + 1$$

$$D6T6: \; M[AR] \leftarrow DR, \; \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), \; SC \leftarrow 0$$

We can skip one instruction by incrementing the PC twice. Once the PC is incremented during T1, now if we again increment it at T6, it is equivalent to skipping one instruction. Also, the content of DR are saved to main memory and the SC is cleared.

The operation of Memory Reference Instruction could be summarized with the help of a flowchart given at

Figure 21.

Can you think of a computer which cannot communicate with its environment? Most of us will have the same answer, NO. A computer does not have any practical application if it cannot intract with its environment via input and output devices. The input devices like keyboard, scanner, etc. are used to feed input/data to the CPU or memory and the output devices like monitor, printer are used to display the processed information. The basic input-output configuration of a computer is explaind with the help of Figure 22.

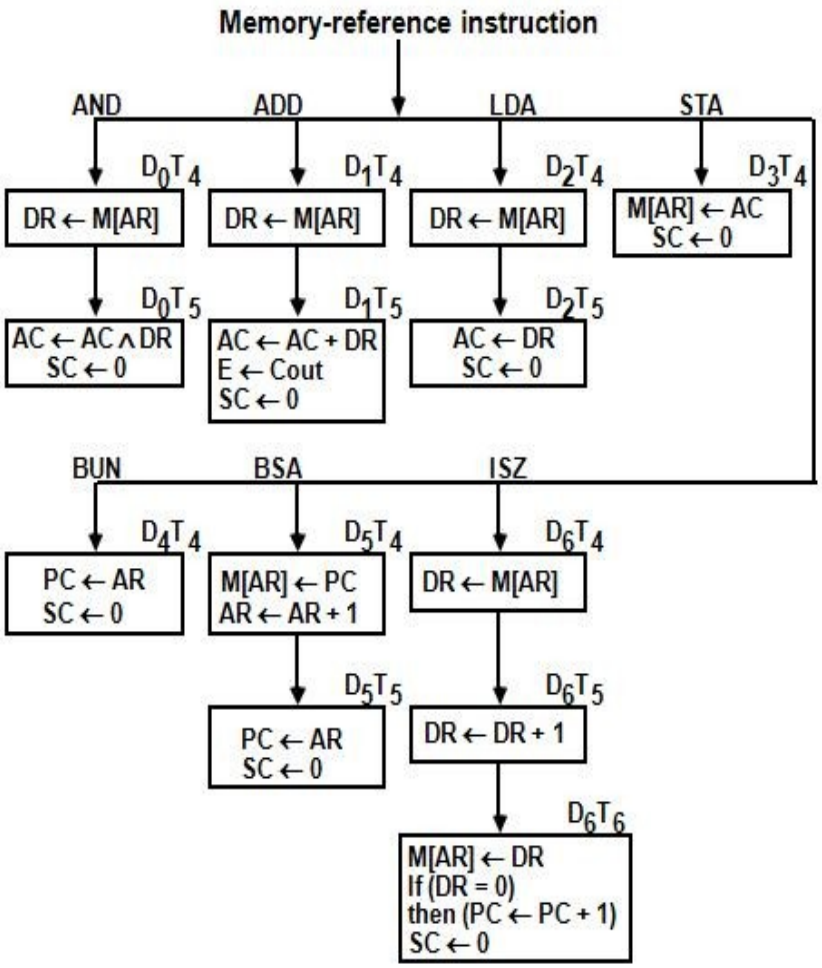


Figure 21: Flowchart of Memory Reference Instruction

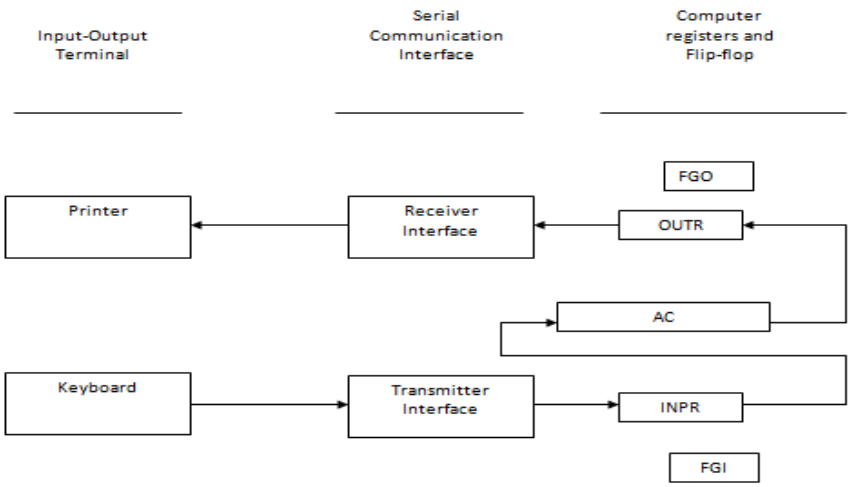


Figure 22: Input-Output Configuration

The input/output devices receives and transmits data at much slower rate as compared to CPU. To compensate this speed mismatch, an input and output interfaces are required, which receives and transmit the data at different rates. Input devices are attached to input interface, which is connected to INPR register of CPU. INPR is able to receive the data from input device serially and when the whole of the data item is received, it can transmit it to the CPU parallely. Whenever we press an key in the keyboard, it transmit the 8-bit alphanumeric code to INPR serially. INPR is attached to a 1-bit status flag FGI. This FGI bit represents the status of the INPR, whether it is ready to accept data or not. We can imagine a situation when the INPR already holds previous data, which is not yet transferred to CPU. If another key in the keyboard is stiked, the alphanumeric bits of the new data can currupt the old data. To overcome this situation, FGI status flag is used. FGI is set when INPR already holds the data. In case, a new key is pressed, its alphanumeric input are not transmitted to INPR until and unless flag FGI is 0, which is a signal that the INPR is empty and ready to store new data. Whenever the INPR transmits the data to AC, it clears FGI.

Similarly, output devices are attached to output interface, which is connected to OUTR of CPU. OUTR have the capability to receive the data from CPU parallely and transmit it to the output devices serially. Status flag FGO in set state is used to represent the OUTR is ready to receive data from AC. Once the data from data is transmitted from AC to OUTR, FGO is cleared to 0. Now once the data from OUTR is transferred to output device, again FGO is set.

7.11 INPUT-OUTPUT REFERENCE INSTRUCTIONS

As discussed in the previous unit, we can recognize register reference instructions when: D7=1 and the mode selection bit I=1.

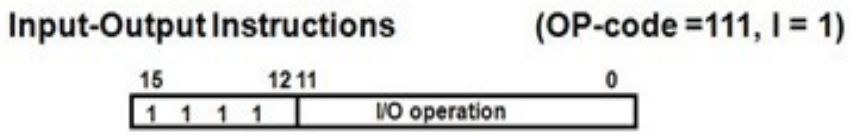


Figure 23: Input-Output Reference Instruction

In input-output reference instructions, memory reference is not required since the operand is present in the register. Hence, the 12 bits of the address part of the instruction code can be used to specify one of the various input-output operations. Various input-output reference instructions are shown in the table 11 below.

p = D7 IT3 => Input-Output Reference Instruction

For convenience, D7 IT3 is represented as *p*. It represent that input-output reference instruction occurs when D7 is 1, I is 1 during clock transition T3. B donotes i^{th} bit of address part of the instruction code. For instance,

B6 represents 0000 0010 0000

B7 represents 0000 0100 0000

.

.

.

B11 represents 1000 0000 0000

The first element in the table 11 is *p*B11, which can be represented in binary as

*p*B11 =>D7 IT3B11 => T3: D7 IB11 => 1111 1000 0000 0000 => INP

INP transfers the content of INPR to AC. After transferring the content, FGI is cleared. Likewise there is different register reference instructions operation explained in Table 5 below.

Table 5 : Input-Output Instructions

D7 I T3 = p(common to all input-output instructions) IR(i)=Bi(Bit in IR(6-11) that specifies the instruction)			
	p:	**SC←0**	**Clear SC**
INP	**pB11:**	**AC(0-7) ←INPR, FGI←0**	**Input Character**
OUT	**pB10:**	**OUTR←AC(0-7), FGO←0**	**Output Character**
SKI	**pB9:**	**If(FGI=1) then (PC←PC+1)**	**Skip on Input flag**
SKO	**pB8:**	**If(FGO=1) then (PC←PC+1)**	**Skip on Output flag**
ION	**pB7:**	**IEN←1**	**Interrupt enable on**
IOF	**pB6:**	**IEN←0**	**Interrupt enable off**

Check Your Progress

1. In reference instructions, memory reference is not required since the operand is present in the register.
2. The input/output devices receives and transmits data at rate as compared to CPU.
3. To compensate the speed mismatch, an input and output are required, which receives and transmit the data at different rates.
4. In …….. reference instructions, memory reference is not required since the operand is present in the register.

7.12 2 SUMMARY

Instructions are processed under the direction of the control unit in a step-by-step manner.

There are four fundamental steps in the instruction cycle:

1. **Fetch the instruction**

   - The next instruction is fetched from the memory address that is currently stored in the Program Counter (PC), and stored in the Instruction register (IR). At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.

2. **Decode the instruction**

   - The decoder interprets the instruction. During this cycle the instruction inside the IR (instruction register) gets decoded.

3. **Execute**

   - The Control Unit of CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU.

4. **Store result**

   - The result generated by the operation is stored in the main memory, or sent to an output device. Based on the condition of any feedback from the ALU, Program Counter may be updated to a different address from which the next instruction will be fetched.

7.13 ANSWERS TO CHECK YOUR PROGRESS

1. Program
2. Program Counter(PC)
3. Register
4. Slower
5. Interfaces
6. input-output

7.14 TERMINAL QUESTIONS

1. What is the difference between a direct and an indirect address instruction? How many references to memory are required for each type of instruction to bring an operand into a processor register?
2. What is instruction cycle? What are the sub-phases of an instruction cycle.
3. Explain ISZ memory reference instruction in detail.
4. Explain BSA memory reference instruction in detail.
5. What is an interface? Why is it required?

UNIT-8 ADDRESSING TECHNIQUES

Structure

8.1 LEARNING OBJECTIVES

After reading this unit, you will be able to:

- Explain various addressing modes.
- Define immediate addressing mode
- Differentiate between direct and indirect addressing mode.
- Explain Register mode.

8.2 INTRODUCTION

The operation field of an instruction specifies the operation to be performed.

This operation must be executed on some data stored in computer registers or memory words. Computers use addressing mode techniques for the purpose of accommodating one or both of thefollowing provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

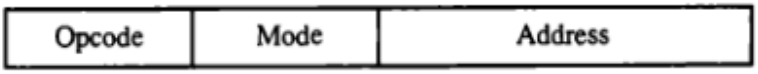


Figure 24: Instruction Code with Mode Field

The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Execute the instruction.

There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. The **mode field** is used to locate the operands needed for the operation.

8.3 ADDRESSING MODES

There are various addressing modes which are discussed in length in the following section.

8.2.1 IMPLIED MODE

In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions.

8.2.2 IMMEDIATE MODE

In this mode the operand is specified in the instruction itself i.e. the address part of the instruction code contains the operand itself.

Instruction Code

Opcode	Operand

Figure 25: Immediate mode

8.2.3 REGISTER MODE

In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

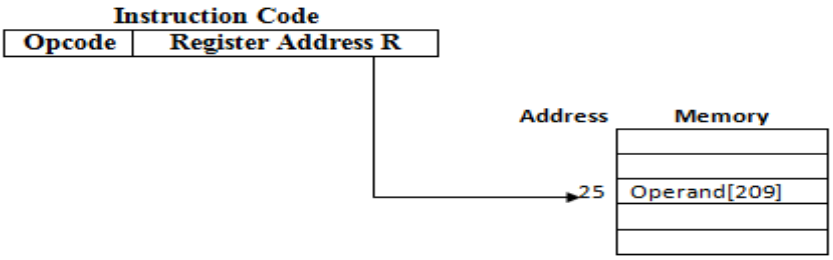


Figure 26: Register Mode

8.2.4 REGISTER INDIRECT MODE

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.

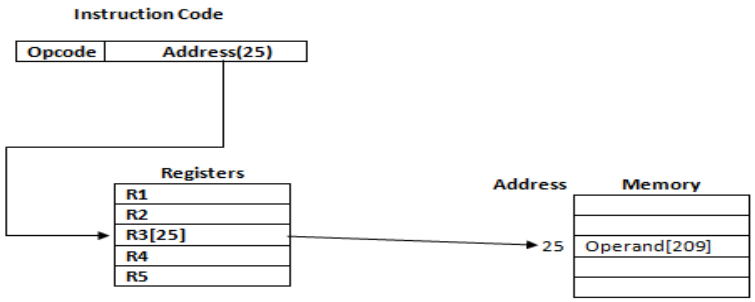


Figure 27: Register Indirect mode

8.2.5 AUTO-INCREMENT OR AUTO-DECREMENT MODE

This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. The effective address is defined to be the memory address obtained from the

computation dictated by the given addressing mode.

8.2.6 DIRECT ADDRESS MODE

In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.

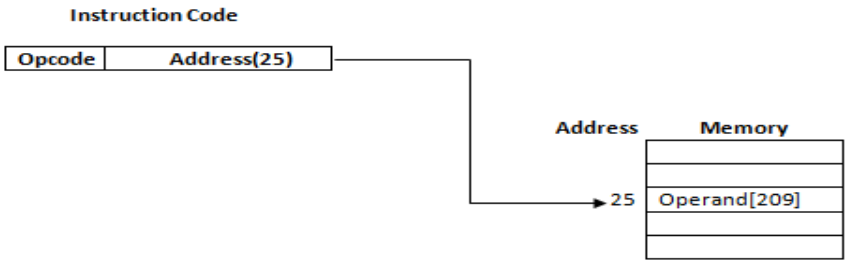


Figure 39: Direct addressing mode

8.2.7 INDIRECT ADDRESS MODE

In this mode the address field of the instruction gives the address where the effective address is stored in memory.A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following computation: effective address = address part of instruction + content of CPU register

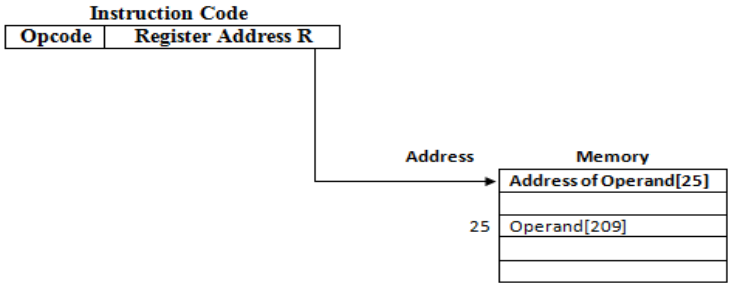


Figure 28: Indirect Address Mode

8.2.8 RELATIVE ADDRESS MODE

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

8.2.9 INDEXED ADDRESSING MODE

In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory.

8.2.10 BASE REGISTER ADDRESSING MODE

In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.

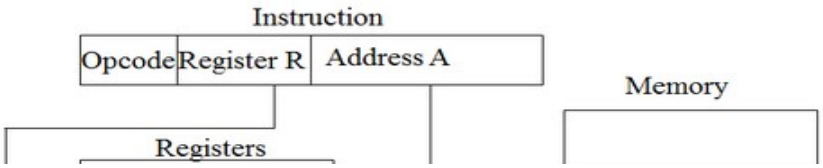


Figure 29: Displacement addressing diagram[2]

Check Your Progress

1. An ……. mode instruction has an operand field rather than the address field.
2. In ……. mode, the instruction has the address of the Register where the operand is stored.
3. In …….. mode, the register contains the address of operand rather than the operand itself.
4. ……. mode is a version of Displacement addressing mode.
5. …… mode is most suitable to change the normal sequence of execution of instructions.

8.3 8085 REGISTER

Microprocessor is a programmable device which reads binary instructions from memory, accepts binary data as input and processes the data as per the instruction to produce output as a result. The 8085 is an 8 bit microprocessor which is used as a basis for studying all microprocessors in the market. The registers in 8085 microprocessor are classified into following categories:

1. **General purpose registers:** The general purpose registers are used to store temporary data during execution of a program. The 8085

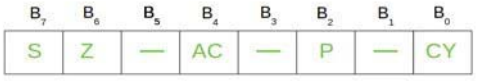
microprocessor has 6 general purpose registers to store 8 bit data. These registers are called as B, C, D, E, H, and L. These registers can be used in an instruction if one of the operands presents in A register (i.e.accumulator register). For example, there is no any instruction to add the content of registers B and C. To add the content of B and C registers and store the result in B, we need to perform following operations-

- Move the contents of B register to A.
- Add the content of A and C registers and store the result back in A.
- Move the result from register A to register B.

These registers can also be combined as a pair to store 16 bit information. But, only BC, DE and HL can be used as register pairs. When these registers are used as a pair, the left register stores the most significant byte and the right register stores the least significant byte. For example, in the HL pair resistors, the content of the H register is most significant byte and the content of L register is the least significant byte.

2. Special purpose registers:

- Accumulator (Register A): The accumulator is an 8 bit register which is present in ALU and directly communicates with ALU. It is used to store input and output of ALU after performing arithmetic or logical operations.
- Status or Flag Register: The flag register is a 8 bit special purpose register which is completely different from other registers in the microprocessor. It consists of 8 bits out of which only 5 bits are useful, while other 3 bits are left unused.



fig(a)-Bit position of various flags in flag registers of 8085

Figure 3.3 : Bit position of different flags in the flag register.

These 5 bits are called flags which can be set or reset (when flag value is 1, it is called as set and when flag value is 0, it is called as reset) based on the results in accumulator and other registers. These 5 flags are as follows:

i. Sign Flag: It is present at 7th position and most significant position (MSB) in the flag register. It helps us to know whether the value stored in the accumulator register is positive or negative. This flag bit is set to 1 if the number stored in the accumulator is negative. Otherwise, it is set to 0 if the number stored in the accumulator register is positive.

ii. Zero Flag: This flag presents at 6th position in the flag register. This flag bit is set to 1 if the result of ALU operation which is stored in the accumulator register is zero. Otherwise, it is set to 0. This flag bit can be useful in checking whether two numbers are equal or not. Parity flag: This flag bit is present at 2nd position in the flag register. This flag bit tells us whether the number of 1 present in the accumulator register is even or odd i.e. even or odd parity. If the number of 1 present in the accumulator is even, this flag bit is set to 1. Otherwise, if the number of 1 present in the accumulator is odd then this flag bit is set to 0.

iii. Carry flag: This flag bit presents at 0th position. When the result of ALU computation which is stored in the accumulator gives a carry i.e. more than 8 bits, then this flag bit is set to 1. Otherwise, it is set to 0.

iv. Auxiliary carry Flag: This flag presents at 4th position in the flag resister. When ALU performs a arithmetic operation and this results in a carry flag is generated and passed on to 4th bit, then the auxiliary carry Flag bit is set to 1. Otherwise, this flag bit is set to 0. This flag bit is used internally for BCD operations.

3. Memory Registers :

The 8085 microprocessor has two memory registers each of 16 bits. These registers are used to store 16 bits memory addresses.

Program counter: This register holds the address of the next instruction to be executed. Once the instruction is fetched from the memory, this register is automatically incremented by one to hold the address of the memory location of the next instruction to be executed. This way this register is used to sequence the execution of instructions.

Stack pointer: It is a 16 bits register which holds memory address. This register holds the address of the top location of the stack and the register is incremented by 2 for every push and pop operation of the stack. In push operation of the stack a new address is inserted on the stack and the stack pointer decremented by 2. In pop operation, the address present at the top of the stack is deleted and the stack pointer is incremented by 2. This way the content of the stack pointer gives the top most useful location with the smallest memory address present on the stack.

8.4 SUMMARY

1. Addressing modes are an aspect of the instruction set architecture in most central processing unit (CPU) designs.

2. The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand(s) of each instruction.

3. An addressing mode specifies how to calculate the effective memory

address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.

4. In immediate mode, the operand is specified in the instruction itself.

5. In register mode, the operand is stored in the register and this register is present in CPU.

6. In register indirect mode, the instruction specifies the register whose contents give us the address of operand which is in memory.

7. In autoincrement/autodecrement mode, the register is incremented or decremented after or before its value is used.

8. In direct addressing mode, effective address of operand is present in instruction itself.

9. In indirect addressing mode, the address field of instruction gives the address where the effective address is stored in memory.

10. In relative addressing mode, the contents of PC (Program Counter) is added to address part of instruction to obtain the effective address.

11. There are several general purpose registers present in the processor referred to as processor registers. These registers are used by the processors to perform various operations on the data that is fetched from the memory to processor.

12. Data register holds the data in which the operation is to be performed.

13. The processor has a register, the *Program Counter* (PC) that holds the memory address of the next instruction to be executed.

8.5 ANSWERS TO CHECK YOUR PROGRESS

1. Immediate

2. Register

3. Register indirect

4. Relative addressing

5. Relative addressing

8.6 TERMINAL QUESTIONS

1. What are the three phases of instruction cycle?

2. What is a difference between register mode and auto-increment/auto-

decrement mode?

3. What is an effective address? How it is computer?
4. Compare index address mode with base register addressing mode.
5. Explain the various address modes using an example.

Uttar Pradesh Rajarshi Tandon Open University

Master of Computer Application

MCA-105/MCS-106 /PGDCA-105

Computer Organization

BLOCK

3

MEMORY AND I/O

UNIT-9

Memory

UNIT-10

I/O System

UNIT-11

Introduction to 8085 Microprocessor and Microcontrollers

Course Design Committee

Course Preparation Committee

Mr. Manoj Kumar Balwant **Author Block 1 (Unit 1,2,3,4,5)**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad.
Dr. JitendraPande **AuthorBlock 2, 3 (Unit 6,7,8,9,10,11)**
Associate Professor
School of Computer Sciences & Information Technology
Haldwani, Uttarakhand 263139
Prof. Ashutosh Gupta **Editor Block 1 (Unit 1, 2, 3, 4, 5)**
Director (In-Charge)
School of Computer & Information Sciences, UPRTOU Allahabad
Prof. Abhay Saxena **Editor Block 2, 3 (Unit 6, 7, 8, 9, 10, 11)**
Professor and Head, Department of Computer Science
Dev SanskritiVishwavidyalya, Hardwar, Uttrakhand
Mr. Manoj Kumar Balwant **Coordinator**
Assistant Professor (computer science),
School of Sciences, UPRTOU Allahabad.

ISBN :

Printed and Published by Dr. Arun Kumar Gupta Registrar, Uttar Pradesh Rajarshi Tandon Open University, 2020.
MCA-105/150
Printed By : Chandrakala Universal Pvt. 42/7 Jawahar Lal Neharu Road, Prayagraj.

BLOCK INTRODUCTION

This block describes you different types of memory and the various peripheral devices of a computer system. The last unit presents you 8085 microprocessor along with few examples of instructions to understand addressing techniques.

Unit-9 concentrates on memory. This unit discusses main memory, cache memory and virtual memory. It also describes various mapping techniques in detail.

Unit-10 discusses different types of peripheral devices, Asynchronous Data Transfer and I/O cards of a personal computer.

Unit-11 is the last unit of this course. This unit focuses on architecture of 8085 Microprocessor. Instruction set of 8085 Microprocessor is covered in this unit.

Each unit includes SUMMARY of the unit at the end of the Unit. You will get "CHECK YOUR PROGRESS" questions. These have been designed to make you self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with "ANSWERS TO CHECK YOUR PROGRESS" given at the end of each unit.

UNIT-9 MEMORY

Structure

9.1 LEARNING OBJECTIVES

After reading this unit, you will be able to:

- Explain memory hierarchy.
- Define Main Memory.
- Differentiate between RAM and ROM.

- Define Cache memory.
- Different mapping techniques in cache memory.
- Define Virtual memory.
- Different mapping techniques in virtual memory.

9.2 INTRODUCTION

A computer system have various type of memories like CPU registers, cache memory, main memory, secondary memory, etc. All the memories have same function i.e. to store the data, but they differ in their speed and cost. These memories are arranges in hierarchies. The access speed of the expensive memory is fast. By using a hierarchy of memories, each with different access speeds and storage capacities, a computer system can exhibit performance above what would be possible without a combination of the various types. The base types that normally constitute the hierarchical memory system include registers, cache, main memory, and secondary memory.

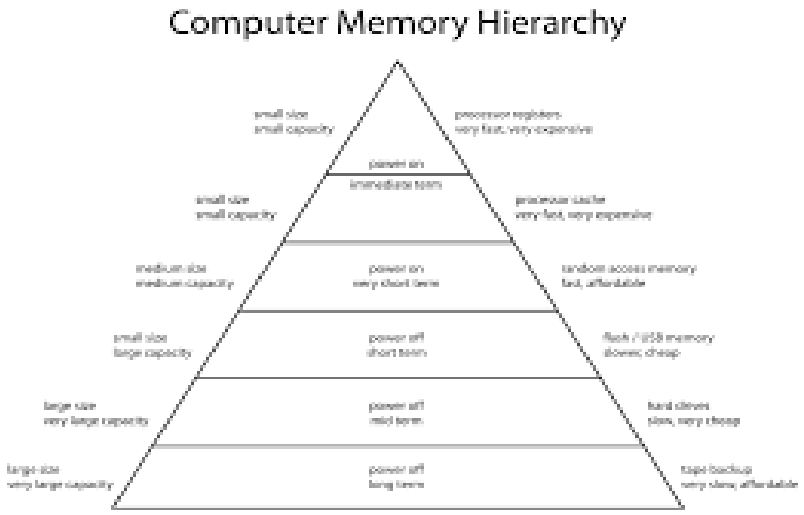


Figure 30 : Memory Hierarchy

In the top of this hierarchy, there are CPU registers. Their access speed is comparable to the speed of CPU. They are used by CPU for storing temporary data during calculations. At second level, a very high-speed memory, called a *cache,* is attached to the computer. It is used to store data temporarily that is very frequently used from memory locations may. It is connected to *main memory,* which is typically a medium-speed memory. This memory is complemented by a very large secondary memory, composed of a hard disk and various removable media. By using such a hierarchical scheme, one can improve the effective access speed of the memory, using only a small number of fast (and expensive) chips. This allows designers to create a computer with acceptable performance at a reasonable cost.

9.3 MAIN MEMORY

The main memory constitutes of RAM and ROM, is the central storage unit in a computer system. It is a temporary storage medium and the data is lost in case the power is lost(except in the case of ROM). RAM and ROM are available in a variety in size. If the memory needed for the computer is large than the capacity of one chip it is necessary to combine a number of chips to obtain the required memory size. It is called as main memory because it can be accessed directly by the CPU. Now let us discuss the different variants of main memory.

9.3.1 RAM (RANDOM ACCESS MEMORY)

It is read write memory. It is just like a page of a notebook, where you can write something to or read something from. All the programs are brought into RAM just before execution. The block diagram of a RAM can be represented as:

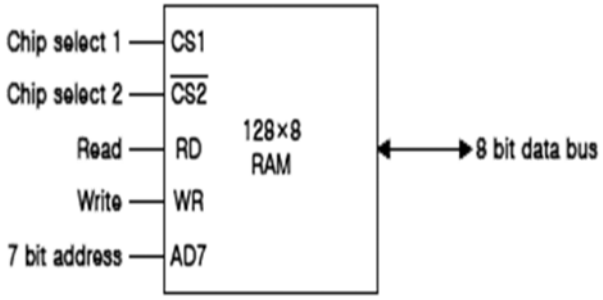


Figure 31 : Block diagram of a RAM

The functional table of the RAM is shown below:

Table 6 : Functional Table of RAM

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedance
0	1	x	x	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedance

There are many variants of RAM. Some of them are:

- *DRAM (Dynamic RAM):* This is the most common type of computer memory. It is called dynamic because it must be refreshed, or re-energized, hundreds of times each second in order to retain the data in its words. Each bit in a word in DRAM is designed around a tiny capacitor that can store an electrical charge. A charged capacitor indicates a 1-bit. However, the capacitor loses its charge rapidly, which is why DRAM must be refreshed.

- *SRAM (Static RAM):* This type of memory is about five times faster, twice as expensive, and twice as big as a DRAM. SRAM does not need to be refreshed like a DRAM. Each bit in SRAM is a flip-flop; a circuit that has two stable states. Once placed in a certain state, it stays in that state. Flip-flops are faster to read and write than the capacitors of the DRAM, but they consume more power.

9.3.2 ROM (READ ONLY MEMORY)

It is Nonvolatile memory i.e. the content remains in the memory even if the power is switched off. It stores mainly the monitor programs and BIOS (Basic Input Output System) Programs.

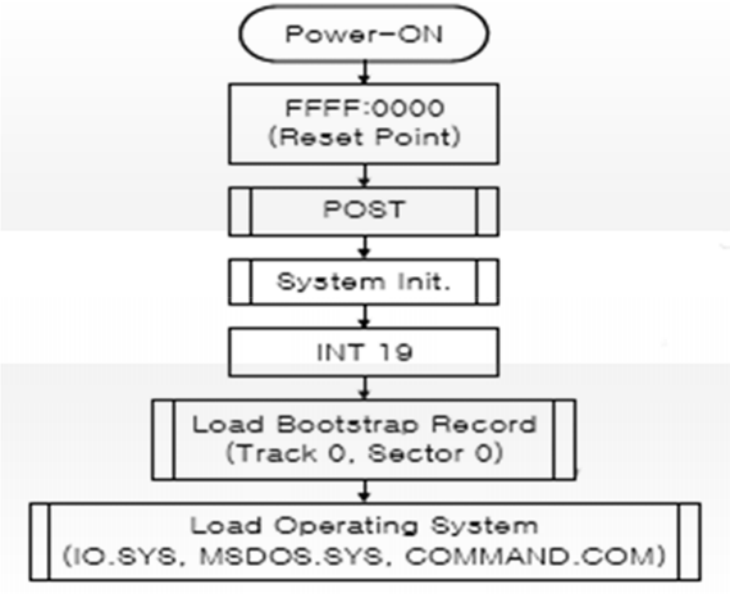


Figure 32 : Bootstrap loader flow chart

The information stored in ROM can only be read but cannot be modified. The contents of ROM can be programmed under special conditions. It is manufacturer programmed memory. The block diagram of a ROM can be represented as:

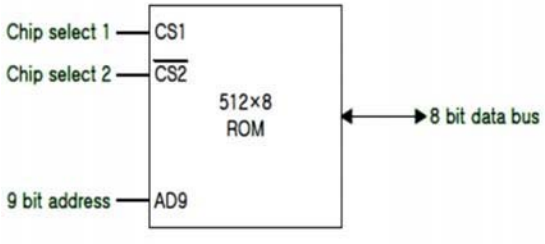


Figure 33: Block Diagram of a ROM

There are many variants of ROM available in the market. Some of them are:

- *PROM - Programmable Read Only Memory*: this is write-once, read-many type of memory, which could be programmed after fabrication. The data is

written to the memory electrically using some special circuitry.

- *EPROM – Erasable Programmable Read Only Memory:* this type of ROM has multiple read and writes facility. The old data present in the ROM can be erased by exposing the ROM chip to UV radiation, after which new data can be written into the ROM.
- *EEPROM – Electrically Erasable Programmable Read Only Memory:* This is the most expensive variant of the ROM and the per-bit storage cost of this variant is more as it is less dense than EPROM. It performs Byte-level writing, in which any part(s) of the memory can be written at any time.

9.3.3 DIFFERENCE BETWEEN RAM AND ROM

RAM	ROM
Read- Write Memory	**Read Only Memory**
Volatile memory i.e. the contents of the RAM are lost when power is turned off	Non-volatile memory i.e. the contents of the ROM are not lost when power is turned off
Temporary storage medium	Permanent storage medium
The data can be read and written	The data can only be read, but the data cannot be written
The programs are brought into RAM just before execution	BIOS and monitor programs are stored

9.3.4 MEMORY ADDRESS TABLE

While designing a computer system, the designer of the system must foresee in advance, the total memory required by the system and the type of memory that will be required for that application. For example, for permanent programs, ROM is an ideal choice. In practical, both the types of memory is used in designing a system. Based on the type of application, the designer of the system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a pictorial representation of assigned address space for each chip in the system.

Now let us demonstrate the above situation using an example. Suppose a computer uses RAM chips of 128x8 and ROM chips of 512 x 8. The memory address map with the chips needed for constructing system memory with 512x8 RAM and 512x8 ROM can be represented as in Table 7. We need 4 RAM and 1 ROM memory address map is:

Table 7 : Memory Address Table

Component	Address in hexadecimal	Address Bus									
		10	9	8	7	6	5	4	3	2	1
RAM I	0000-007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080-OOFF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100-017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180-01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200-03 FF	1	x	x	x	x	x	x	x	x	x

The above memory connection to the CPU can be designed as shown in **Figure 34**.

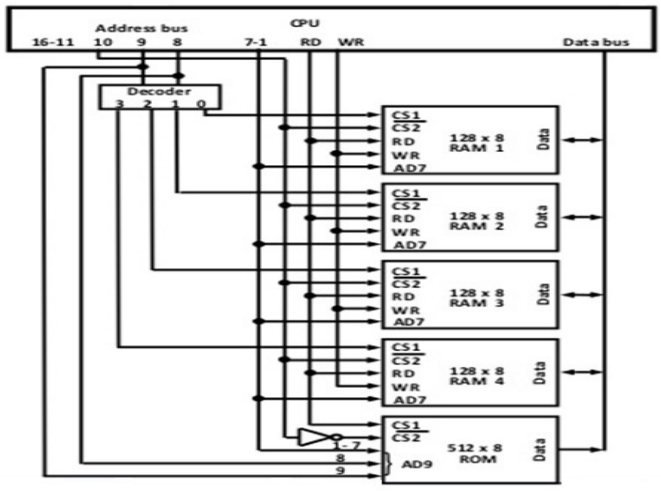


Figure 34 : Memory Connections to the CPU

9.3 CACHE MEMORY

The CPU processing speed is very fast as compared to the time taken by a memory-access. The speed of the CPU to perform an operation is limited by the memory-access time. If, by any mechanism, the memory access can be limited, the speed of the CPU to perform an operation can be increased considerably. Moreover, it has been found after analyzing a large number of programs that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference. If the content of these "local" areas of the memory are brought closer to CPU i.e. placed in a memory whose access time is comparable to CPU speed, the CPU performance can increase to many folds. For this purpose, a fast small memory is referred to as a cache memory is employed. The cache memory access time is less than the access time of main memory by a factor of 5

to 10. The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

As the cache is the amongst the topmost element of the memory hierarchy, its cost per bit storage is also very high. Due to economics, it is only a small fraction of the size of main memory. Therefore, the data from the main memory must be exchanged frequently to keep the frequently and most accessed data in the cache. The transformation of data from main memory to cache memory is referred to as a *mapping process*. Three types of mapping procedures are of practical interest when considering the organization of cache memory:

- Associative mapping.
- Direct mapping.
- Set-associative mapping.

9.3.1 ASSOCIATIVE MAPPING

This is one of the optimal cache organization which employees associative memory for searching, and thus making the mapping process very fast. The associated mapping can be explained using Figure 35. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory. The diagram shows three words presently stored in the cache. The address value of15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

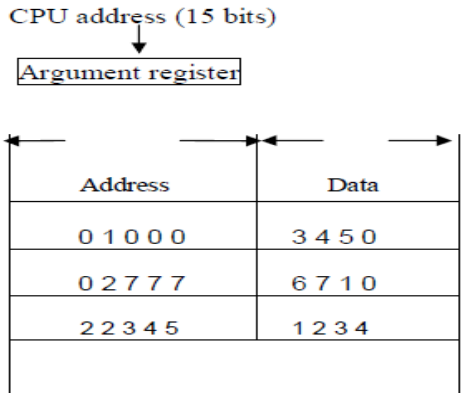


Figure 35 : Associative Mapping

9.3.2 DIRECT MAPPING

Direct mapping can be implemented using random access memory as shown below in **Figure**. Consider a main memory of size 32K X l2 and cache of size 512 X 12. 15 bits address is required to access each location uniquely. The 15- bit address part is divided into two part. The 9 least significant bits (2^9=512) are required to address each location of cache of size 512 X 12 uniquly. Remaining 15-9= 6 bits are used for tag field.

The **Figure** shows that main memory needs an address that includes both the tag and the index bits.

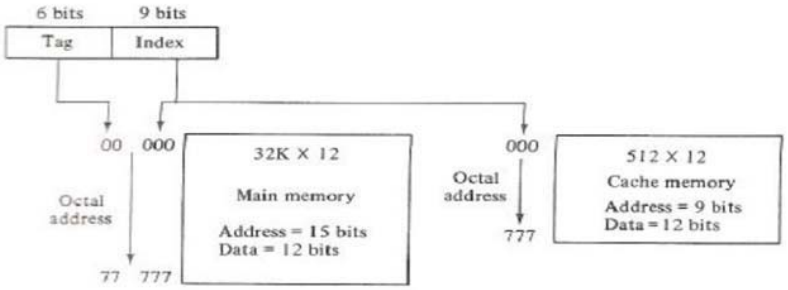
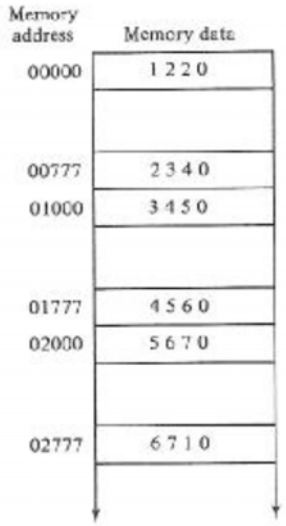


Figure 48 : Direct Mapping

The direct mapping cache organization uses the n-bits address to access the main memory and the k-bits index to access the cache. The internal organization of the words in the cache memory is as shown in **Figure**. Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in the cache. If there is no match, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value.

The direct mapping technique has one disadvantage. The hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly. However, this possibility is minimized by the fact that such words are relatively far apart in the address range (multiples of 512 locations in this example.).



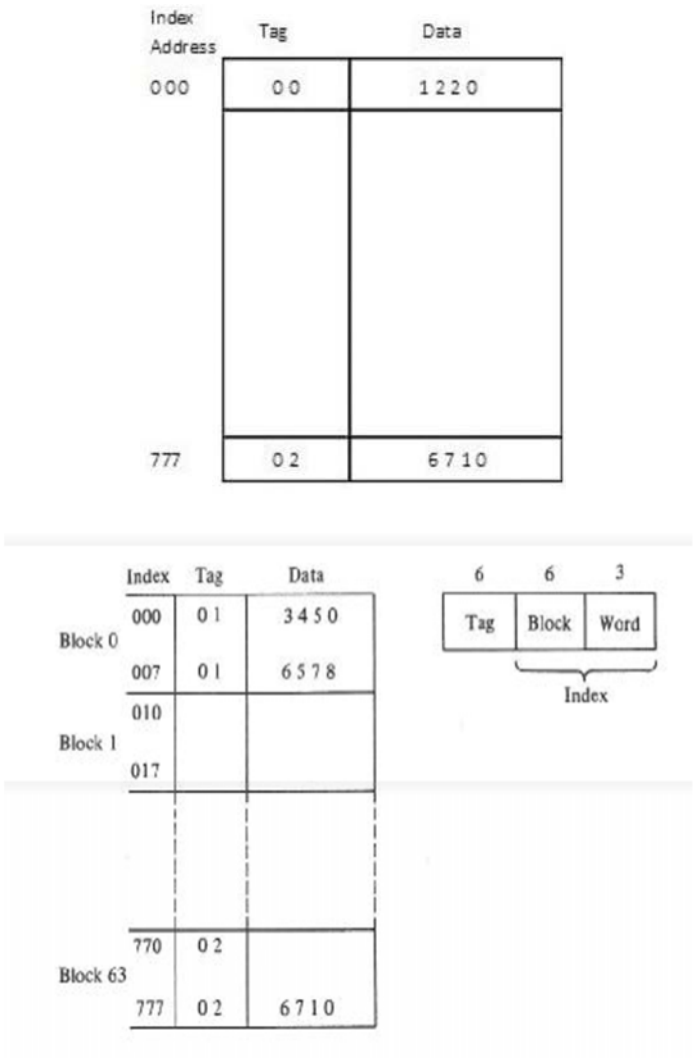


Figure 49 : Direct Mapping Example

9.3.3 SET-ASSOCIATIVE MAPPING

To overcome the limitation of direct mapping i.e. each word of cache cannot store two or more words of memory under the same index address, a new mapping technique named as set-associative mapping is introduced. In this mapping technique, each word of cache can store two or more words of memory under the same index address. This can be explained using Figure 36**.**

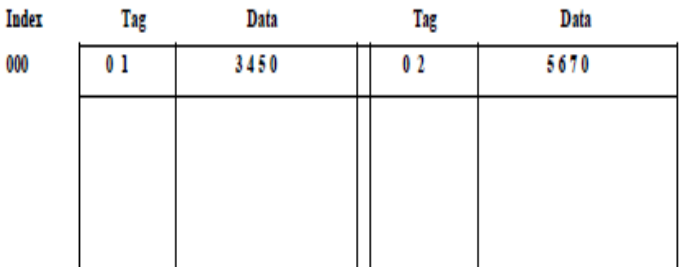


Figure 36 : Set-Associative Mapping

Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set. Each index address refers to two data words and their associated tags. For a main memory of size 32K X l2 and cache memory of size 512 X 12, each tag requires six bits and each data word has 12 bits, so the word length is 2(6 + 12) = 36 bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is512 X 36. It can accommodate 1024 words of main memory since each word of cache contains two data words. In general, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.

9.3.4 WRITING INTO CACHE

The very idea of using cache memory is to limit memory-access so that the operation can be performed at a much faster speed. If there is a data read request, this problem of frequent data access could be easily resolved as the data is residing in the cache. But if there is a data write request, we cannot deny memory access. The main memory is updated with every memory write operation; with cache memory being updated in parallel if it contains the word at the specified address. This is called the ***write-through*** method. This method has the advantage that main memory always contains the same data as the cache. This characteristic is important in systems with direct memory access transfers. It ensures that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

However, there are some techniques by which the frequency at which the memory is accessed can be controlled through a procedure called the ***write-back*** method. In this method, only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory. The reason for the write- back method is that during the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache.

9.3.5 PAGE REPLACEMENT ALGORITHM

When a miss occurs in a Cache memory and the Cache is full, it is necessary to replace one word with a new word from main memory. The most common replacement algorithms are:

- Random Replacement: Select the item randomly.
- FIFO (First-In First-Out): Select the item has been in the Cache the longest.
- LRU (Least Recently Used): Select the item that has been least recent used by the CPU.

Check Your Progress

1. The last on the hierarchy scale of memory devices is ……..…
2. The effectiveness of the cache memory is based on the property of ……
3. The correspondence between the main memory blocks and those in the cache is given by ……. .
4. The algorithm to remove and place new contents into the cache is called …….. .
5. The bit used to signify that the cache location is updated is ……..
6. In …… protocol the information is directly written into main memory.
7. The method of mapping the consecutive memory blocks to consecutive cache blocks is called …….. .
8. While using the direct mapping technique, in a 16 bit system the higher order 5 bits is used for …… .
9. The technique of searching for a block by going through all the tags is …….. search.
10. A control bit called …….. set-associative. bit has to be provided to each block in
11. The bit used to indicate whether the block was recently used or not is …… bit.

9.4 VIRTUAL MEMORY

Initially the program resides in the secondary storage device like hard disk. Whenever the program is executed, a copy of that program is brought into main memory. When a program is large enough to fit into the main memory in one go, the operating system need to move program and data constantly between main memory and secondary storage. This gives an illusion to the programmer that has a very large main memory at his disposal. This concept is that automatically swaps programmed data blocks between main memory and secondary storage device are called virtual memory.

This concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory. Each address that is referenced by the CPU goes through an address mapping from the so called virtual address to a physical address in main memory. A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations. This is done dynamically, while programs are being executed in the CPU. The translation or mapping is handled automatically by the hardware by means of a mapping table.

9.4.1 ADDRESS SPACE AND MEMORY SPACE

Let us start our discussion by defining address space and memory space.

- ***Virtual Address :*** An address used by a programmer is called virtual address.
- ***Address Space :*** Set of virtual address is known as address space. It is the set of addresses generated by programs as they reference instructions and data.
- ***Physical Address :*** Address of main memory is known as Physical Address.
- ***Physical Space :*** Set of main memory addresses is know as Physical Space. It is the memory space consists of the actual main memory locations directly addressable for processing.

The virtual address is used by programmers who have the illusion that he have a very large main memory, equal to the size of secondary memory, at his disposal. Therefore, the computers with virtual memory have address space greater than memory space. This can be further clerified using an example. If we have an auxiliary memory of size 1024K and main memory of size 32K.

Number of bits required to specify a virtual address uniquly will be 1024K= 1024 X 1024= 2^{10} X 2^{10}= 2^{20}

Therefore, 20 bits are required to specify any location in virtual space uniquly. Simillerly, number of bits required to specify a physical address will be 32K=32 X 1024 = 2^5 X 2^{10}= 2^{15}

Therefore, 15 bits are required to specify any location in physical space uniquly. But the instruction code is designed with address feld of 20 bits and the main memory address is of 15 bits only. Hence, a mapping logic is required to map these 20 bits generated by the programmer to 15 bits main memory address. A table, know as mapping table, is used to map the virtual addresses to physical address. This mapping need to be stored in the computer for translation to take place. There are three alternatives to store this table, with each alternative having its own merits and demerits.

(a) A separate memory is used to store the mapping table. This will require an additional memory as well as an additional access to this memory for address translation. This increases the cost as well as it decreses the performance as two accesses to memory is required.

(b) The mapping table is stored in the main memory. This does not increases the cost but, descreses the performance because this alternative also requires two accesses to the main memory.

(c) Associative memory is used for storing the mapping table. In this case, the

speed of the operation increases considerable as the associative memory facilitates the search operation. However, associate memory is one of the costliest memory therefore, the cost also increases.

The emorysy to store **Figure 37** below explains the address translation process in the virtual m stem when an additional memory as explained in (a) is employed apping table.

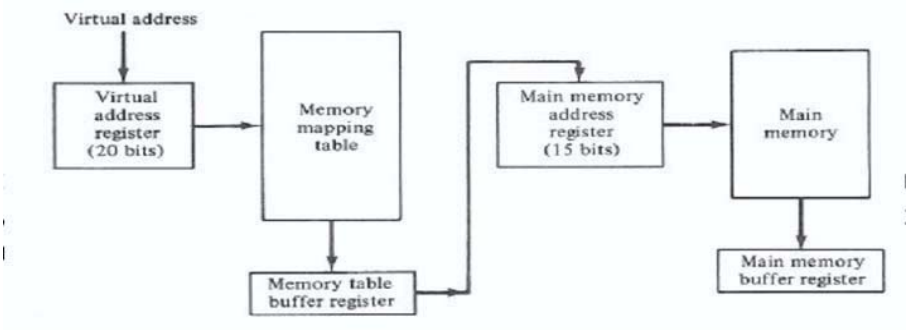


Figure 37: Virtual memory address translation

9.4.2 ADDRESS MAPPING USING PAGES

To facilitates the storing and reference the data, the memory is further divided into small groups. In the case of address space, these equal-sized group of memory is known as pages. And in the case of memory, these equal size group of memory is known as block. The address translation process is greatly facilitated if the size of the block and the page is same. For example: Consider a computer with an address space of 8K and a memory space of 4K. If we divide the address space and memory space into blocks and pages with size 1K i.e., the block size and page size is same(1K). This is shown in the F**igure 38** below:

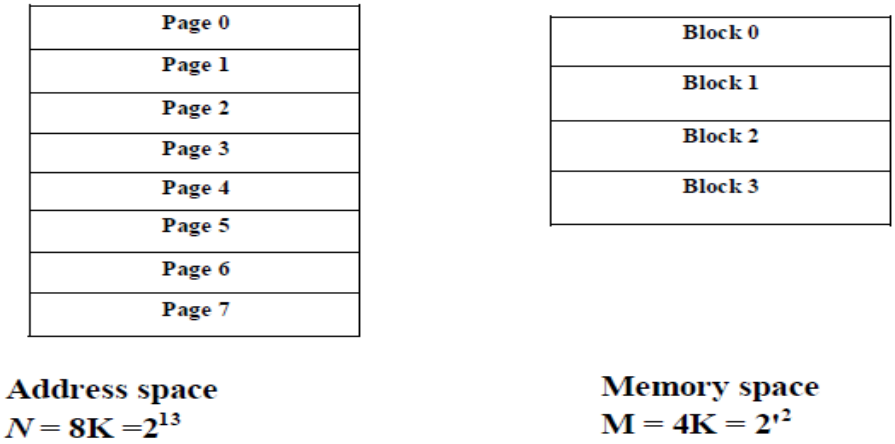


Figure 38 : Address and Memory Space

Since the address space is 8K, therefore we have 8 pages of 1K each. Similarly, memory space is of 4K therefore, we have 4 blocks of 1K each. Since the page and block size is the same, with this arrangement, at any point of time, the memory space can accommodate 4 out of 8 pages in any of its 4 blocks. Now we

will observe how the equal size of block and page facilitates the mapping process. The page and block size is 1K i.e. there are 1024 address locations ranging from 0 to 1023. Therefore, if we transfer any page from address space to memory space and we want to refer that page in the memory space, we need to map only the corresponding block number where the page is stored.

The 20-bit virtual address is divided into two parts. The first part specifies the page number and the second part denotes the word within the page (0 to1023). If a page is transferred to memory space in the block, we need to map the page with the associated block. The word need not to be mapped since the block and the page size are equal. 4^{th} word of the 2^{nd} page, if transferred to the 3^{rd} block in the memory space, it will be the same 4^{th} word in the 3^{rd} block. Mapping is required only for the 2^{nd} page to 4^{th} block i.e., the only mapping required is from a page number to a block number. This is explained in **Figure 39** below:

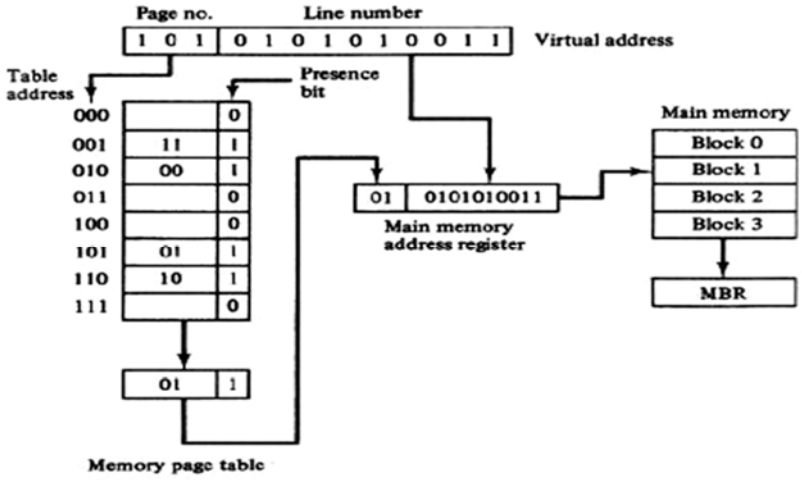


Figure 39 : Memory table in paged system

In the above figure, an auxiliary memory of 8K and main memory of 4K is considered. The 13-bit virtual address generated by the programmer, is divided into two parts, page number and line number. Page number consists of 3 bits which are used to denote one of the 8 pages.

The page table consists of eight locations ranging from 000 to 111. Since the memory space is of 4K therefore, only 4 pages can be accommodated by the main memory at any point of time. Therefore, the address of the block numbers corresponding to the pages that are transferred to those blocks are written in adjacent to the page numbers. The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively. A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory. Using this scheme, the virtual address can be used to generate the main memory address.

9.4.3 ASSOCIATIVE MEMORY PAGE TABLE

The above implementation is useful when the size of address space and the memory space is small. As the size increases, the random-access memory page table is inefficient with respect to storage utilization. In the example cited above, when the address space and the memory space is 8K and 4 K respectively, we have clearly seen that eight words of memory are needed, one for each page, but at least four words will always be marked empty because main memory cannot accommodate more than four blocks. But if we consider an address space of 1024K words and memory space of 32K words with page and block size equal to 1K, the number of pages is 1024 and the number of blocks 32. The capacity of the memory-page table must be 1024 words and only 32 locations may have a presence bit equal to 1. At any given time, at least 992 locations will be empty and not -in use, which is clearly a wastage of memory.

The above problem could be overcome, if the associative memory is used to store the page table as shown in Figure 40**.**

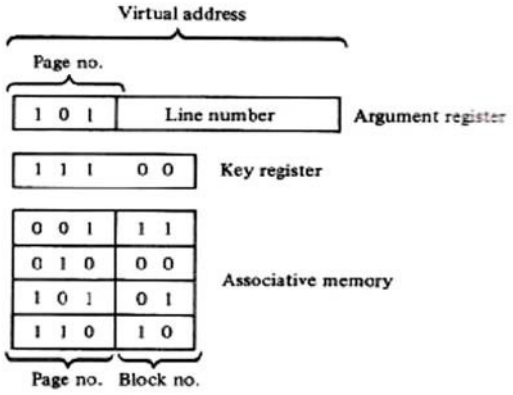


Figure 40 : Page table stored in associative memory

In this case, the size of the memory is chosen according to the block size of the memory space, in contrast to the example in section 13.2.2, where the page table was selected according to the number of the pages in the address space. This will reduce the size of the page table considerably.

In associative memory, each block number is stored along with the associated page in the address space. Since the associated memory facilitates the content search, the page field ineach word is compared with the page number in the virtual address. If a match occurs, the word is read from memory and its corresponding block number is extracted.

Check Your Progress

1. The techniques which move the program blocks to or from the physical memory is called as ……. .
2. The binary address issued to data or instructions are called as …… address.

3. ……. is used to implement virtual memory organisation.
4. ……. translates logical address into physical address.
5. The virtual memory basically stores the next segment of data
6. to be executed on the ……..
7. The asscociatively mapped virtual memory makes use of …….

9.5 SUMMARY

1. A computer system have various type of memories like CPU registers, cache memory, main memory, secondary memory, etc.
2. All the momories have same function i.e. to store the data, but they differ in their speed and cost.
3. The base types that normally constitute the hierarchical memory system include registers, cache, main memory, and secondary memory.
4. The main memory constitutes of RAM and ROM, is the central storage unit in a computer system.
5. The CPU processing speed is very fast as compared to the time taken by a memory-access.
6. It has been found after analyzing a large number of programs that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of *locality of reference.*

9.6 ANSWERS TO CHECK YOUR PROGRESS

1. Secondary memory
2. Locality of reference
3. Mapping function
4. Replacement algorithm
5. Dirty bit
6. Write through
7. Direct
8. Tag
9. Associative
10. Valid
11. Dirty

12. Virtual memory organization
13. Logical
14. Memory Management Unit(MMU)
15. Memory Management Unit(MMU)
16. Secondary storage
17. TBL

9.7 TERMINAL QUESTIONS

1. What is principle of locality of reference?
2. What is memory hierarchy? Why hierarchy concept is used in memory?
3. What is an address space and the memory space?
4. Explain hit ratio and miss ratio.
5. Explain associative mapping in cache organization.
6. Explain direct mapping in cache organization.
7. Explain set-associative mapping in cache organization.
8. How the cache is initialized?
9. Explain the difference between the write-through and write-back method in cache.
10. What is virtual memory?
11. Explain virtual address mapping process using a diagram.
    (a) An address space is specified by 24 bits and the corresponding memory space by 16 bits.
    (b) How many words are there in the address space?
    (c) How many words are there in the memory space?
    (d) If a page consists of 2K words, how many pages and block are there in the system?

UNIT-10 I/O SYSTEM

Structure

10.1 LEARNING OBJECTIVES

After reading this unit, you will be able to:

- Define peripheral devices.
- Understand the necessity of an interface.
- Define synchronous data transfer.
- Define asynchronous data transfer.
- Identify various peripheral devices.

10.2 INTRODUCTION

A CPU is referred to as the brain of the computer. For any processing on data, the data need to be brought inside the CPU. The data is transferred inside the CPU

via input devices and the processed information is displayed via output devices. These input/output devices like, printer, keyboard, mouse, monitor, etc. are collectively known as Peripheral devices. At times the information is stored in the secondary storage devices for future reference. Clearly, the CPU needs to communicate with memory and peripheral devices. Therefore, the CPU requires some path, known as bus, through which it can communicate to these devices. **Figure 41** shows how CPU is connected to these devices.

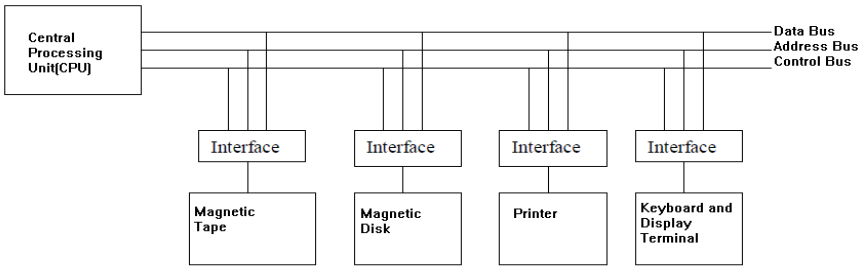


Figure 41 : Processor Interconnection with Peripheral Devices

In the figure above, the peripheral devices are connected to CPU via interface. There are many differences in the implementation of CPU and the peripherals devices, some of the major differences are listed below:

1. CPU is an electronic device whereas the peripheral devices are electromechanical (like printer, which have mechanical motors for page movement) and electromagnetic devices (like secondary memory). Therefore, the technology behind these two entities and the manner of operation is different.

2. CPU transmits and processes the data at very fast rate whereas the data transfer rate of peripheral devices is comparatively very slow. Moreover, CPU transmits the data in parallel whereas peripheral devices usually transmit data serially.

3. Data codes and formats in peripherals differ from the word format in the CPU and memory.

To overcome these mismatches, a special hardware, known as interface is connected between CPU and these peripherals device to take care of all the above issues and to supervise and synchronize all input and output transfers. These components are called **interface.**

CPU needs to communicate with memory and other peripherals devices. There are many peripheral devices attached to a computer, so to locate a specific device, an address need to be specified. Similarly, to read/write data from/to memory, an address need to be specified. Also, CPU need to specify the control information i.e., what operation is to be performed. Like in case of memory, CPU needs to specify whether it is a read operation or a write operation. After specifying the address and the control information, the data need to be transferred between CPU and peripheral devices/memory. For this purpose, bus is required. Bus is a

conducting path that connects the CPU with other devices. Based on the purpose for which the bus is used, it is categorized into three categories.

a. *Address Bus:* An address bus is used by the CPU to transmit the address of the peripherals/memory location. Since, it is used by CPU only, this bus is unidirectional.

b. *Control Bus:* It is used by the CPU to control and regulate the various devices attached to it. This bus is bidirectional.

c. *Data Bus:* It is used by CPU and the devices attached to the CPU to transfer data. This Bus is bidirectional.

Apart from peripheral devices, the CPU is also connected to main memory and needs to frequently communicate with it. So bus is also required to connect the CPU with memory also. There are three possible architectures to connect the CPU with memory and peripheral devices.

Different set of address, data and control buses are used to connect memory and other Peripheral Devices (PD), as shown in. **Figure 42**. In this case a separate Input/Output processor (IOP) is required to control the peripheral devices. The memory communicates with CPU and IOP using memory bus whereas the IOP communicates with its separate address data and control bus.

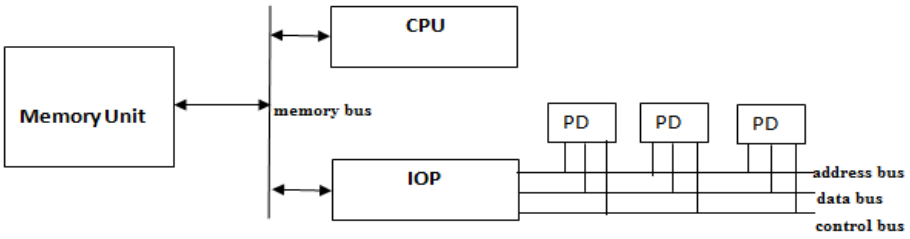


Figure 42 : Interconnection of Peripheral Devices and Memory using different set of Buses. The second alternative is to connect CPU to memory and peripheral is using different set of control buses but data and address buses are common.

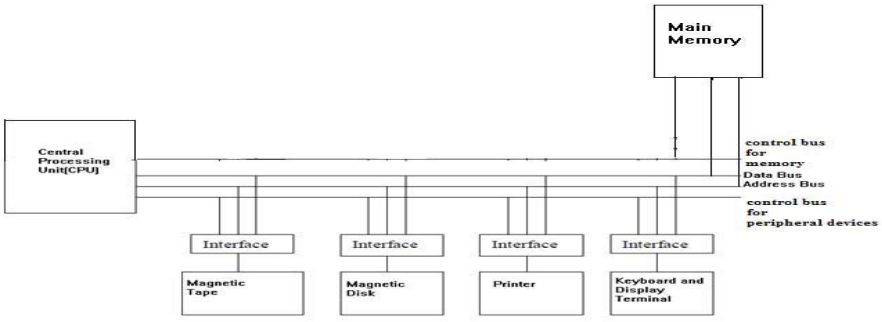


Figure 43 : Interconnection of CPU and Peripheral Devices using Common Address & Data Bus and different Control Buses. The same is shown in **Figure 43**. When CPU needs to communicate with memory or I/O device, it places the data in the data bus and specifies the address using address bus. The distinction between I/O device and memory is made via control lines. If CPU wants to interact with memory, it will use the control lines dedicated for memory. This will isolate all the I/O devices and the memory will clearly recognize that the address is directed to memory, not I/O devices. Similarly, if CPU wants to interact with I/O devices, it will use the control lines dedicated for I/O devices. Whenever the CPU communicates with memory, the I/O device remains isolated and when CPU communicates with I/O devices, memory becomes isolated. Therefore this configuration is also known as Isolated I/O method.

The third alternative is to use all the busses i.e., data, addresses and control bus to connect the CPU with peripheral and memory. The same is shown below in 9 **Figure**.

In this case, some of the address range of the main memory is reserved to peripheral devices. This configuration is known as memory-mapped I/O. In this case, the CPU does not make any distinction between memory and peripheral devices and use the same set of read and write signals for both memory-read/ peripheral-read and memory write/peripheral-write operations as the interface of the peripheral devices are treated similar to memory address (refer **Figure**). Thus, this reduces the number of instructions in the instruction set of the computer as the same instructions can be used for memory as well as peripheral devices. The drawback of this scheme is, the memory cannot be fully utilized as some of the address from the address range of memory is reserved for peripheral address.

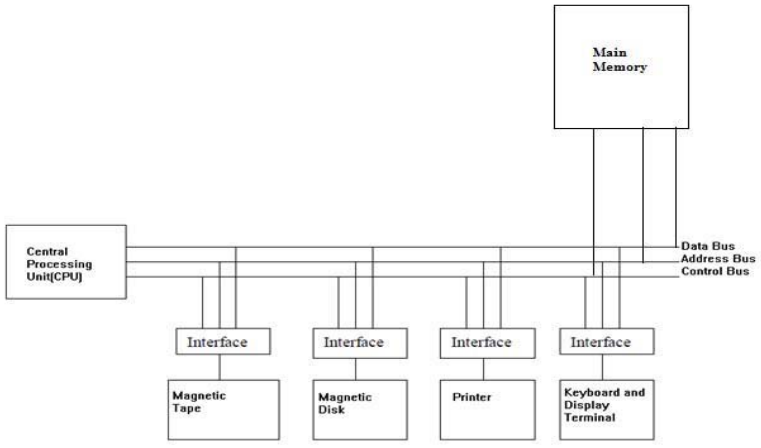


Figure 58 : Interconnection of CPU and Peripheral Devices using Common Address, Data Bus Control Bus Address

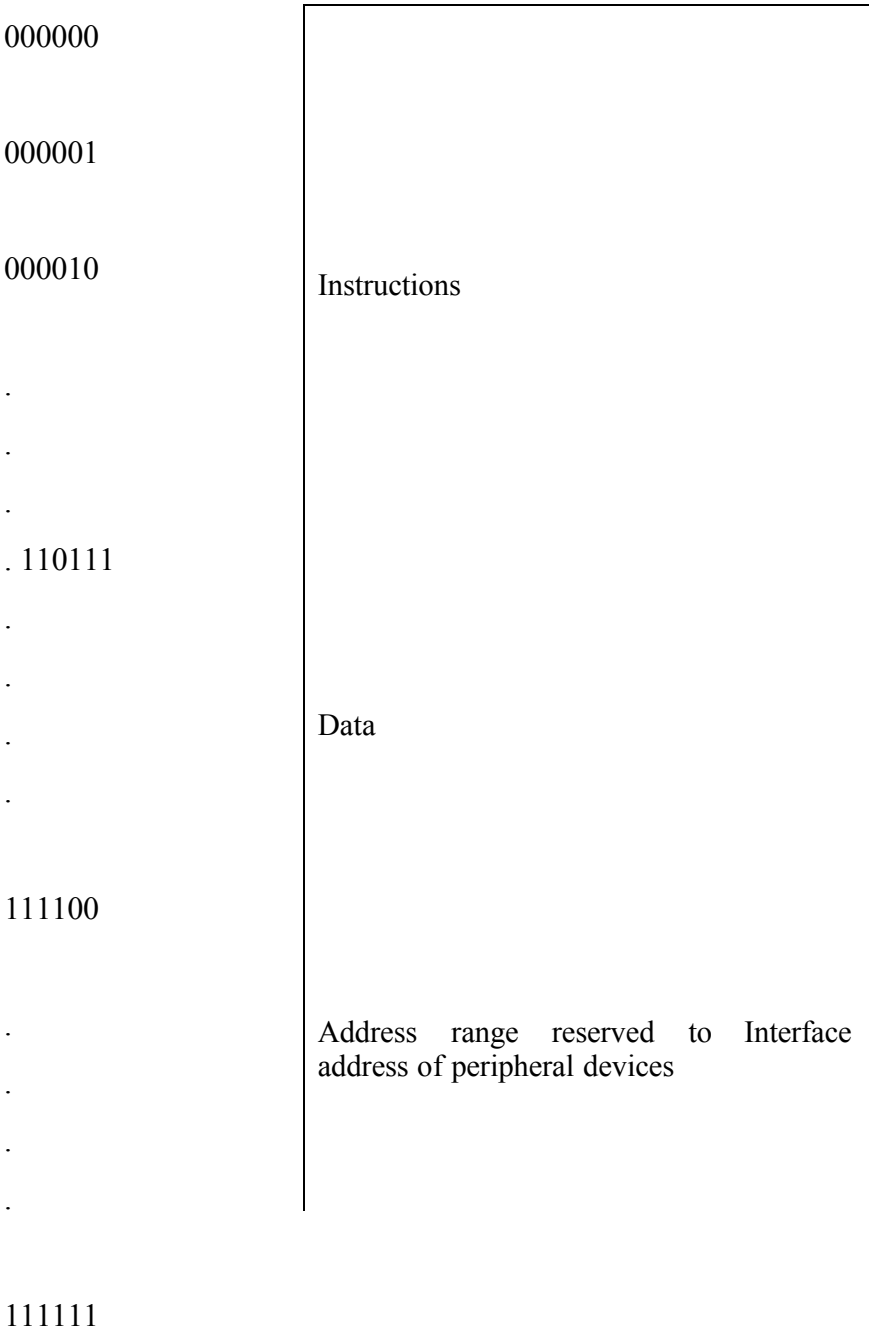


Figure 59 : Address Range in Memory-Mapped I/O Configuration

10.3 PERIPHERAL DEVICES

The CPU of a computer interacts with the outer world using Input/ Output devices, which are attached to the computer and are commonly known as peripheral devices. Some of the most commonly used peripherals devices are keyboard, printer, magnetic tapes, magnetic disks, display unit, etc. Now let us discuss some of these I/O and peripheral devices in detail.

10.2.1 KEYBOARD

The CPU needs data for processing. Depending on the type of data to be processed, there are several ways in which the data can be transmitted to CPU. If the data is an image which is to be processed, then the preferred input devices would be camera, scanner, etc. If an audio is to be processed, then microphone would be an idle choice. However, in most of the cases the input data is text and a keyboard is used to input the alphanumeric information into computer. **Figure 44** below shows a normal keyboard of a computer.

Figure 44: A keyboard

There are many variant of a computer available in the market nowadays and most of them have between 80-110 keys which consists of set of alphabet keys, numeric keys, function keys, control keys, arrow keys and operator keys. The keys of the typewriter are laid out on the same pattern as typewriter. This pattern is know as QWERTY, which is the first six alphabets of the top row from left to right. The logic behind this arrangement was to avoid colliding and jamming the metal arms of the typewriter while typing. There are several other varients of the keyboards like ADCBS, XpeRT, QWERTZ, AZERTY etc.

10.2.2 MAGNETIC TAPE

After the data is processeds by the CPU, the information may be stored for future reuse. For this purpose, a computer have a variety of internal and external memory elements. Some of the popular options for external storage are magnetic disks and megnetic tapes. The per bit storage cost of magnetic disk is higher than the magnetic tape. But Megnetic disks have less access time (faster) as compared to magnetic tapes. Generally these are used for system backup.

10.2.3 DISPLAY SYSTEM

When the information is generated by the CPU, we need to confirm whether the information generated is correct and relevent. For this purpose, variety of display devices are used. For example, monitor, printer, plotter, etc.

As discussed earlier, these peripheral devices are implemented using different technologies, varies in speed and operation. Therefore, an interface, which acts as a mediator is required between the CPU and the peripheral device to communicate effectively. Now let us discuss the operation and architecture of an interface unit in detail.

10.3 INPUT/OUTPUT INTERFACE

The importance of interface unit is already pointed out in the introduction section. Now let us discuss an example of an interface device to bring more clarity on the working of an interface unit.

Figure-45 shown below explains the block diagram of interface unit. The right side of the Interface unit is connected to peripheral devices. It consists of four register namely, Port A register, Port B register, Control register and Status register. These registers can communicate with the peripheral device attached to it. Port A and Port B are bi-directional registers used for sending and receiving data. Control register is used by the CPU to send the control information to the peripheral device. Status register is used by the CPU to check the status of the device or the data transfer that is currently taking place. The left side of the I/O interface unit consists of timing and control unit which receives the signals from CPU. The timing and control unit consists of a chip select(CS) signal, which is used by the CPU to select the interface of a particular device. This chip select logic is generally connected to the address bus of the CPU via decoded. Whenever the CPU generate the address of the device, it place the address of that device in the address bus. The decoded attached to the Chip select logic continuously monitors the address line and as soon as the address in the address bus of the CPU matches the address of a device the decoder attached to the chip select logic enables CS signal and the interface is selected and available to the CPU for further orders.

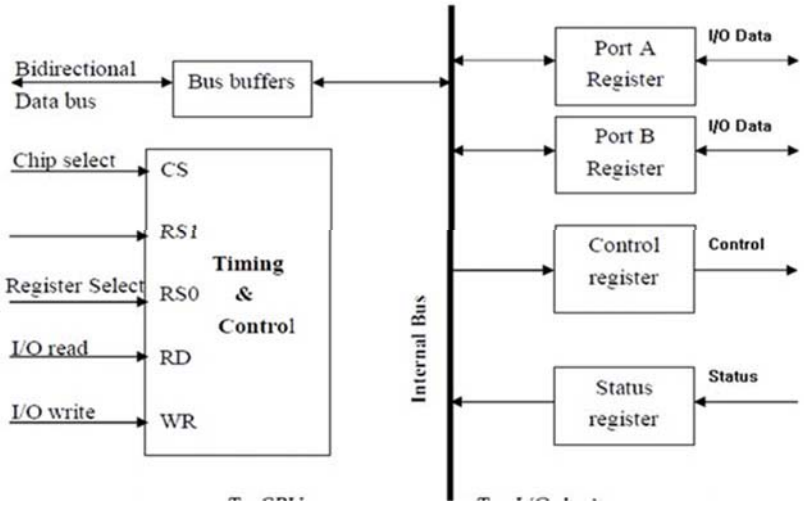


Figure 45 : Block diagram of an I/O interface

The CPU communicates to the device via interface registers. If CPU wants to send the control information, it will use control register. Similarly, if the CPU wants to check the status of the device, it will check the bits of status register. In case, the data is to be transmitted/received, CPU uses port A and port B for this purpose. Now the question arises, how the CPU select a particular registers among the four registers. The Answer to this question is, CPU uses two register select lines, RS0 and RS1 to select between the four register. The Table 8 show how the register select lines are used to select a particular register.

Table 8 : Chip and Register Selection Logic

CS	RS1	RS0	Selected Register
0	X	X	None: as the chip is not selected
1	0	0	Port A
1	0	1	Port B
1	1	0	Control Register
1	1	1	Status Register

When the chip select logic is 0, the data bus is in high-impedance state. When the address bus contains the address of a particular device, the decoder attached to the chip select logic of the device enables the CS logic and one of the register can be selected using the RS1 and RS0 line. There are two additional lines RD and WR to differentiate between the read and write operation.

10.4 ASYNCHRONOUS DATA TRANSFER

A computer is a digital system which is composed of variety of sub- systems like CPU, memory, I/O units. The operations of these units are synchronized by a clock pulses, which are generated by a common pulse generator. If the CPU and the interface unit shares a common clock then these two unit are said to be *synchronous* to each other. Whereas if the two units have its own private clock, the two units are said to be *asynchronus* to each other. Whenever two asynhcronous units communicate with each other, some control signals are needed to be sent along with the data signals. These control signals consists of the time information at which the data is being transmitted. One of the mechanism to achieve this is by sending a strobe pulse from the source, which consists of the information for the destination unit regarding the time of data transfer.

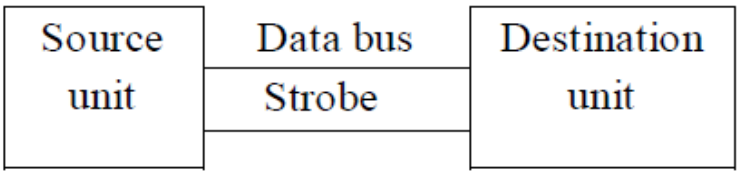


Figure 46 : Source-initiated strobe for data transfer

The strobe may be activated by either source or the destination unit. The Strobe signal is disabled indicates that the data bus does not contain valid data. New valid data will be available only after the strobe is enabled again.

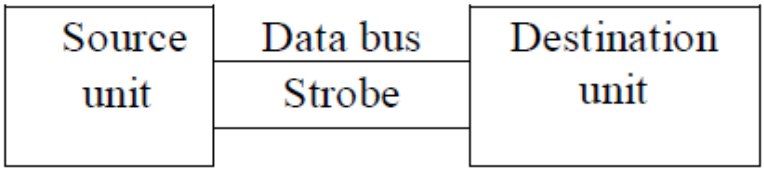


Figure 47: Destination-initiated strobe for data transfer

But this technique have a serious drawback. The source does not have any mechanism to know whether the data sent to the destination is received by the destination unit, and if yes, when? To overcome this problem, another technique, known as *handshaking* is used. In this method, whenever a source need to transmit data, it sends a strobe signal to the destination unit which is a signal for the destination unit that the data bus now contains valid data. After this, the data is placed in the data bus by the source. Once the data is received by the destination, the destination returns the acknowledgement signal which is a signal to the source that the destination unit has received data successfully.

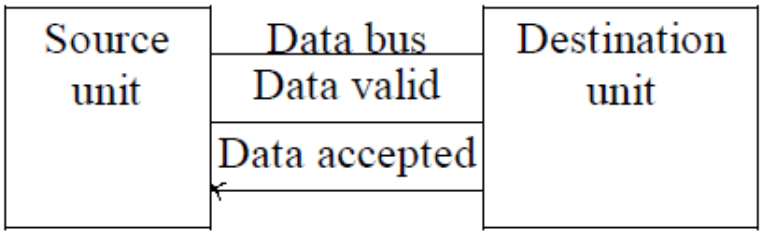


Figure 48: Source-initiated transfer using handshaking

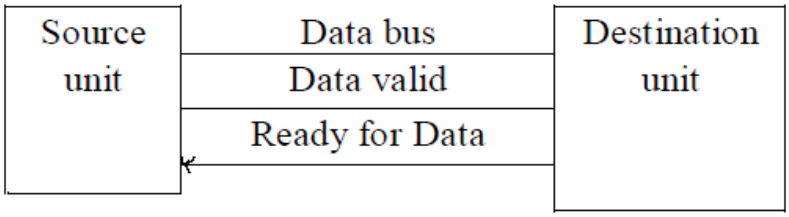


Figure 49: Destination-initiated transfer using handshaking

10.5 I/O CARDS IN PERSONAL COMPUTERS

Personal Computer is built on open architecture i.e. it can be expanded in future, if need arises. The expansion can be done using expansion slots, which are the sockets present in the motherboard of the computer wherein expansion cards, also known as interface cards, plug-in boards, or adapter cards can be plugged-in. Graphic card, sound card, network interface card etc are some of the popular I/O cards that are popularly known. These expansion cards are connected to the buses of the motherboard so that the device attached to the expansion card can

communicate with the CPU. As we already know that the bus that connects the CPU with the main memory is known as system bus. The bus that connects the CPU with the expansion slot is known as expansion bus. Some of the well-known expansion buses are ISA (Industry Standard Architecture), PCI(Peripheral Component Interconnect), and AGP(Accelerated Graphic Port). Now we will discuss some of the popular I/O cards used in PC.

10.5.1 GRAPHIC CARD

Graphic card is already chipped in the motherboard of the personal computer. It is a card through with the monitor of the computer is connected. It is also known as video card or video adapter. Its function is to convert the signals received from the CPU into video signals or image so that it can be displayed in an output device like monitor. It comes in the variety of configuration like 8 MB, 16 MB or 32 MB.

Figure 50: Graphic card

As shown in **Figure 50**, like CPU, the graphic cards have its own processor, BIOS, and RAM and are designed for performing complex mathematical and geometric calculation necessary for graphics rendering.

10.5.2 SOUND CARD

Like Graphic cards, the sound cards are also preinstalled in the motherboard of PC nowadays and it is used to transmit the digital sound through speakers and headphones.

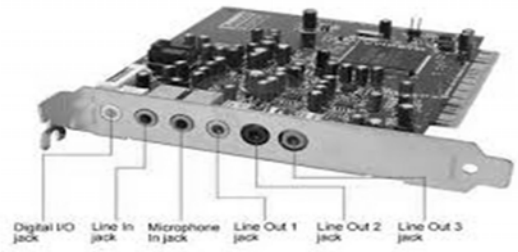


Figure 51: Sound card The basic function of a sound card is to convert digital data into analog information and analog information to digital data. This can be accomplished either by using separate A to D (Analog to Digital) and D to A(Digital to Analog) convertor or a single CODEC(coder/decoder) chip which

can perform both the functions. Music is produced by sound card using a process called wavetable synthesis. It is a method of creating music based on wave table, which is a collection of digitalized sound samples taken from recording of actual instruments. A sound card converts analog information it has received through one of its inputs into digital data by taking precise measurements of an analog sound wave at a rate of thousands of times per second. These sound samples are mixed and stored on the sound card. When the user wants to create sound/music from computer, the sound card reproduces analog sound waves from those digital measurements.

10.5.3 NETWORK INTERFACE CARD (NIC)

This card is also pre-installed now-a-days on the motherboard of a computer and its basic purpose is to provide a physical and logical link for a PC to a network.

Figure 68 : NIC card

Networks transmit data in a serial data format (1 bit at a time), and the data bus of the PC moves data in a parallel format (8 bits at a time). The NIC acts as an interface between the PC and the network and its primary task is to convert the signal from serial to parallel format or from parallel to serial format, depending on its direction. The NIC also formats the data as required by the network architecture.

Check Your Progress

1. The input/output devices like, printer, keyboard, mouse, monitor, etc. are collectively known as ……. devices.
2. If the CPU and the interface unit shares a common clock then these two unit are said to be …… to each other.
3. ISA stands for ……. .
4. PCI stands for ……. .
5. AGP stands for……. .
6. CODEC stands for……. .
7. NIC stands for……. .

10.6 SUMMARY

1. A CPU is referred to as the brain of the computer. For any processing on

data, the data need to be brought inside the CPU.

2. The data is transferred inside the CPU via input devices and the processed information is displayed via output devices.
3. CPU needs to communicate with memory and other peripherals devices. There are many peripheral devices attached to a computer, so to locate a specific device, an address need to be specified.
4. Apart from peripheral devices, the CPU is also connected to main memory and needs to frequently communicate with it.
5. In this case, some of the address range of the main memory is reserved to peripheral devices. This configuration is known as memory-mapped I/O.
6. The CPU of a computer interacts with the outer world using Input/ Output devices, which are attached to the computer and are commonly known as peripheral devices.
7. Some of the most commonly used peripherals devices are keyboard, printer, magnetic tapes, magnetic disks, display unit, etc.
8. A computer is a digital system which is composed of variety of sub-systems like CPU, memory, I/O units. The operations of these units are synchronized by a clock pulses, which are generated by a common pulse generator.

10.7 ANSWERS TO CHECK YOUR PROGRESS

1. Peripheral
2. Synchronous
3. Industry Standard Architecture
4. Peripheral Component Interconnect
5. Accelerated Graphic Port
6. coder/decoder
7. Network Interface Card

10.8 TERMINAL QUESTIONS

1. What are the three possible architecture to connect the CPU with memory and peripheral devices? Explain in details using diagram.
2. What is wavetable synthesis?
3. What is isolated I/O method?
4. What is memory mapped method?
5. How isolated I/O method is different from memory mapped method.

6. What is the purpose of using interface unit?
7. Explain the working of I/O interface using a diagram.
8. Discuss some of the popularly used I/O cards in personal computers.
9. What is asynchronous data transfer?
10. What is Handshaking?

UNIT-11 INTRODUCTION TO 8085 MICROPROCESSOR AND MICROCONTROLLERS

Structure

11.1 LEARNING OBJECTIVES

After reading this unit, you will be able to:

- Define a Microprocessor.
- Explain the pin diagram of a 8085 microprocessor.
- Understand the instruction set of 8085 microprocessor.

11.2 INTRODUCTION

A microprocessor (μp) is an electronic device that is used by the computer to its processing. The term CPU and microprocessor are often used interchangeably, but there is a difference. The CPU of a computer consists of Arithmetic Logic Unit (ALU) and Control Unit (CU). In the earlier days, when the chip fabrication technology was in its initial phase, it was not possible to integrate the ALU and

CU into the same chip. The ALU and CU were connected to each other and the combined unit was known as CPU. With the advent of LSI, VLSI and VVLSI technologies, it was possible to fabricate both ALU and CU into the same chip. When the ALU and CU were fabricating into the same chip, it is known as a microprocessor. So a microprocessor, similar to a CPU, performs the basic arithmetical, logical, and input/output operations of the system. Unlike, PC which is designed to be a general purpose machine, a microprocessor can be designed for both general purpose and specific purpose. Almost all of your automatic home appliances like an automatic washing machine, dishwasher, digital set-top box, music system, air conditioners, etc. Even it finds place in your car, motorcycle and scooter. This small magical device has affected our day-to- day life to such an extent that today we cannot think of our life without it. Be it our public transportation infrastructure, where it find place in traffic lights, railways, transport and fright management. Medical sciences' sophisticated equipments and automatic patient monitoring systems are designed using microprocessors. Most of the communication and handheld devices like smartphones, GPS systems, etc. have a microprocessor inside it. Now in the next section, we will take a closer look at the working and functionality of a microprocessor.

11.3 ARCHITECTURE OF MICROPROCESSOR

A microprocessor is a programmable device which can perform different sets of operations on the data it receives as an input depending on the sequence of instructions supplied in the given program.

The processing power of a microprocessor depends on its instruction set. Like a CPU, a microprocessor can perform only those operations for which it is designed. The collection of all such instructions for which the operations are defined is called an instruction set. Larger the number of instructions in the instruction set, better is the execution power of a microprocessor. A microprocessor primarily consists of three units:

- The Arithmetic/Logic Unit (ALU)
- The Control Unit.
- Internal Register Set

Depending on the width of the data bus of a microprocessor, it can be categorized into 8-bit, 16-bit, 32-bit or 64-bit microprocessor. The width of the data bus signifies the number of data bits a microprocessor can process simultaneously. Our aim here is to introduce with the basics of microprocessor and its internal working, therefore we will discuss a simple 8-bit microprocessor.

There are variety of microprocessors available in the market manufactured by different companies like from Motorola 6800, ZiLOG Z80 and Intel 8085. We will discuss the most popular one i.e., Intel 8085 in detail.

11.2.1 8085 MICROPROCESSOR

Intel launched its first 8-bit microprocessor in 1972 and names it Intel 8008. Soon after, it improved version, Intel 8080 was launched. Finally, Intel

launched 8085 in 1977, which was much powerful then its two earlier versions. Intel 8085 is a 8-bit general purpose microprocessor capable of addressing 64K memory. It has 40 pins and runs on +5 V power supply. It operates with 3 MHz clock. In 8085, the 8-bit data bus was multiplexed with the lower part of 16-bit address bus so that the number of pins are limited to 40. It has a 16-bit address bus, hence it is capable of addressing 2^{16}= 64 KB memory.

It consists of:

- **Control unit** : control microprocessor operations.
- **ALU** : performs data processing function.
- **Registers** : provide storage internal to CPU.
- **Interrupts**
- **Internal data bus**

The pin diagram of Intel 8085 is shown in fig. 97. The pins on the chip can be grouped into 6 groups:

- Address Bus.
- Data Bus.
- Control and Status Signals.
- Power supply and frequency.
- Externally Initiated Signals.
- Serial I/O ports.

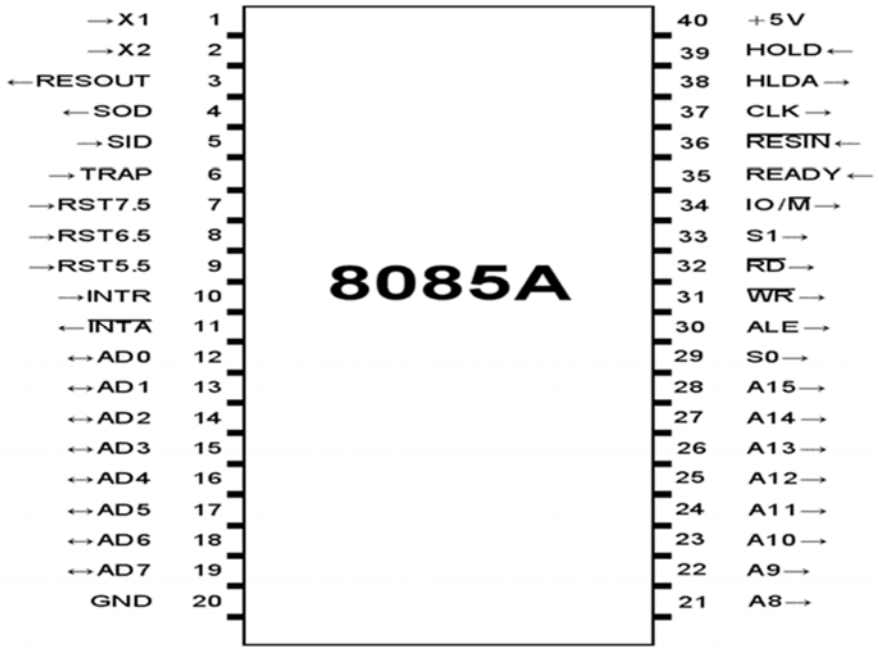


Figure 69: Pin Diagram of Intel 8085[3]

1. *Address and Data Signals:* The address bus has 8 signal lines A8 – A15 which are unidirectional. The other 8 address bits are multiplexed (time shared) with the 8 data bits. The bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time. During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits. In order to separate the address from the data, we use a latch to save the value before the function of the bits changes.

2. *Control and Status Signals*: There are 4 main control and status signals. These are:

    a. ALE(pin 30): Address Latch Enable. This signal is a pulse that become 1 when the AD0 – AD7 lines have an address on them. It becomes 0 after that. This signal can be used to enable a latch to save the address bits from the AD lines.

    b. RD(pin 32): Read (Active low). WR(pin 31): Write(Active low).

    c. IO/M(pin 34): This signal specifies whether the operation is a memory operation (IO/M=0) or an I/O operation (IO/M=1).

    d. S1(pin 33) and S0(pin 29) : Status signals to specify the kind of operation being performed. Usually not used in small systems.

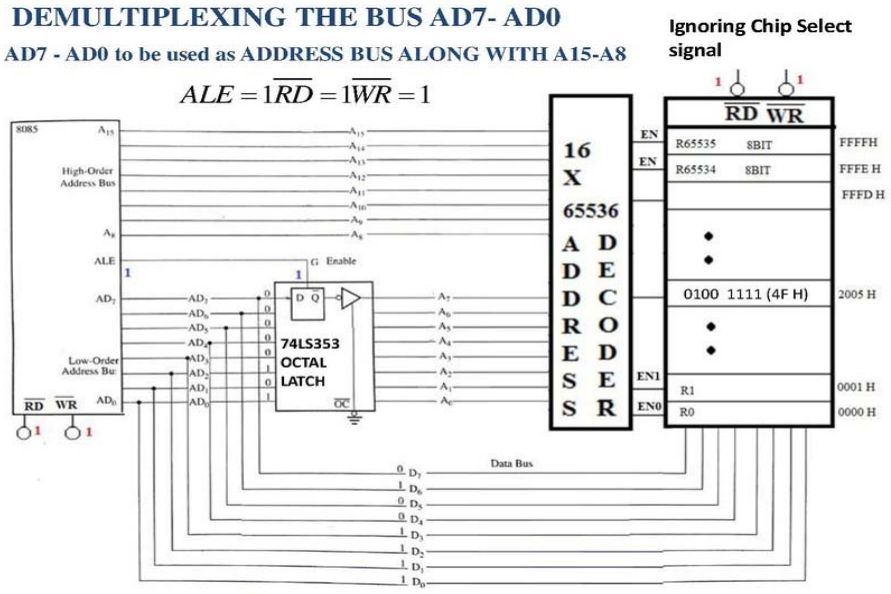


Figure 52: Multiplexed Data and Address Bus in Intel 8085[4]

[3] Image adopted from:

https://commons.wikimedia.org/wiki/File:Anschlussbelegung_8085.gif

3. *Power Supply and Frequency:* There are 3 important pins in the frequency control group.

   a. X1(pin 1) and X2(pin 2) are the inputs from the crystal or clock generating circuit. The frequency is internally divided by 2. So, to run the microprocessor at 3 MHz, a clock running at 6 MHz should be connected to the X1 and X2 pins.

   b. CLK (OUT)(Pin 37): An output clock pin to drive the clock of the rest of the system.

4. *Interrupts and Externally initiated signals*:

   a. INTR(pin 10): Input Request. This signal is used as a general-purpose interrupt.

   b. INTA(Active Low)(pin 11): Interrupt Acknowledge. This signal is used to acknowledge an interrupt.

   RST 7.5(pin7)

   Restart Interrupts. These are vectored interrupts

   RST 6.5 pin8

   c. RST 5.5(pin9) that transfer the control of the program to specified memory locations.

d. TRAP(pin 6): This is a non-maskable interrupt and has a highest priority.

e. HOLD(pin 39): Whenever a peripheral device, such as DMA want the hold of the address and the data bus, this signal is initiated by the peripheral device.

f. HLDA(pin 38): Hold Acknowledge. This signal acknowledges the HOLD request.

g. READY(pin 35): This signal is used to delay the microprocessor Read or Write cycles until a slow peripheral device is ready to send or receive data. When this signal is low, the microprocessor waits for a integral number of clock cycles until it goes high.

There are two kinds of RESET signals in 8085:

h. RESET IN(pin 36): an active low input signal, Program Counter (PC) will be set to 0 and thus MPU will reset.

i. RESET OUT(pin 3): an output reset signal to indicate that the µp was reset (i.e. RESET IN=0). It also used to reset external devices.

---

4 Image adopted from: https://commons.wikimedia.org/w/index.php?title=File:8085_microprocessor_Presentation.pdf&p age=2

5. *Serial I/O Ports:*

   a. SID (Input)(pin5): Serial input data line The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

   b. SOD (output)(pin 4): Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

   There are two more pins left:

   a. Vcc(pin 40): +5 volt supply.

   b. Vss(pin 20): Ground Reference.

   The signal groups of Intel 8085 microprocessor is shown in Figure 53**.**

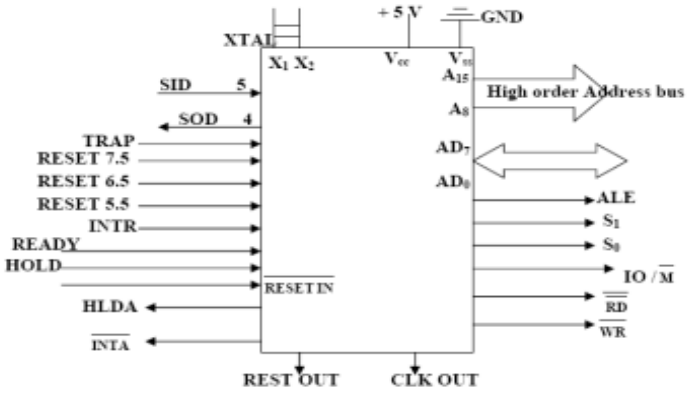


Figure 53: Signal Groups of Intel 8085

11.2.2 THE INTERNAL ARCHITECTURE OF 8085

Now let discuss the internal architecture of the 8085 microprocessor in detail.

Figure 54 explains the internal architecture of the 8085.

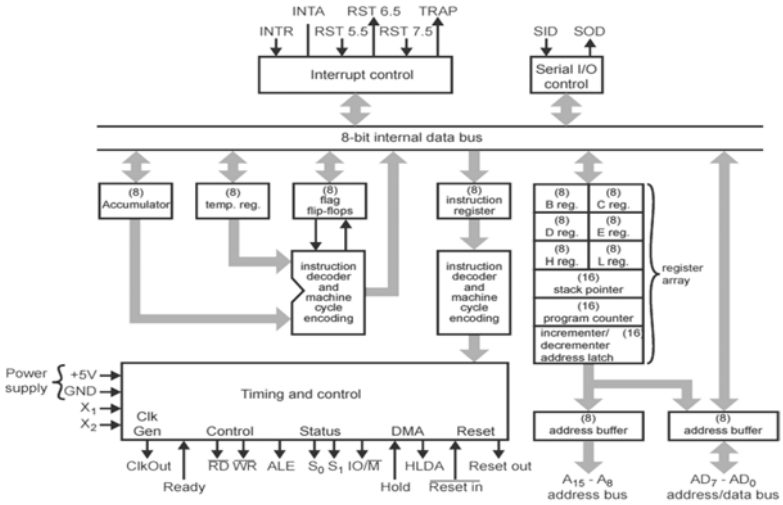


Figure 54 : Internal Architecture of 8085 Microprocessor

The ALU consists of arithmetic and logic circuits to perform arithmetic and logic operations. It also consists of register set to perform these operations on the data. The details of these registers are discussed below.

The 8085 microprocessor contains seven 8-bit register which are directly accessible to the programmer. These registers are named as A, B, C, D, E, H, and L.

A	Status Register
B	C
D	E
H	L
Program Counter	
Stack Pointer	

Figure 55 : Register Set of 8085 Microprocessor

A is the 8-bit accumulator where all the operation on data takes place. The other six registers can be used as 8- bit register or a 16-bit register pair to manipulate 16-bit data. The register pair is as follows: BC, DE, and HL. In case a operation on a 16-bit data is to be performed, the HL pair can be used as a 16-bit accumulator. Apart from that, it also consists of two 16-bit registers. PC, program counter, controls the sequencing of the execution of instructions and is used to store the address of the next instruction to be executed. And SP, stack pointer, is used to point the address of the top-most element of the stack. The register set is shown in Figure 55.

It register set also contains 8-bit status register. Each bit of this status register contains a flag, which is a 1–bit flip-flop. These status flags are affected by the arithmetic and logic operations before or after the operation. There are six status flags in the status register and these are S (sign flag), Z (zero flag), AC (auxiliary carry flag), P (parity flag) & CY (carry flag).

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
S	Z		AC		P		CY

Figure 56: Status Flags The use of these flags is explained below:

a. S(sign flag): The sign flag is set if bit D7 of the accumulator is set after an arithmetic or logic operation.

b. Z(zero flag): Set if the result of the ALU operation is 0. Otherwise is reset. This flag is affected by operations on the accumulator as well as other registers. (DCR B).

c. AC(Auxiliary Carry): This flag is set when a carry is generated from bit D3 and passed to D4 . This flag is used only internally for BCD operations.

d. P(Parity flag): After an ALU operation, if the result has an even # of 1s, the p-flag is set. Otherwise it is cleared. So, the flag can be used to indicate even parity.

e. CY(carry flag): This flag is set when a carry is generated from bit D7 after an unsigned operation.

f. OV(Overflow flag): This flag is set when an overflow occurs after a signed operation.

Whenever an Instruction from the memory is fetched, the instruction is placed inside a 8-bit register, known as Instruction Register (IR). A decoder is attached to the Instruction Register which enables the CPU to decode the instruction and take appropriate action. Suppose, after decoding the instruction, it was found that it was an addition instruction, the CPU will generate necessary control signals to initialize the adder and fetch the operand either from the memory or the input device.

The 8085 microprocessor has 8-bit data bus and 16-bit address bus. The address bus has 8 signal lines A8 –A15 which are unidirectional. The other lower order 8 address bits are multiplexed (time-shared) with the 8 data bits. So, the bits AD0 –AD7 are bi-directional and serve as A0–A7and D0 –D7at the same time. During the execution of the instruction, these lines carry the address bits during the early part, and then during the late parts of the execution, they carry the 8 data bits. In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.

Now let us discuss how the demultiplexing of AD7-AD0 is done to serve the dual purpose i.e. the same line are used as address lines and data lines. The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally. Therefore, an external latch is used to save the value of AD7–AD0 when the lines are carrying the address bits. Address Latch Enable(ALE) signal is used to enable the latch. Whenever AD7- AD0 is to be used for the data bus, the ALE goes low.

Direct Memory Access (DMA) technique is used when a fast speed I/O device want to transmit the data to memory at high speed and the speed of CPU limits the speed of transfer. In this case, the CPU is bypassed and the control of the data and address buses is given to the transmitting device. Once the transfer is complete, the control of Data and Address bus is relinquished to the CPU. To facilitate DMA transfer, HOLD and HLDA signals are used. Whenever an I/O device request

for DMA transfer, it enables the HOLD line. As soon as the HOLD line is enabled, the microprocessor data and the address bus of the CPU are placed in the high impedance state and the control of the buses is transferred to I/O device. After this, the HLDA signal is initialized by the CPU which is a signal for the I/O device to start the transfer of data. Once the data transfer is complete, the HOLD signal is disabled and the control of buses is returned to CPU.

An Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced. 8085 microprocessor has few maskable and non-maskable Interrupts.

There are four hardware interrupts in 8085:

- TRAP
- RST 7.5
- RST6.5
- RST5.5

Interrupts are generally used to stop the normal execution sequence of the instructions by the CPU and address a higher priority task first. This usually happens when the peripheral device want to transmit data to either to memory or CPU. The device which seeks CPU attention sends an interrupt signal INTR to CPU. The CPU holds the operation which it was performing, save the intermediate data and the registers value so that it could resume the task later. The CPU sends the interrupt acknowledgment INTA to the device, which is a signal that the CPU is now ready to serve the request of the device which initiated the interrupt and the vectored address, the address of the Interrupt Service Routine to handle the interrupts is stored in the Program Counter(PC). After this the CPU executes the *Interrupt Service Routine(ISR)*, which is a small program/routine to service the corresponding interrupting source. The interrupts are of two categories, maskable and non-maskable.

a. *Maskable interrupts*: these are the class of interrupts which a CPU can ignore if it is servicing an important task. TRAP is an example of a non-maskable interrupt.

b. *Non-maskable interrupts*: these are the class of interrupts which the CPU cannot afford to ignore and has to be serviced immediately, come what may. Typically this category of interrupts is used for critical condition. RST 7.5, RST 6.5 and RST 5.5 are the examples of maskable interrupt.

The priority order of the interrupts is as follows: TRAP > RST 7.5 > RST 6.5> RST5.5> INTR

11.3 INSTRUCTION SET OF 8085 MICROPROCESSOR

A computer can perform all the operations for which it is designed i.e. perform operations that are defined in its instruction set. To perform a specific task, a program, which is a sequence of instructions, is written and through this sequence of instructions we instruct the computer to perform what operation is to be

performed, on what data and in which sequence. The programmer can write a program in assembly language using these instructions. These instructions have been classified into the following groups:

- Data Transfer Group
- Arithmetic Group
- Logical Group
- Branch Control Group
- I/O and Machine Control Group

11.3.1 DATA TRANSFER GROUP

This category of instructions are used to transfer the content of source register to another destination register and after the transfer of the data, the content of the source remains unaltered. It is similar to copy command, where the selected contents from the source are transferred to destination without the affecting of content at the source. The example of this category instructions are MOV, MVI, LDA, LXI, STA, etc.

Example #1: MOV R1, R2

This instruction moves/copies the content of the source register R2 to Destination register R1.

MOV [Destination Register], [Source Register]

Suppose before the execution of the above instruction, the content of register R1 and R2 are 20 and 30 respectively.

Destination Register R1	**Source** Register R2
20	30

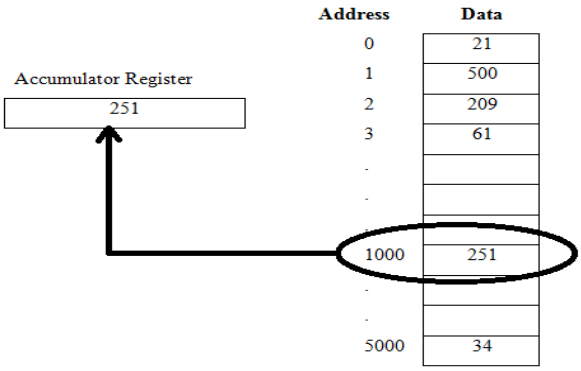
After the execution of the statement MOV R1, R2, the content of R1= 30 and R2=30 i.e. the content of the source register remains same even after the execution of the instruction. Only the content of destination register is altered.

Destination Register R1	Source Register R2
30	30

Example #2: LDA 1000

This instruction load the content of the memory location, as specified in the instruction to the accumulator register and the after the execution of the

instruction, the content of the memory location remains intact.



LDA [Address of Memory Location]

In the above example, after the LDA 1000 instruction is executed, the content of location 1000 is transferred to Accumulator register and after the transfer; the content at memory location remains unchanged.

The list of instructions which falls in this category is:

- MOV r1, r2 (Move Data; Move the content of the one register to another). [r1] ← [r2]
- MOV r, m (Move the content of memory register). r ← [M]
- MOV M, r. (Move the content of register to memory). M ← [r]
- MVI r, data. (Move immediate data to register). [r] ← data.
- MVI M, data. (Move immediate data to memory). M ← data.
- LXI rp, data 16. (Load register pair immediate). [rp] ← data 16 bits, [rh] ← 8 LSBs of data.
- LDA addr. (Load Accumulator direct). [A] ← [addr].
- STA addr. (Store accumulator direct). [addr] ← [A].
- LHLD addr. (Load H-L pair direct). [L] ← [addr], [H] ← [addr+1].
- SHLD addr. (Store H-L pair direct) [addr] ← [L], [addr+1] ← [H].
- LDAX rp. (LOAD accumulator indirect) [A] ← [[rp]]
- STAX rp. (Store accumulator indirect) [[rp]] ← [A].
- XCHG. (Exchange the contents of H-L with D-E pair) [H-L] ↔ [D-E].

11.3.2 ARITHMETIC GROUP

All the arithmetic operations like addition, subtraction; increment or decrement comes under this category.

Example #1:

DAA (Decimal adjust accumulator)- The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in the decimal system. It uses carry and auxiliary carry for decimal adjustment. 6 is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

The example of this category instructions are:

- ADD r. (Add register to accumulator) [A] ← [A] + [r].
- ADD M. (Add memory to accumulator) [A] ← [A] + [[H-L]].
- ADC r. (Add register with carry to accumulator). [A] ← [A] + [r] + [CS].
- ADC M. (Add memory with carry to accumulator) [A] ← [A] + [[H-L]] [CS].
- ADI data (Add immediate data to accumulator) [A] ← [A] + data.
- ACI data (Add with carry immediate data to accumulator) [A] ← [A] + data + [CS].
- DAD rp. (Add register paid to H-L pair) [H-L] ← [H-L] + [rp].
- SUB r. (Subtract register from accumulator) [A] ← [A] – [r].
- SUB M. (Subtract memory from accumulator) [A] ← [A] – [[H-L]].
- SBB r. (Subtract register from accumulator with borrow) [A] ← [A] – [r] – [CS].
- SBB M. (Subtract memory from accumulator with borrow) [A] ← [A] – [[H-L]] – [CS].
- SUI data. (Subtract immediate data from accumulator) [A] ← [A] – data.
- SBI data. (Subtract immediate data from accumulator with borrow). [A] ← [A] – data – [CS].
- INR r (Increment register content) [r] ← [r] +1.
- INR M. (Increment memory content) [[H-L]] ← [[H-L]] + 1.
- DCR r. (Decrement register content). [r] ← [r] – 1.
- DCR M. (Decrement memory content) [[H-L]] ← [[H-L]] – 1.
- INX rp. (Increment register pair) [rp] ← [rp] – 1.
- DCX rp (Decrement register pair) [rp] ← [rp] -1.

11.3.3 LOGICAL GROUP

All the instructions related to logical operations like AND, OR, compare, etc. in data are grouped under logic group instructions.

The example of this category instructions are:

- ANA r. (AND register with accumulator) [A] ← [A] ^ [r].
- ANA M. (AND memory with accumulator). [A] ← [A] ^ [[H-L]].
- ANI data. (AND immediate data with accumulator) [A] ← [A] ^ data.
- ORA r. (OR register with accumulator) [A] ← [A] v [r].
- ORA M. (OR memory with accumulator) [A] ← [A] v [[H-L]]
- ORI data. (OR immediate data with accumulator) [A] ← [A] v data.
- XRA r. (EXCLUSIVE – OR register with accumulator) [A] ← [A] v [r]
- XRA M. (EXCLUSIVE-OR memory with accumulator) [A] ← [A] v [[H-L]]
- XRI data. (EXCLUSIVE-OR immediate data with accumulator) [A] ← [A]
- CMA. (Complement the accumulator) [A] ← [A]
- CMC. (Complement the carry status) [CS] ← [CS]
- STC. (Set carry status) [CS] ← 1.
- CMP r. (Compare register with accumulator) [A] – [r]
- CMP M. (Compare memory with accumulator) [A] – [[H-L]]
- CPI data. (Compare immediate data with accumulator) [A] – data.

  The 2nd byte of the instruction is data, and it is subtracted from the content of the accumulator. The status flags are set according to the result of subtraction. But the result is discarded. The content of the accumulator remains unchanged.

- RLC (Rotate accumulator left) [An+1] ← [An], [A0] ← [A7],[CS] ← [A7].

  The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected.

- RRC. (Rotate accumulator right) [A7] ← [A0], [CS] ← [A0], [An] ← [An+1].

The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected.

- RAL. (Rotate accumulator left through carry) [An+1] ← [An], [CS] ← [A7], [A0] ← [CS].
- RAR. (Rotate accumulator right through carry) [An] ← [An+1], [CS] ← [A0], [A7] ← [CS]

11.3.4 BRANCH CONTROL GROUP

Normally the order of execution of the instructions of the program is sequential. Often we encounter a situation in real world programming where we want the control of the program jump to some location as specified by the instruction. For which, a condition is tested. If the condition holds true, the instruction at the location specified in the instruction is executed, else the instruction in the next sequential location is executed. This is known as a conditional jump. At times, we encounter a situation where we want to execute an instruction which is not located at the next sequential location at any cost and we don't want any condition to hold true for that. This is an unconditional jump. And such situation arises when the CPU is executing a program and an urgent operating system related subroutine needs to be executed, or an interrupt occurs. Branch control group contains such instructions which are used for conditional and unconditional jump, subroutine call and return, and restart purposes.

This group includes the instructions. Examples are:

- JMP addr (label)- (Unconditional jump: jump to the instruction specified by the address). [PC] ←Label.
- Conditional Jump addr (label)- After the execution of the conditional jump instruction the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled. The program proceeds further in the normal sequence if the specified condition is not fulfilled. If the condition is true and program jumps to the specified label, the execution of a conditional jump takes 3 machine cycles: 10 states. If condition is not true, only 2 machine cycles; 7 states are required for the execution of the instruction.
- JZ addr (label)- (Jump if the result is zero)
- JNZ addr (label)- (Jump if the result is not zero)
- JC addr (label)- (Jump if there is a carry)
- JNC addr (label)- (Jump if there is no carry)
- JP addr (label)- (Jump if the result is plus)
- JM addr (label)- (Jump if the result is minus)

- JPE addr (label)- (Jump if even parity)
- JPO addr (label)- (Jump if odd parity)
- CALL addr (label)- (Unconditional CALL: call the subroutine identified by the operand)
- CALL instruction is used to call a subroutine- Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by two to indicate the new stack top. Then the program jumps to subroutine starting at address specified by the label.
- RET (Return from subroutine)
- RST n (Restart)- Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location.

11.3.5 I/O AND MACHINE CONTROL GROUP

This group includes the instructions for input/output ports, stack and machine control.

The example of this category instructions are:

- IN port-address. (Input to accumulator from I/O port) [A] ← [Port]
- OUT port-address (Output from accumulator to I/O port) [Port] ← [A]
- PUSH rp (Push the content of register pair to stack)
- PUSH PSW (PUSH Processor Status Word)
- POP rp (Pop the content of register pair, which was saved, from the stack)
- POP PSW (Pop Processor Status Word)
- HLT (Halt)
- XTHL (Exchange stack-top with H-L)
- SPHL (Move the contents of H-L pair to stack pointer)
- EI (Enable Interrupts)
- DI (Disable Interrupts)
- SIM (Set Interrupt Masks)
- RIM (Read Interrupt Masks)
- NOP (No Operation)

Check Your Progress

1. The ALU and CU were connected to each other and the combined unit was known as ……..
2. Depending on the …….. of the data bus of a microprocessor, it can be categorized into 8-bit, 16-bit, 32-bit or 64-bit microprocessor.
3. 8085 operates with …….. MHz clock.
4. ALE stands for ……
5. Whenever an Instruction from the memory is fetched, the instruction is placed inside a 8-bit register, known as ……..
6. The 8085 microprocessor has 8-bit data bus and …… bit address bus.

11.4 MICROCONTROLLER

Microcontroller is a small computer fabricated on a single IC which consists of a processor, RAM, ROM and I/O pins. The microcontroller is used in an embedded system and it is a key component. A microcontroller is specially designed for a specific task which runs one specific program. The program is stored on ROM which cannot be changed. A microcontroller has an input device and often has a LCD display for output. For example, the microcontroller inside a tv system takes input from remote control and displays output in tv screen.

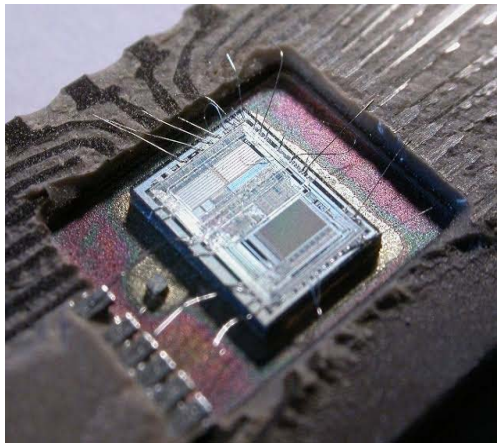


Figure-75: A microcontroller chip.

Difference between microprocessor and microcontroller :

1. Microprocessors are bulky due to the presence of external peripheral devices. While microcontrollers are compact with only RAM, ROM and EEPROM are embedded on a single chip. Microprocessors are more expensive than microcontrollers.
2. Microprocessors work faster than microcontrollers. A typical processing speed of a microprocessor above 1GHz. The processing speed of microcontrollers is generally upto 60 MHz.

3. Microprocessors consume more power as compared to microcontrollers. Microcontrollers are power efficient and consume less power.

4. Microprocessors are generally designed for personal computers while microcontrollers are generally designed for embedded systems. Microprocessors are generally used in software development, game development, websites and other heavy computation intensive applications. While, microcontrollers are used in embedded systems such as washing machines, microwave ovens, cameras and speedometers.

5. Microprocessors are based on von Neumann architecture where code memory and data memory are stored at the same memory. While, microcontrollers are based on Harvard architecture where code and data memory are stored separately.

11.4 SUMMARY

1. A microprocessor (μp) is an electronic device that is used by the computer to its processing.

2. A microprocessor is a programmable device which can perform different sets of operations on the data it receives as an input depending on the sequence of instructions supplied in the given program.

3. The processing power of a microprocessor depends on its instruction set.

4. The width of the data bus signifies the number of data bits a microprocessor can process simultaneously.

5. Intel launched its first 8-bit microprocessor in 1972 and names it Intel 8008.

6. Intel 8985 is an 8-bit general purpose microprocessor capable of addressing 64K memory.

7. 8085 has 40 pins and runs on +5 V power supply.

8. In 8085, the 8-bit data bus was multiplexed with the lower part of 16-bit address bus so that the number of pins are limited to 40.

9. It has a 16-bit address bus, hence it is capable of addressing 2^{16}= 64 KB memory.

10. The address bus has 8 signal lines A8 – A15 which are unidirectional.

11. There are 4 main control and status signals.

12. An Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced.

11.5 ANSWERS TO CHECK YOUR PROGRESS

1. CPU

2. Width
3. 3
4. Address Latch Enable
5. Instruction Register(IR)
6. 16

11.6 TERMINAL QUESTIONS

1. What is a microprocessor? What is a difference between a microprocessor and CPU?
2. Draw and explain the pin-diagram of 8085 microprocessor.
3. Explain the classification of the instructions of 8085 microprocessor.
4. What is conditional and unconditional jump.
5. What is an interrupt? What is the order of priority of these interrupts?
6. Explain the MOV and LDA instruction with the help of diagram.
7. Explain JMP instruction.
8. What is an interrupt?
9. What is the different between a maskable and non-maskable interrupt? What is the priority of various interrupts?

REFERENCES

(s.j.). Onttrek Dec. 15, 2015 uit http://www.cs.umd.edu

(s.j.).Onttrek Dec. 15, 2015 uit http://www.technicalsymposium.com/MICROPROCESSOR_Instruction_Set_of_Intel_8085.doc

(s.j.). Onttrek Dec. 15, 2015 uit https://en.wikipedia.org/wiki/Astable

(s.j.). Onttrek Dec. 15, 2015 uit http://mcqquestion.blogspot.in/2012/08/computer-system-architecture_6.html

Anand, A. (2012, Nov.). *Memory Organization*. Onttrek Dec. 15, 2015 uit http://www.slideshare.net/rashcommuz/memory-organization-16934580

Arivazhagan, S. S. (2012). *Digital Circuits and Design.* S.Chand & Company.

AspenCore. (2016). *Bipolar Transistor*. Onttrek Oct. 29, 2017 uit http://www.electronics-tutorials.ws/transistor/tran_1.html

Astable Multivibrator. (s.j.). Onttrek Dec. 15, 2015 uit http://evalidate.in/lab1/pages/IC555/AstableMultivibrator/AstableMultivibrator_I.html

Basheer, N. (2011). *ZENER DIODE*. Onttrek Oct. 29, 2017 uit https://mediatoget.blogspot.in/2011/10/zener-diode.html available under Creative Commons Attribution 3.0 Unported License.

Belurkar, V. (2012, Dec. 07). *Lithium-ion Battery*. Onttrek Dec. 15, 2015 uit http://simpleelectronic-project.blogspot.com/

Burke, T. (2006). *Resistor Codes*. Onttrek Oct. 29, 2017 uit https://commons.wikimedia.org/wiki/File:Resistor-Codes.svg available under the Creative Commons Attribution-Share Alike 2.5 Generic, 2.0 Generic and 1.0 Generic license.

Community, E. e. (2014). *Diode Charachteristics*. Onttrek Oct. 29, 2017 uit http://engineering.electrical-equipment.org/electrical-distribution/

diode- characteristics.html available under Creative Commons By Attribution License.

Dutt, S. (2012). *An introduction to Microprocess Architecture using Intel 8085 as a classical processor*. Onttrek Dec. 15, 2015 uit Slideshare: http://www.slideshare.net/sdutt36/8085-14257924

Elcap. (2012). *Ceramic disc capacitor*. Onttrek Oct. 29, 2017 uit https://commons.wikimedia.org/wiki/File:Ceramic_disc_capacitor.png available under Creative Commons CC0 1.0 Universal Public Domain Dedication.

ETHW. (2017). *Integrated Circuits*. Onttrek Oct. 30, 2017 uit http://ethw.org/Integrated_Circuits?gclid=EAIaIQobChMIjaXJ36KX1wIV0hFoCh0IPwUAEAAYAyAAEgLj3PD_BwE available under Creative Commons Attribution-ShareAlike License.

Find a Tech Definition. (s.j.). Onttrek Dec. 15, 2015 uit http://whatis.techtarget.com/

Gao, Y. (s.j.). *Review of Flip Flops*. Onttrek Dec. 15, 2015 uit http://www.readbag.com/maxwell-ict-griffith-au-yg-teaching-dns-dns-module3-p1

Halfwave rectifier. (2005). Onttrek Oct. 29, 2017 uit https://simple.wikipedia.org/wiki/File:Halfwave.rectifier.en.png available under Public Domain License.

Hamacher. (2011). *Computer Organization.* Tata McGraw Hill.

Instruction Seti of Intel 8085. (s.j.). Onttrek Dec. 15, 2015 uit http://www.daenotes.com/electronics/digital-electronics/instruction-set-intel-8085

Learning, L. (2012). *The Central Processing Unit*. Onttrek Oct. 30, 2017 uit Introductionj to Computer Applications and Concepts: https://courses.lumenlearning.com/zeliite115/chapter/reading-the-central-processing-unit/ available under Creativr Commons Attribution-ShareAlike License.

LibreTexts. (2016). *Bipolar Junction Transistor*. Onttrek Oct. 29, 2017 uit https://eng.libretexts.org/Core/Materials_Science/Materials_and_Devices/Bipolar

_Junction_Transistor available under Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License.

LOGIC CIRCUIT AND SWITCHING THEORY. (2010, Feb. 05). Onttrek Dec. 15,

2015 uit Sequential Logic Basics: http://logicckt.blogspot. in/2010/02/week-5.html

Mano, M. M. (2008). *Computer System Architecture.* Pearson. Mano, M. M. (2008). *Digital Logic and Computer Design.* Pearson.

Multivibrator. (2014, Jan. 11). Onttrek Dec. 15, 2015 uit Slideshare: http://www.slideshare.net/nakulrtm/multivibrators-including-monostable-astable-and-bistable

Omegatron. (2006). *Diode symbol*. Onttrek Oct. 29, 2017 uit Wikibooks: https://commons.wikimedia.org/wiki/File:Diode_symbol.svg available under Creative Commons Attribution-Share Alike 3.0 Unported, 2.5 Generic, 2.0 Generic and 1.0 Generic license.

Omegatron. (2015). *Zener Diode Figure*. Onttrek Oct. 29, 2017 uit https://commons.wikimedia.org/wiki/File:Zener_diode_symbol.svg available under Creative Commons Attribution-Share Alike 3.0 Unported, 2.5 Generic, 2.0 Generic and 1.0 Generic license.

Peripitus. (2007). *Toroidal inductor*. Onttrek Oct. 29, 2017 uit https://commons.wikimedia.org/wiki/File:Toroidal_inductor.jpg available under Creative Commons Attribution-Share Alike 4.0 International, 3.0 Unported, 2.5 Generic, 2.0 Generic and 1.0 Generic license.

Powley, R. (2011). *Introduction to Computers*. Onttrek Oct. 30, 2017 uit http://doer.col.org/bitstream/123456789/8192/1/2011_VUSSC_Intro-Computers.pdf avaialble under CC-BY-SA license.

Storr, W. (2015, Dec.). *Basic Electronics Tutorials*. Onttrek Dec. 10, 2015 uit http://www.electronics-tutorials.ws/

Storr, W. (2015, Dec. 13). *Multivibrators*. Onttrek Dec. 15, 2015 uit Electronics Tutorials: http://www.electronics-tutorials.ws/sequential/seq_3.html

TES. (2017). *Explain the machine instruction cycle*. Onttrek Oct. 31, 2017 uit https://compsci2014.wikispaces.com/2.1.4+Explain+the+machine+instruction+cycle available under Creative Commons Attribution License.

Types of capacitor. (2014). Onttrek Oct. 29, 2017 uit https:// commons.wikimedia.org/wiki/File:Types_of_capacitor.svg available under Creative Commons CC0 1.0 Universal Public Domain Dedication.

What is a computer? (2017). Onttrek Oct. 30, 2017 uit https://en. wikiversity.org/wiki/What_is_a_computer%3F available under Creative Commons Attribution-ShareAlike License.

Wikibooks. (2017, Aug. 16). *Electronics Handbook/Components/ Diodes/Photo*. Onttrek Oct. 28, 2017 uit

https://en.wikibooks. org/wiki/Electronics_Handbook/Components/Diodes/Photo available under the Creative Commons Attribution-ShareAlike License.

Wikibooks. (2017). *Transistor Basics*. Onttrek Oct. 29, 2017 uit https://en.wikibooks.org/wiki/Digital_Circuits/Transistor_Basics available under Creative Commons Attribution-ShareAlike License.

Wikipedia. (2017). *Diode*. Onttrek Oct. 29, 2017 uit http://sciencewise.info/resource/Diode/Diode_by_Wikipedia available under Creative Commons Attribution-ShareAlike License.

Wikipedia. (2017). *Optical isolator*. Onttrek Oct. 29, 2017 uit https://en.wikipedia.org/wiki/Optical_isolator avaialble under Creative Commons Attribution-ShareAlike License.

Wikispaces. (s.j.). *Computer Architecture*. Onttrek Oct. 31, 2017 uit https://isscs.wikispaces.com/3.2+-+Computer+Architecture available under Creative Commons Attriution license.

Wikiversity. (2017). *Field-Effect Transistors*. Onttrek Oct. 29, 2017 uit https://en.wikiversity.org/wiki/Fundamental_Physics/Electronics/Field-Effect_Transistors available under Creative Commons Attribution-ShareAlike License.

Yewale, J. (2011). *Diode Clamping Circuit*. Onttrek Oct. 29, 2017 uit http://todayscircuits.blogspot.com/2011/06/diode-clamping-circuits.html available under Creative Commons Attribution-ShareAlike 2.5 India License.

Yewale, J. (2011). *Diode Clippers – A study of various Clipping Circuits*. Onttrek Oct. 29, 2017 uit http://todayscircuits.blogspot.com: http://todayscircuits.blogspot.com/2011/06/diode-clippers-overview-of-clipping.html available under Creative Commons Attribution-ShareAlike 2.5 India License.

Zener Diode. (2009). Onttrek Oct. 29, 2017 uit https://en.wikibooks. org/wiki/Semiconductors/Zener_Diode available under Creative Commons Attribution-ShareAlike License.

(n.d.). Retrieved Dec. 15, 2015, from http://www.cs.umd.edu

(n.d.). Retrieved Dec. 15, 2015, from http://www.technicalsymposium. com/MICROPROCESSOR_Instruction_Set_of_Intel_8085.doc

(n.d.). Retrieved Dec. 15, 2015, from https://en.wikipedia.org/wiki/Astable

(n.d.). Retrieved Dec. 15, 2015, from http://mcqquestion.blogspot.in/ 2012/08/computer-system-architecture_6.html

Anand, A. (2012, Nov.). *Memory Organization*. Retrieved Dec. 15, 2015, from http://www.slideshare.net/rashcommuz/memory-organization-16934580

Arivazhagan, S. S. (2012). *Digital Circuits and Design.* S.Chand & Company .

Astable Multivibrator. (n.d.). Retrieved Dec. 15, 2015, from http:// evalidate.in/lab1/pages/IC555/AstableMultivibrator/AstableMultivibrator_I.html

Belurkar, V. (2012, Dec. 07). *Lithium-ion Battery*. Retrieved Dec. 15, 2015, from http://simpleelectronic-project.blogspot.com/

Dutt, S. (2012). *An introduction to Microprocess Architecture using Intel 8085 as a classical processor*. Retrieved Dec. 15, 2015, from Slideshare: http://www.slideshare.net/sdutt36/8085-14257924

Find a Tech Definition. (n.d.). Retrieved Dec. 15, 2015, from http://whatis.techtarget.com/

Gao, Y. (n.d.). *Review of Flip Flops*. Retrieved Dec. 15, 2015, from http://www.readbag.com/maxwell-ict-griffith-au-yg-teaching-dns-dns-module3- p1

Hamacher. (2011). *Computer Organization.* Tata McGraw Hill.

Instruction Seti of Intel 8085. (n.d.). Retrieved Dec. 15, 2015, from http://www.daenotes.com/electronics/digital-electronics/instruction-set-intel-8085

LOGIC CIRCUIT AND SWITCHING THEORY. (2010, Feb. 05). Retrieved Dec.

15, 2015, from Sequential Logic Basics: http://logicckt.blogspot.in/2010/02/week-5.html

Mano, M. M. (2008). *Computer System Architecture.* Pearson. Mano, M. M. (2008). *Digital Logic and Computer Design.* Pearson.

Multivibrator. (2014, Jan. 11). Retrieved Dec. 15, 2015, from Slideshare: http://www.slideshare.net/nakulrtm/multivibrators-including-monostable-astable-and-bistable

Storr, W. (2015, Dec.). *Basic Electronics Tutorials*. Retrieved Dec. 10, 2015, from http://www.electronics-tutorials.ws/

Storr, W. (2015, Dec. 13). *Multivibrators*. Retrieved Dec. 15, 2015, from Electronics Tutorials: http://www.electronics-tutorials.ws/sequential/seq_ 3.html

Wikibooks. (2017, Aug. 16). *Electronics Handbook/Components/ Diodes/Photo*. Retrieved Oct. 28, 2017, from https://en.wikibooks. org/wiki/Electronics_Handbook/Components/Diodes/Photo available under the Creative Commons Attribution-ShareAlike License.

ROUGH WORK

ROUGH WORK