

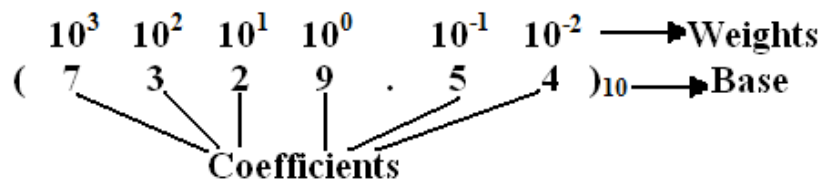


## Lecture 1

### 1. Digital Computers:

The characteristic of digital systems is its manipulation of discrete elements of information. Examples of discrete elements may be electrical impulses, decimal digits, the letters of the alphabet, arithmetic operations, punctuation marks, and any other meaningful symbols. Discrete elements of information are represented in a digital system by physical quantities called signals. Electrical signals such as voltages are the most common. The signals in all present-day electronic digital systems have only two discrete values and are said to be binary.

### 2. Number Systems ( Binary, Octal , Decimal and Hexadecimal):



$$\text{Example: } (7392.54)_{10} = 10^3 * 7 + 10^2 * 3 + 10^1 * 9 + 10^0 * 2 + 10^{-1} * 5 + 10^{-2} * 4 \\ = 7000 + 300 + 20 + 9 + 0.5 + 0.04$$

- When the **base** is equal to **10** the numbering system is named **Decimal** and the **coefficients range** is **(0,1,2,3,4,5,6,7,8,9)**

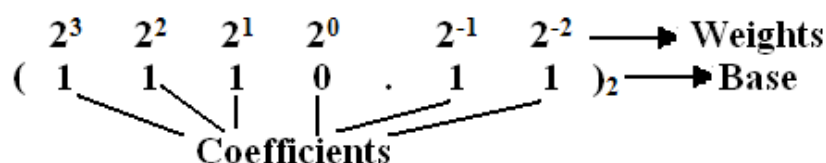
In general a number with decimal points is represented by a series of coefficients as follows:

$$(a_n \dots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \dots a_{-m})_{10}$$

The  $a_j$  coefficients are one of the digits (0, 1, 2, 3, ..., 9)

$$a_3 * 10^3 + a_2 * 10^2 + a_1 * 10^1 + a_0 * 10^0 + a_{-1} * 10^{-1} + a_{-2} * 10^{-2} + a_{-3} * 10^{-3}$$

- The **Binary** number system is a different number system, the **coefficients** are **(0 and 1)** only and the **base** or radix **2**, Ex:



- When the **base** is equal to **8** the numbering system is named **Octal** and the **coefficients range** is **(0, 1, 2, 3, 4, 5, 6, 7)**, Ex:

$$\begin{array}{ccccccc}
 8^3 & 8^2 & 8^1 & 8^0 & & 8^{-1} & 8^{-2} & \longrightarrow \text{Weights} \\
 ( & 7 & 4 & 3 & 0 & . & 2 & 5 & )_8 \longrightarrow \text{Base} \\
 & \swarrow & \swarrow & | & | & \swarrow & \swarrow & \\
 & \text{Coefficients} & & & & & & 
 \end{array}$$

- When the **base** is equal to **16** the numbering system is named **Hexadecimal** and the **coefficients range** is **(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)**, where A=10, B=11, C=12, D=13, E=14, F=15. Ex :

$$\begin{array}{ccccccc}
 16^2 & 16^1 & 16^0 & & 16^{-1} & 16^{-2} & \longrightarrow \text{Weights} \\
 ( & F & 8 & 9 & . & B & 5 & )_{16} \longrightarrow \text{Base} \\
 & \swarrow & \swarrow & | & \swarrow & \swarrow & \\
 & \text{Coefficients} & & & & & 
 \end{array}$$

In general a number expressed in base (r) system has coefficients (0, 1...r-1), multiplied by power of r

$$\begin{array}{ccccccccccc}
 r^n & & r^3 & r^2 & r^1 & r^0 & & r^{-1} & r^{-2} & & r^{-m} & \longrightarrow \text{Weights} \\
 ( & a_n & \dots & a_3 & a_2 & a_1 & a_0 & . & a_{-1} & a_{-2} & \dots & a_{-m} & )_r \longrightarrow \text{Base}
 \end{array}$$

$$r^n \cdot a_n + \dots + r^3 \cdot a_3 + r^2 \cdot a_2 + r^1 \cdot a_1 + r^0 \cdot a_0 + r^{-1} \cdot a_{-1} + r^{-2} \cdot a_{-2} + \dots + r^{-m} \cdot a_{-m}$$

When the base of the number is less than (10) the needed (r) digit of the coefficients are borrowed from the decimal system. If the base is greater than (10) then the letters of the alphabet are used.

### 3. Conversion from Decimal to Other Bases and vice versa.

#### 3. A The conversion from any base r to decimal:

A number expressed in base r can be converted to its decimal equivalent by multiplying each coefficient with corresponding power of r and adding.

Ex:

$$\begin{array}{ccccccc}
 7^2 & 7^1 & 7^0 & & 7^{-1} & & \\
 ( & 6 & 3 & 0 & . & 4 & )_7 = 6 \cdot 7^2 + 3 \cdot 7^1 + 0 \cdot 7^0 + 4 \cdot 7^{-1} \\
 & & & & & & = 49 \cdot 6 + 21 + 0 + 4/7 = 294 + 21 + 0 + 0.571 = (315.571)_{10}
 \end{array}$$

$$\begin{array}{cccc}
 8^2 & 8^1 & 8^0 & 8^{-1} \\
 (6 & 3 & 0 & . 4)_{8}
 \end{array}
 = 6 \cdot 8^2 + 3 \cdot 8^1 + 0 \cdot 8^0 + 4 \cdot 8^{-1}$$

$$= 64 \cdot 6 + 24 + 0 + 4/8 = 384 + 24 + 0 + 0.5 = (408.5)_{10}$$

$$\begin{array}{ccccccc}
 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\
 (1 & 1 & 1 & 0 & . 1 & 0 & 1)_{2}
 \end{array}
 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$

$$= 8 + 4 + 2 + 0 + 1/2 + 0 + 1/8 = 14 + 0.5 + 0 + 0.125$$

$$= (14.625)_{10}$$

$$\begin{array}{cccc}
 16^2 & 16^1 & 16^0 & 16^{-1} \\
 (F & 3 & A & . B)_{16}
 \end{array}
 = F \cdot 16^2 + 3 \cdot 16^1 + A \cdot 16^0 + B \cdot 16^{-1}$$

$$= 15 \cdot 16^2 + 3 \cdot 16^1 + 10 \cdot 16^0 + 11 \cdot 16^{-1}$$

$$= 15 \cdot 256 + 3 \cdot 16 + 10 \cdot 1 + 11/16$$

$$= 3840 + 48 + 10 + 0.6875 = (3898.6875)_{10}$$

### 3. B The Conversion from decimal to any base r:

Note: The conversion is more convenient if the number is separated into an integer part and a fraction part so the conversion of each part is done separately.

Ex: Convert the decimal number (14.625) to binary (base 2)

Integer	Remainder	
$14 \div 2$		
$7 \div 2$	0	$a_0$
$3 \div 2$	1	$a_1$
$1 \div 2$	1	$a_2$
0	1	$a_3$

$$(14)_{10} = (1110)_2$$

	Integer	Fraction	
		$0.625 \cdot 2$	$=1.25$
$a_{-1}$	1	$0.25 \cdot 2$	$=0.5$
$a_{-2}$	0	$0.5 \cdot 2$	$=1.0$
$a_{-3}$	1	0.00	

$$(0.625)_{10} = (0.101)_2$$

$$(14.625)_{10} = (1110.101)_2$$

Ex: Convert the decimal number (315.571) to (base 7)

Integer	Remainder	
$315 \div 7$		
$45 \div 7$	0	$a_0$
$6 \div 7$	3	$a_1$
0	6	$a_2$

$$(315)_{10} = (630)_7$$

	Integer	Fraction	
		$0.571$ $* 7$	$=3.997$
$a_{-1}$	3	$0.997$ $* 7$	$=6.979$
$a_{-2}$	6	$0.979$ $* 7$	$=6.853$
$a_{-3}$	6	0.853	

$$(0.571)_{10} = (0.366)_7$$

$$(315.571)_{10} = (630.366)_7$$

Ex: Convert the decimal number (314.21) to Hexadecimal (base 16)

Integer	Remainder	
$314 \div 16$		
$19 \div 16$	A	$a_0$
$1 \div 16$	3	$a_1$
0	1	$a_2$

$$(314)_{10} = (13A)_{16}$$

	Integer	Fraction	
		$0.21$ $* 16$	$=3.36$
$a_{-1}$	3	$0.36$ $* 16$	$=5.76$
$a_{-2}$	5	$0.76$ $* 16$	$=12.16$
$a_{-3}$	C	0.16	

$$(0.21)_{10} = (0.35C)_{16}$$

$$(314.21)_{10} = (13A.35C)_{16}$$

**HW. Convert the following number to the indicated bases**

1.  $(214.3)_{10}$  to base 4.
2.  $(10101.101)_2$  to decimal.
3.  $(124.03)_5$  to base 7.

4.  $(346.67)_{10}$  to base 16.
5.  $(124.34)_{10}$  to base 12.
6.  $(110101.1101)_2$  to decimal.
7.  $(42F.CB)_{16}$  to decimal
8.  $(111011010.001101)_2$  to Octal.
9.  $(12A.8)_{12}$  to Decimal.

#### 4. A number with different bases:

Decimal (0,1,..,9)		Binary (0,1)					Octal (0,1,..,7)		Hexadecimal (0,1,..,9,A..F)	
$10^1$	$10^0$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$8^1$	$8^0$	$16^1$	$16^0$
<b>10</b>	<b>1</b>	<b>16</b>	<b>8</b>	<b>4</b>	<b>2</b>	<b>1</b>	<b>8</b>	<b>1</b>	<b>16</b>	<b>1</b>
	0					0		0		0
	1					1		1		1
	2				1	0		2		2
	3				1	1		3		3
	4			1	0	0		4		4
	5			1	0	1		5		5
	6			1	1	0		6		6
	7			1	1	1		7		7
	8		1	0	0	0	1	0		8
	9		1	0	0	1	1	1		9
1	0		1	0	1	0	1	2		A
1	1		1	0	1	1	1	3		B
1	2		1	1	0	0	1	4		C
1	3		1	1	0	1	1	5		D
1	4		1	1	1	0	1	6		E
1	5		1	1	1	1	1	7		F
1	6	1	0	0	0	0	2	0	1	0
1	7	1	0	0	0	1	2	1	1	1
1	8	1	0	0	1	0	2	2	1	2
1	9	1	0	0	1	1	2	3	1	3
2	0	1	0	1	0	0	2	4	1	4
2	1	1	0	1	0	1	2	5	1	5
2	2	1	0	1	1	0	2	6	1	6
2	3	1	0	1	1	1	2	7	1	7
2	4	1	1	0	0	0	3	0	1	8
2	5	1	1	0	0	1	3	1	1	9

## 5. Octal, Hexadecimal and Binary numbers:

The conversion from and to binary (base 2), Octal (base 8) and hexadecimal (base 16) plays an important part in digital computers, since  $2^3=8$  and  $2^4=16$  each octal digit corresponds to 3-binary digits and each hexadecimal digit corresponds to 4-binary digits.

$$\text{Ex:(1) } (10110001101011.11110000011)_2 = (?)_8 = (26153.7406)_8 \\ = (?)_{16} = (2C6B.F06)_{16}$$

(010	110	001	101	011	.	111	100	000	110)	<sub>2</sub>
(2	6	1	5	3	.	7	4	0	6)	<sub>8</sub>

(0010	1100	0110	1011	.	1111	0000	0110)	<sub>2</sub>
(2	C	6	B	.	F	0	6)	<sub>16</sub>

$$(2) \quad (673.124)_8 = (?)_2 = (110111011.0010101)_2$$

(6	7	3	.	1	2	4)	<sub>8</sub>
(110	111	011	.	001	010	100)	<sub>2</sub>

$$(3) \quad (306.D)_{16} = (?)_2 = (1100000110.1101)_2$$

(3	0	6	.	D)	<sub>16</sub>
(0011	0000	0110	.	1101)	<sub>2</sub>

### HW. Convert the following number to the indicated bases

1.  $(456.7)_8$  to hexadecimal using the binary as intermediate base.
2.  $(98FE.0AB)_{16}$  to Octal using the binary as intermediate base.
3.  $(10AB.FE)_{16}$  to octal using the binary as intermediate base.
4.  $(6754.231)_8$  to Hexadecimal using the binary as intermediate base
5.  $(111011010.001101)_2$  to Octal.
6.  $(AD09.3C)_{16}$  to Binary.

## 6. Arithmetic Operation:

Addition (+), Subtraction (-), Multiplication (\*)

Carry	(1)	(1)	(1)	(1)		
Augend	(1	0	1	1	0	1) <sub>2</sub>
Addend	(1	0	0	1	1	1) <sub>2</sub> +
Sum	(10	1	0	1	0	0) <sub>2</sub>

			10			
Borrow			0	<del>1</del>	10	
Minuend	(1	0	<del>1</del>	<del>1</del>	<del>0</del>	1) <sub>2</sub>
Subtrahend	(1	0	0	1	1	1) <sub>2</sub> -
Difference	(0	0	0	1	1	0) <sub>2</sub>

Multiplicand	(	1	0	1	1) <sub>2</sub>	
Multiplier	(		1	0	1) <sub>2</sub>	*
	(1)	1	0	1	1	
		0	0	0	0	+
	1	0	1	1		
Product	(1	1	0	1	1	1) <sub>2</sub>

**HW. Perform the following operation without converting to decimal or any other base:**

1.  $(1101.01 * 10.1)_2 =$
2.  $(D39 + 16A)_{16} =$
3.  $(243.13 - 144.02)_5 =$
4.  $(11100.01 - 1110.11)_2 =$
5.  $(243 * 24)_5 =$
6.  $(FB09 - ED32)_{16} =$
7.  $(227.65 + 624.31)_8 =$
8.  $(FD0D - AE21)_{16} =$
9.  $(323 * 32)_4 =$

## 7. Complement and Subtraction using 1's and 2's Complement.

### 7.1 Complement:

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements for each base ( $r$ ) system:

- I. The  $r$ 's complements
- II. The  $(r-1)$ 's complements

**7.1.1 The  $r$ 's complements:** A positive number  $N$  in base  $r$  with an integer part of  $n$  digits, the  $r$ 's complement of  $N$  is defining as follows:

$r^n - N$	for $N \neq 0$
0	for $N = 0$

Ex:

- (1) The 10's complement of  $(925.67)_{10}$  is  $N=925.67$ ,  $r=10$ ,  $n=3 \rightarrow$  the 10's complement equal to:  $(10^3 - 925.67)_{10} = 1000 - 925.67 = (74.33)_{10}$

$$\begin{array}{r}
 \begin{array}{cccccc}
 & 9 & 9 & 9 & & 9 \\
 0 & \cancel{10} & \cancel{10} & \cancel{10} & & \cancel{10} & 10 \\
 (\cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & . & \cancel{0} & \cancel{0} )_{10} - \\
 ( & 9 & 2 & 5 & . & 6 & 7 )_{10} \\
 \hline
 (0 & 0 & 7 & 4 & . & 3 & 3 )_{10}
 \end{array}
 \end{array}$$

- (2) The 10's complement of  $(0.3267)_{10}$  is  $N=0.3267$ ,  $r=10$ ,  $n=0 \rightarrow$  the 10's complement equal to:  $(10^0 - 0.3267)_{10} = 1 - 0.3267 = (0.6733)_{10}$

$$\begin{array}{r}
 \begin{array}{cccccc}
 & 9 & 9 & 9 & & 9 \\
 0 & \cancel{10} & \cancel{10} & \cancel{10} & & 10 \\
 (\cancel{1} & . & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} )_{10} - \\
 (0 & . & 3 & 2 & 6 & 7)_{10} \\
 \hline
 (0 & . & 6 & 7 & 3 & 3)_{10}
 \end{array}
 \end{array}$$

- (3) The 2's complement of  $(101100)_2$  is  $N=101100$ ,  $r=2$ ,  $n=6 \rightarrow$  the 2's complement equal to:  $(2^6)_{10} - (101100)_2 = (1000000 - 101100)_2 = (10100)_2$

$$\begin{array}{r}
 \begin{array}{cccccc}
 & 1 & 1 & 1 & & 1 \\
 0 & \cancel{10} & \cancel{10} & \cancel{10} & & 10 \\
 (\cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & \cancel{0} & 0 & 0)_2 - \\
 ( & 1 & 0 & 1 & 1 & 0 & 0)_2 \\
 \hline
 (0 & 0 & 1 & 0 & 1 & 0 & 0)_2
 \end{array}
 \end{array}$$

- (4) The 2's complement of  $(0.0110)_2$  is  $N=0.0110$ ,  $r=2$ ,  $n=0 \rightarrow$  the 2's complement equal to:  $(2^0)_{10} - (0.0110)_2 = (1 - 0.0110)_2 = (0.1010)_2$

$$\begin{array}{r}
 \begin{array}{cccccc}
 & 1 & 1 & & & 1 \\
 0 & . & \cancel{10} & \cancel{10} & & 10 \\
 (\cancel{1} & . & \cancel{0} & \cancel{0} & \cancel{0} & 0)_2 - \\
 ( & 0 & 1 & 1 & 0)_2 \\
 \hline
 (0 & . & 1 & 0 & 1 & 0)_2
 \end{array}
 \end{array}$$



- (5) The 9's complement of  $(0.3267)_{10}$  is  $N=0.3267$ ,  $r=10$ ,  $n=0, m=4 \rightarrow$  the 9's complement equal to:  $(10^0 - 10^{-4} - 0.3267)_{10} = (1 - 0.0001) - 0.3267 = 0.9999 - 0.3267 = (0.6732)_{10}$

$$\begin{array}{r}
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \hline
 (0 \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{1})_{10} \\
 - \\
 (0 \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{1})_{10} \\
 \hline
 (0 \phantom{.} \phantom{9} \phantom{9} \phantom{9} \phantom{9} \phantom{9})_{10} \\
 - \\
 (0 \phantom{.} \phantom{3} \phantom{2} \phantom{6} \phantom{7})_{10} \\
 \hline
 (0 \phantom{.} \phantom{6} \phantom{7} \phantom{3} \phantom{2})_{10}
 \end{array}$$

### 7.1.2 The (r-1)'s complements:

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 follows:

$$r^n - r^{-m} - N$$

- (1) The 9's complement of  $(925.67)_{10}$  is  $N=925.67$ ,  $r=10$ ,  $n=3, m=2 \rightarrow$  the 9's complement equal to:  $(10^3 - 10^{-2} - 925.67)_{10} = (1000 - 0.01) - 925.67 = 999.99 - 925.67 = (74.32)_{10}$

$$\begin{array}{r}
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{.} \phantom{0} \phantom{0} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{.} \phantom{0} \phantom{0} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{.} \phantom{0} \phantom{0} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{.} \phantom{0} \phantom{0} \\
 \hline
 (0 \phantom{.} \phantom{9} \phantom{9} \phantom{9} \phantom{.} \phantom{9} \phantom{9})_{10} \\
 - \\
 (0 \phantom{.} \phantom{9} \phantom{2} \phantom{5} \phantom{.} \phantom{6} \phantom{7})_{10} \\
 \hline
 (0 \phantom{.} \phantom{7} \phantom{4} \phantom{.} \phantom{3} \phantom{2})_{10}
 \end{array}$$

- (2) The 9's complement of  $(0.3267)_{10}$  is  $N=0.3267$ ,  $r=10$ ,  $n=0, m=4 \rightarrow$  the 9's complement equal to:  $(10^0 - 10^{-4} - 0.3267)_{10} = (1 - 0.0001) - 0.3267 = 0.9999 - 0.3267 = (0.6732)_{10}$

$$\begin{array}{r}
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \phantom{0} \phantom{.} \phantom{0} \phantom{0} \phantom{0} \phantom{1} \\
 \hline
 (0 \phantom{.} \phantom{9} \phantom{9} \phantom{9} \phantom{9} \phantom{9})_{10} \\
 - \\
 (0 \phantom{.} \phantom{3} \phantom{2} \phantom{6} \phantom{7})_{10} \\
 \hline
 (0 \phantom{.} \phantom{6} \phantom{7} \phantom{3} \phantom{2})_{10}
 \end{array}$$



**Note:**

1. The 9's complement of a decimal number is formed simply by subtracting every digit from 9.
2. The 1's complement of a binary number is even simpler to form: 1's are changed to 0's and 0's to 1's .
3. The r's complement can be obtained from the (r-1)'s complement after the addition of  $r^m$  to the least significant digit.

Ex: the 2's complement of  $(10110100)_2$  obtained from the 1's complement as follows:

$$\begin{aligned} 1's \text{ complement of } (10110100)_2 &= (2^5 - 2^0)_{10} - (10110)_2 = (100000 - 1 - 10110)_2 = \\ &= (11111 - 10110)_2 = (01001)_2 \end{aligned}$$

$$2's \text{ complement} = 1's \text{ complement} + r^m = 1001 + 1 = (1010)_2$$

$$2's \text{ complement directly} = (2^5)_{10} - (10110)_2 = (100000 - 10110) = (1010)_2$$

$$\begin{array}{r} \phantom{0} \phantom{1} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \\ \phantom{0} \phantom{1} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \\ 0 \phantom{1} \phantom{1} \phantom{1} \phantom{1} \phantom{1} \phantom{0} \\ (\cancel{1} \ \cancel{0} \ \cancel{0} \ \cancel{0} \ \cancel{0} \ 0)_2 \quad - \\ (\phantom{1} \ 1 \ 0 \ 1 \ 1 \ 0)_2 \\ \hline (0 \ 0 \ 1 \ 0 \ 1 \ 0)_2 \end{array}$$

**5.1.2** The complement of the complement is restoring the number its original value.

Ex: the r's complement of N is  $r^n - N$

$$\text{The } r's \text{ complement of } (r^n - N) \text{ is } r^n - (r^n - N) = \cancel{r^n} - \cancel{r^n} + N = N$$

**H.W Prove the above example using (r-1)'s complement.**

## 8. Alphanumeric Codes:

Many applications of digital computers require handling name as well as letters. An alphanumeric code is a binary code of a group of elements consisting of 10 decimal digits, the 26 letter of alphabet, certain number of special symbols such as \$,;,',...etc. The total number of elements in an alphanumeric group is >36. So it must be coded with minimum number of bits ( $2^6=64$ ).

- ASCII code (American Standard Code for Information Interchange) 7-bit Length code.
- EBCDIC code (Extended BCD Interchange Code) 8-bit Length code.

Letters	ASCII code		Letters	ASCII code	
	decimal	binary		decimal	binary
A	65	100 0001	a	97	110 0001
B	66	100 0010	b	98	110 0010
:					
Z	90	101 1010	z	122	111 1010
0	48	011 0000			
1	49	011 0001			
:					
9	57	011 1001			
\$	36	010 0100			
(	40	010 1000			
?	47	010 1111			

## Lecture 2

### 9. Binary Logic and Gates

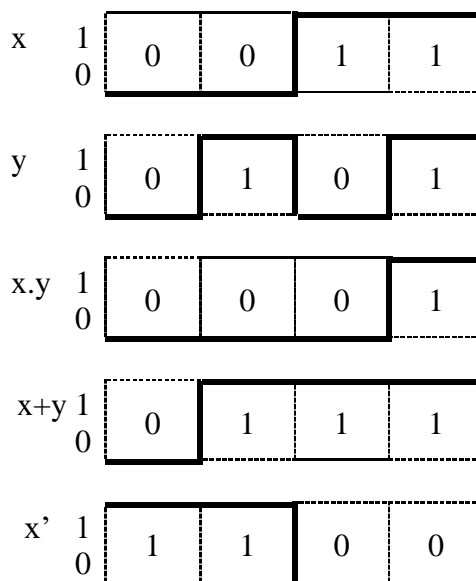
#### 7.1 Binary logic:

Binary logic deals with the variables take two discrete values true and false, yes and no. Binary logic is used to describe, in a mathematical way, the manipulation and processing of binary information. It particularly suited for analysis and design of digital systems. Binary Logic is equivalent to Boolean algebra.

#### 7.2 Definition of Binary logic:

Binary logic consists of binary variables and logic operation. The variables are designated by letters e.g. A, B, C, r, w, x, y with only two distinct values: 1 and 0. The basic logical operators are AND, OR and NOT.

AND similar to \* in binary arithmetic  
 OR similar to + in binary arithmetic but  $1+1=10$  (in binary arithmetic)  
 $1+1=1$  (in binary logic)

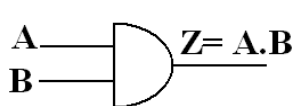


Timing diagram illustrate the response of each circuit to each of 4 input binary combinations (00,01,10,11)

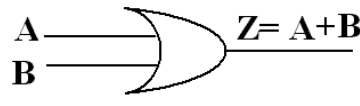
#### 7.3 Logic Gates:

Digital Circuits, Switching Circuits, Logic Circuits and Logic Gates are the same. Gates are block of hardware that produces a logic-1 or logic-0 output signal if input logic requirement are satisfied.

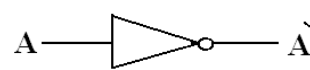
AND			OR			NOT	
Inputs		Output	Inputs		Output	Input	Output
x	y	x.y	x	y	x+y	x	$\bar{x}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		



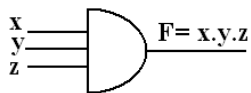
2-input AND gate



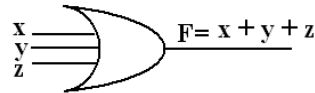
2-input OR gate



NOT gate or inverter



3-input AND gate



3-input OR gate

The mathematical system of binary logic is better known as Boolean or switching algebra. This algebra is conveniently used to describe the operation of complex network of digital circuits. Designers of digital systems use Boolean algebra to transform circuits diagram to algebraic expression and vice versa.

## 10.Integrated Circuits:

Is a small silicon semiconductor crystal, called a chip, containing electrical components such as transistor, diodes, resistor and capacitor. The various components are interconnected inside the chip to form electronic circuits. The chip is mounted on a metal or plastic packaged, and connection are welded to external pins to form the IC. IC's come in two types: 1) Flat package 2) dual package

As the technology of ICs has improved, number of gates that can be put on a single silicon chip has increased considerably.

SSI chip contains less than 10 gates.

MSI chip contains between 10 to 100 gates.

LSI chip contains between 100 to 1000 gates.

VLSI chip contains between thousands of gates.

### 11. Basic Identity of Boolean Algebra:

	a	b	
1	$X+0=X$	$X.1=X$	Identity Element
2	$X+1=1$	$X.0=0$	
3	$X+X=X$	$X.X=X$	
4	$\forall X \in B \exists \bar{X} \text{ s.t. } X+\bar{X}=1$	$X.\bar{X}=0$	Complement Definition
5	$\overline{\bar{X}}=X$		
6	$X+Y=Y+X$	$X.Y=Y.X$	Commutative Law
7	$X+(Y+Z)=(X+Y)+Z$	$X.(YZ)=(XY).Z$	Associative Law
8	$X.(Y+Z)=(X.Y)+(X.Z)$	$X+(Y.Z)=(X+Y).(X+Z)$	Distributive Law
9	$\overline{(X+Y)} = \bar{X}.\bar{Y}$	$\overline{(X.Y)} = \bar{X} + \bar{Y}$	Demorgan's Theorem
10	$X+XY=X$	$X(X+Y)=X$	Absorption Theorem

### 12. Boolean Functions:

Boolean functions are formed from binary variables, logic operators and equal sign. the function value can be either 1 or 0:

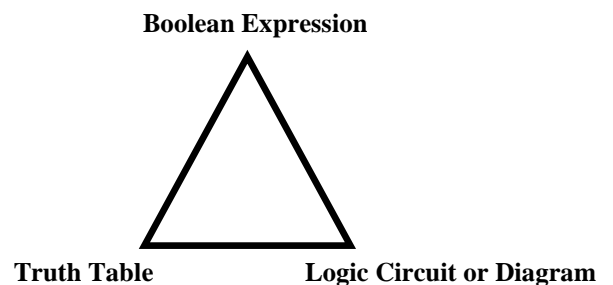
Ex:  $F1 = xy\bar{z}$

$F2 = x + \bar{y}z$

A Boolean function also represented in truth table. A Boolean function may be transformed from an algebraic expression into a logic diagram or circuit composed of AND, OR and NOT get.

**NOTE:** The operator precedence for evaluating Boolean expression is:

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



Boolean Expression	Logic Circuit	Truth Table (T.T.)																																			
$F1 = xy\bar{z}$		<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="5">Truth Table of F1</th> </tr> <tr> <th>x</th> <th>y</th> <th>z</th> <th>z'</th> <th>F1=xyz'</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	Truth Table of F1					x	y	z	z'	F1=xyz'	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	0	1	0	0	1	0
Truth Table of F1																																					
x	y	z	z'	F1=xyz'																																	
0	0	0	1	0																																	
0	0	1	0	0																																	
0	1	0	1	0																																	
0	1	1	0	0																																	
1	0	0	1	0																																	

		1	0	1	0	0																																																												
		1	1	0	1	1																																																												
		1	1	1	0	0																																																												
$F2 = x + \bar{y}z$		<table border="1"> <thead> <tr> <th colspan="6">Truth Table of F2</th> </tr> <tr> <th>x</th> <th>y</th> <th>z</th> <th>y'</th> <th>y'z</th> <th>x+y'z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> </tbody> </table>					Truth Table of F2						x	y	z	y'	y'z	x+y'z	0	0	0	1	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1
Truth Table of F2																																																																		
x	y	z	y'	y'z	x+y'z																																																													
0	0	0	1	0	0																																																													
0	0	1	1	1	1																																																													
0	1	0	0	0	0																																																													
0	1	1	0	0	0																																																													
1	0	0	1	0	1																																																													
1	0	1	1	1	1																																																													
1	1	0	0	0	1																																																													
1	1	1	0	0	1																																																													

### Complement of a function:

The complement of any function  $F$  is  $F'$  or  $\bar{F}$  its value can be obtained by interchange the (1's to 0's) and (0's to 1's) in the value of  $F$ . the complement of a function may be obtained algebraically through Demerger's theory.

Ex: Fined  $\bar{F}$  of  $F1$  and  $F2$ .

$$\bar{F1} = \overline{(xyz)} = \bar{x} + \bar{y} + \bar{z} = \bar{x} + \bar{y} + z$$

$$\bar{F2} = \overline{(x + \bar{y}z)} = \bar{x} \cdot \overline{\bar{y}z} = \bar{x} \cdot (\bar{\bar{y}} + \bar{z}) = \bar{x} \cdot (y + \bar{z})$$

## Lecture 3

### 13.Simplification of Boolean Functions using Boolean algebra.

Literal: is a single variable within the term that may be complemented or not. When a Boolean function is implemented with logic gates, each literal in Boolean function is designated an input to gate and each terms is implemented with gate. The minimization of a number of literal and the number of terms results in circuit with less equipment. Boolean algebra is a useful tool for simplifying digital circuit. Functions below simplified by using Boolean algebra.

Reduce the following Boolean expression using Boolean algebra to the indicated number of literals:

- $F1 = x'yz + x'yz' + xy$  to one literal
  - $= \underline{x'yz} + \underline{x'yz'} + xy$
  - $= x'y(z+z') + xy$  (Distributive law)
  - $= x'y \cdot 1 + xy$  (Complement definition)
  - $= x'y + xy$  (Identity element)



$$=y(x'+x) \quad (\text{Distributive law})$$

$$=y.1 \quad (\text{Complement definition})$$

$$F1=y \quad (\text{Identity element})$$

2)  $F2=x+(x'y)$  to two literals

$$=(x+x')(x+y) \quad (\text{Distributive law})$$

$$=1.(x+y) \quad (\text{Complement definition})$$

$$F2=x+y \quad (\text{Identity element}).$$

3)  $F3=xy+x'z+yz$  to four literals

$$=xy+x'z+yz.1 \quad (\text{Identity element})$$

$$=xy+x'z+yz(x+x') \quad (\text{Complement definition})$$

$$=xy+x'z+xyz+x'yz \quad (\text{Distributive law})$$

$$=xy(1+z)+x'z(1+y) \quad (\text{Distributive law})$$

$$=xy.1+x'z.1 \quad (x+1)=1$$

$$F3=xy+x'z \quad (\text{Identity element})$$

Truth Table of $F3=xy+x'z$						
x	Y	z	x'	xy	x'z	$F3=xy+x'z$
0	0	0	1	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	0	1
1	1	1	0	1	0	1

Logic Circuit of  $F3=xy+x'z$

4)  $F4 = x + y(z + \bar{x} + \bar{z})$  to two literals

$$=x + y(z + \bar{x}\bar{z}) \quad (\text{Demorgan's theory})$$

$$=x + y(z + \bar{x})(z + \bar{z}) \quad (\text{Distributive law})$$

$$=x + y(z + \bar{x}).1 \quad (\text{Complement definition})$$

$$=x + yz + y\bar{x} \quad (\text{Identity element})$$

$$=(x + y\bar{x}) + yz \quad (\text{Commutative law})$$

$$=(x + y)(x + \bar{x}) + yz \quad (\text{Distributive law})$$

$$=(x + y).1 + yz \quad (\text{Complement definition})$$

$$=x + y + yz \quad (\text{Identity element})$$

$$=x + y(1 + z) \quad (\text{Distributive law})$$

$$=x + y.1 \quad (\text{Complement definition})$$

$$F4 = x + y \quad (\text{Identity element})$$

**HW.**

**Q1)** Represent the decimal number (1729) using the following codes:

- 1)BCD                  2)Excess-3
- 3)(8,4,-2,-1)    4) (2,4,2,1) self-complemented

**Q2)** Reduce the following Boolean expression using Boolean algebra to the indicated number of literals:

- 1.  $XYZ + \bar{X}Y + XY\bar{Z}$     to one literal
- 2.  $(\bar{A} + \bar{C})(\bar{A} + C)(A + B + \bar{C}D)$     to four literals.
- 3.  $\overline{(X + Y)}(\bar{X} + Y)$     to two literals
- 4.  $[A\bar{B}(C + BD) + \bar{A}\bar{B}]C$     to two literals

**Q3)** Draw the logic diagram of the following Boolean function **without simplifying** them

- 1.  $F1 = B\bar{C} + AD$
- 2.  $F2 = B(\bar{C} + A)$
- 3.  $F3 = \overline{(X + Y)}(\bar{X} + Y)$

**14.Canonical and Standard forms:**

**12. A Canonical form:** canonical forms (Sum of Minterms or Product of Maxterms) are used to obtain the function from the given truth table

x	y	z	Minterms	Notation	Maxterms	Notation
0	0	0	$x'y'z'$	m <sub>0</sub>	$(x+y+z)$	M <sub>0</sub>
0	0	1	$x'y'z$	m <sub>1</sub>	$(x+y+z')$	M <sub>1</sub>
0	1	0	$x'yz'$	m <sub>2</sub>	$(x+y'+z)$	M <sub>2</sub>
0	1	1	$x'yz$	m <sub>3</sub>	$(x+y'+z')$	M <sub>3</sub>
1	0	0	$xy'z'$	m <sub>4</sub>	$(x'+y+z)$	M <sub>4</sub>
1	0	1	$xy'z$	m <sub>5</sub>	$(x'+y+z')$	M <sub>5</sub>
1	1	0	$xyz'$	m <sub>6</sub>	$(x'+y'+z)$	M <sub>6</sub>
1	1	1	$xyz$	m <sub>7</sub>	$(x'+y'+z')$	M <sub>7</sub>
			Variable Primed if =0 Imprimed=1		Variable Primed if =1 Imprimed=0	

**12. A .1 Sum of Minterms:**

A Boolean function may be expressed algebraically as a sum of minterms from a given truth table by:

- Step1: forming a minterm for each combination of the variables which produce 1 in the function.
- Step2: OR all of the minterms in step1.

Example: From the given truth table express F as a sum of minterms

Given				Solution	
Inputs			Output	Step1	Step2
x	y	z	F	minterms	Sum of minterms
0	0	0	0		$F = x'y'z + x'yz + xyz$
0	0	1	1	$x'y'z$	$F = m_1 + m_3 + m_7$
0	1	0	0		$F(x,y,z) = \Sigma(1,3,7)$
0	1	1	1	$x'yz$	
1	0	0	0		
1	0	1	0		
1	1	0	0		
1	1	1	1	$xyz$	

From the table F' can be expressed as a sum of minterms as follows:

Step1: forming a minterm for each combination of the variables which produce 0 in the function.

Step2: OR all of the minterms in step1.

Example: From the given truth table express F' as a sum of minterms

Given				Solution	
Inputs			Inputs	Step1	Step2
x	y	z	F	minterms	Sum of minterms
0	0	0	0	$x'y'z'$	$F' = x'y'z' + x'yz' + xy'z' + xy'z + xyz'$
0	0	1	1		$F' = m_0 + m_2 + m_4 + m_5 + m_6$
0	1	0	0	$x'yz'$	$F'(x,y,z) = \Sigma(0,2,4,5,6)$
0	1	1	1		
1	0	0	0	$xy'z'$	
1	0	1	0	$xy'z$	
1	1	0	0	$xyz'$	
1	1	1	1		

## 12. A .2 Product of Maxterms:

A Boolean function may be expressed algebraically as a product of maxterms from a given truth table by:

Step1: forming a maxterms for each combination of the variables which produce 0 in the function.

Step2: form the AND of all the maxterms in step1.

Example: From the given truth table express F as a product of maxterms

Given				Solution	
Inputs			Inputs	Step1	Step2
x	y	z	F	maxterms	Product of maxterms
0	0	0	0	$(x+y+z)$	$F=(x+y+z)(x+y'+z)(x'+y+z)(x'+y+z')(x'+y'+z)$
0	0	1	1		$F= M_0.M_2.M_4. M_5.M_6$
0	1	0	0	$(x+y'+z)$	$F(x,y,z)=\Pi(0,2,4,5,6)$
0	1	1	1		
1	0	0	0	$(x'+y+z)$	
1	0	1	0	$(x'+y+z')$	
1	1	0	0	$(x'+y'+z)$	
1	1	1	1		

From the table F' can be expressed as follows:

Step1: forming a maxterm for each combination of the variables which produce 1 in the function.

Step2: form the AND of all the maxterms in step1.

Example: From the given truth table express F' as a product of maxterms

Given				Solution	
Inputs			Inputs	Step1	Step2
x	y	z	F	maxterms	Product of maxterms
0	0	0	0		$F'=(x+y+z')(x+y'+z')(x'+y'+z')$
0	0	1	1	$(x+y+z')$	$F'=M_1.M_3.M_7$
0	1	0	0		$F'(x,y,z)=\Pi(1,3,7)$
0	1	1	1	$(x+y'+z')$	
1	0	0	0		
1	0	1	0		
1	1	0	0		
1	1	1	1	$(x'+y'+z')$	

## 12. B. Standard Forms:

Sum terms: single variable or logical sum of several variables such as (A, B, (x+y), (A+C')).

Product terms: single variable or logical product of several variables such as (x, y, AB', CD')

Note: the expression  $x+y'z$  (not sum term nor product terms)

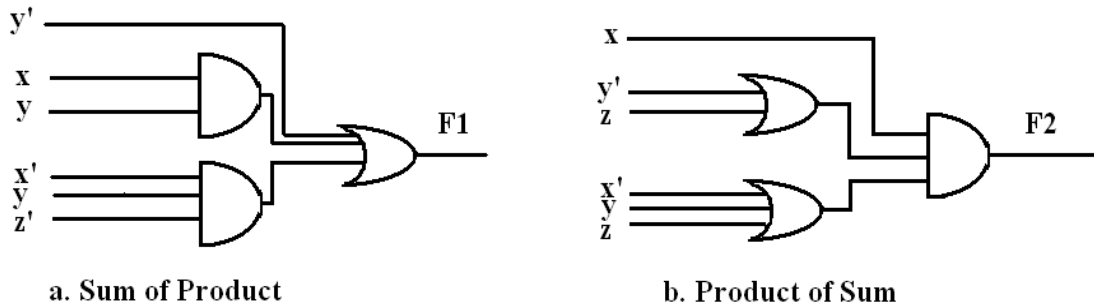
**12. B.1. Sum of Products (SOP):** is a Boolean expression containing AND terms, called product terms, of one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of product is  $F1=y'+xy+x'yz'$

The expression contains three product terms ( $y'$  one literal,  $xy$  two literals and  $x'yz'$  three literals). Their sum is in effect an OR operation. The logic diagram if a sum-of-product expression consist of a group of AND gates followed by a single OR gate.

**12. B.2 Product of Sums (POS):** is a Boolean expression containing OR terms, called sum terms, of one or more literals each. The product denotes the ANDing of these terms. An example of a function expressed as a product of sum is

$$F2=x(y'+z)(x'+y+z)$$

The expression contains three sum terms (x one literal,  $y'+z$  two literals and  $x'+y+z$  three literals). The product is an AND operation. The logic diagram if a product-of-sum expression consist of a group of OR gates followed by a single AND gate.



### Two-level implementation

Note: a Boolean function may be expressed in a nonstandard form

Ex:  $F = (AB + CD)(A'B' + C'D')$  is neither in sum of products nor product of sums. It can be change to the standard form using the distributive law:

$$F = ABA'B' + ABC'D' + CDA'B' + CDC'D'$$

$$= 0 + ABC'D' + A'B'CD + 0$$

$$F = ABC'D' + A'B'CD \quad \text{sum of products}$$

**H.W. Express F in a product of sums.**

Example: From the given truth table express F as a sum of minterms then simplify as a sum of product

Given			
x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Solution: the function equal to 1 in  $\{(2,x'yz'),(3,x'yz),(6,xyz')\}$

So  $F=x'yz'+x'yz+xyz'$  (Sum of Minterms)

Simplification of F

$$F=x'yz'+x'yz+xyz'$$

$$=x'y(z'+z)+xyz' \quad (\text{distributive law})$$

$$=x'y.1+xyz' \quad (\text{complement definition})$$

$$=x'y+xyz' \quad (\text{identity element})$$

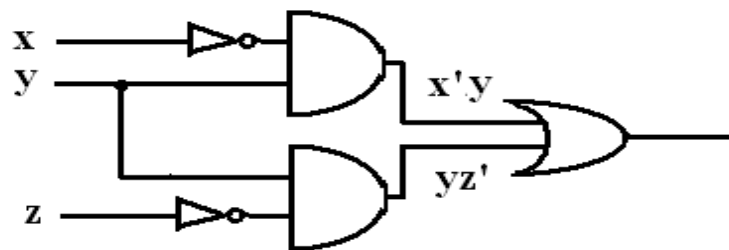
$$=y(x'+xz') \quad (\text{distributive law})$$

$$=y(x'+x)(x'+z') \quad (\text{distributive law})$$

$$=y.1.(x'+z') \quad (\text{complement definition})$$

$$=y(x'+z') \quad (\text{identity element})$$

$$=x'y+yz' \quad (\text{distributive law})$$



**Logic Circuit of  $F = x'y + yz'$**

Example: From the given truth table express F

as a product of maxterms then simplify as a product of sum

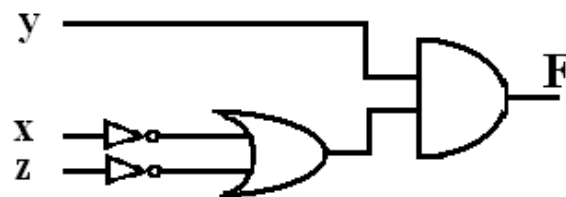
Solution: the function equal to 0 in  $\{(0,x+y+z),(1,x+y+z'),(4,x'+y+z), (5,x'+y+z'),(7,x'+y'+z')\}$

So  $F=(x+y+z)(x+y+z')(x'+y+z)(x'+y+z')(x'+y'+z')$  product of maxterms

Given			
x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Simplification of F

$$\begin{aligned}
F &= (\underline{x+y+z}) (\underline{x+y+z'}) (\underline{x'+y+z}) (\underline{x'+y+z'}) (\underline{x'+y'+z'}) \\
&= [(x+y)+zz'] [(x'+y)+zz'] (\underline{x'+y'+z'}) && \text{(distributive law)} \\
&= [(x+y)+0] [(x'+y)+0] (\underline{x'+y'+z'}) && \text{(complement definition)} \\
&= (\underline{x+y}) (\underline{x'+y}) (\underline{x'+y'+z'}) && \text{(identity element)} \\
&= (y+xx') (\underline{x'+y'+z'}) && \text{(distributive law)} \\
&= (y+0) (\underline{x'+y'+z'}) && \text{(complement definition)} \\
&= y(\underline{x'+y'+z'}) && \text{(identity element)} \\
&= x'y+yy'+yz' && \text{(distributive law)} \\
&= x'y+0+yz' && \text{(complement definition)} \\
&= x'y+yz' && \text{(identity element)} \\
&= y(x'+z') && \text{(distributive law) to convert the function from SOP to POS}
\end{aligned}$$



Logic Circuit of  $F=y(x'+z')$

## 12.C convert functions to the canonical forms:

### 12.C.1 conversion to sum of minterms:

It is sometimes convenient to express function in its sum of minterms form by:

- Expanding the expression in to sum of AND terms (SOP)
- Each AND term is inspected to see if it contains all the variables. If it missing one or more variables, it is ANDed with an expression  $(x+x')$ , where x is one of the missing variable

**Example:** Express the Boolean function  $F=A+B'C$  in a sum of minterms.

**Solution:** the function F has three variables A,B and C. it is in SOP standard form the first product term (A) missing two variable (B,C); therefore

$$A = A(B+B') = AB + AB'$$

This terms still missing one variable (C):

$$\begin{aligned} A &= AB(C+C') + AB'(C+C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term (B'C) missing one variable (A):

$$B'C = B'C(A+A') = B'CA + B'CA' \quad \text{rearrange the variable alphabetically}$$

$$B'C = AB'C + A'B'C$$

Combining all terms, we have:

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + \underline{AB'C} + AB'C' + \underline{AB'C} + A'B'C \end{aligned}$$

Since  $(x+x=x)$  we can eliminate one of the underlined term

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

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

### 12.C.2 Conversion to product of maxterms:

To express function in its product of maxterms form by:

- Expanding the expression in to product of **OR** terms (POS), using distributive law
- Each **OR** term is inspected to see if it contains all the variables. If it missing one or more variables, it is **OR**ed with an expression  $(xx')$ , where  $x$  is one of the missing variable

Example: Express the Boolean function  $F = xy + x'z$  in a product of maxtermes.

First: convert the function into OR terms (POS) by using distributive law:

$$\begin{aligned} F &= xy + x'z = \quad \quad \quad \text{let } a=xy \\ F &= a + x'z = (a + x')(a + z) = (xy + x')(xy + z) \\ &= (x' + x)(x' + y)(z + x)(z + y) \\ F &= (x' + y)(z + x)(z + y) \text{POS} \end{aligned}$$

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

$$\begin{aligned} (x' + y) &= (x' + y) + zz' = (x' + y + z)(x' + y + z') \\ (z + x) &= (z + x) + yy' = (z + x + y)(z + x + y') = (x + y + z)(x + y' + z) \\ (z + y) &= (z + y) + xx' = (z + y + x)(z + y + x') = (x + y + z)(x' + y + z) \end{aligned}$$



Combining all terms and removing all those that appear more than once, we finally obtain:

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$

$$= M_0 M_2 M_4 M_5$$

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

### 12.C.3 Conversions between Canonical forms:

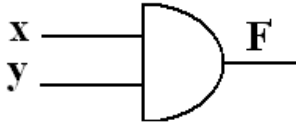
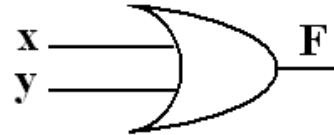
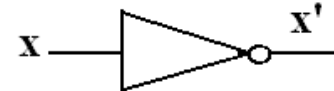
To convert from one canonical form to another: interchanging the symbols  $\Pi$  and  $\Sigma$  then list those numbers missing in the original form. In order to find the missing terms, one must realize the total number of minterms and maxterms is  $2^n$ , where  $n$  is the number of binary variables in the function.


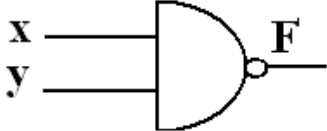
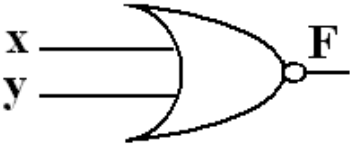

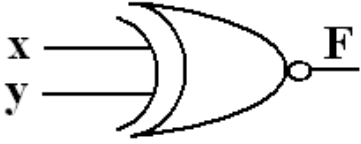
Example: convert the Boolean function  $F(x,y,z) = \Sigma(0,2,4,5)$  to the other canonical form.

The number of variables is three ( $x, y, z$ ), therefore the total numbers is in the range  $(0..2^3-1) = (0..7)$  therefore,  $F(x,y,z) = \Pi(1,3,6,7)$

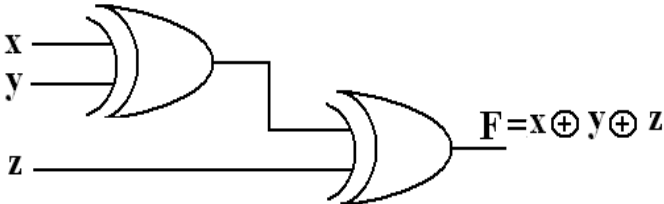
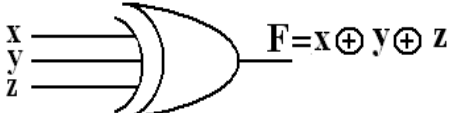
## Lecture 4

### 15. Other Logical Operations:

Name	Graphical symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table border="1"> <thead> <tr> <th>x</th> <th>y</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT, Inverter			<table border="1"> <thead> <tr> <th>x</th> <th>x'</th> </tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	x	x'	0	1	1	0									
x	x'																	
0	1																	
1	0																	

Buffer			<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>x</td><td>x</td></tr> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td></tr> </table>	x	x	0	0	1	1									
x	x																	
0	0																	
1	1																	
NAND		$F = (xy)'$ $= x \uparrow y$	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>x</td><td>y</td><td>F</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x+y)'$ $= x \downarrow y$	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>x</td><td>y</td><td>F</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = x'y + xy'$ $= x \oplus y$	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>x</td><td>y</td><td>F</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR		$F = x'y' + xy$ $= (x \oplus y)'$ $= x \odot y$	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>x</td><td>y</td><td>F</td></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Exclusive OR (XOR): odd function equal to 1 if the numbers the input variables have an odd number of 1's and 0 otherwise. The construction of 2-input Exclusive-OR function is shown below. It is normally implemented by cascading 2-input gates, as shown in below:

<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>x</td><td>y</td><td>z</td><td><math>F = x \oplus y \oplus z</math></td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td></tr> </table>	x	y	z	$F = x \oplus y \oplus z$	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	0	1	0	0	1	1	0	1	0	1	1	0	0	1	1	1	1	 <p style="text-align: center;">Using 2-input gates</p>  <p style="text-align: center;">3-input gate</p>
x	y	z	$F = x \oplus y \oplus z$																																		
0	0	0	0																																		
0	0	1	1																																		
0	1	0	1																																		
0	1	1	0																																		
1	0	0	1																																		
1	0	1	0																																		
1	1	0	0																																		
1	1	1	1																																		

The following identities apply to Exclusive –OR operation:

$$x \oplus 0 = x \quad , \quad x \oplus x = 0 \quad , \quad x \oplus y' = x' \oplus y = (x \oplus y)'$$

$$x \oplus 1 = x' \quad , \quad x \oplus x' = 1 \quad ,$$

Equivalence or Exclusive NOR: even function equal to 1 if the number of 0's in the input variables are even and 0 otherwise.

XOR and Equivalence are both commutative and associative.

$$x \uparrow y = (xy)' = x' + y' \quad \quad \quad y \uparrow x = (yx)' = y' + x' = x' + y' \quad \therefore \text{NAND is commutative}$$

$$x \downarrow y = (x+y)' = x'y' \quad \quad \quad y \downarrow x = (y+x)' = y'x' = x'y' \quad \therefore \text{NOR is commutative}$$

$$(x \uparrow y) \uparrow z = (xy)' \uparrow z = ((xy)' z)' = (xy)'' + z' = xy + z'$$

$$x \uparrow (y \uparrow z) = x \uparrow (yz)' = (x' (yz)')' = x' + (yz)'' = x' + yz \quad \therefore \text{NAND is not associative.}$$

H.W. Neither prove that NOR is associative or not.

### 16.NAND and NOR gates.

Two other gates, NAND and NOR are often used in computer. To show any function can be implemented with NAND or NOR gates, we need only show that the logical operations of AND & OR can be obtained from NAND or NOR only.

	AND gate	OR gate
Using NAND		
Using NOR		

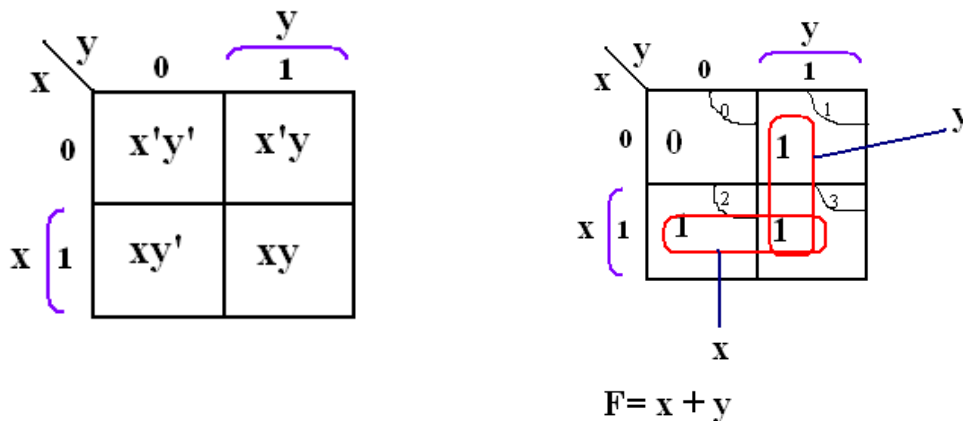
### 13. Map Simplification:

Boolean function may be simplified by algebraic means, but this procedure lacks specific rules to predict each succeeding step in the manipulation process. The map method provides a simple straight forward procedure for minimizing Boolean functions. The map is a diagram made up of squares each square represents one minterms.

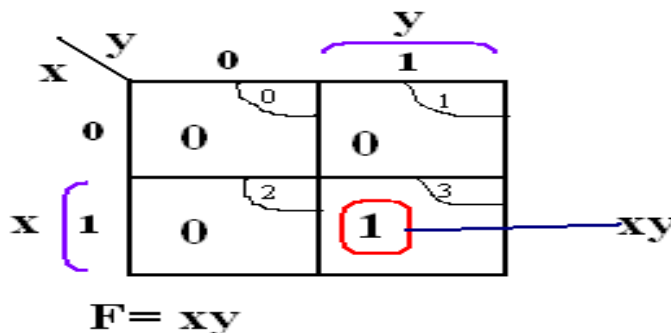
### 13.1 Two and Three variables maps:

in two variable map, there are 4 minterms hence the map consists of four squares , one for each minterm.

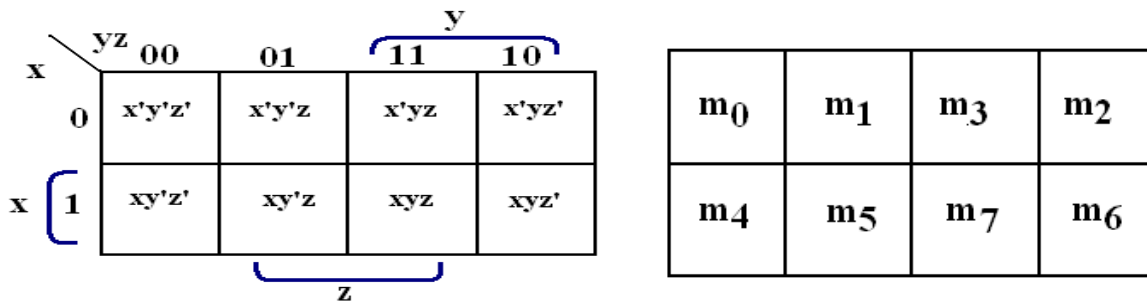
Example: simplify F using map method, where  $F(x,y)=\sum(1,2,3)$



Example: simplify F using map method, where  $F(x,y)=\sum(3)$



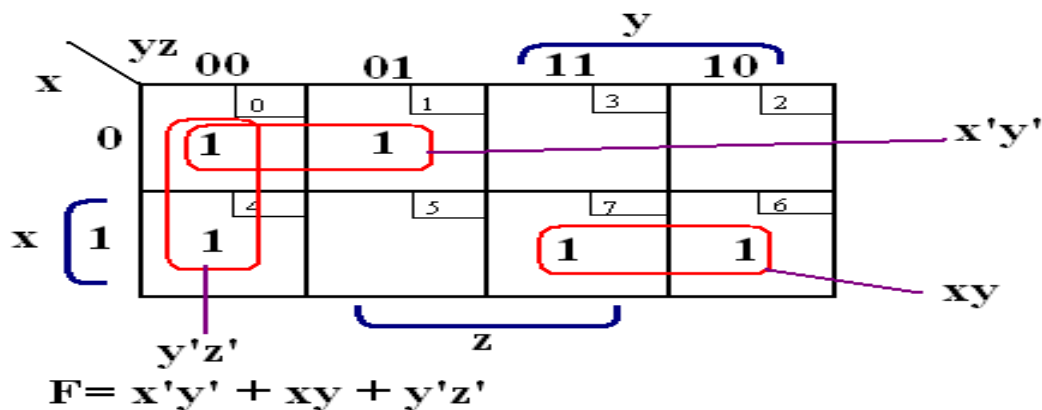
A three variable map is shown below, note that the minterms are arranged, not in binary sequence, the characteristic of this sequence is that the only one bit changed from (1 to 0) or from (0 to 1) in the listing sequence.



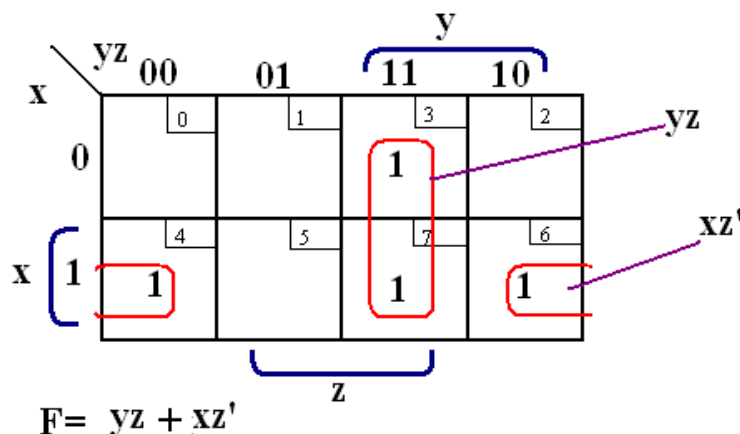
The basic property proposed by the adjacent squares that only two adjacent squares in the map differ by only one variable which is primed in one square and unprimed in the other. For example  $m_6$  and  $m_7$  hence, the sum of two minterms in the adjacent square can be simplified to a simple AND term consisting of only two literals

$$m_6 + m_7 = xy'z' + xyz = xy(z' + z) = xy \cdot 1 = xy$$

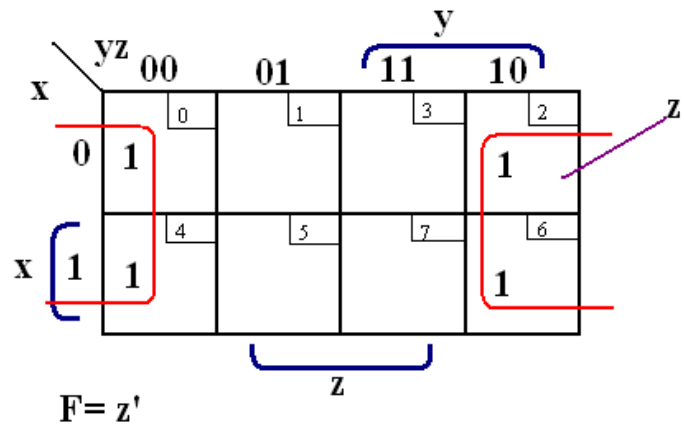
Example: simplify F using map method, where  $F(x,y,z) = \sum(0,1,4,6,7)$



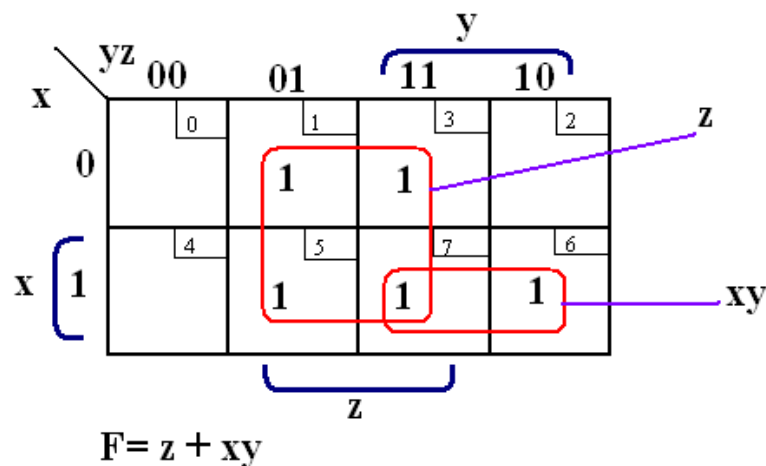
Example: simplify F using map method, where  $F(x,y,z) = \sum(3,4,6,7)$



Example: simplify F using map method, where  $F(x,y,z) = \sum(0,2,4,6)$



Example: simplify F using map method, where  $F(x,y,z) = \sum(1,3,5,6,7)$



Note:- any combination of 4 adjacent squares in the 3-variables map, which represents the ORing of four adjacent minterms will result in an expression of only one literals.

Example: simplify F using map method,

where  $F = A'B'C' + A'C + A'B + AB'C + BC$

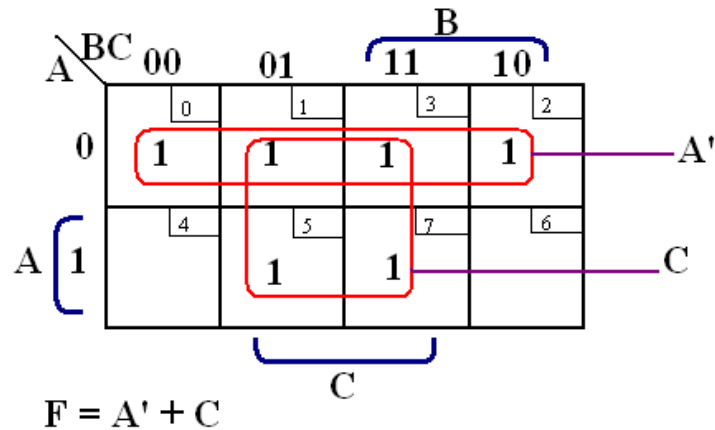
Sol.: the terms need to be expressed as a sum of minterms as explain before, so A'C missing B, A'B missing C and BC missing A

$$F = A'B'C' + A'C(B+B') + A'B(C+C') + AB'C + BC(A+A')$$

$$= A'B'C' + A'BC + A'B'C + A'BC + A'BC' + AB'C + ABC + A'BC$$

$$= A'B'C' + A'B'C + A'BC' + A'BC + AB'C + ABC$$

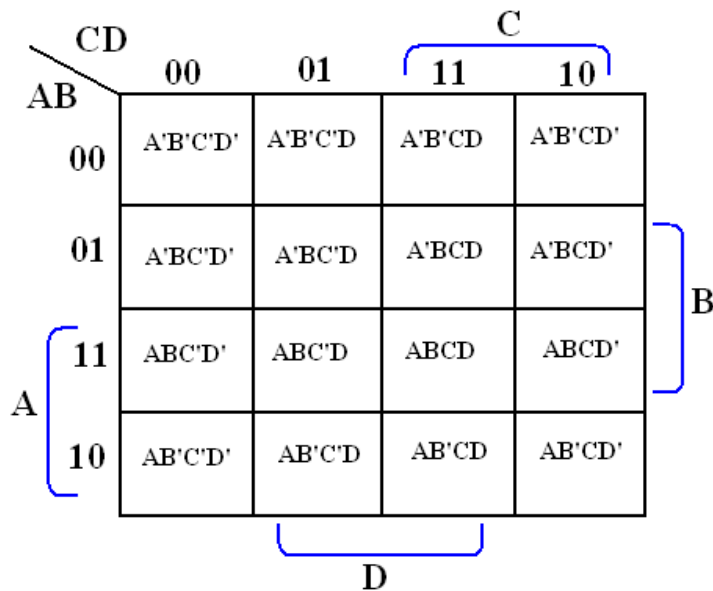
$\therefore F(A,B,C) = \sum(0,1,2,3,5,7)$  and the simplification of F using the map methods is as follows:



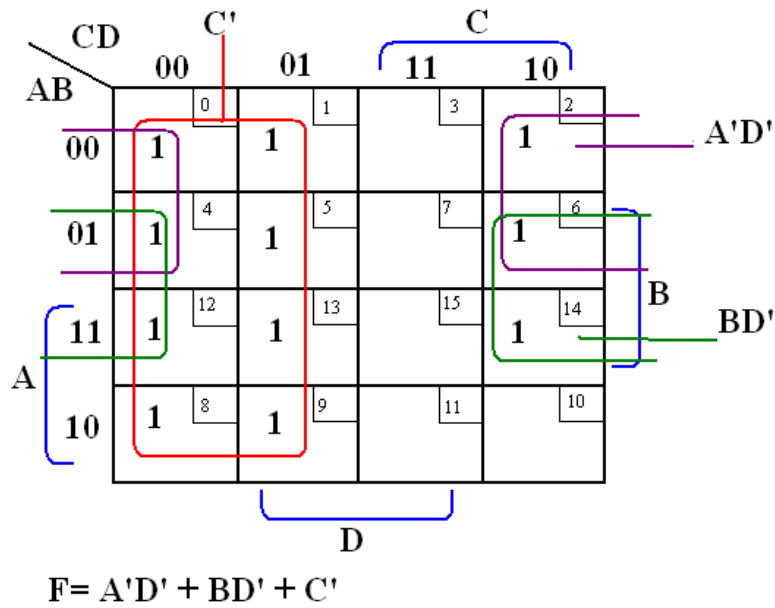
### 13.2 Four variables map:-

The combinations of adjacent squares that is useful during the simplification process easily determined for inspection of the 4-variable map

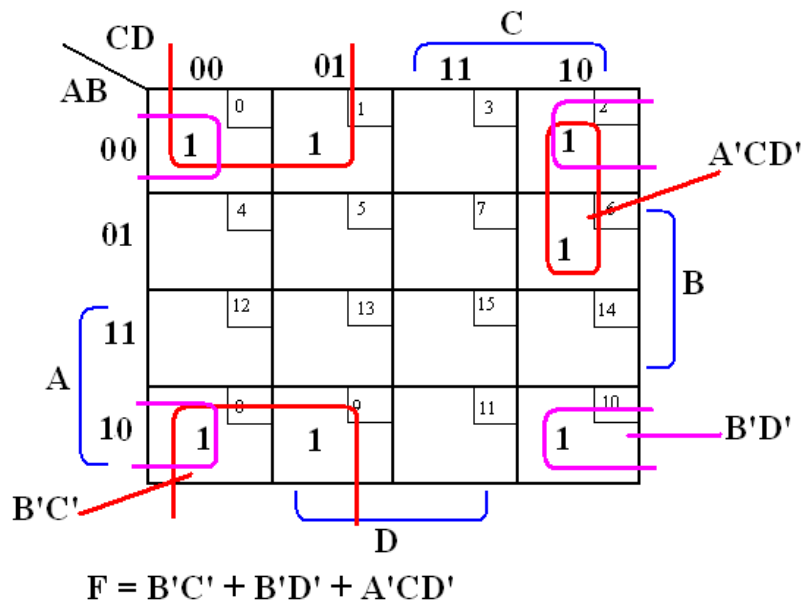
- ☒  $(2^0)$  One square represents one minterm, given a term of 4 literals
- ☒  $(2^1)$  Two adjacent squares represents a term of 3 literals.
- ☒  $(2^2)$  Four adjacent squares represents a term of 2 literals.
- ☒  $(2^3)$  Eight adjacent squares represents a term of 1 literal.
- ☒  $(2^4)$  Sixteen adjacent squares the function equal to 1 (constant)



Example: simplify F using map method, where  $F(A,B,C,D)=\sum(0,1,2,4,5,6,8,9,12,13,14)$



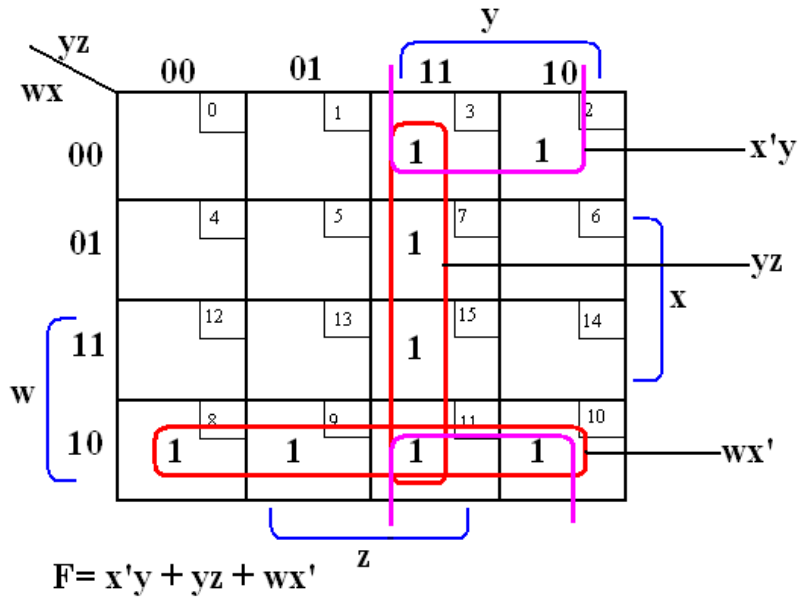
Example: simplify F using map method, where  $F(A,B,C,D) = \sum(0,1,2,6,8,9,10)$



Example: simplify F using map method,

Where,  $F(w,x,y,z) = \sum(2,3,7,8,9,10,11,15)$

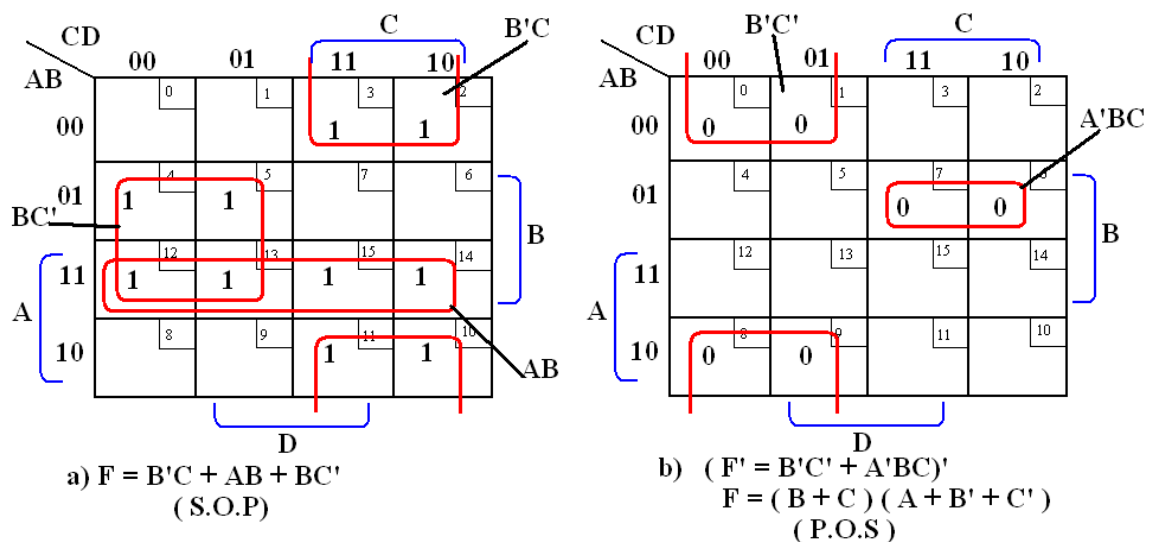




### 14. Product of Sum Simplification.

The minimized Boolean function derived from the map in all previous example were expressed in the sum of product form, with a minor or modification the product of sums forms can be obtained.

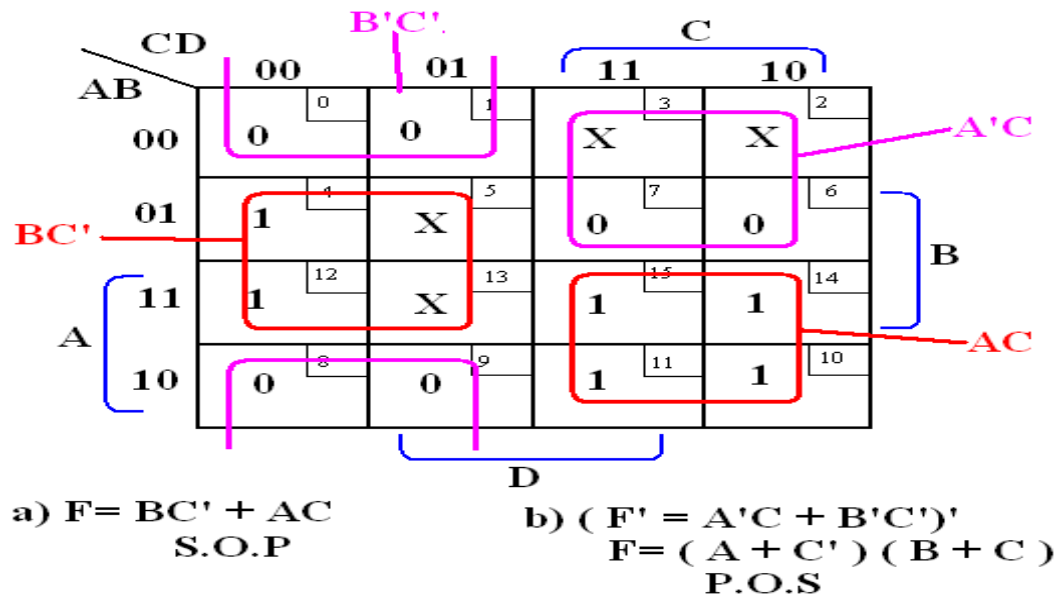
Example: simplify  $F$  using map method as a) sum of products b) product of sums, where  $F(A,B,C,D) = \sum(2,3,4,5,10,11,12,13,14,15)$



### 15. Don't care Conditions:

The don't care conditions can be used on a map to provide further simplification of the function. To distinguish the don't care conditions from 1's and 0's, an X will be used. When choosing the adjacent square to simplify the function in a map, the X's may be assumed to be either (0 or 1) whichever gives the simple expression.

Example: simplify F using map method as a) sum of products b) product of sums, where  $F(A,B,C,D)=\sum(4,10,11,12,14,15)$  and the don't care conditions  $d(A,B,C,D)=\sum(2,3,5,13)$



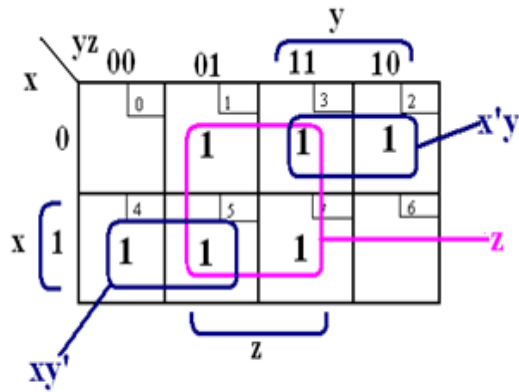
## 16. NAND and NOR implementations.

### 16.1. NAND circuits:

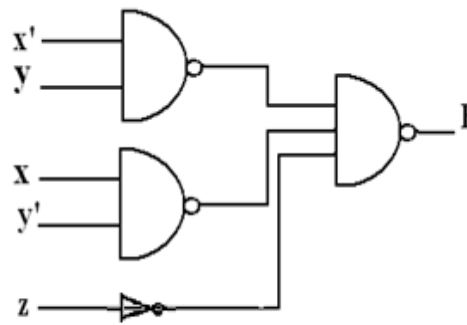
The Boolean functions can be implemented with two levels of NAND gates by:

- Simplifying the function and express in sum of product term.
- Draw NAND gate for each product term of expression that has at least two literals. The inputs of each of each NAND gates are the literals of the term. This constitutes a group of first-level gates.
- Draw single gate using the AND-invert graphic symbol in the second level, with inputs coming from outputs of the first level gates.
- A term with single literal requires an inverter in the first level.

Example: implement F with NAND gates, where  $F(x,y,z)=\sum(1,2,3,4,5,7)$



$$F = x'y + xy' + z$$

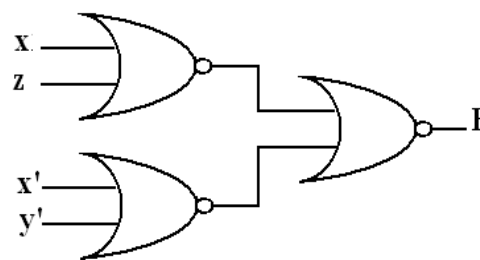
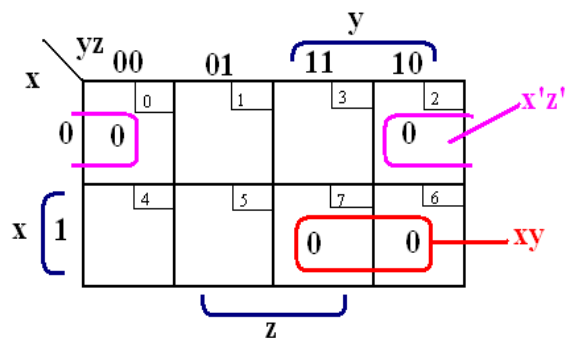


### 16.2. NOR circuits:

The Boolean functions can be implemented with two levels of NOR gates by:

- Simplifying the function and express in product of sums term.
- Draw NOR gate for each sum term of expression that has at least two literals. The inputs of each of each NOR gates are the literals of the term. This constitutes a group of first-level gates.
- Draw single gate using the OR-invert graphic symbol in the second level, with inputs coming from outputs of the first level gates.
- A term with single literal requires an inverter in the first level.

Example: implement F with NOR gates, where  $F(x,y,z) = \sum(1,3,4,5)$

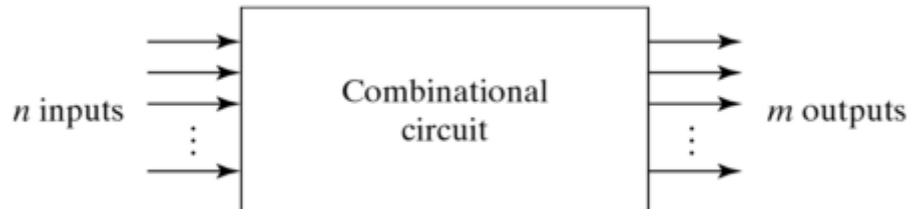


- $(F' = x'z' + xy)'$   
 $F = (x + z)(x' + y')$   
 P. O. S

## Lecture 5

### 17. Combinational Circuits

Logic circuits whose outputs at any time are determined directly and only from the present input combination

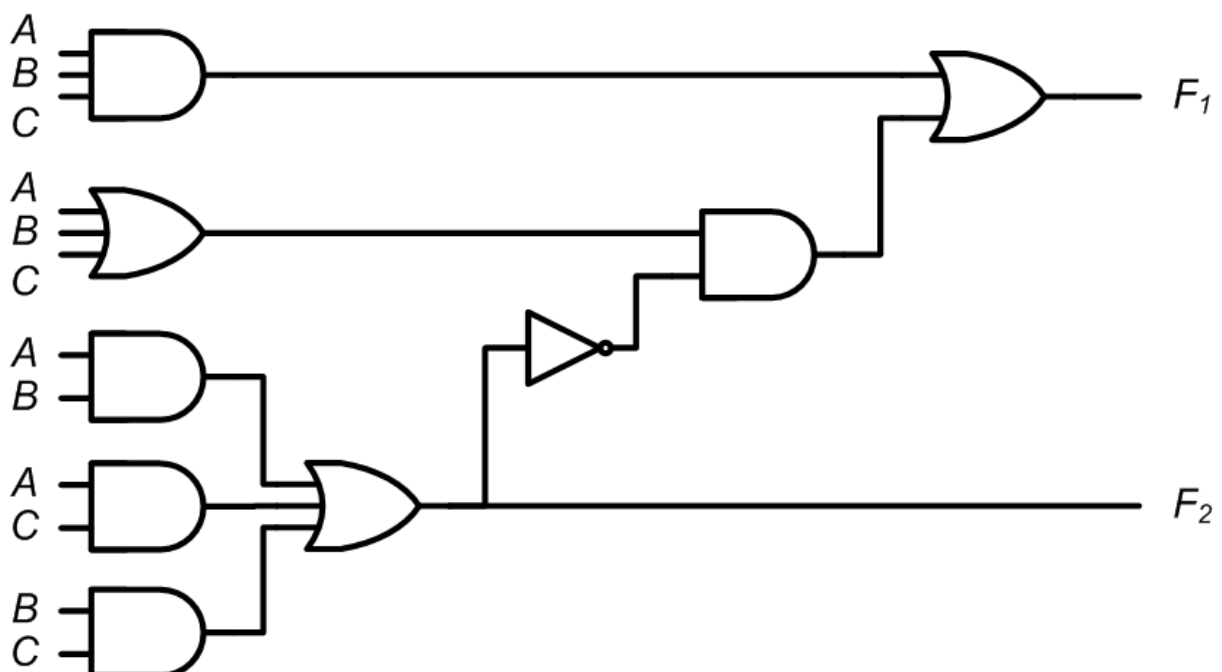


#### A. Analysis Procedure:

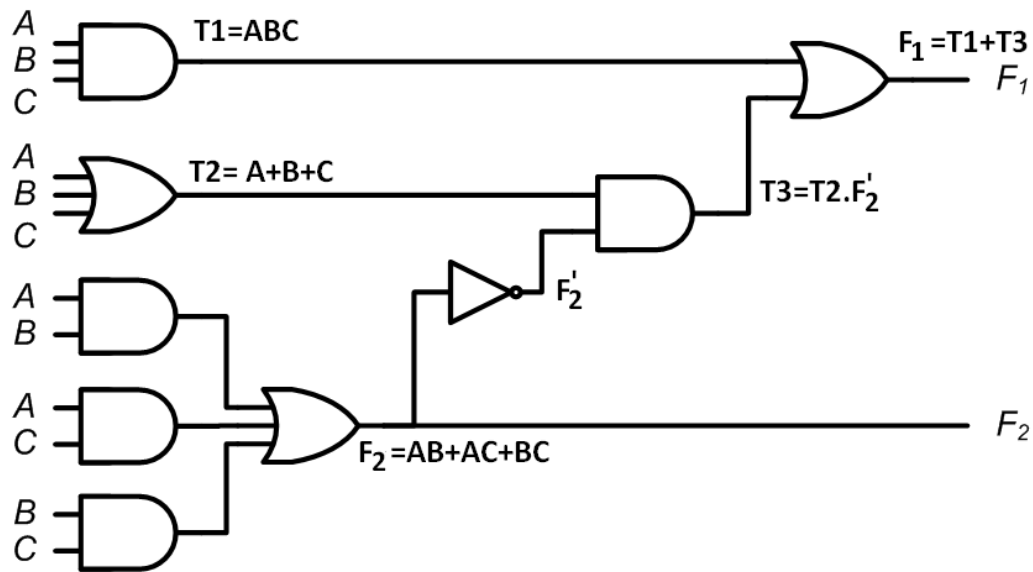
##### Boolean Expression Approach:

- ☒ Label all gate outputs that are functions of the input variables only. Determine the functions.
- ☒ Label all gate outputs that are functions of the input variables and previously labeled gate outputs, and find the functions.
- ☒ Repeat previous step until all the primary outputs are obtained.

**Ex.1** Analyze the circuit shown in the following figure:



Solution: Step1:



Solution : Step2:

$$\bar{F}_2 = (\bar{A} + \bar{B})(\bar{A} + \bar{C})(\bar{B} + \bar{C})$$

$$T3 = T2 \cdot \bar{F}_2$$

$$T3 = (A + B + C)(\bar{A} + \bar{B})(\bar{A} + \bar{C})(\bar{B} + \bar{C})$$

$$T3 = (A + B + C)(\bar{A} + \bar{B}\bar{C})(\bar{B} + \bar{C})$$

$$T3 = (A + B + C)(\bar{A}\bar{B} + \bar{A}\bar{C} + \bar{B}\bar{C}\bar{B} + \bar{B}\bar{C}\bar{C})$$

$$T3 = (A + B + C)(\bar{A}\bar{B} + \bar{A}\bar{C} + \bar{B}\bar{C})$$

$$T3 = \overbrace{A\bar{A}\bar{B}}^{=0} + \overbrace{A\bar{A}\bar{C}}^{=0} + A\bar{B}\bar{C} + \overbrace{B\bar{A}\bar{B}}^{=0} + B\bar{A}\bar{C} + \overbrace{B\bar{B}\bar{C}}^{=0} + C\bar{A}\bar{B} + \overbrace{C\bar{A}\bar{C}}^{=0} + \overbrace{C\bar{B}\bar{C}}^{=0}$$

$$T3 = A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$$

$$F_1 = T1 + T3$$

$$F_1 = \underbrace{ABC}_7 + \underbrace{A\bar{B}\bar{C}}_4 + \underbrace{\bar{A}\bar{B}C}_2 + \underbrace{\bar{A}\bar{B}C}_1$$

**HW. Derive the T.T. of the F1 and F2**

## B. Design Procedure:

The steps to design combinational circuits are as the following:

1. Understand the problem
2. Determine the number of input and output variables that are needed.
3. Give symbols for the stated input and output.
4. Construct a truth table that defines the relationship between the input and output.
5. Obtain the Boolean function or the logical expression from the truth table in step 4, using K-map map or other known methods.
6. Draw a logic circuit based on the expression obtained from step 5 above.

**Ex.2.** Design a circuit that accept 3-bit binary number and produce 1 if the number of 0's greater than the number of 1's in the input combination and produce 0 otherwise.

**Solution:**

☒ **Generate the truth table**

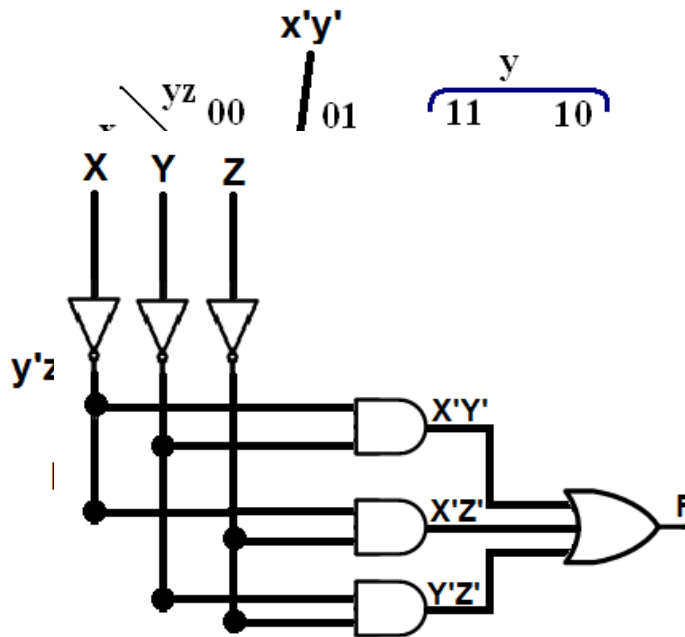
Inputs			NO. of 0's and 1's in the input combinations		Condition	Output
$2^2$	$2^1$	$2^0$	NO. of 0's	NO. of 1's	Is NO. Of 0's > NO. of 1's?	F
0	0	0	3	0	Is (3>0)=T	1
0	0	1	2	1	Is (2>1)=T	1
0	1	0	2	1	Is (2>1)=T	1
0	1	1	1	2	Is (1>2)=F	0
1	0	0	2	1	Is (2>1)=T	1
1	0	1	1	2	Is (1>2)=F	0
1	1	0	1	2	Is (1>2)=F	0
1	1	1	0	3	Is (0>3)=F	0

Inputs			Output
X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

☒ **Express each output function as sum of minterms:**

$$F(X,Y,Z)=\sum(0,1,2,4)$$

☒ **Simplify each output function using( Boolean algebra or Map method)**



☒ Draw the logic circuit of the output function  $\neg X'Z'$

**Ex.3.** Design a combinational circuit that accepts 3-bit binary number and produces the number of 1's in each input combination in binary.

**Solution:**

Inputs			Output
$2^2$	$2^1$	$2^0$	N0. Of 1's
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	0	2
1	1	1	3

Number of output variables=2 {since maximum value =3 needs 2 binary digits to be written  $(3)_{10}=(11)_2$

Assign symbols to the input and output columns

Inputs			Outputs in binary	
$2^2$	$2^1$	$2^0$	$2^1$	$2^0$
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

Inputs			Outputs in binary	
x	y	z	A	B
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

☒ Express each output function as sum of minterms:

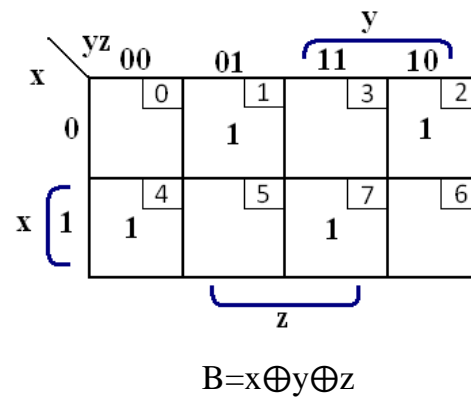
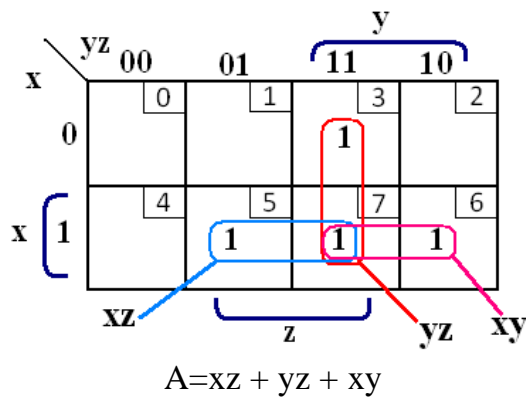
$$A(X,Y,Z)=\sum(3,5,6,7)$$

$$B(X,Y,Z)=\sum(1,2,4,7)$$

☒ Simplify each output function (A and B) using Map method

∴ There are 3 input variables ∴ use 3-Var. map  $2^3=8$  squares

Draw 2-maps ( one for each output functions)



$$A = xz + yz + xy$$

$$B = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$

$$B = \bar{x}(y\bar{z} + \bar{y}z) + x(\bar{y}\bar{z} + xy)$$

$$B = \bar{x}(y \oplus z) + x(y \odot z)$$

$$\text{Let } A = (y \oplus z) \quad \therefore \bar{A} = (y \odot z)$$

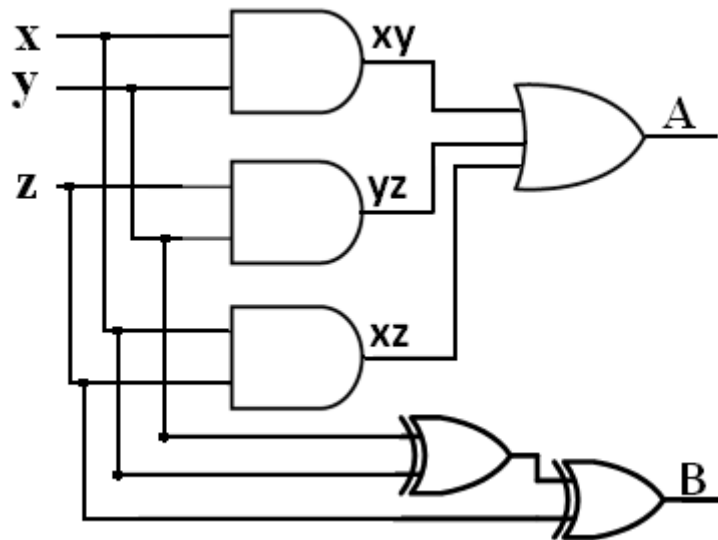
$$B = \bar{x}A + x\bar{A}$$

$$B = x \oplus A$$



$$B = x \oplus y \oplus z$$

☒ Draw the circuit



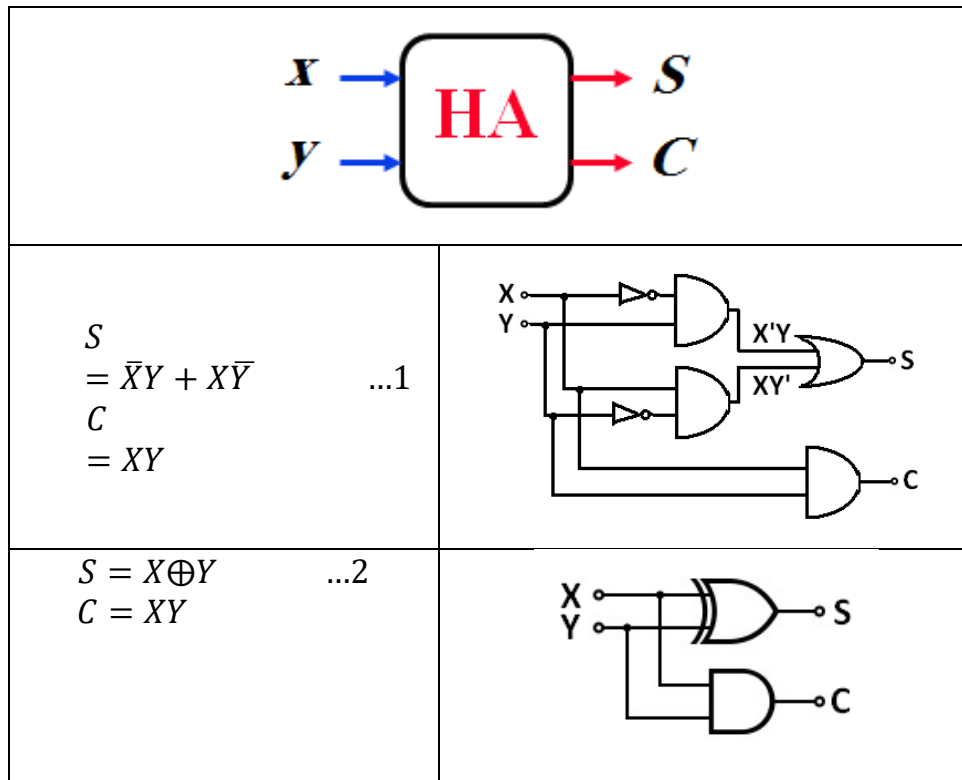
## Lecture 6

### 18. Adder

#### 18.1. Half Adder (HA)

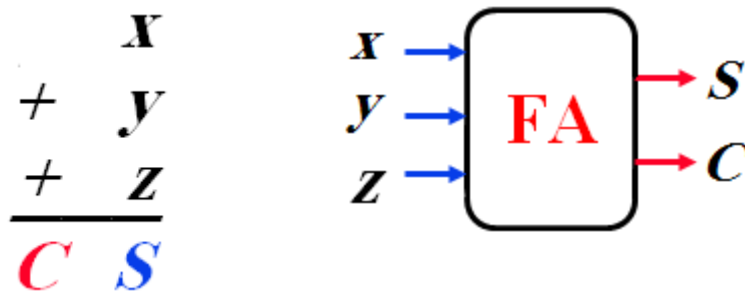
A half adder is a logical circuit that performs an addition operation on two binary bits.

X	0	0	1	1	inputs		Outputs in Binary			
Y	0	1	0	1	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>1</sup>	2 <sup>0</sup>		
	+	+	+	+	X	Y	addition	Carry	Sum	Minterms
Carry				1	0	0	0	0	0	
Sum	0	1	1	0	0	1	1	0	1	$\bar{X}Y (S)$
					1	0	1	0	1	$X\bar{Y}(S)$
					1	1	2	1	0	$XY (C)$



### 18.2. Full adder (FA)

A full adder is a logical circuit that performs the arithmetic sum of three binary bits (two significant and the previous carry)



Inputs			Outputs in binary	
x	y	z	Carry	Sum
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

		<b>y</b>		
	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>x</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
		<b>z</b>		

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

$$= x \oplus y \oplus z$$

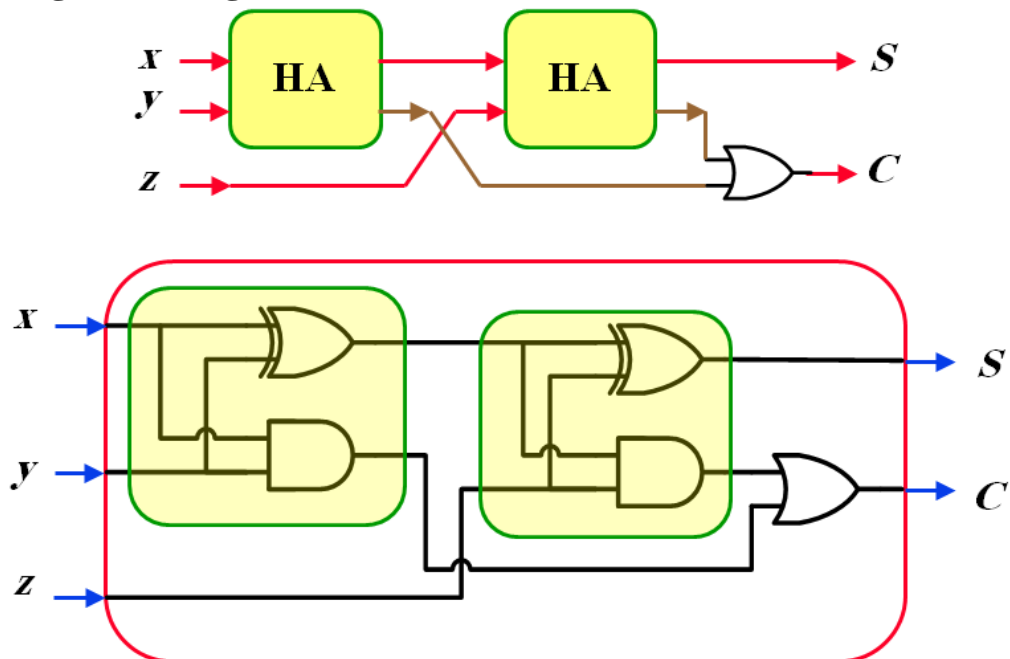
		<b>y</b>		
	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>x</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>
		<b>z</b>		

$$C = xy + xz + yz$$

**Sum using S.O. minterms**

**Sum using XOR gates**

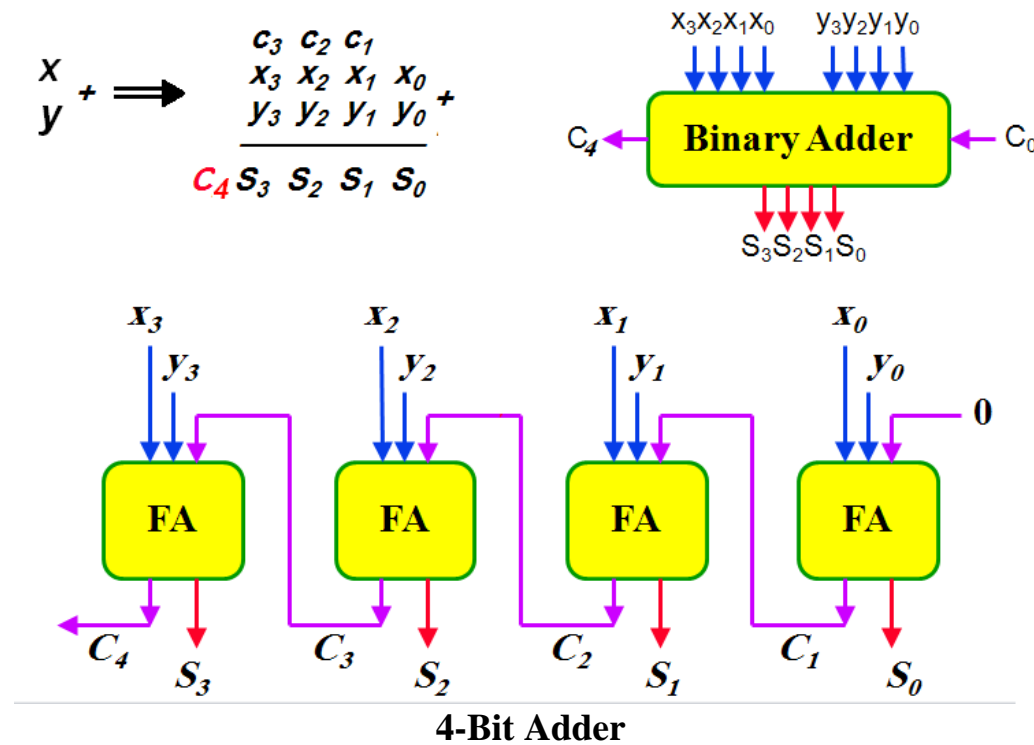
**Design FA using HA**



## Design FA using 2-HA and external OR gate

### 2.3 Binary Adder:

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adder in cascaded.



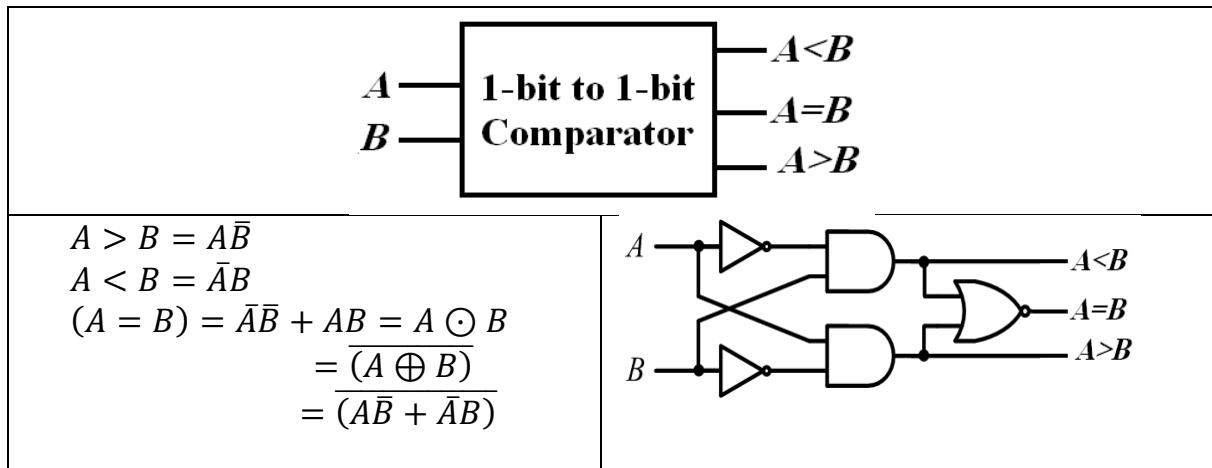
## 19. Magnitude Comparator

The comparison of two numbers is an operation that determine if one number is greater than, less than, or equal to the other number. A magnitude comparator is combinational circuit that compares two numbers, A and B, and determines their relative magnitude. The outcome of the comparison is specified by three binary variables that indicate whether  $A > B$ ,  $A = B$ , and  $A < B$ .

The circuit of comparing two n-bit binary numbers has  $2^{2n}$  entries in the truth table

### 19.1. 1-bit comparator circuit

A	B	A>B	A<B	A=B
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1



### 19.2. 2-bit comparator circuit

A comparator circuit processes a certain amount of regularity. Digital function which processes an inherent well-defined regularity can usually be designed by means of an algorithmic procedure if one is found to exist. An algorithm is a procedure that specifies a finite set of steps which, if followed, give the solution to the problem. We illustrate this method by deriving an algorithm for the design of a 2-bit magnitude comparator.

Consider two numbers A and B where

$$A = A_1 A_0$$

$$B = B_1 B_0$$

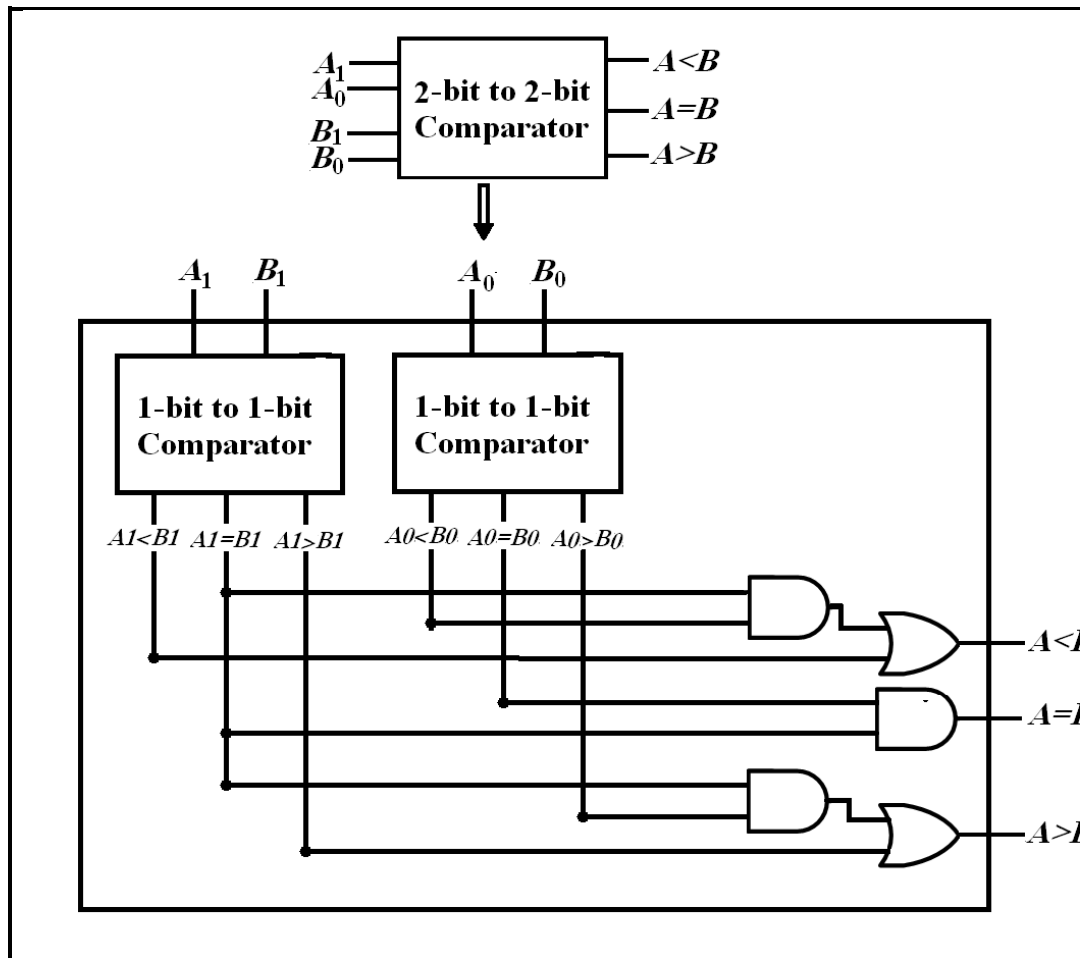
- The two numbers are equal if all pairs of significant digit are equal (i.e.  $A_1 = B_1$  and  $A_0 = B_0$ )  
 $X_1 = (A_1 = B_1)$   
 $X_0 = (A_0 = B_0)$
- A greater than B if  $(A_1 > B_1)$  OR  $(A_1 = B_1)$  AND  $(A_0 > B_0)$
- A less than B if  $(A_1 < B_1)$  OR  $(A_1 = B_1)$  AND  $(A_0 < B_0)$

$$(A=B) = X_1 X_0$$

$$(A>B) = (A_1 > B_1) + X_1 (A_0 > B_0)$$

$$(A<B) = (A_1 < B_1) + X_1 (A_0 < B_0)$$

The circuit of **2-bit comparator** is derived by repeating 1-bit comparator circuit for each pair of numbers (i.e.  $(A_1, B_1)$  and  $(A_0, B_0)$ ) as shown in the following figure



### 19.3. 4-bit comparator circuit

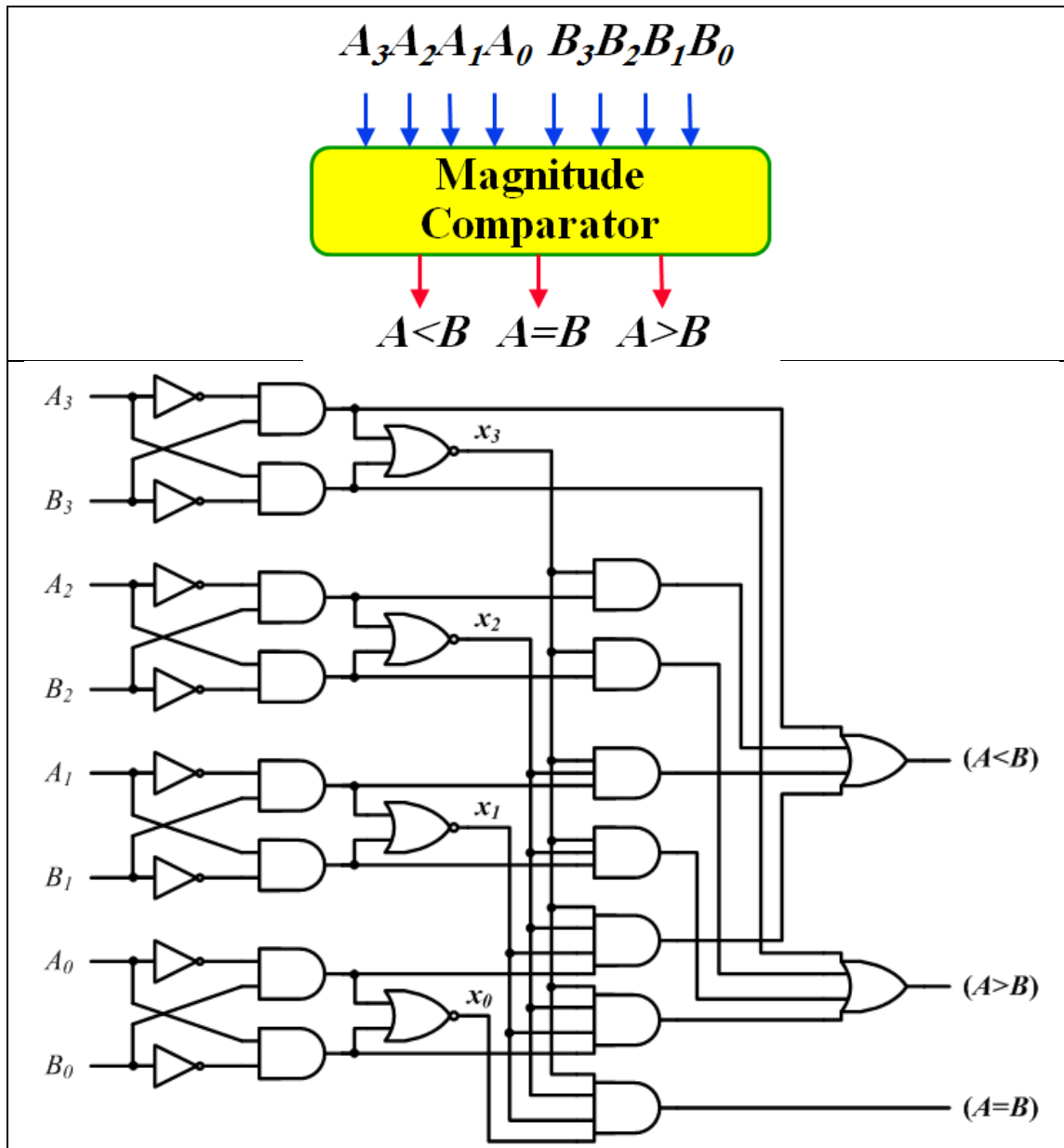
Consider two numbers A and B where

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

- The two numbers are equal if all pairs of significant digit are equal (i.e.  $A_3 = B_3$ ,  $A_2 = B_2$ ,  $A_1 = B_1$  and  $A_0 = B_0$ )  
 $X_i = (A_i = B_i)$  where  $i = 0, 1, 2, 3$
  - A greater than B if  $(A_3 > B_3) \text{ OR } X_3 \text{ AND } (A_2 > B_2) \text{ OR } X_3 X_2 \text{ AND } (A_1 > B_1) \text{ OR } X_3 X_2 X_1 \text{ AND } (A_0 > B_0)$
  - A less than B if  $(A_3 < B_3) \text{ OR } X_3 \text{ AND } (A_2 < B_2) \text{ OR } X_3 X_2 \text{ AND } (A_1 < B_1) \text{ OR } X_3 X_2 X_1 \text{ AND } (A_0 < B_0)$
- $(A=B) = X_3 X_2 X_1 X_0$   
 $(A>B) = (A_3 > B_3) + X_3(A_2 > B_2) + X_3 X_2 (A_1 > B_1) + X_3 X_2 X_1 (A_0 > B_0)$   
 $(A<B) = (A_3 < B_3) + X_3(A_2 < B_2) + X_3 X_2 (A_1 < B_1) + X_3 X_2 X_1 (A_0 < B_0)$

The circuit of **4-bit comparator** is derived by repeating 1-bit comparator circuit for each pair of numbers (i.e.  $(A_3, B_3)$ ,  $(A_2, B_2)$ ,  $(A_1, B_1)$  and  $(A_0, B_0)$ ) as shown in the following figure

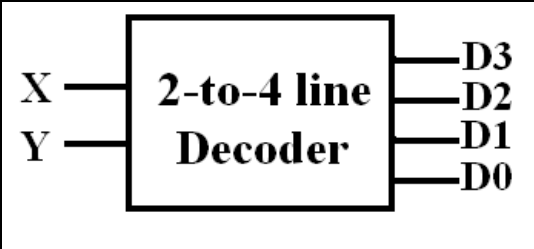
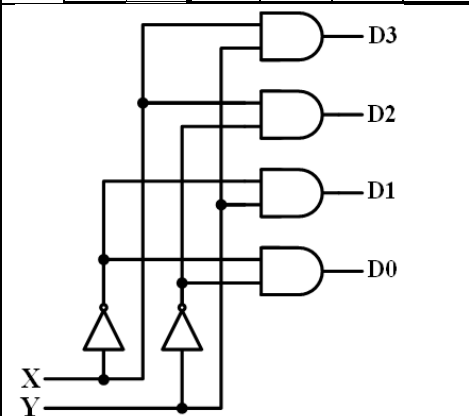


Lecture 7

20. Decoder

A binary code of  $n$  bits is capable of representing up to  $2^n$  distinct binary code. A decoder is a combinational circuit that converts binary information from  $n$  input lines to a maximum of  $2^n$  unique output lines. If the  $n$ -bit decoded information has unused or don't-care combinations, the decoder output will have less than  $2^n$  outputs.

**Ex6. Design 2-to-4 line decoder**

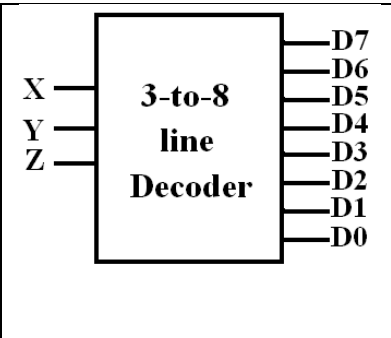
	<table border="1" style="margin: auto; border-collapse: collapse;"> <thead> <tr> <th colspan="2">Inputs</th> <th colspan="4">Outputs</th> </tr> <tr> <th>x</th> <th>y</th> <th>D3</th> <th>D2</th> <th>D1</th> <th>D0</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	Inputs		Outputs				x	y	D3	D2	D1	D0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	1	1	1	0	0	0
Inputs		Outputs																																			
x	y	D3	D2	D1	D0																																
0	0	0	0	0	1																																
0	1	0	0	1	0																																
1	0	0	1	0	0																																
1	1	1	0	0	0																																
$D3 = XY$ $D2 = X\bar{Y}$ $D1 = \bar{X}Y$ $D0 = \bar{X}\bar{Y}$																																					

**Ex7. Design 3-to-8 line decoder or design binary to Octal decoder**

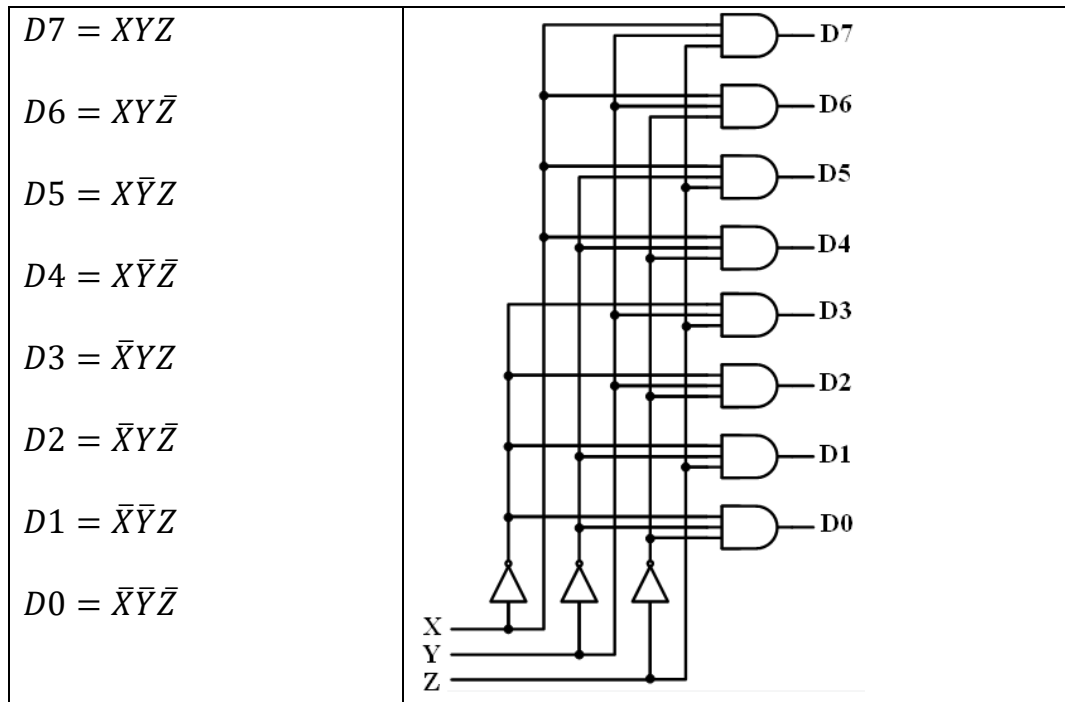
- How many bits in binary needs to configure one Octal digit?

Three binary bits needs to configure one octal digit, thus the input variables are three

- The octal system coefficients are (0,1,2,3,4,5,6,7), thus the output lines are eight

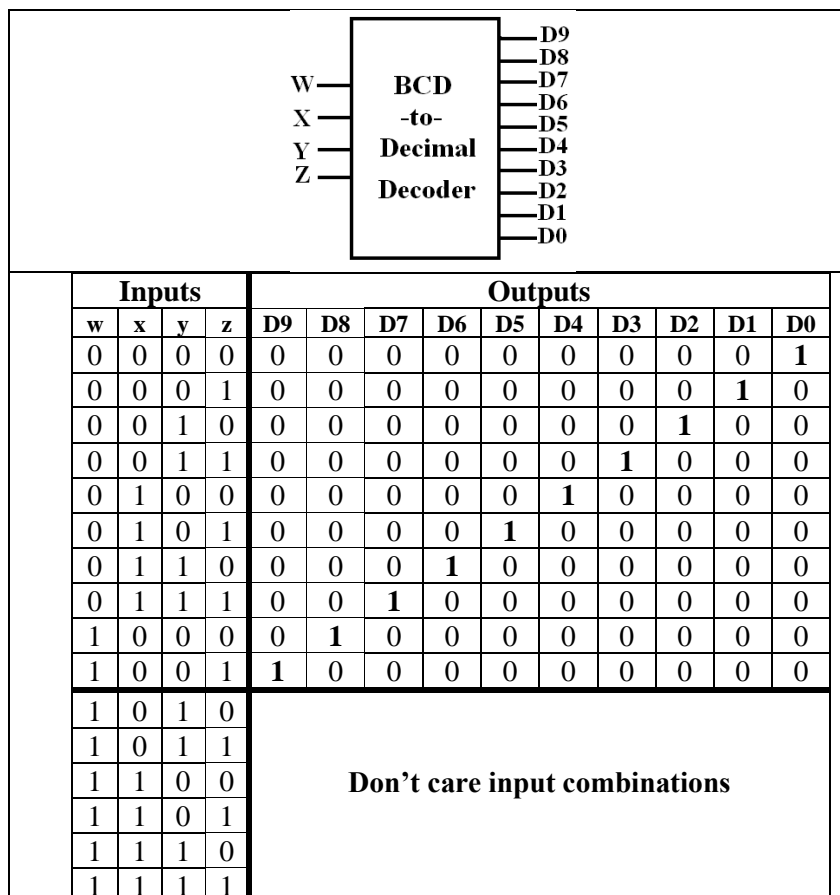
	<table border="1" style="margin: auto; border-collapse: collapse;"> <thead> <tr> <th colspan="3">Inputs</th> <th colspan="8">Outputs</th> </tr> <tr> <th>x</th> <th>y</th> <th>z</th> <th>D7</th> <th>D6</th> <th>D5</th> <th>D4</th> <th>D3</th> <th>D2</th> <th>D1</th> <th>D0</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </tbody> </table>	Inputs			Outputs								x	y	z	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
Inputs			Outputs																																																																																																												
x	y	z	D7	D6	D5	D4	D3	D2	D1	D0																																																																																																					
0	0	0	0	0	0	0	0	0	0	1																																																																																																					
0	0	1	0	0	0	0	0	0	1	0																																																																																																					
0	1	0	0	0	0	0	0	1	0	0																																																																																																					
0	1	1	0	0	0	0	1	0	0	0																																																																																																					
1	0	0	0	0	0	1	0	0	0	0																																																																																																					
1	0	1	0	0	1	0	0	0	0	0																																																																																																					
1	1	0	0	1	0	0	0	0	0	0																																																																																																					
1	1	1	1	0	0	0	0	0	0	0																																																																																																					

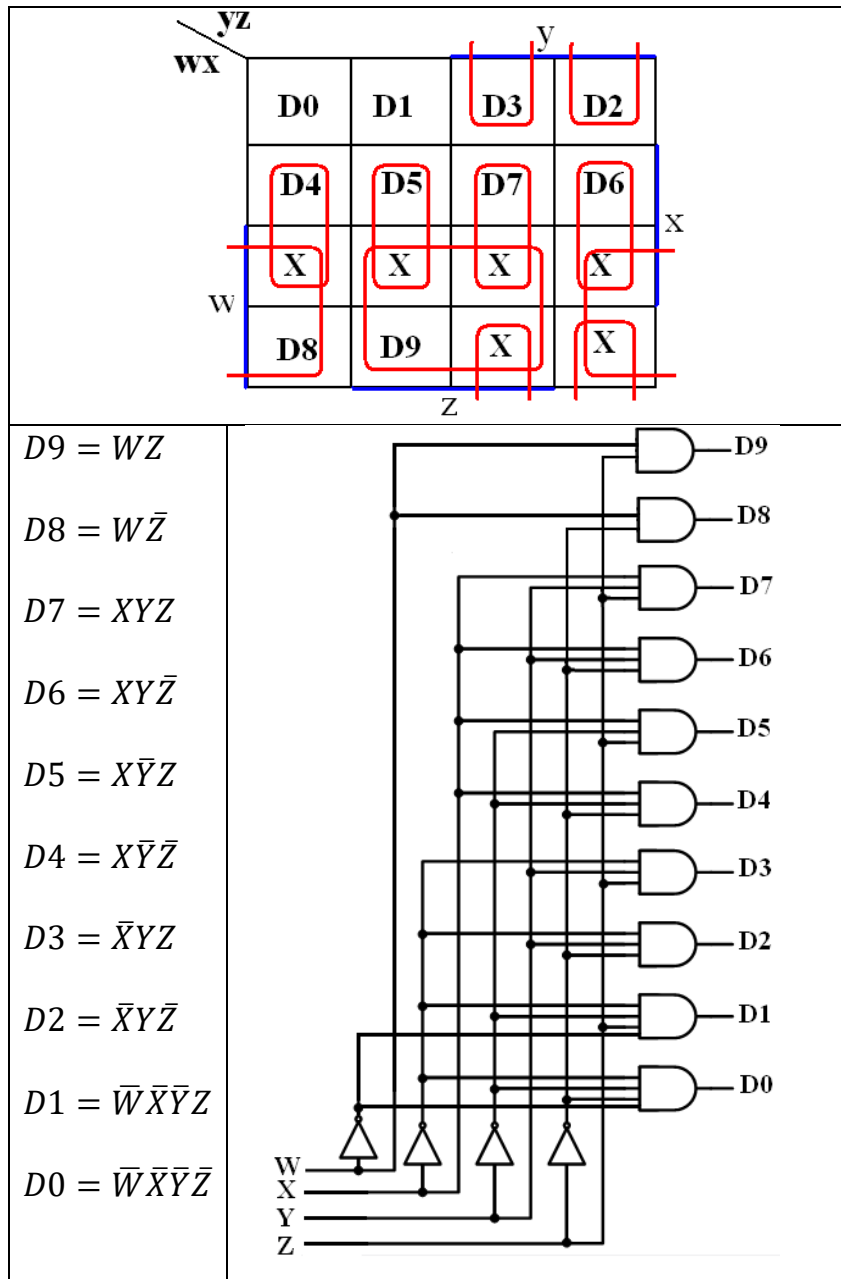




**Ex8.** Design BCD-to-Decimal decoder

The elements of information in this case are ten decimal digits represented by BCD code. The code itself has four bits. Therefore, the decoder should have four inputs to accept the coded digit and ten outputs, one for each decimal digit. This will give a 4-line to 10-line BCD-to-Decimal decoder



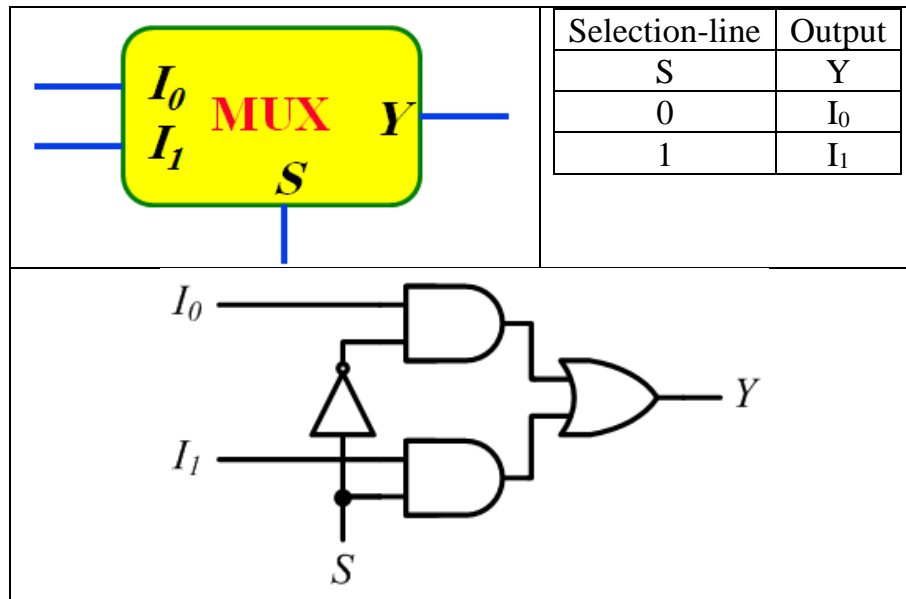


**21. Multiplexer:**

Is also called the data selector, since it selects one of many inputs. A digital multiplexer (MUX) is a combinational circuit that selects binary information from one of many inputs lines and directs it to a single output line. The selection of particular input lines is controlled by a set of selection lines. Normally there are  $2^n$  input lines and  $n$  selection lines whose bit combinations determine which input is selected.

**Ex12.** Design 2-to-1 MUX

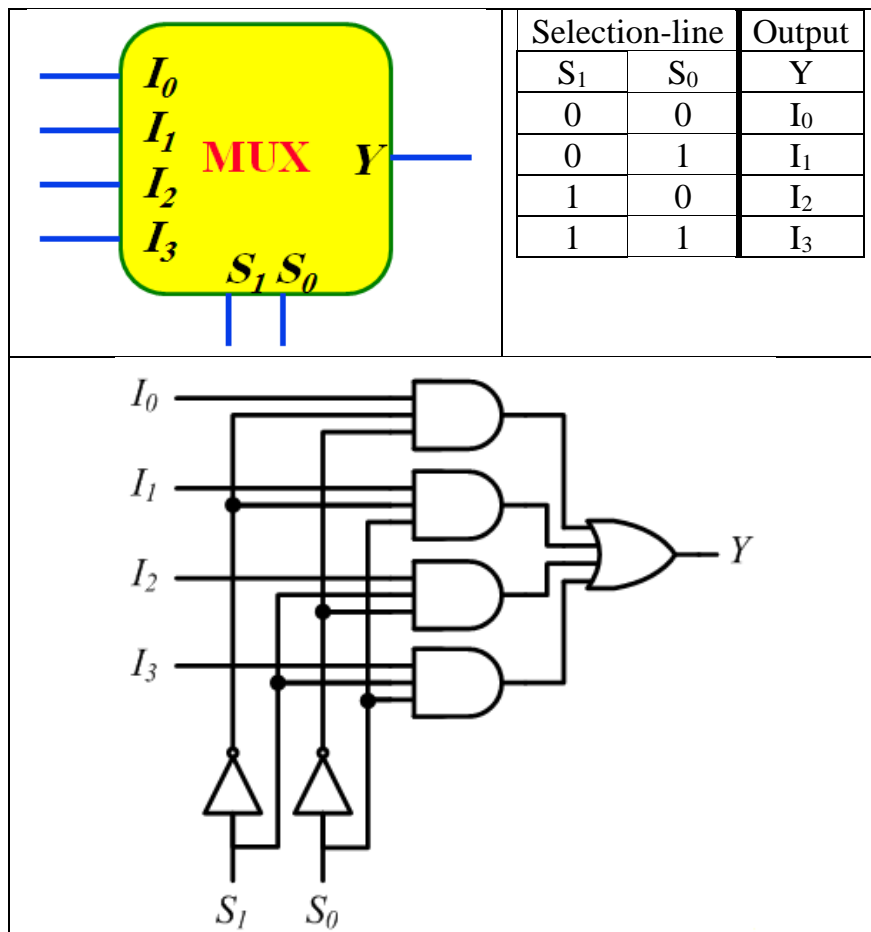
2 is the number of the input variable ( $I_1, I_0$ )  
 $2=2^n \rightarrow n=1$  where  $n$  is the selection variable ( $S$ )



**Ex13.** Design 4-to-1 MUX

4 is the number of the input variable ( $I_3, I_2, I_1, I_0$ )

$4=2^n = 2^2 \rightarrow n=2$  where n is the selection variable ( $S_1, S_0$ )



The AND gates and inverter in multiplexer resemble a decoder circuit and they decode the input selection lines. In general  $2^n$  to 1 line MUX is constructed from  $n$ -to- $2^n$  decoder by adding to its  $2^n$  input lines, one to each input lines.( see Ex12. 2-inputs  $s_1s_0$  , 4 AND gets each has 3- input variables ( $s_1, s_0$  and either  $\{ I_3, I_2, I_1, I_0 \}$  ). The size of a multiplexer is specified by the  $2^n$  of its input lines and a single output lines. It is then implied that it also contain  $n$  selection lines.

**21.1. Boolean Function Implementation**

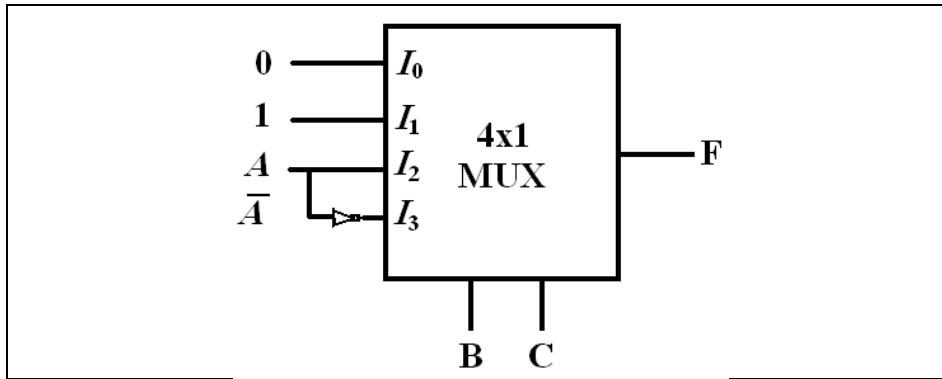
A decoder can be used to implement a Boolean function by employing an external OR gate. The multiplexer is a decoder with OR gate already available. If we have a Boolean function of  $n+1$  variable, we take  $n$  of these variables and connect them to the selection lines of a multiplexer. The remaining single variable of the function is used for the input lines of the multiplexer.

**Ex.** Implement  $F(A,B,C)=\sum(1,3,5,6)$  using MUX, select A as input variable.

$\therefore$  A is used as input variable.  $\therefore$  The reaming two variables (B,C) are used as selection lines

$\therefore$   $n=2$  is the number of selection lines (B,C)  $\rightarrow 2^2=4$  is the number of input lines ( $I_3, I_2, I_1, I_0$ ) that must be written in terms of input variable (A)

<b>Minterms</b>		<b>A</b>	<b>B</b>	<b>C</b>	<b>F</b>	<p style="text-align: center;"><b>Input lines</b></p> <table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td></td> <td style="text-align: center;"><math>I_0</math></td> <td style="text-align: center;"><math>I_1</math></td> <td style="text-align: center;"><math>I_2</math></td> <td style="text-align: center;"><math>I_3</math></td> </tr> <tr> <td style="text-align: center;"><math>\bar{A}</math></td> <td style="text-align: center;">0</td> <td style="text-align: center;">①</td> <td style="text-align: center;">2</td> <td style="text-align: center;">③</td> </tr> <tr> <td style="text-align: center;">A</td> <td style="text-align: center;">4</td> <td style="text-align: center;">⑤</td> <td style="text-align: center;">⑥</td> <td style="text-align: center;">7</td> </tr> <tr> <td></td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">A</td> <td style="text-align: center;"><math>\bar{A}</math></td> </tr> </table>		$I_0$	$I_1$	$I_2$	$I_3$	$\bar{A}$	0	①	2	③	A	4	⑤	⑥	7		0	1	A	$\bar{A}$
	$I_0$	$I_1$	$I_2$	$I_3$																						
$\bar{A}$	0	①	2	③																						
A	4	⑤	⑥	7																						
	0	1	A	$\bar{A}$																						
0	$\bar{A}$	0	0	0																						
1	$\bar{A}$	0	1	1																						
2	$\bar{A}$	1	0	0																						
3	$\bar{A}$	1	1	1																						
4	A	1	0	0																						
5	A	1	0	1																						
6	A	1	1	0																						
7	A	1	1	1																						
	Selection-line	Output																								
	B	C	F																							
	0	0	$I_0=0$																							
	0	1	$I_1=1$																							
	1	0	$I_2=A$																							
	1	1	$I_3=\bar{A}$																							
		<b>Input lines</b>																								
		$I_0$	$I_1$	$I_2$	$I_3$																					
$\bar{A}$	Input var.	0	①	0	①																					
A	Input var.	0	①	①	0																					
		0	1	A	$\bar{A}$																					



**Ex.** Implement  $F(A,B,C,D)=\sum(0,1,3,4,8,9,15)$  using MUX, select B as input variable.

$\because$  B is used as input variable.  $\therefore$  The remaining three variables (A,C,D) are used as selection lines

$\because$  n=3 is the number of selection lines (A,C,D)  $\rightarrow 2^3=8$  is the number of input lines ( $I_7, I_6, I_5, I_4, I_3, I_2, I_1, I_0$ ) that must be written in terms of input variable (B)

Minterms	A	B	C	D	F
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

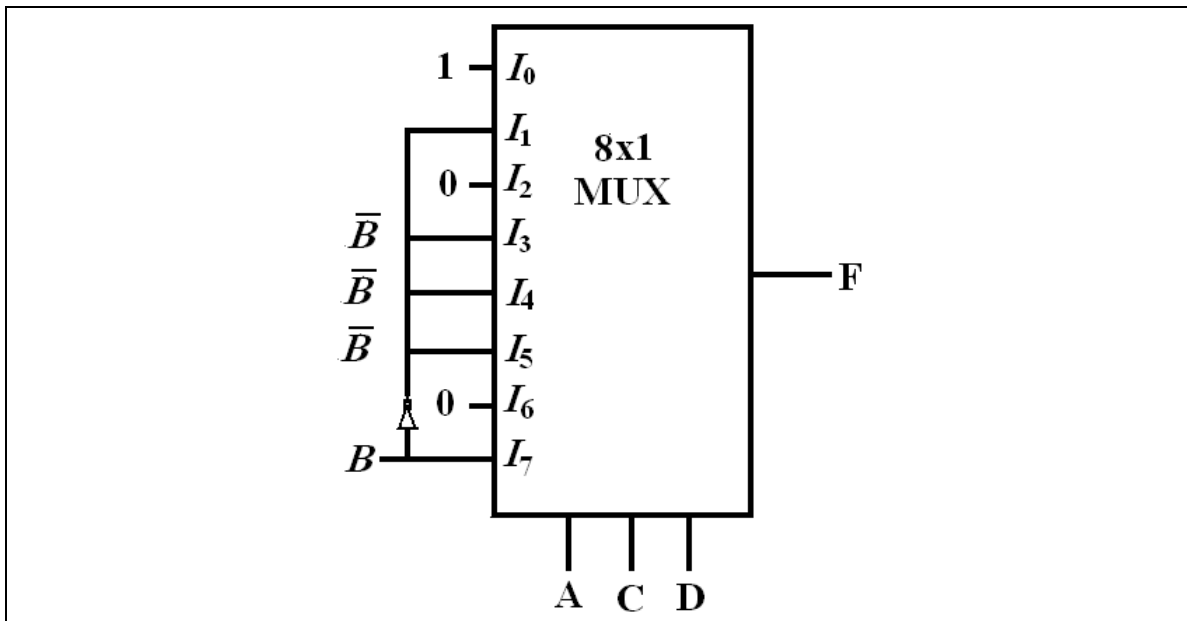
		Input lines							
		I <sub>0</sub>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>
Input	$\bar{B}$	0	1	2	3	8	9	10	11
	B	4	5	6	7	12	13	14	15
		1	$\bar{B}$	0	$\bar{B}$	$\bar{B}$	$\bar{B}$	0	B

Selection-line			Output
A	C	D	F
0	0	0	$I_0=1$
0	0	1	$I_1=\bar{B}$
0	1	0	$I_2=0$
0	1	1	$I_3=\bar{B}$
1	0	0	$I_4=\bar{B}$
1	0	1	$I_5=\bar{B}$
1	1	0	$I_6=0$
1	1	1	$I_7=B$

		Input lines							
		I <sub>0</sub>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>
Input var.	$\bar{B}$	1	1	0	1	1	1	0	0
	B	1	0	0	0	0	0	0	1
		1	$\bar{B}$	0	$\bar{B}$	$\bar{B}$	$\bar{B}$	0	B



**Ex.** Implement F-A using Dual-MUX, let the most significant variable is the input variable.

Inputs			Outputs in binary	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

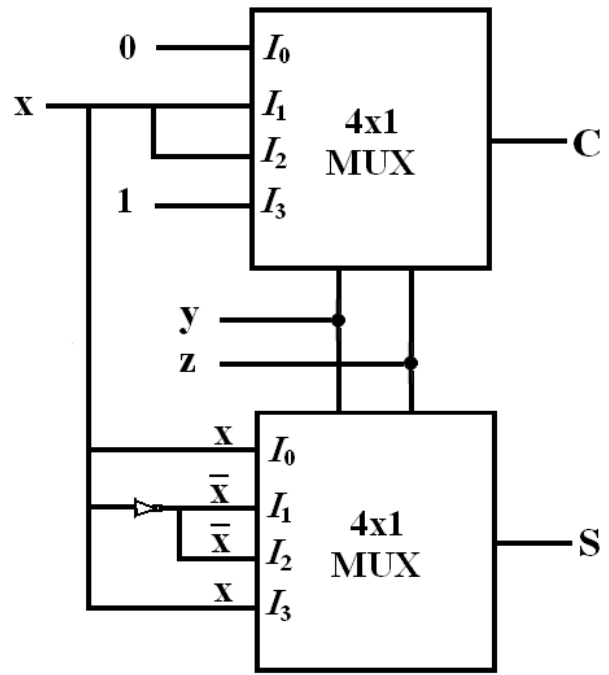
• The output functions must be written as S.O. minterms as:  
 $C(x,y,z) = \sum(3,5,6,7)$   
 $S(x,y,z) = \sum(1,2,4,7)$

∴ x is used as input variable. ∴ the remaining two variables (y,z) are used as selection lines → n=2

$2^2=4$  is the number of input lines ( $I_3, I_2, I_1, I_0$ ) that must be written in terms of input variable (x)

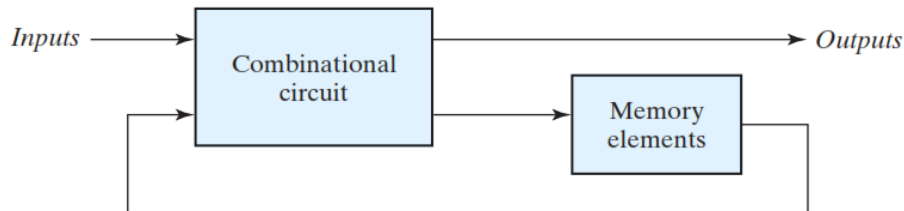
<b>C</b>		Input lines				<b>S</b>		Input lines			
		$I_0$	$I_1$	$I_2$	$I_3$			$I_0$	$I_1$	$I_2$	$I_3$
Input var.	$\bar{x}$	0	1	2	③	Input var.	$\bar{x}$	0	①	②	3
	x	4	⑤	⑥	⑦		x	④	5	6	⑦
		0	x	x	1			x	$\bar{x}$	$\bar{x}$	x

Selection-lines		Output	
y	z	C	S
0	0	$I_0=0$	$I_0=x$
0	1	$I_1=x$	$I_1=\bar{x}$
1	0	$I_2=x$	$I_2=\bar{x}$
1	1	$I_3=1$	$I_3=x$



## Lecture 8

### 22.Sequential Circuits



Block diagram of sequential circuit

As shown in the block diagram of a sequential circuit. It consists of a combinational circuit to which storage elements are connected to form a feedback path. The storage elements are devices capable of storing binary information. The binary information stored in these elements at any given time defines the *state* of the sequential circuit at that time. The sequential circuit receives binary information from external inputs that, together with the present state of the storage elements, determine the binary value of the outputs. These external inputs also determine the condition for changing the state in the storage elements.

- The block diagram demonstrates that the outputs in a sequential circuit are a function not only of the inputs, but also of the present state of the storage elements.
- The next state of the storage elements is also a function of external inputs and the present state. Thus, a **sequential circuit is specified by a time sequence of inputs, outputs, and internal states.**
- In contrast, the outputs of combinational logic depend only on the present values of the inputs.

There are two main types of sequential circuits, and their classification is a function of the timing of their signals. A synchronous sequential circuit is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time. The behavior of an asynchronous sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change.

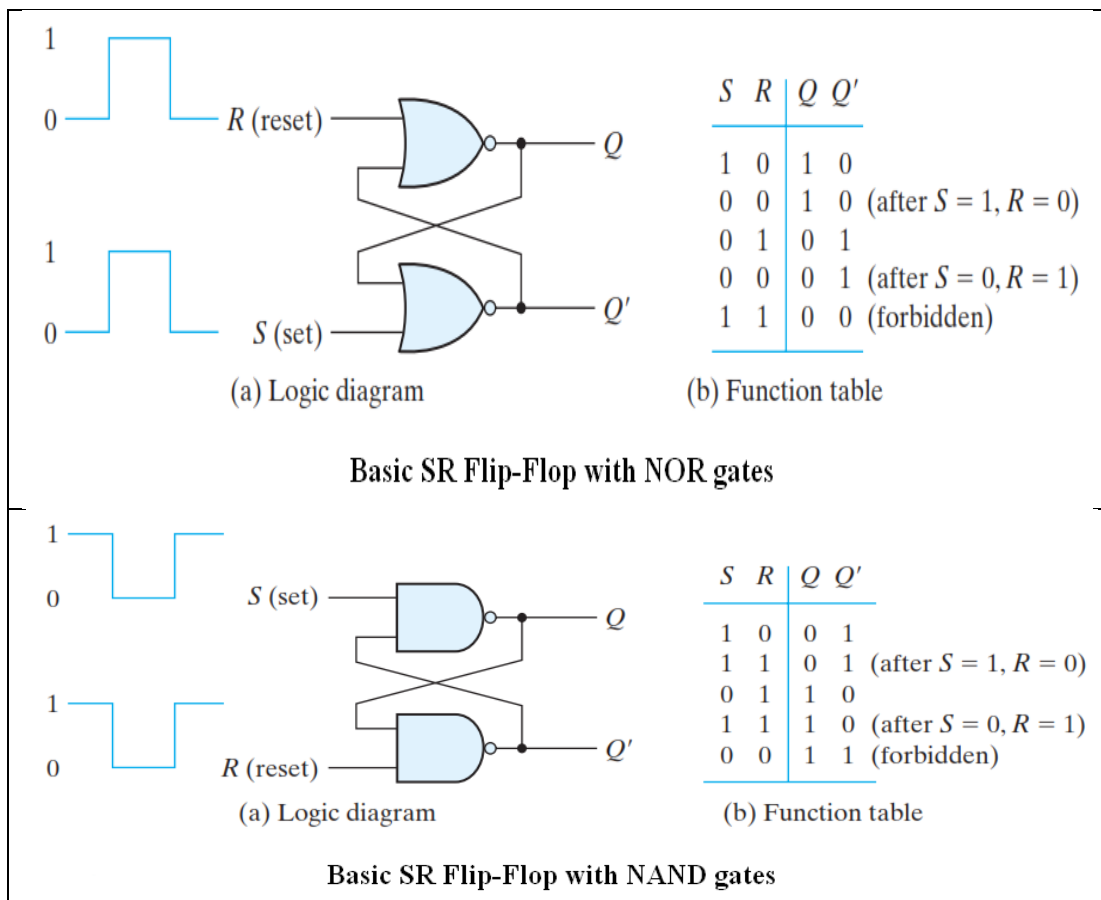
A synchronous sequential circuit employs signals that affect the storage elements at only discrete instants of time. Synchronization is achieved by a timing device called a clock *generator*, which provides a clock signal having the form of a periodic train of *clock pulses*. Synchronous sequential circuits that use clock pulses to control storage elements are called clocked *sequential circuits*



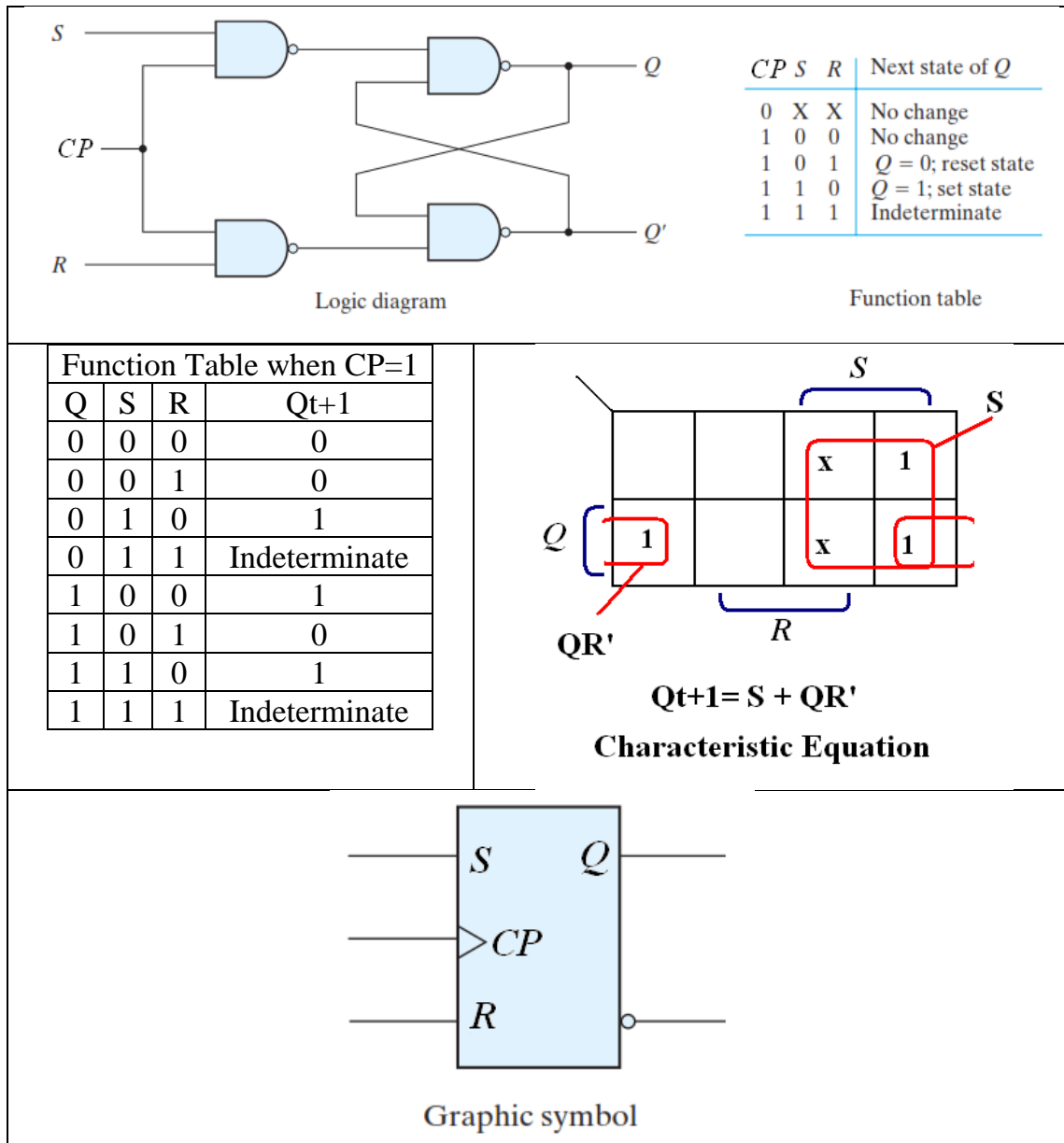
The storage elements (memory) used in clocked sequential circuits are called flip *flops*. A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1. A sequential circuit may use many flip-flops to store as many bits as necessary

## 22.1 Basic Flip-Flops

The basic flip-flop can be constructed from two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled S for set and R for reset.



## 20.2 Clocked SR Flip-Flop



## 20.3 D Flip-Flop

One way to eliminate the undesirable condition of the indeterminate state in the SR flip-flop is to ensure that inputs  $S$  and  $R$  are never equal to 1 at the same time. This is done in the D flip-flop.

Logic diagram

$CP$	$D$	Next state of $Q$
0	X	No change
1	0	$Q = 0$ ; reset state
1	1	$Q = 1$ ; set state

Function table

$Q$	$D$	$Q_{t+1}$
0	0	0
0	1	1
1	0	0
1	1	1

$Q_{t+1} = D$

**Characteristic Equation**

Graphic symbol

### 20.4 JK Flip-Flop

Circuit diagram

Graphic symbol

Function Table when CP=1			
Q	J	K	Qt+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$Q_{t+1} = Q'J + QK'$

**Characteristic Equation**

Function Table			
CP	J	K	Qt+1
0	X	X	Q No change
1	0	0	Q No change
1	0	1	0
1	1	0	1
1	1	1	Q' Complement

### 20.5 T Flip-Flop

**Circuit diagram**

**Graphic symbol**

Function Table when CP=1		
Q	T	Qt+1
0	0	0
0	1	1
1	0	1
1	1	0

$Q_{t+1} = QT' + Q'T = Q \oplus T$

**Characteristic Equation**

Function Table		
CP	T	Qt+1
0	X	Q No change
1	0	Q No change
1	1	Q' Complement