DIGITAL LOGIC DESIGN

BY

P.GOPALA KRISHNA

M.Sc., M.Tech., (Ph.D)

Associate Professor

Department of IT

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

UNIT – I

BINARY SYSTEMS

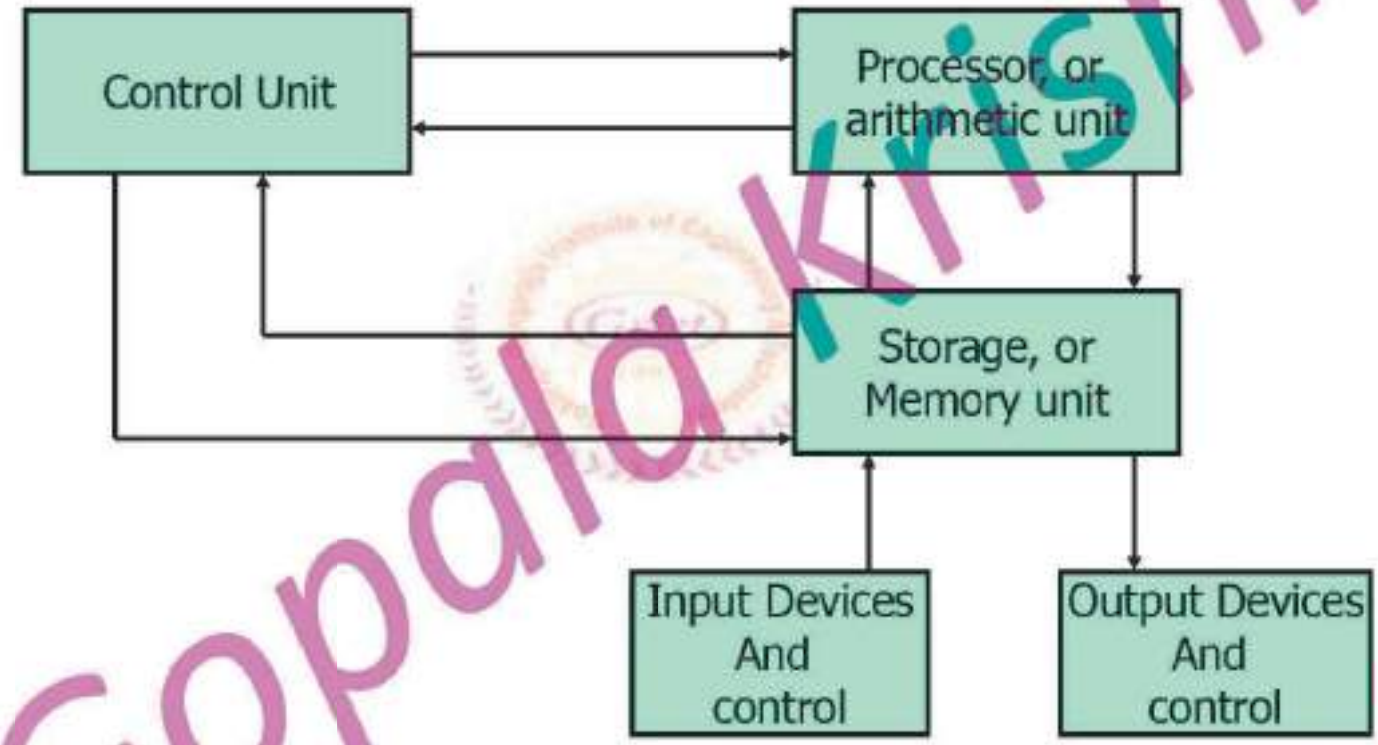
DIGITAL COMPUTERS

- Digital computers have made possible many scientific, industrial, and commercial advances that would have been unattainable otherwise.
- Computers are used in scientific calculations, commercial and business data processing, air traffic control, space guidance, the education field, and many other areas.
- The most striking property of a digital computer is it's generality.
- It can follow a sequence of instructions, called a program, that operates on given data.

DIGITAL SYSTEMS

- The general purpose digital computer is the best example of a digital system.
- Other examples include telephone switching exchanges, digital volt meters, digital counters, electronic calculators and digital displays.
- Characteristic of a digital system is its manipulation of discrete elements of information.
- Such discrete elements may be electric impulses, the decimal digits, the letters of an alphabet, arithmetic operations or any other set of meaningful operations.
- Discrete elements of information are represented in a digital system by physical quantities called signals.

Block Diagram of a digital computer



- The Memory Unit stores programs as well as input, output, and intermediate data.
- The processor unit performs arithmetic and other data processing tasks as specified by a program.
- The control unit supervises the instruction, one by one, from the program that is stored in memory. For each instruction, the control unit informs the processor to execute the operation specified by the instruction.
- The program and data prepared by the user are transferred into the memory unit by means of an input device such as a key board.
- An out device, such as a printer, receives the result of computations and the printed results are presented to the user.
- Digital computer manipulates discrete elements of information and that these elements are represented in the binary form.

BINARY NUMBERS

- In general, a number with a decimal point is represented by a series of coefficients as follows:

$$a_5a_4a_3a_2a_1a_0 \cdot a_{-1}a_{-2}a_{-3}$$

- The a_j coefficients are one of the digits of the number system and the subscript value j gives the place value and hence the power by which the coefficients must be multiplied.
- The decimal number system is said to be of base, 10 because it uses ten digits, and the coefficients are multiplied by powers of 10.
- The coefficients of binary number system have two possible values: 0 and 1. Each coefficient a_j is multiplied by 2^j.

- In general, a number expressed in base –r system has coefficients multiplied by powers of r:

$$a_n r^n + a_{n-1} r^{n-1} + \ldots\ldots + a_1 r + a_0 + a_{-1} r^{-1} + a_{-2} r^{-2} + \ldots\ldots\ldots + a_{-m} r^{-m}$$

- The coefficients a_j range in value from 0 to r-1.
- For Example,

$$(465.27)_8 = 4x8^2 + 6x8^1 + 5x8^0 + 2x8^{-1} + 7x8^{-2}$$

NUMBERS WITH DIFFERENT BASES

Decimal	Binary	Octal	Hexadecimal	BCD
0	0	0	0	0000
1	1	1	1	0001
2	10	2	2	0010
3	11	3	3	0011
4	100	4	4	0100
5	101	5	5	0101
6	110	6	6	0110
7	111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

Arithmetic operations in base r

- Arithmetic operations with numbers in base r, follow the same rules as for decimal numbers.
- Examples of addition, subtraction, and multiplication of two binary numbers are shown below:

Binary Addition

- Binary addition is very simple.
- This is best shown in an example of adding two binary numbers...

```
   1  1  1  1  1  1 <------------ carries
      1  1  1  1  0  1
+        1  0  1  1  1
-----------------------
   1  0  1  0  1  0  0
```

Binary Subtraction

- We can also perform subtraction (with borrows in place of carries).
- Let's subtract (10111)2 from (1001101)2…

```
        1         10          ← borrows
      0 10 10  0  0 10

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

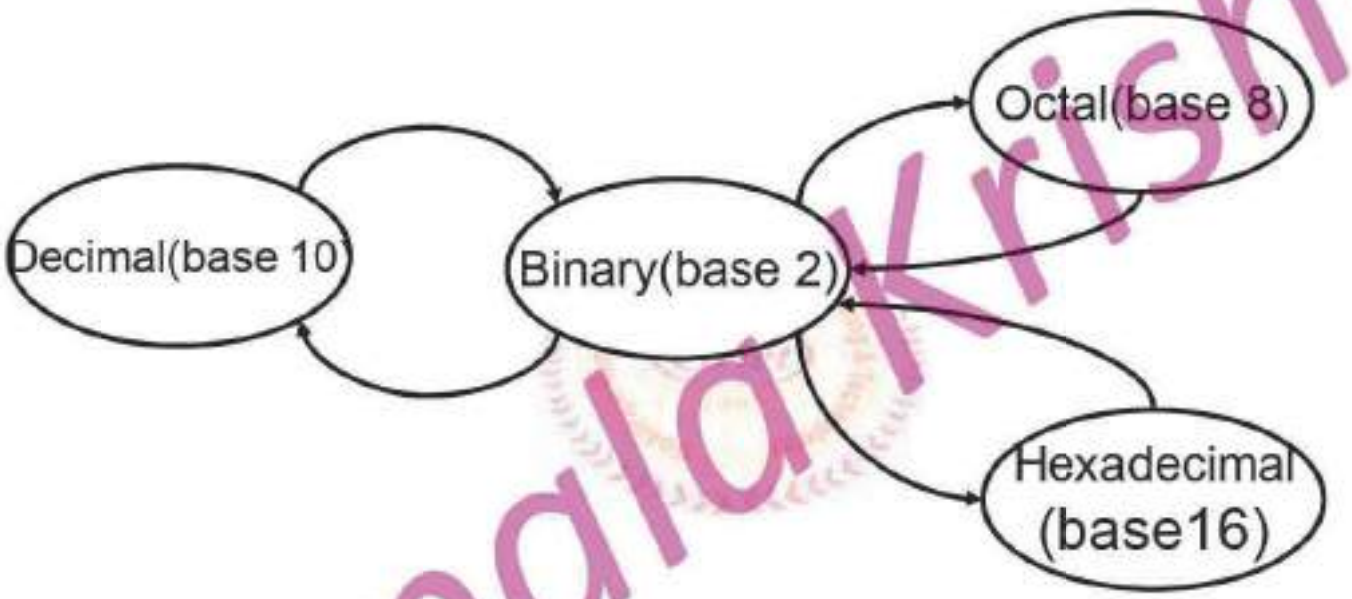
(In the original, the digits 1 0 0 1 1 0 of the top number and the first 10 and the second 0 in the borrow row are struck through.)

Binary Multiplication

- Binary multiplication is much the same as decimal multiplication, except that the multiplication operations are much simpler...

```
                1   0   1   1   1
X               1   0   1   0
-----------------------------------
                0   0   0   0   0
            1   0   1   1   1
        0   0   0   0   0
    1   0   1   1   1
-----------------------------------
    1   1   1   0   0   1   1   0
```

NUMBER BASE CONVERSIONS



Convert an Integer from Decimal to Another Base

- For each digit position,
  - Divide decimal number by the base (e.g. 2)
  - The remainder is the lowest-order digit
  - Repeat first two steps until no divisor remains
  - Example for $(13)_{10}$:

	Integer Quotient		Remainder	Coefficient
13/2 =	6	+	1	$a_0 = 1$
6/2 =	3	+	0	$a_1 = 0$
3/2 =	1	+	1	$a_2 = 1$
1/2 =	0	+	1	$a_3 = 1$

Answer $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

Convert an Fraction from Decimal to Another Base

- For each digit position:
  - Multiply decimal number by the base (e.g. 2)
  - The integer is the highest-order digit
  - Repeat first two steps until fraction becomes zero.
  - Example for $(0.625)_{10}$:

	Integer		Fraction	Coefficient
0.625 x 2 =	1	+	0.25	$a_{-1} = 1$
0.250 x 2 =	0	+	0.50	$a_{-2} = 0$
0.500 x 2 =	1	+	0	$a_{-3} = 1$

Answer $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

Conversion from Decimal to any base r

- The conversion from decimal integers to any base-r system is similar to the above examples, except that division is done by r instead of 2.
- The conversion of decimal fraction to any base-r system is similar except multiplication is done by r instead of 2, and the coefficients found from the integers may range in value from 0 to r-1 instead of 0 and 1.

Convert a Fraction from Decimal to Octal

For each digit position:

- Multiply decimal number by the base (e.g. 8)
- The integer is the highest-order digit
- Repeat first two steps until fraction becomes zero.

Example for $(0.3125)_{10}$:

	Integer		Fraction	Coefficient
$0.3125 \times 8 =$	2	+	5	$a_{-1} = 2$
$0.5000 \times 8 =$	4	+	0	$a_{-2} = 4$

Answer $(0.3125)_{10} = (0.24)_8$

Octal and Hexadecimal Numbers

- The conversion from and to binary, octal and hexadecimal plays an important role in digital computers. Since $2^3=8$, $2^4=16$, each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits.
- The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding to the left and to the right. The corresponding octal digit is then assigned to each group.
- The conversion binary to hexadecimal is similar, except that the binary number is divided into groups of four digits.

Converting between base 16 and base 2

$3A9F_{16} = \underline{0011}\ \underline{1010}\ \underline{1001}\ \underline{1111}_2$

3 A 9 F

Determine 4-bit value for each hex digit

- Note that there are $2^4 = 16$ different values of four bits
- Easier to read and write in hexadecimal.
- Representations are equivalent!

Converting between base16 and base 8

$3A9F_{16} = \underline{0011}\ \underline{1010}\ \underline{1001}\ \underline{1111}_2$

3 A 9 F

$35237_8 = \underline{011}\ \underline{101}\ \underline{010}\ \underline{011}\ \underline{111}_2$

3 5 2 3 7

- Convert base 16 to base 2
- Regroup bits into groups of three starting from right
- Ignore leading zeros
- Each group of three bits forms an octal digit.

COMPLEMENTS

- Complements are used in digital computers for simplifying the subtraction operation and for logical manipulations.
- There are two types of complements for each base-r system:
  - The r's complement
  - The (r-1)'s complement
- When the value of the base is substituted, the two types receive the names 2's and 1's complement for binary numbers, or 10's and 9's complement for decimal numbers.

The r's Complement

- Given a positive number N in base r with an integer part of n digits, the r's complement of N is defined as
  - r^n-N for N≠0 and
  - 0 for N=0.

Numerical Example:

- The 10's complement of $(52520)_{10}$ is
  10^5-52520=47480
- The 10's complement of $(0.3267)_{10}$ is
  1-0.3267=0.6733.
- The 10's complement of $(25.639)_{10}$ is
  10^2-25.639=74.361
- The 2's complement of $(101100)_2$ is
  $(2^6)_{10}$- $(101100)_2$=$(1000000-101100)_2$=010100
- The 2's complement of (0.0110)2 is (1-0.0110)=0.1010

The (r-1)'s Complement

- Given a positive number N in base r with an integer part of n digits and a fraction part of m digits, the (r-1)'s complement of N is defined as $r^n - r^{-m} - N$.

Numerical Example:

- The 9's complement of $(52520)_{10}$ is
  $(10^5-1-52520)=99999-52520=47479$
- The 9's complement of $(0.3267)_{10}$ is
  $(1-10^{-4}-0.3267)=(0.9999-0.3267)=0.6732.$
- The 9's complement of $(25.639)_{10}$ is
  $(10^2-10^{-3}-25.639)=74.360$
- The 1's complement of $(101100)_2$ is
  $(2^6-1)_{10}-(101100)_2=(111111-101100)_2=010011$
- The 1's complement of (0.0110)2 is
  $(1-2^{-4})_{10}-(0.0110)=(0.1111-0.0110)=0.1001$

- From the examples, we see that the 9's complement of a decimal number is formed simply by subtracting every digit from 9.

- The 1's complement of a binary number is even simpler to form: the 1's are changed to 0's and the 0's to 1's.

Subtraction with r's complements

- The borrowing method of subtraction is found to be less efficient than the method that uses complements and addition as stated:
- The subtraction of two positive numbers (M-N), both of base r, may be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N.
2. Inspect the result obtained in step1 for an end carry:
   - If an end carry occurs, discard it.
   - If an end carry does not occur, take the r's complement of the number obtained in step1 and place negative sign in front.

Example

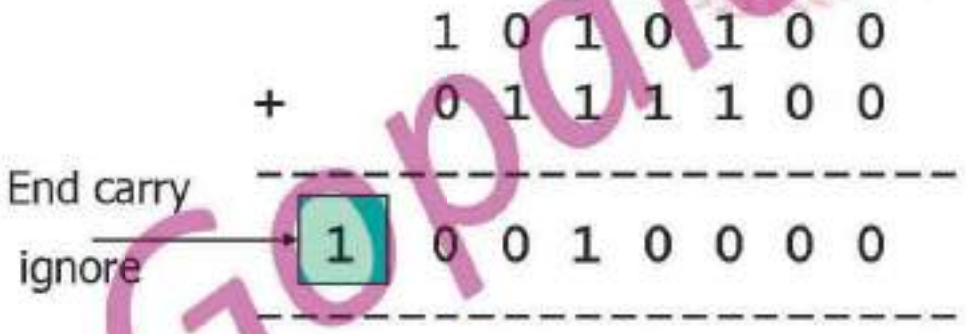
- Using 10's complement, subtract 72532-3250.
  - M= 72532
  - N= 03250
  - 10's complement of N=96750.

```
              7 2 5 3 2
   +          9 6 7 5 0
          -------------
End carry
ignore ->  1  6 9 2 8 2
          -------------
```

  - Answer: 69282

Example

- Using 2's complement, subtract 1010100-1000100.
  - M= 1010100
  - N= 1000100
  - 2's complement of N=0111100.



- Answer: 10000

Example

- Using 2's complement, subtract 1000100-1010100.
  - M= 1000100
  - N= 1010100
  - 2's complement of N=0101100.

```
        1 0 0 0 1 0 0
+       0 1 0 1 1 0 0
-----------------------
      1 1 1 0 0 0 0
-----------------------
```

- Answer: -10000= (2's complement of 1110000)

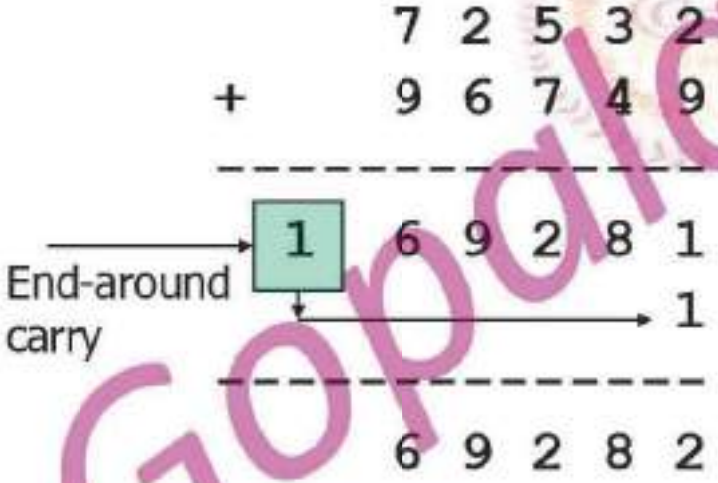
Subtraction with (r-1)'s complements

- The procedure for subtraction with the (r-1)'s complement is exactly the same as the one used with the r's complement except for one variation, called end-around carry.
- The subtraction of two positive numbers (M-N), both of base r, may be done as follows:

1. Add the minuend M to the (r-1)'s complement of the subtrahend N.
2. Inspect the result obtained in step1 for an end carry:
   - If an end carry occurs, add 1 to the least significant digit (end-around carry).
   - If an end carry does not occur, take the (r-1)'s complement of the number obtained in step1 and place negative sign in front.

Example

- Using 9's complement, subtract 72532-3250.
  - M= 72532
  - N= 03250
  - 9's complement of N=96749.



- Answer: 69282

Example

- Using 9's complement, subtract 3250-72532.
  - M= 03250
  - N= 72532
  - 9's complement of N=27467.

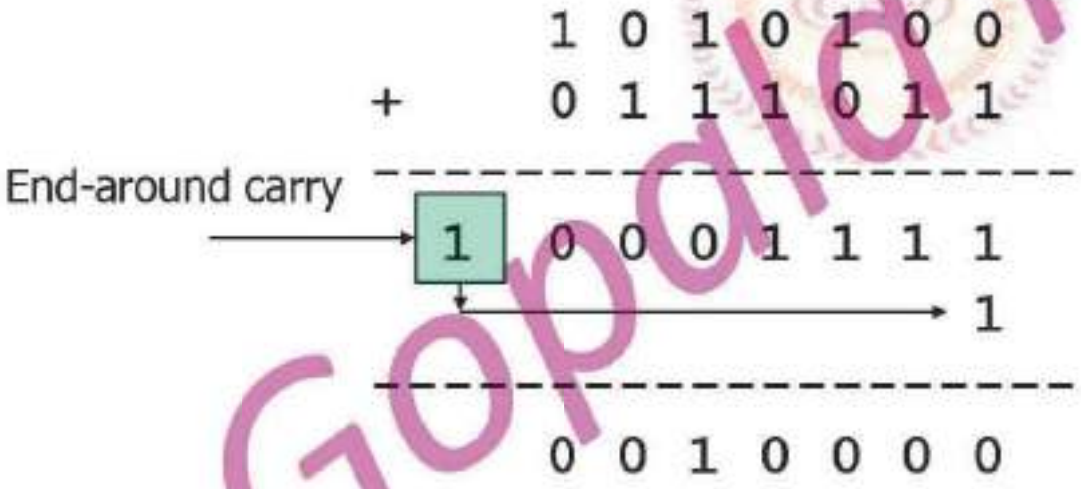
```
          0  3  2  5  0
    +     2  7  4  6  7
    ----------------------
          3  0  7  1  7
    ----------------------
```

No carry →

  - Answer: -69282= - (9's complement of 30717)

Example

- Using 1's complement, subtract 1010100-1000100.
  - M= 1010100
  - N= 1000100
  - 1's complement of N=0111011.



- Answer: 10000

Example

- Using 1's complement, subtract 1000100-1010100.
  - M= 1000100
  - N= 1010100
  - 1's complement of N=0101011

```
          1 0 0 0 1 0 0
    +     0 1 0 1 0 1 1
    ---------------------
No carry  1 1 0 1 1 1 1
    ---------------------
```

  - Answer: -10000= (1's complement of 1101111)

Binary Codes

- A bit by definition, is a binary digit. When used in conjunction with a binary code, it is better to think of it as denoting a binary quantity equal to 0 or 1.
- To represent a group of 2^n distinct elements in a binary code requires a minimum of n bits.
- For example, the ten decimal digits can be coded with ten bits, and each decimal digit assigned a bit combination of nine 0's and a 1. In this particular binary code, the digit 6 is assigned the bit combination 000100000.

Binary codes for Decimal Digits

decimal	B C D	Excess3	84-2-1	2421	5034210 Biquinary
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

Error-Detection Codes

- Binary information, be it pulse modulated signals or digital computer input or output, may be transmitted through some form of communication medium such as wires or radio waves.
- Any external noise introduced into physical medium changes bit values from 0 to 1 or vice versa.
- An error detection code can be used to detect errors during transmission. The detected error cannot be corrected, but it's presence can be indicated.

Parity bit generation for error detection

- A parity bit is an extra bit included with a message to make the total number of 1's either odd or even.
- In the sending end, the message is applied to a parity generator where the required P bit is generated. The message, including P bit is transferred to the destination.
- In the receiving end, all the incoming bits are applied to parity checker to check the proper parity adopted.

Parity Bit Generation

Message	P (Odd)	P (Even)
0000	1	0
0001	0	1
0010	0	1
0011	1	0
0100	0	1
0101	1	0
0110	1	0
0111	0	1

Message	P (Odd)	P (Even)
1000	0	1
1001	1	0
1010	1	0
1011	0	1
1100	1	0
1101	0	1
1110	0	1
1111	1	0

The Reflected or Gray Code

- The advantage of the reflected code over the normal binary code is that a number in the reflected code changes by only one bit as it proceeds from one number to the next.

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

Binary Number Representations

- A binary number can be represented in one of the following ways:
  - Signed magnitude representation
  - Signed 1's complement representation
  - Signed 2's complement representation
- Representation of positive number is same in all the three cases, however representation of negative number is different in all the three cases.

- In signed magnitude representation, for a negative number sign bit is 1 and magnitude bits are unchanged.
- In signed 1's complement representation, sign bit is 1, and magnitude bits are in 1's complement form.
- In signed 2's complement representation, sign bit is 1, and magnitude bits are in 2's complement representation.
- For example, +5 and -5 in 8-bit form:

	+5	-5
Signed magnitude	0 000 0101	1 000 0101
Signed 1's complement	0 000 0101	1 111 1010
Signed 2's complement	0 000 0101	1 111 1011

Binary Storage and Registers

- The binary information in a digital computer must have a physical existence in some information storage medium for storing individual bits.
- A binary cell is a device that possesses two stable states and is capable of storing one bit of information.
- The out put of a cell is a physical quantity that distinguishes between the two states. The information stored in a cell is 1 when it is in one stable state and 0 when in the other stable state.

Registers

- A register is a group of binary cells. A register with n cells can store any discrete quantity of information that contains n bits.
- The state of a register is an n tuple number of 1's and 0's. with each bit designating the state of one cell in the register.
- The contents of a register is a function of the interpretation given to the information stored in it.

- For example, a 16-bit register with the following content:

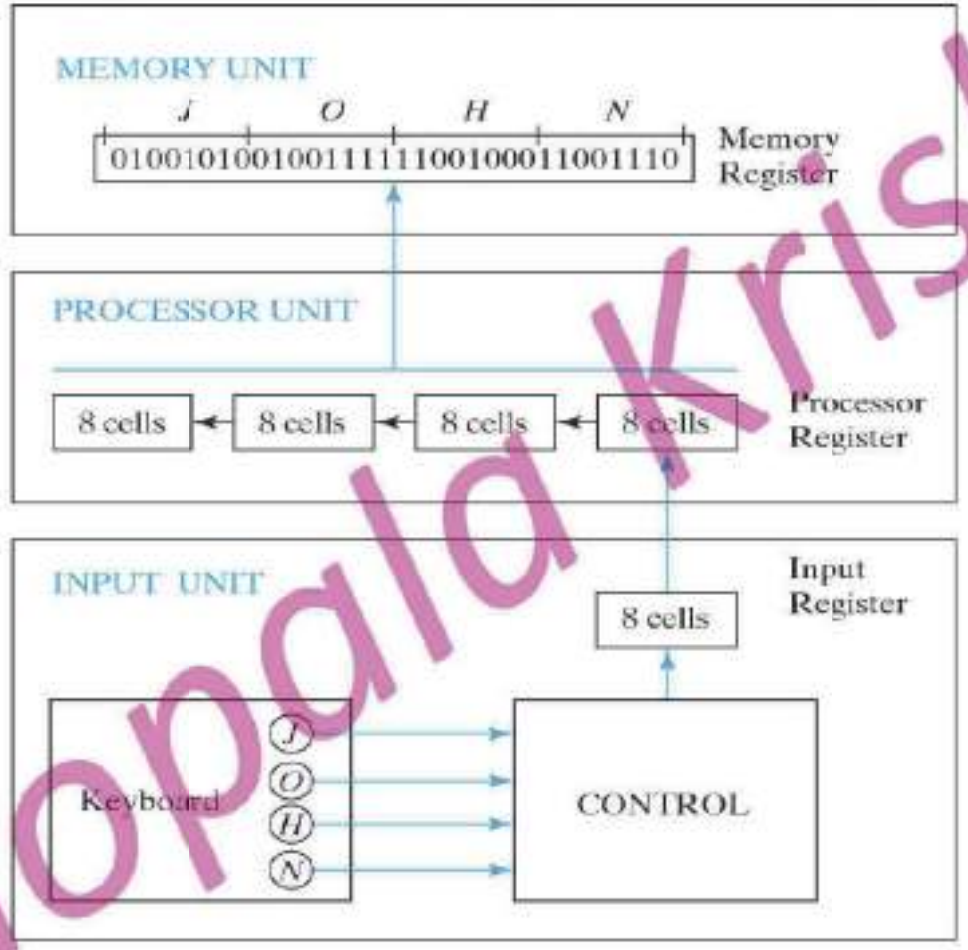
  1 1 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1

- If the content of the register is assumed to be binary equivalent: 50121
- If the content of the register is assumed to be alpha numeric 8 bit code equivalent: two meaningful characters
- If the content of the register is assumed to be ASCII equivalent: C I
- If the content of the register is assumed to be Excess - 3 equivalent: 9096

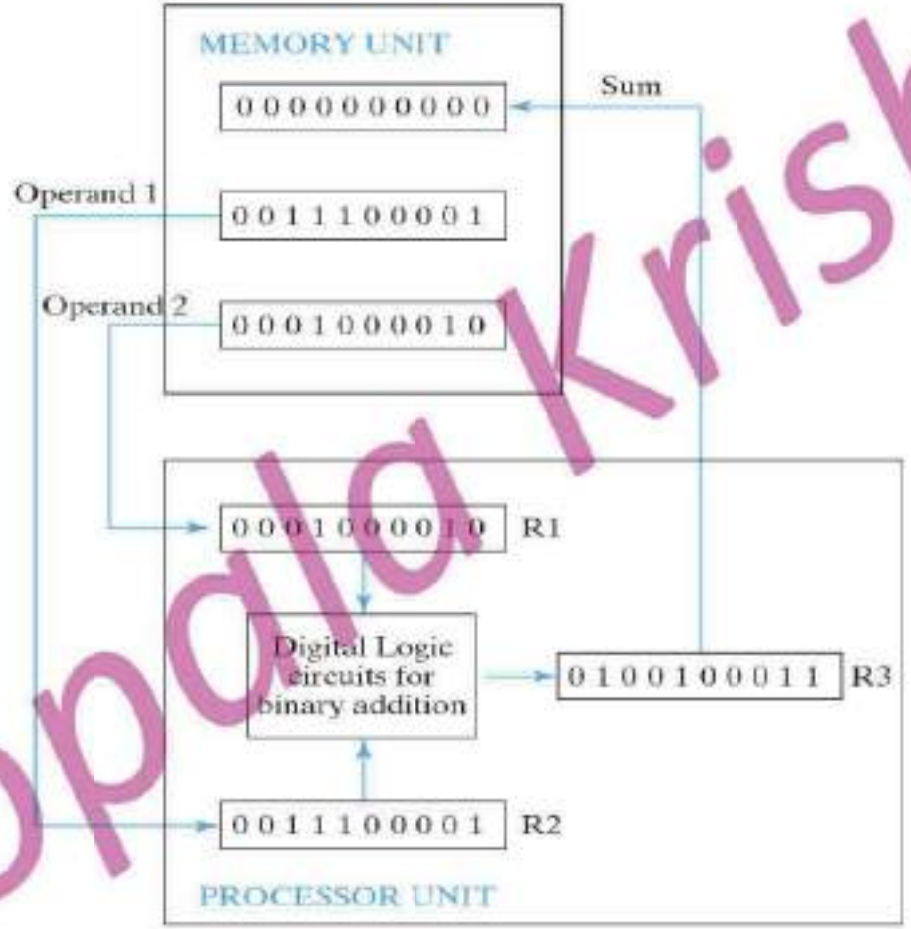
Register Transfer

- A digital system is characterized by its registers and the components that perform data processing.
- A register transfer operation is a basic operation in digital systems.
- It consists of transfer of binary information from one set of registers into another set of registers.

Transfer of information with registers



Binary information processing



Binary Logic

- Binary logic consists of binary variables and logical operations. The variables are designed by the letters of the variables such as A,B,C,x,y,z, etc., with each variable having two and only two distinct possible values: 1 and 0.
- There are three basic logic operations: AND, OR, and NOT.

Basic Logical operations

- AND: This operation is represented by a dot or by the absence of an operator. For example, x.y=z or xy=z is read " x AND y is equal to z." The logical AND is interpreted to mean that z=1 if and only if x=1 and y=1; otherwise z=0.
- Truth Table:

X	Y	Z=X . Y
0	0	0
0	1	0
1	0	0
1	1	1

Basic Logical operations

- OR: This operation is represented by a plus sign. For example, x+y=z is read "x OR Y is equal to z," meaning that z=1 of x=1 or y=1 or if both x=1 and y=1. if both x=0 and y=0 then z=0.
- Truth Table:

X	Y	Z=X + Y
0	0	0
0	1	1
1	0	1
1	1	1

Basic Logical operations

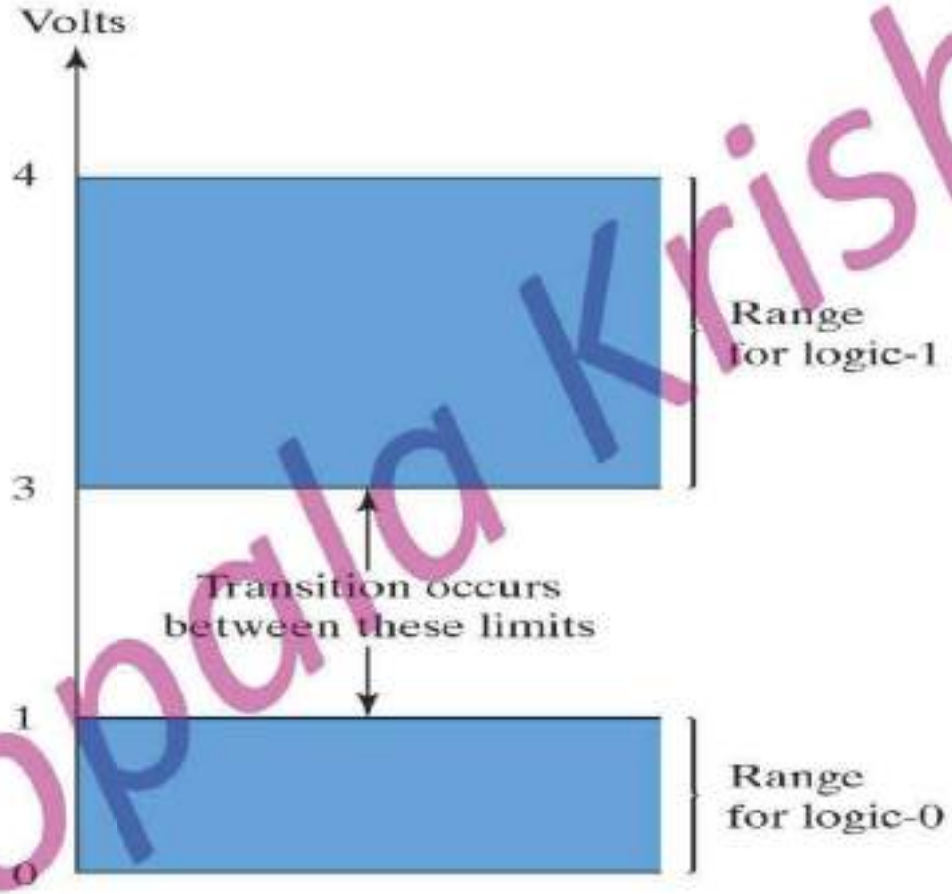
- NOT: This operation is represented by a prime. For example, $x^1 = z$ is read " not x is equal to z" meaning that z is what x is not. In other words, if x=1 then z=0; but if x=0, then z=1. The NOT operation is also called as complementation.
- Truth table:

X	$Z=X^1$
0	1
1	0

Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal.
- Electric signals such as voltages or currents exist throughout a digital system in either of two recognizable values.
- Voltage operated circuits respond to two separate voltage levels that represent a binary variable equal to logic 1 or logic 0.

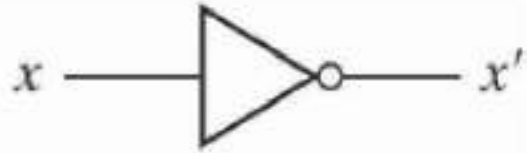
Binary Signals



Symbols for digital logic gates

(a) Two-input AND gate

(b) Two-input OR gate



(c) NOT gate or inverter

Input – Output signals of Gates

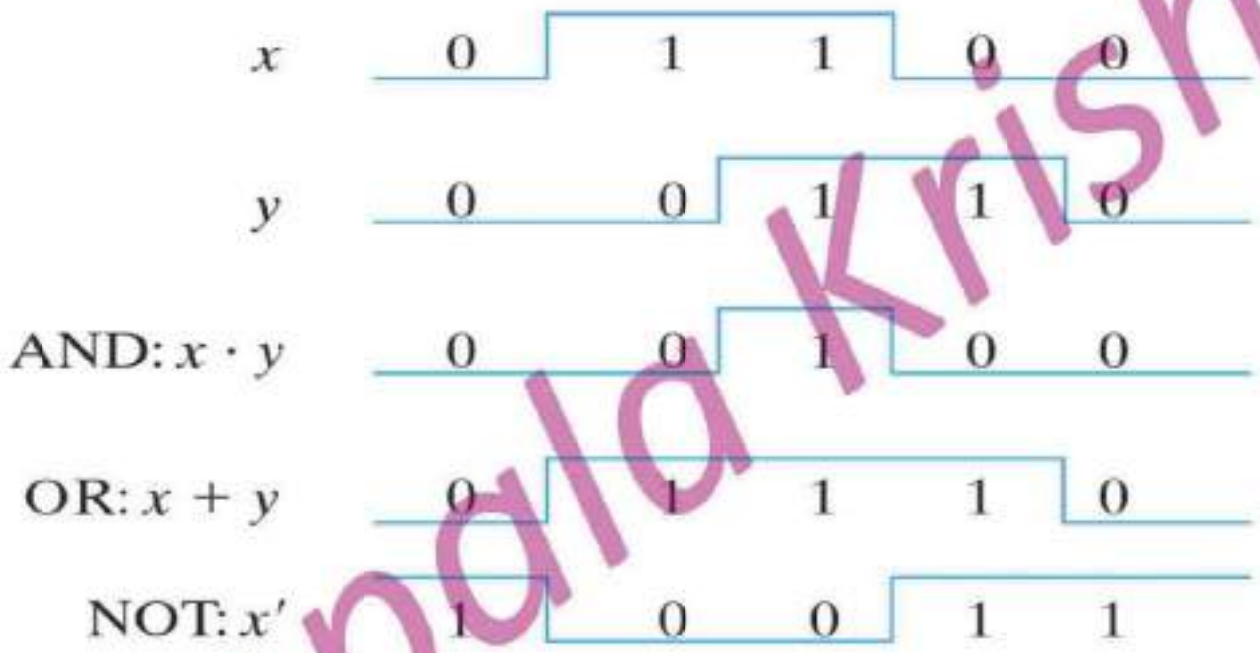
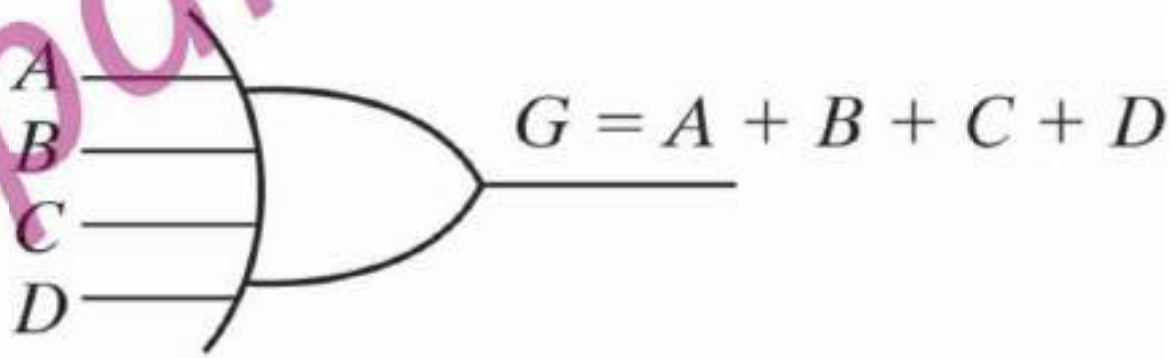


Fig. 1-5 Input-output signals for gates

Gates with multiple inputs

Three Input AND Gate

Four Input OR Gate



BOOLEAN ALGEBRA AND LOGIC GATES

Basic Definitions

- Boolean Algebra is like a deductive mathematical system defined with a set of elements, a set of operators, and a number of unproved axioms or postulates.
- A set of elements is any collection of objects having a common property.
- If S is a set, x and y are certain objects, then

  $x \in S$ denotes that x is a member of the set S, and $y \notin S$ denotes that y is not a member of the set S.

- A binary operator defined on a set S of elements is a rule that assigns to each pair of elements from S a unique element from S.
- As an example, consider the relation a*b=c. we say that * is a binary operator if a,b,c $\in$ S. However, * is not a binary operator if a,b $\in$ S, when the rule finds c $\notin$ S

Properties of a Mathematical System

- The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems, and properties of the system. The most common postulates used to formulate various algebraic structures are:
- Closure: A set S is closed with respect a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.
- Associative Law: a binary operator * on a set S is said to be associative whenever

$$(x*y)*z=x*(y*z) \text{ for all } x,y,z \in S$$

- Commutative Law: a binary operator * on a set S is said to be commutative whenever

$$x*y=y*x \text{ for } x,y \in S$$

- Identity Element: a set S is said to have an identity element with respect to a binary operator *on S if there exists an element e with the property e *x=x*e =x.
- Inverse: A set S having the identity element e with respect to a binary operator * is said to have an inverse whenever, for x of S, there exists an element y in S such that

$$x*y=e$$

- Distributive Law: if * and . are two binary operators on a Set S, * is said to be distributive over . whenever

$$x*(y.z)=(x*y) . (x*z)$$

Axiomatic Definition of Boolean Algebra

- In 1854, George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic structure now called Boolean Algebra.
- For the formal definition of Boolean Algebra, we shall employ the postulates formulated by E.V.Hunntigton in 1904.
- Boolean algebra is an algebraic structure defined by a set of elements B, together with two binary operators, + and *, provided that the following postulates are satisfied.

Huntington Postulates

1(a). Closure with respect to the operator +.

(b). Closure with respect to the operator *.

2(a). An identity element with respect to + designated by 0.

(b). An identity element with respect to * designated by 1.

3(a). Commutative with respective to +.

(b). Commutative with respective to *.

4(a). * is distributive over +

(b). + is distributive over *

5. For every element x in B, there exists an element y in B such that x+y=1 and x*y=0.

6. There exists at least two element x and y in B such that $x \neq y$

Two – valued Boolean Algbera

- A Two-valued Boolean algebra is defined on a set of two elements, B={0,1}, with rules for the two binary operators + and * as shown in the following operator tables:

x	y	x*y
0	0	0
0	1	0
1	0	0
1	1	1

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

x	x^1
0	1
1	0

Huntington postulates are valid for the set B={0,1} and the two binary operators

1 Closure is obvious from the tables since the result of each operation is either 1 or 0

2 From the table we see that

$0+0=0$ $0+1=1+0=1$

$1*1=1$ $1*0=0*1=0$

This establishes the two identity elements 0 and 1 for + and *.

3 The commutative laws are obvious from the symmetry of the binary operator tables.

4(a) The distributive law $x*(y+z)=(x*y)+(x*z)$ can be shown to hold true from the operator tables by forming a truth table of all possible values of x, y, and z.

(b) The distributive law of + over * can be shown to hold true by means of a truth table.

5 From the complement table it is easily shown that

$x+x^1=1$, since $0+0^1=0+1=1$ and $1+1^1=1+0=1$

$X*x^1=0$, since $0*0^1=0*1=0$ and $1*1^1=1*0=0$

6 Postulates 6 is satisfied because the two-valued Boolean algebra has two distinct elements, 1 and 0, with $1\neq0$

Basic Theorems and Postulates

Postulate 2	(a) $x+0=x$	(b) $x*1=x$
Postulate 5	(a) $x+x^1=1$	(b) $x*x^1=0$
Theorem 1	(a) $x+x=x$	(b) $x*x=x$
Theorem 2	(a) $x+1=1$	(b) $x*0=0$
Theorem 3, Involution	(a) $((x)^1)^1=x$	
Postulate 3, Commutative	(a) $x+y=y+x$	(b) $x*y=y*x$
Theorem 4, Associative	(a) $x+(y+z)=(x+y)+z$	(b) $x*(y*z)=(x*y)*z$
Postulate 4, Distributive	(a) $x*(y+z)=(x*y)+(x*z)$	(b) $x+yz=(x+y)(x+z)$
Theorem 5, De Morgan	(a) $(x+y)^1=x^1y^1$	(b) $(xy)^1=x^1+y^1$
Theorem 6, Absorption	(a) $x+xy=x$	(b) $x(x+y)=x$

Operator Precedence

- The operator precedence for evaluating Boolean Expression is:
  - Parenthesis
  - NOT
  - AND
  - OR

Boolean Functions

- A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.
- For a given value of the binary variables, the function can be equal to either 1 or 0.

Example: $F_1 = x + y^1 z$

- A Boolean function expresses the logical relationship between binary variables. It is evaluated by determining the binary value of the expression for all possible values of the variable.

- A Boolean function can be represented in a truth table.
- A truth table is a list of combinations of 1's and 0's assigned to the binary variables and a column that shows the value of the function for each binary combination.
- The number of rows in the truth table is 2^n, where n is the number of variables in the function.

Truth table of a Boolean Function

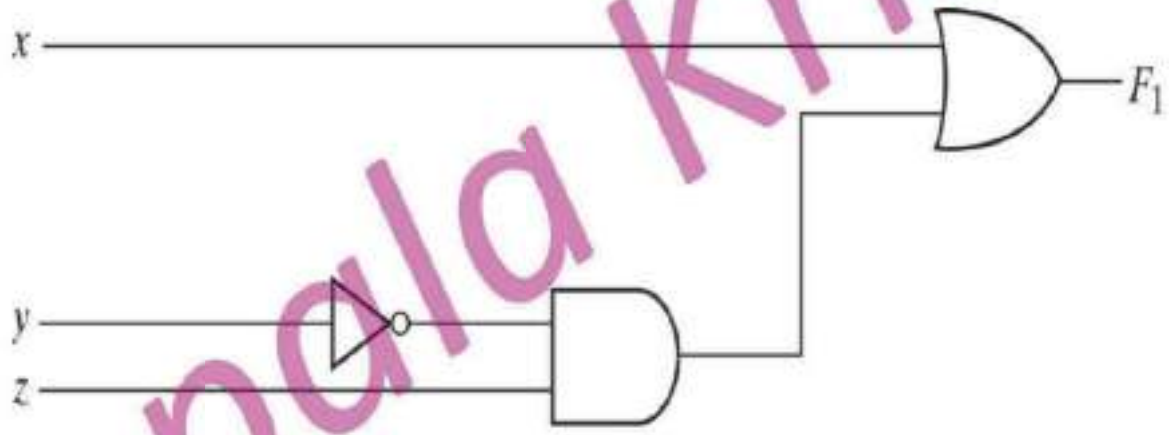
Example: $F_1 = x + y^1 z$

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

- A Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates.
- The variables of the function are taken as inputs of the circuit and the left hand side variable is taken as the output.
- For example for the function $F_1=x+y^1z$, in which x,y and z are inputs and F_1 is the output of the circuit diagram.
- An inverter is used for y input to generate y^1 and an AND gate used for the term y^1z and an OR gate to form the sum $x+y^1z$

Gate Implementation of a Boolean Function

- $F_1 = x + y^1 z$

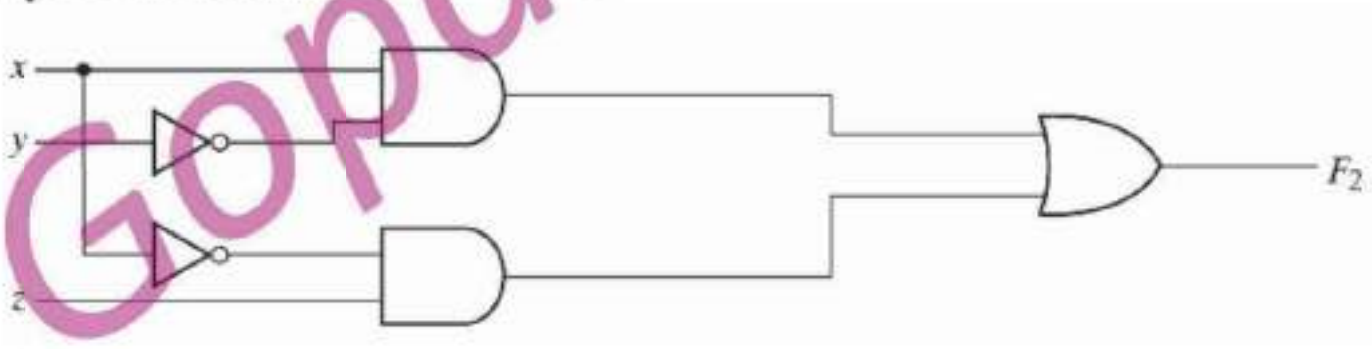


Manipulation of Boolean Expressions

- There is only one way that a Boolean Function can be represented in a truth table.
- When the function is in algebraic form, it can be expressed in a variety of ways.
- By manipulating a Boolean expression according to Boolean Algebra rules, it is some times possible to obtain a simpler expression for the same function and thus reduce the number of gates in the circuit and the number of inputs to the gate.

- For example, $F2=x^1y^1z+x^1yz+xy^1$
- The implementation of the above expression:

- Simplification of the function
  $F2=x^1y^1z+x^1yz+xy^1$
  $=x^1z(y^1+y)+xy^1$
  $= x^1z(1)+xy^1 = x^1z+xy^1$
- The implementation of simplified expression



Algebraic Manipulation

- When a Boolean expression is implemented with logic gates, each term requires a gate and each variable within the term designates an input to the gate.
- A Literal is a single variable within a term that may be complemented or not.
- For example, $F2=x^1y^1z+x^1yz+xy^1$ has three terms and eight literals.
- By reducing the number of terms, the number of literals, or both in a Boolean expression, it is often possible to obtain a simpler circuit.
- The manipulation of Boolean Expression mostly consists of reducing an expression for the purpose of obtaining a simpler circuit.

Complement of a Function

- The complement of a function F is F^1 and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F.
- The complement of a function may be derived algebraically through De Morgan's theorem.
- For example,

$$(A+B+C)^1=(A+X)^1 \quad \text{LET } B+C=X$$
$$= A^1X^1= A^1(B+C)^1= A^1B^1C^1$$

Canonical Forms

Minterms and Maxterms

- A binary variable may appear either in its normal form (x) or in its complement form (x^1).
- Consider two variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations: x^1y^1, x^1y, xy^1, xy each of these four AND terms is called a minterm or a standard product.
- In a similar manner, n variables can be combined to form 2^n minterms.

Minterms & Maxterms for three variables

Mint erm					Maxt erm	
X	Y	Z	Term	Designation	Term	designation
0	0	0	$x^1 y^1 z^1$	m_0	$x+y+z$	M_0
0	0	1	$x^1 y^1 z$	m_1	$x+y+z^1$	M_1
0	1	0	$x^1 y z^1$	m_2	$x+y^1+z$	M_2
0	1	1	$x^1 y z$	m_3	$x+y^1+z^1$	M_3
1	0	0	$x y^1 z^1$	m_4	x^1+y+z	M_4
1	0	1	$x y^1 z$	m_5	x^1+y+z^1	M_5
1	1	0	$x y z^1$	m_6	x^1+y^1+z	M_6
1	1	1	$x y z$	m_7	$x^1+y^1+z^1$	M_7

Minterm Formation

- The 2^n different minterms are determined by the following procedure:
  - The binary numbers from 0 to 2^n-1 are listed under the n variables.
  - Each minterm is obtained from an AND term of the n variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1.
  - Each minterm is also symbolized by m_j, where j denotes the decimal equivalent of the binary number of the minterm designated.

Maxterm Formation

- In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called Maxterms or standard sums.
  - Each Maxterm is obtained from an OR term of the variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1.
  - Note that, each maxterm is the complement of its corresponding minterm, and vice versa.

Minterm and Maxterm representations of Boolean functions

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

- A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function, and then taking OR of all those terms.
- $F_1 = x^1 y^1 z + x y^1 z^1 + x y z$

  $= m_1 + m_4 + m_7$
- This example, demonstrates an important property of boolean algebra: Any boolean function can be expressed as a sum of minters.

$F_1 = x^1\ y^1\ z + x\ y^1\ z^1 + x\ y\ z$

- The complement of F_1 can be formed from the truth table by OR ing all the minterms that produces a 0.

$F_1^1 = x^1\ y^1\ z^1 + x^1\ y\ z^1 + x^1\ y\ z + x\ y^1\ z + x\ y\ z^1$

- If we take the complement of F_1^1, we obtain the function F_1.

$(F_1^1)^1 = (x^1y^1z^1 + x^1yz^1 + x^1yz + xy^1z + xyz^1)^1$

$F_1 = (x+y+z)(x+y^1+z)(x+y^1+z^1)(x^1+y+z^1)(x^1+y^1+z)$

$= M_0M_2M_3M_5M_6$

Sum of Minterms

- It is sometimes convenient to express the Boolean Function in its sum of minterms form.
- If not in this form, it can be made so by first expanding the expression into a sum of AND terms.
- Each of the term is then inspected to see if it contains all the variables.
- If it misses one or more variables, it is AND ed with an expression such as $(x+x^1)$, where x is one of the missing variable.

Example

- Express the Boolean function $F=A+B^1C$ in a sum of minterms. The function has three variables. A,B,and C.
- The first A is missing two variables; therefore:

$$A=A(B+B^1)(C+C^1)$$
$$=ABC+ABC^1+AB^1C+AB^1C^1$$

- The second term B^1C is missing one variable: that is A

$$B^1C=B^1C(A+A^1)=AB^1C+A^1B^1C$$

- Combining all terms we have,

$$F=A+B^1C=ABC+ABC^1+AB^1C+AB^1C^1+AB^1C+A^1B^1C$$
$$=A^1B^1C+AB^1C^1+AB^1C+ABC^1+ABC$$

- The Boolean Function, when it is in its Sum of Minterms form, can be expressed in short notation:

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

- An alternative way of obtaining sum of minterms form of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.
- Consider, the Boolean Function $F=A+B^1C$.
- The truth table can be obtained from the algebraic function
- $F(A,B,C)=\Sigma(1,4,5,6,7)$

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

Product of Maxterms

- To express the Boolean function as a product of maxterms, it must first be brought into a form of OR terms.
- This may be done by using the distributive law, x+yz=(x+y)(x+z).
- Then any missing variable x in each OR term is ORed with xx^1.

Example

- Express the boolean function $F=xy+x^1z$ in a product of maxterm form.
- First, convert the function into OR terms using the distributive law.

$F=xy+x^1z=(xy+x^1)(xy+z)$

$=(x+x^1)(y+x^1)(x+z)(y+z)$

$=(x^1+y)(x+z)(y+z)$

- The function has three variables: x,y and z. each OR term is missing one variable; therefore

$(x^1+y)= x^1+y+zz^1 = (x^1+y+z)(x^1+y+z^1)$

$(x+z) = x+z+yy^1 = (x+y+z)(x+y^1+z)$

$(y+z) = y+z+xx^1 = (x+y+z)(x^1+y+z)$

- Combining all the terms and removing those that appear more than once, we finally obtian:

$F=(x^1+y+z)\ (x^1+y+z^1)\ (x+y^1+z)\ (x+y+z)=M_0M_2M_4M_5$

- A convenient way to express this function as follows:

$F(x,y,z)=\Pi(0,2,4,5)$

- The product symbol Π, denotes the ANDing of maxterms; the numbers are the maxterms of the function

Conversion between Canonical Forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
- This is because the original function is expressed by those minterms that make the function equal to 1, whereas its complement is a 1 for those minterms that the function is a 0.
- For example consider the function

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

This has a complement that can be expressed as

$$F^1(A,B,C)= \Sigma(0,2,3)=m_0+m_2+m_3$$

Now if we take the complement of F^1 by DeMorgan's theorem, we obtain F in a different form:

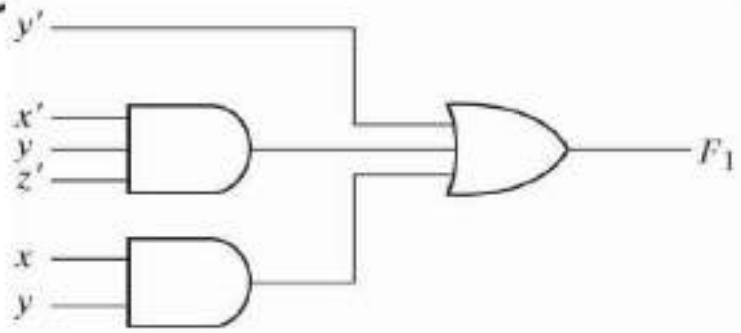
$$F(A,B,C)= (m_0+m_2+m_3)^1=M_0M_2M_3= \Pi(0,2,3)$$

Standard Forms

- The two canonical forms of Boolean algebra are basic forms that one obtains from reading a function from the truth table.
- Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two or any number of literals.
- There are two types of standard forms:
  - The sum of products
  - The product of sums

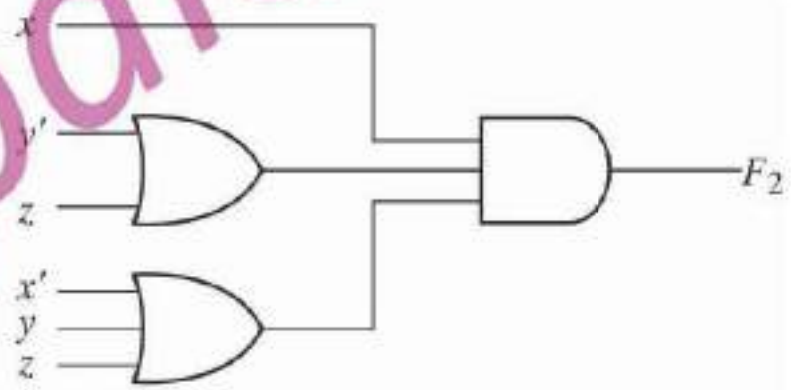
Sum of Products

- The sum of products is a Boolean expression containing AND terms, called product terms, of one or more literals each. The sum denotes the ORing of these terms.
- An example of a function expressed in sum of products is $F_1=y^1+xy+x^1yz^1$
- The logic diagram of a sum of products expression consists of a group of AND gates followed by a single OR gate



Product of Sums

- A product of sum is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals
- The product denotes the ANDing of these terms.
- An example of a function expressed in product of sum is $F_2=x(y^1+z)(x^1+y+z^1)$



Other Logic Operations

Boolean Functions	Name
$F_0=0$	Null
$F_1=xy$	AND
$F_2=xy^1$	Inhibition
$F_3=x$	Transfer
$F_4=x^1y$	Inhibition
$F_5=y$	Transfer
$F_6=xy^1+x^1y$	Exclusive-OR
$F_7=x+y$	OR

Boolean Functions	Name
$F_8=(x+y)^1$	NOR
$F_9=xy+x^1y^1$	Equivalence
$F_{10}=y^1$	Complement
$F_{11}=x+y^1$	Implication
$F_{12}=x^1$	Complement
$F_{13}=x^1+y$	Implication
$F_{14}=(xy)^1$	NAND
$F_{15}=1$	Identity

Digital Logic Gates

- Since Boolean functions are expressed in terms of AND, OR and NOT operations, it is easier to implement a Boolean function with these type of gates.
- The possibility of constructing gates for the other logic operations are based on the following factors:
  - The feasibility and economy of producing the gate with physical components.
  - The possibility of extending the gate to more than two inputs.
  - The basic properties of the binary operators, such as commutativity, and associativity, and
  - The ability of the gate to implement Boolean functions alone or in conjunction with other gates.

AND Gate

- The AND gate consists of two inputs named x and y and F is the Output function which equals to F=xy.

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

- The OR gate consists of two inputs named x and y and F is the Output function which equals to F=x+y.

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

- The NOT Gate consists of only one input x whose value can be either 1 or 0.
- F is the output function of NOT Gate which is equal to x^1

Inverter

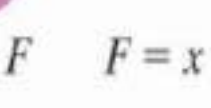
$F = x'$

x	F
0	1
1	0

Buffer Gate

- The Buffer gate consists of one input named x and F is the Output function which equals to F=x.

Buffer



$F = x$

x	F
0	0
1	1

NAND Gate

- The NAND gate consists of two inputs named x and y and F is the Output function which equals to $F=(xy)^1$.

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

- The NOR gate consists of two inputs named x and y and F is the Output function which equals to $F=(x+y)^1$.

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR GATE

- The Ex-OR gate consists of two inputs named x and y and F is the Output function which equals to F=x ⊕ y.

Exclusive-OR (XOR)

$F = xy' + x'y$
$= x \oplus y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

EX-NOR Gate

- The EX-NOR gate consists of two inputs named x and y and F is the Output function which equals to $F=(x \oplus y)^1$.

Exclusive-NOR or equivalence

$$F = xy + x'y'$$
$$= (x \oplus y)'$$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Integrated Circuits

- An integrated circuit is a silicon semiconductor crystal, called a chip, containing the electronic components for constructing digital gates.
- The various gates are interconnected inside the chip to form the required circuit.
- The chip is mounted in a ceramic or plastic container, and connections are welded to external pins to form the integrated circuit.
- Each IC has a numeric designation printed on the surface of the package for identification.

Levels of Integration

- Digital IC's are often categorized according to their circuit complexity as measured by the number of gates in a single package.
- Small Scale Integration devices contain several independent gates in a single package. The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.
- Medium Scale Integration devices have a complexity of 10 to 1,000 gates in a single package. Example of these devices are decoders, adders and multiplexers.
- Large Scale Integration devices contain thousands of gates in a single package. They include digital systems such as processors, memory chips, and programmable logic devices.
- Very Large Scale Integration devices contain hundred of thousands of gates within a single package. Examples are memory arrays and microcomputer chips.

Digital Logic Families

- Digital integrated circuits are categorized not only by their complexity but also by their circuit technology.
- The circuit technology is referred to as a digital logic family.
  - TTL Transistor Transistor logic
  - ECL Emitter coupled logic
  - MOS Metal oxide semiconductor
  - CMOS Complementary metal oxide semiconductor