Chapter one: Number systems

Introduction

A digital computer stores data in terms of digits (numbers). and proceeds in discrete steps from one state to the next. The states of a digital computer typically involve binary digits which may take the form of the presence or absence of magnetic markers in a storage medium, on-off switches or relays. In digital computers, even letters, words and whole texts are represented digitally.

Positional number system is the type of number system in which the weight or value of the digit (or symbol) depends upon its position in the number. The positional number system is also known as **weighted number system.** This is because, in the positional number system, there is a weight associated with the position in the number.Therefore, in the positional number system, each digit of the number is weighted according to its position of occurrence in the number. When we travel toward left along the number, the weights increase by a constant factor that is equivalent to the base of the number system. Also, in the positional number system, a **radix point (.)** is used to differentiate the positions corresponding to integral weights from the positions corresponding to the fractional weights.

Types of Positional Number Systems

There are four very popular positional number systems, which are:

1-Decimal system

This system has ten coefficients (0,1,2,3,4,5,6,7,8,9) and the base of this system is equal to the number of coefficients, so the base of decimal is (10)

2- Binary system

This system has only two coefficients (0, 1) and the base is (2)

3- Octal system

This system has eight coefficients (0, 1, 2, 3, 4, 5, 6, 7) and the base is (8)

4- Hexadecimal system

The coefficients of this system are (0,1,2,3,4,5,6,7,8,9,A,B,C,E,F) and the base is (16)

Decimal System:

The **decimal system** is composed of *10* numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; using these symbols as *digits* of a number, we can express any quantity. The decimal system, also called the *base-10* system because it has 10 digits. The decimal system is a positional-value system in which the value of a digit depends on its position. For example, consider the decimal number 453 .

We know that the digit 4 actually represents 4 hundreds, the 5 represents 5 tens, and the 3 represents 3 units. In essence, the 4 carries the most weight of the three digits; it is referred to as the most significant digit (MSD). The 3 carries the least weight and is called the least significant digit (LSD).

Consider another example, 27.35 . This number is actually equal to 2 tens plus 7 units plus 3 tenths plus 5 hundredths, or 2 x 10 + 7 x 1 + 3 x 0.1 + 5 x 0.01. The decimal point is used to separate the integer and fractional parts of the number.

Moreover, the various positions relative to the decimal point carry weights that can be expressed as powers of 10. This is illustrated in Figure (1), where the number 2745.214 is represented. The decimal point separates the positive powers of 10 from the negative powers. The number 2745.214 is thus equal to

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

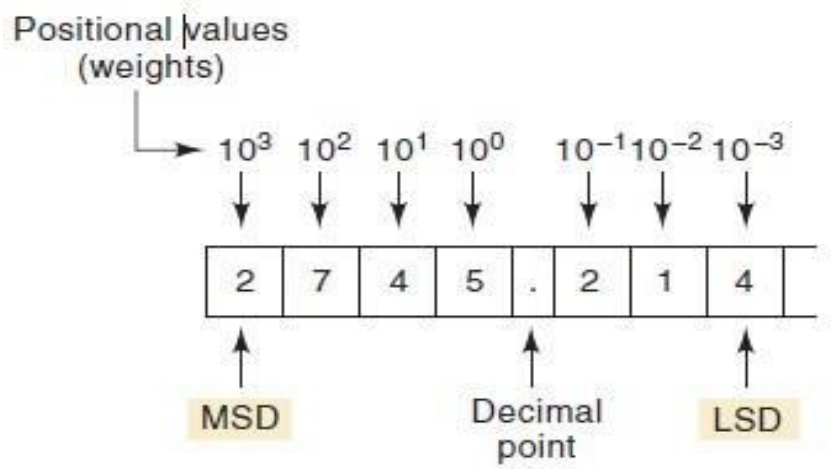


Figure (1): Decimal position values as powers of 10.

Binary System:

In the **binary system** there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems. In general though, it will take a greater number of binary digits to express a given quantity. All of the statements

made earlier concerning the decimal system are equally applicable to the binary system. The binary system is also a positional value system, wherein each binary digit has its own value or weight expressed as a power of 2. This is illustrated in Figure (2).

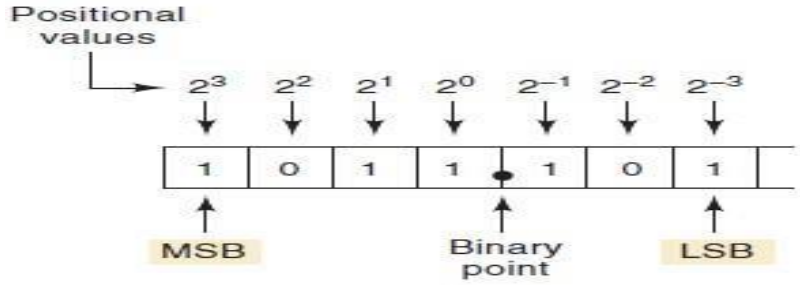


Figure (2): Binary position values as powers of 2.

Here, places to the left of the binary point (counterpart of the decimal point) are positive powers of 2, and places to the right are negative powers of 2. The number 1011.101 is shown represented in the figure. To find its equivalent in the decimal system, we simply take the sum of the products of each digit value (0 or 1) and its positional value:

$$\begin{aligned} 1011.101_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &\quad + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\ &= 11.625_{10} \end{aligned}$$

Octal Number System:

Characteristics

• Uses eight digits, 0,1,2,3,4,5,6,7.

•Also called base 8 number system

• Each position in an octal number represents a 0 power of the base (8).

Example: 80

• Last position in an octal number represents an x power of the base (8).

Example: 8x where x represents the last position - 1.

Example

Octal Number − 125708

Calculating Decimal	**Octal Number**	**Decimal Number**
Step 1	125708	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	125708	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	125708	5496_{10}

Hexadecimal Number System:

Characteristics

• Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

• Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

• Also called base 16 number system.

• Each position in a hexadecimal number represents a 0 power of the base (16). Example 160.

• Last position in a hexadecimal number represents an x power of the base (16). Example 16x where x represents the last position - 1.

Example

Hexadecimal Number: 19FDE16

Calculating Decimal	**Hexadecimal Number**	**Decimal Number**
Step 1	19FDE16	$((1 \times 16^4) + (9 \times 163) + (F \times 162) + (D \times 161) + (E \times 160))10$
Step 2	19FDE16	$((1 \times 164) + (9 \times 163) + (15 \times 162) + (13 \times 161) + (14 \times 160))10$
Step 3	19FDE16	$(65536 + 36864 + 3840 + 208 + 14)10$
Step 4	19FDE16	10646210

Number System Conversion

There are many methods or techniques which can be used to convert numbers from one base to another. We'll demonstrate here the following:

• Decimal to Other Base System
• Other Base System to Decimal
• Binary to Octal
• Octal to Binary
• Binary to Hexadecimal
• Hexadecimal to Binary

Decimal to Other Base System:

To convert any Number from the decimal system to other systems we divided the Number into two parts:

1. The Integer part:

Steps

Step1 − Divide the decimal number to be converted by the value of the new base.

Step2 − Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.

Step3 − Divide the quotient of the previous divide by the new base.

Step 4 − Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Operation	Integer	Remainder (Result)	
29 / 2	14	1	
14 / 2	7	0	
7 / 2	3	1	
3 / 2	1	1	
1 / 2	0	1	

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

Decimal Number $(29)_{10}$= Binary Number $(11101)_2$.

2. The Fraction part:

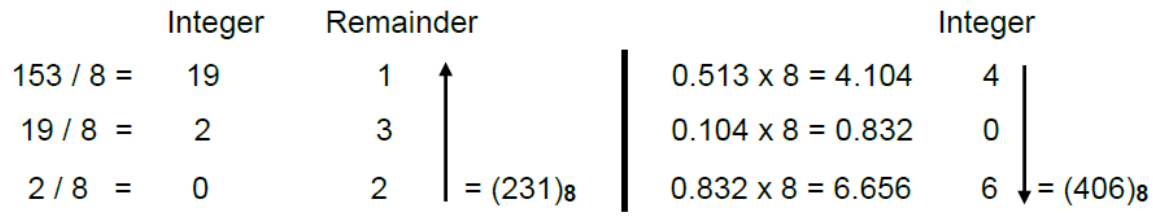
Multiply the decimal number by the new base number to give an integer and a fraction. The integer number after each multiplication will be result, Then the new remainder of fraction is multiplied by the new base number to give a new integer and a new fraction. The process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy.

Example

Convert (0.6875)**10** to binary Operation	Integer	Fraction	Result (Integer)	
0.6875 x 2	1	0.3750	1	
0.3750 x 2	0	0.7500	0	
0.7500 x 2	1	0.5000	1	
0.5000 x 2	1	0.0000	1	

Then $(0.687)_{10}=(1011)_2$

Example
Convert $(153.513)_{10}$ to Octal



The answer is $(153.513)_{10} = (231.406)_8$

Other Base System to Decimal:
Steps
• Step 1 − Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
• Step 2 − Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
• Step 3 − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example
Binary Number $(11101)_2$
Calculating Decimal Equivalent

Step	Binary Number	Decimal Number
Step 1	11101	$(1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$
Step 2	11101	16 + 8 + 4 + 0 + 1
Step 3	11101	29

Binary Number $(11101)_2$ = Decimal Number $(29)_{10}$

Example
Convert $(167)_8$ to Decimal
$(167)_8 = 1x8^2 + 6x8^1 + 7x8^0$
$= 64 + 48 + 7$
$= (119)_{10}$

Example

Convert $(35.62)_8$ to Decimal

$(35.62)_8 = 3x8^1 + 5x8^0 + 6x8^{-1} + 2x8^{-2}$

$= 24 + 5 + 0.75 + 0.031$

$= (29.781)_{10}$

Example

Convert $(3C6E.2AF)_{16}$ to Decimal

$(3C6E.2AF)_{16} = 3x16^3 + 12x16^2 + 6x16^1 + 14x16^0 + 2x16^{-1} + 10x16^{-2} + 15x16^{-3}$

$= 12288 + 3072 + 96 + 14 + 0.125 + 0.039 + 0.003$

$= (15470.167)_{10}$

Binary to Octal:

Steps

• Step 1 − Divide the binary digits into groups of three (starting from the right) as in table below.

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

• Step 2 − Convert each group of three binary digits to one octal digit.

Example

Convert $(10101)_2$ to Octal

$10101_2 = \underbrace{0\,1\,0}_{2}\,\underbrace{1\,0\,1}_{5}$

$= 25_8$

Example

Convert $(111011100)_2$ to Octal

$111011100_2 = \underbrace{1\,1\,1}_{7}\,\underbrace{0\,1\,1}_{3}\,\underbrace{1\,0\,0}_{4}$

$= 734_8$

❖ Octal to Binary:

Steps

- Step 1 − Convert each octal digit to a 3 digit binary number.
- Step 2 − Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Convert $(746)_8$ to binary

$746_8 = \underset{\downarrow}{7}\ \underset{\downarrow}{4}\ \underset{\downarrow}{6}$

$= 111\ \ 100\ \ 110 = (111100110)_2$

Binary to Hexadecimal:

Steps

• Step 1 − Divide the binary digits into groups of four (starting from the right) as in table follow.

• Step 2 − Convert each group of four binary digits to one hexadecimal symbol.

Hexadecimal Number	Groups in Binary	Hexadecimal Number	Groups in Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Example

Convert $(1000011010101100.111111)_2$ to Hexadecimal

$1000011010101100.111111_2 = \underbrace{1000}_{8}\underbrace{0110}_{6}\underbrace{1010}_{A}\underbrace{1100}_{C}.\underbrace{1111}_{F}\underbrace{1100}_{C}$

$= 86AC.FC_{16}$

Hexadecimal to Binary:

Steps

• Step 1 − Convert each octal digit to a 4 digit binary number.

• Step 2 − Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Convert $(59AB.2F)_{16}$ to binary

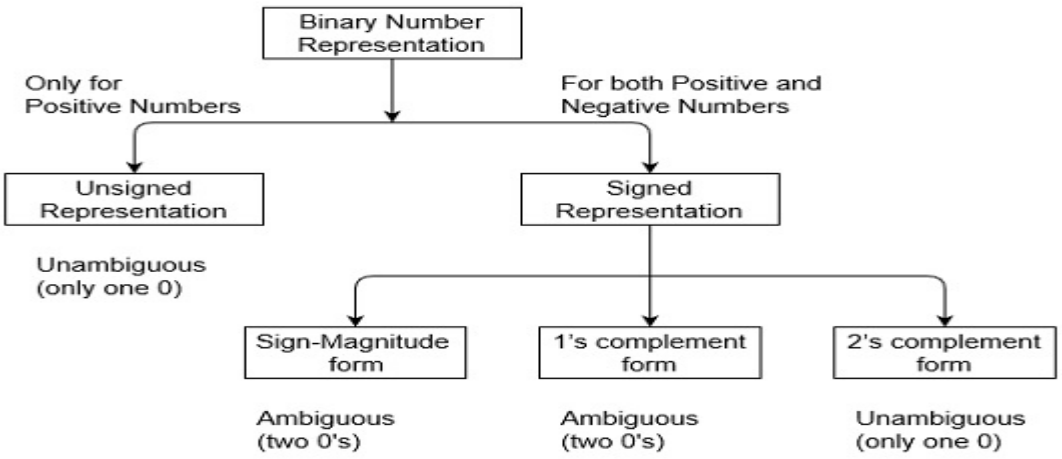
$59AB.2F_{16} = 5\ 9\ A\ B\ .\ 2\ F$

$= 0101\ 1001\ 1010\ 1011\ .\ 0010\ 1111$

$= 0101100110101011.00101111_2$

Signed and unsigned binary numbers

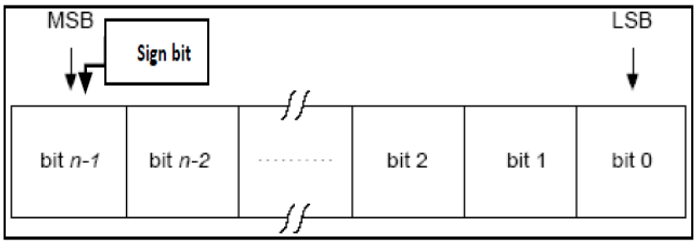
In decimal system, generally a plus (+) sign denotes a positive number whereas a minus (–) sign denotes a negative number. But, the plus sign is usually dropped, and no sign means the number is positive. This type of representation of numbers is known as ***signed numbers.***



But in digital circuits, there is no provision to put a plus or minus sign, since everything in digital circuits have to be represented in terms of 0 and 1. Normally an additional bit is used as the *sign bit*. This sign bit is usually placed as the MSB.

Generally, a 0 is reserved for a positive number and a 1 is reserved for a negative number.

For example, an 8-bit signed binary number 01101001 represents a positive number whose magnitude is $(1101001)_2 = (105)_{10}$. The MSB is 0, which indicates that the number is positive. On the other hand, in the signed binary form, 11101001 represents a negative number whose magnitude is $(1101001)_2 = (105)_{10}$. The 1 in the MSB position indicates that the number is negative and the other seven bits give its magnitude. This kind of representation of binary numbers is called ***sign-magnitude representation.***



Example :

Signed Integer	Sign Magnitude
+2	0000 0010
+1	0000 0001
0	0000 0000
-1	1000 0001
-2	1000 0010

Unsigned Numbers: Unsigned numbers don't have any sign, these can contain only magnitude of the number. So, representation of unsigned binary numbers are all positive numbers only. For example, representation of positive decimal numbers are positive by default. We always assume that there is a positive sign symbol in front of every number.

Representation of Unsigned Binary Numbers: Since there is no sign bit in this unsigned binary number, so N bit binary number represent its magnitude only. Zero (0) is also unsigned number. This representation has only one zero (0), which is always positive. Every number in unsigned number representation has only one unique binary equivalent form, so this is unambiguous representation technique. The range of unsigned binary number is from 0 to (2^n-1).

Example-1: Represent decimal number 92 in unsigned binary number. Simply convert it into Binary number, it contains only magnitude of the given number.
$= (92)_{10}$

$= (1x2^6+0x2^5+1x2^4+1x2^3+1x2^2+0x2^1+0x2^0)_{10}$

$= (1011100)_2$

It's 7 bit binary magnitude of the decimal number 92.

Example-2: Find range of 5 bit unsigned binary numbers. Also, find minimum and maximum value in this range.

Since, range of unsigned binary number is from 0 to (2^n-1). Therefore, range of 5 bit unsigned binary number is *from* 0 to (2^5-1) which is equal from minimum value 0 (i.e., 00000) to maximum value 31 (i.e., 11111).

Binary Arithmetic

Arithmetic operation in digital systems are usually done in binary because design of logic networks to perform binary arithmetic is much easier than for decimal. Binary arithmetic is carried out in much the same manner as decimal, except the addition and multiplication tables are much simpler.

The addition table for binary numbers is

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$ and carry 1 to the next column

Example : add $(13)_{10}$ and $(11)_{10}$ in binary.

$(13)_{10}= 1101$

$(11)_{10}= 1011$

$11000=(24)_{10}$

The subtraction table for binary numbers is

$0 - 0 = 0$

$0 - 1 = 1$ and borrow 1 from the next column

$1 - 0 = 1$

$1 - 1 = 0$

Borrowing 1 from a column is equivalent to subtracting 1 from that column.

Example : sub $(11)_{10}$ from $(13)_{10}$ in binary.

$1 \longleftarrow$ (Indicate a borrow from the 3rd column)

$(13)_{10}= 1101$

$(11)_{10}= 1011$

$0010=(2)_{10}$

Example : sub $(10000)_2$ and $(11)_2$ in binary.

```
 1111 ←——— ( Indicate a borrow )
 10000
    11
 ------
  1101
```

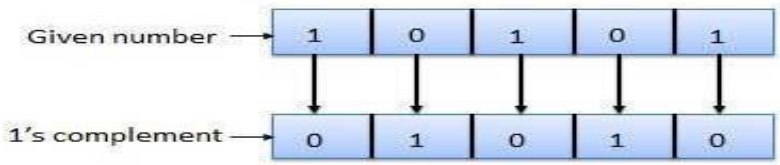
Complements of Numbers

complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations. There for in Binary system complements has base r = 2. So the two types of complements for the binary system are:

 # 1' complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement.

Example:



ones' complement can be used to represent negative numbers. The ones' complement form of a negative binary number is the complement of its positive counterpart, which can be obtained by applying the NOT to the positive counterpart.

Note: The range of signed numbers using ones' complement in a conventional 8-bit byte is −127 to +127.

Signed integer	Unsigned integer	8 bit ones' complement
0	0	00000000
1	1	00000001
…….	…….	……
125	125	01111101
126	126	01111111
-127	128	10000000
-126	129	10000001
-125	130	10000010
…….	…….	……
-1	245	11111110
-0	255	11111111

1's Complement Addition

To add two numbers represented in this system, we use the conventional binary addition, **but it is then necessary to add any resulting carry back into the resulting sum.**

Example: perform (7-3) using 1's complements method ?

```
Decimal      binary
  7    --->  111   ---->  111
- 3    --->  011        + 100
_____________________________
                       ¹ 011
                         +  1
                       _____
                         100  the result
```

Example: perform (8-12) using 1's complements method ?

```
 Decimal      binary
    8    ->    1000  -->    1000
-  12   ->    1100  --> +  0011
 _________              _______
 - 4                      1011
```

2' complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

Note: 2's complement = 1's complement + 1

Example of 2's Complement is as follows.

The Two's complement representation allows the use of binary arithmetic operations on signed integers, yielding the correct 2's complement results.

Positive Numbers

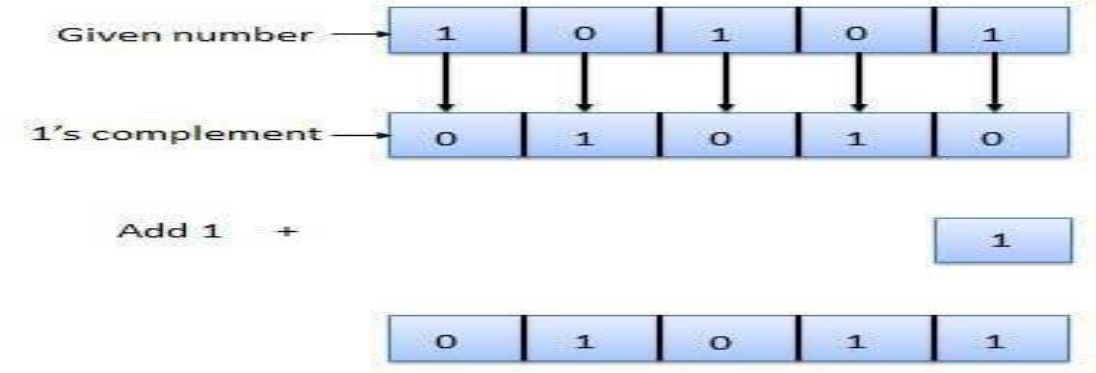
Positive 2's complement numbers are represented as the simple binary.

Negative Numbers

Negative 2's complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

Integer		2's Complement
Signed	Signed	
5	5	0000 0101
4	4	0000 0100
3	3	0000 0011
``2	2	0000 0010
1	1	0000 0001
0	0	0000 0000
-1	255	1111 1111
-2	254	1111 1110
-3	253	1111 1101
-4	252	1111 1100
-5	251	1111 1011

Note: The most significant (leftmost) bit indicates the sign of the integer; therefore it is sometimes called the sign bit.
If the sign bit is zero, then the number is greater than or equal to zero, or positive.
If the sign bit is one, then the number is less than zero, or negative.



For example:

17 → **-17**
0001 0001(binary 17) → 1110 1111(two's complement -17)

NOT(0001 0001) = 1110 1110 (**Invert bits**)

1110 1110 + 0000 0001 = **1110 1111** (Add 1)

2's Complement Addition

Two's complement addition follows the same rules as binary addition.

For example:

5 + (-3) = 2 0000 0101 = +5
\+ 1111 1101 = -3
0000 0010 = +2

2's Complement Subtraction

Two's complement subtraction is the binary addition of the minuend to the 2's complement of the subtrahend (adding a negative number is the same as subtracting a positive one).

For example:

7 - 12 = (-5) 0000 0111 = +7
\+ 1111 0100 = -12
1111 1011 = -5

Binary coded decimal (BCD) codes.

Code is a system of rules to convert information- such as a letter, word, sound, image, or gesture- into another form or representation, sometimes shortened or secret, for use it to security reasons.

There are many types of codes: -

1) Binary-Coded-Decimal Code (BCD – 8421) (Weighted Code)

If each digit of a decimal number is represented by its binary equivalent, the result is a code called binary-coded-decimal (hereafter abbreviated BCD). Since a decimal digit can be represented from (0 – 9), four bits are required to code each digit (the binary code for 9 is 1001).

Table below gives the four-bit code for one decimal digit.

Decimal Symbol	BCD Digit (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Advantages of BCD Code

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Code

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

To illustrate the BCD code, take a decimal number such as 874. Each digit is changed to its binary equivalent as follows:

```
  8      7      4      (decimal)
  ↓      ↓      ↓
1000   0111   0100     (BCD)
```

As another example, let us change 943 to its BCD-code representation:

```
  9      4      3      (decimal)
  ↓      ↓      ↓
1001   0100   0011     (BCD)
```

Example: Convert 0110100000111001 (BCD) to its decimal equivalent.

Solution: Divide the BCD number into four-bit groups and convert each to decimal.

$$\underbrace{0110}_{6}\ \underbrace{1000}_{8}\ \underbrace{0011}_{3}\ \underbrace{1001}_{9}$$

2) Gray Code (non-weighted code)

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

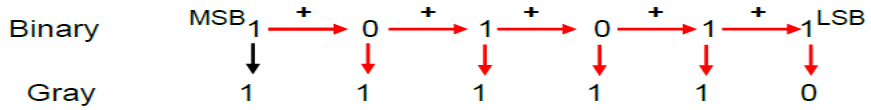
Applications of Gray Code

Gray code is popularly used in the shaft position encoders.

A shaft position encoder produces a code word which represents the angular position of the shaft.

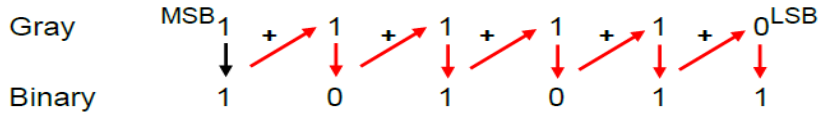
Example: Convert (101011)2 to Gray code.

Solution:



Example: Convert Gray code (111110) to Binary.

Solution:



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

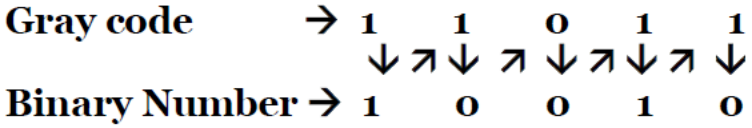
Gray to Binary Conversion

Steps

Step1: The **MSB** in the left is the **MSB** in binary number. In other word they are The same.

Step2: Add the first digit of the binary number to the second digit in Gary code, the carry is ignored , in other word , take XOR operation between them .

Step3: Generally working from the left to right digit, the n'th digit in the binary number is formed from summing the (n+1)'th digit in the binary number with n'th bit in the Gray code .



Then $(11011)_G = (10010)_2$

Binary to Gray Conversion

Steps:

Step1: The **MSB** digit in Gary code is the same as corresponding digit in the binary number.

Step2: going from left to right , add each adjacent pair of binary digit to get the next Gray digits , regardless carries .

Example: Convert the following binary number 100110 to Gray code?

Binary Number → 1 0 0 1 1 0

Gray Code → 1 1 0 1 0 1

Then $(100110)_2 = (110101)_G$

ASCII Character Code

The standard binary code for the alphanumeric characters is called ASCII (American Standard Code for Information Interchange). It uses 7 bits to code 128 characters, as shown in the table. The seven bits of the code are designed by A0 through A6, with A6 being the most significant bit. For example, the letter A is represented n ASCII as (1000001). The ASCII code contains 94 characters that can print and 34 nonprinting used in control functions. The printing characters consist of 26 uppercase letters, the 26 lowercase letters, 10 numerals, and 32 special printable character such as %, @ and $.

Example: What are the character corresponding to the ASCII code?

(1001010100111110010001001110010000010001001000101101010)ASCII

Sol

(1001010 1001111 1001000 1001110 0100000 1000100 1000101 1010110) ASCII

= JOHN DEV

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Chapter three: Karnaugh map (K-map)

Introduction

A karnauph map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as ***the minimum expression***. As you have seen , the effectiveness of the algebraic simplification depends on your familiarity with the laws, rules , and theorems of Boolean algebra and the on your ability to apply them. The Karnauph map on other hand, provide a "cookbook" method for simplification. other simplification techniques include the ***Quine- McClusky*** method and the **Espresso algorithm.**

Another method of simplification of Boolean function is Karnaugh – Map (K-Map). This map is a diagram made of squares, each square represent one minterms, and there are several types of K-|Map depending on the number of variables in Boolean function.

1. Two, three, and four-variable Karnaugh map.

Two variable Karnauph map

- ✓ The 2-variable karnauph map is an array of **four cells**, as shown in table(1) bellow.
- ✓ In this case A and B are used for the variables although other letters could be used.
- ✓ Binary values of A are along the left side and the values of B are across the top. The value of a given cell is the binary values of A at the left in the same row combined with the value of B at the to in the same column.

For example, the cell in the upper left corner has a binary value 00 and the cell in the lower right corner has a binary value 11. Table(2) shows the standard product terms that are represented by each cell in the 2-varable Karnaugh map.

Table(1)

A \ B	0	1
0	00	01
1	10	11

Table(2)

A \ B	0	1
0	$\overline{A}\,\overline{B}$	$\overline{A}B$
1	$A\overline{B}$	AB

three- Variable Karnauph map

- ✓ The 3-variable karnauph map is an array of **eight cells**, as shown in table (1) bellow.
- ✓ In this case A, B and C are used for the variables although other letters could be used.
- ✓ Binary values of A and B are along the left side and the values of C are
- ✓ across the top.
- ✓ The value of a given cell is the binary values of A and B at the left in the
- ✓ same row combined with the value of C at the to in the same column.

For example, the cell in the upper left corner has a binary value 000 and the cell in the lower right corner has a binary value 101. Table (2) shows the standard product terms that are represented by each cell in the 3-varable Karnaugh map.

Table(1)

AB \ C	0	1
00	000	001
01	010	011
11	110	111
10	100	101

Table(2)

AB \ C	0	1
00	$\overline{A}\overline{B}\overline{C}$	$\overline{A}\overline{B}C$
01	$\overline{A}B\overline{C}$	$\overline{A}BC$
11	$AB\overline{C}$	ABC
10	$A\overline{B}\overline{C}$	$A\overline{B}C$

Four- Variable Karnauph map

The 4-variable karnauph map is an array of sixteen cells, as shown in table (1) bellow .

- ✓ In this case A, B C, and D are used for the variables although other letters could be used.
- ✓ Binary values of A and B and C and D are along the left side and the values of C are across the top.
- ✓ The value of a given cell is the binary values of A and B at the left in the same row combined with the value of C and D at the to in the same column.

For example, the cell in the upper right corner has a binary value 0010 and the cell in the lower right corner has a binary value 1010. Table (2) shows the standard product terms that are represented by each cell in the 4-varabl Karnaugh map.

Table(1)

CD / AB	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Table(2)

CD / AB	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$C\bar{B}\bar{C}\bar{D}$	$C\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

2. Minimum SOP expressions using the Karnaugh map.

As stated in the last section, the Karnaugh map is used for simplifying Boolean expressions to their minimum form. A minimized SOP expression contains the fewest possible terms with the fewest possible variables per term.

Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression.

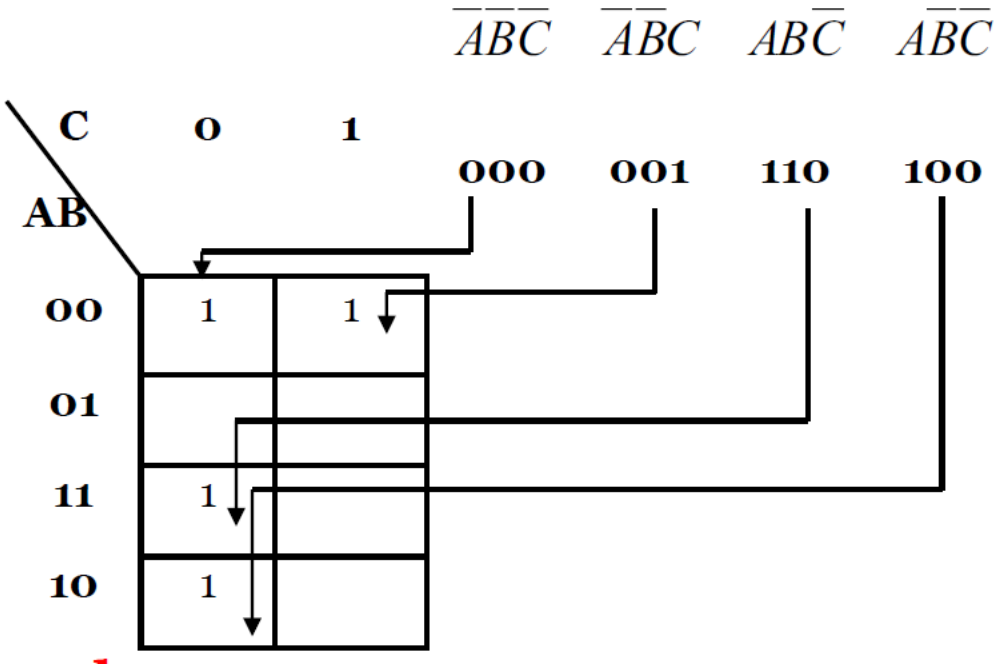
- Mapping a standard SOP Expression

- For an SOP expression in standard form, a 1 is placed on the Karnaugh map for each product term in the expression. Each 1 is placed in a cell corresponding to the value of a product term. For example, for the product term *ABC*, a 1 goes in the 101 cell on a 3-variablemap.
- When an SOP expression is completely mapped, there will be a number of 1's on the Karnaugh map equal to the number of product terms in the standard SOP expression.
- The cells that do not have a 1 are the cells for which the expression is 0.
- Usually, when working with SOP expressions, the 0's are left off the map.

The following steps and the illustration in the next figure show the mapping process.

Step1: Determine the binary value of each product term in the standard SOP expression. After some practice, you can usually do the evaluation of terms mentally.

Step2: As each product term is evaluated, place a 1 on the Karnaugh map in the cell having the same value as the product term.



Karnaugh Map Simplification of SOP expressions

The process that results in an expression containing the fewest possible terms with the fewest possible variables is called **minimization**. After an SOP expression has been mapped , a minimum SOP expression is obtained by ***grouping the 1's*** and ***determining the minimum SOP expression from the map***.

 grouping the 1's : you can group 1s on the Karnaugh map according to the following rules by enclosing those adjacent cells containing 1s.the goal is to maximize the size of the groups and to minimize the number of groups.

1. A group must contain either 1,2,4,8 , or 16 cells,which are all powers of two. In the case of a 3-variables map,2^3=8 cells is the maximum group.

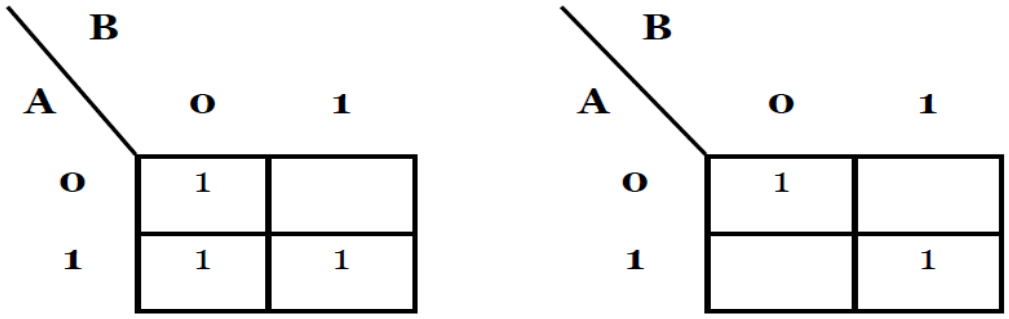
2. Each cell in a group must be adjacent to one or more cells in that some group, but all cells in the group do not have to be adjacent to each other.

3. always include the largest possible number of 1s in a group in accordance with rule 1.

4. Each1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include non common 1s.

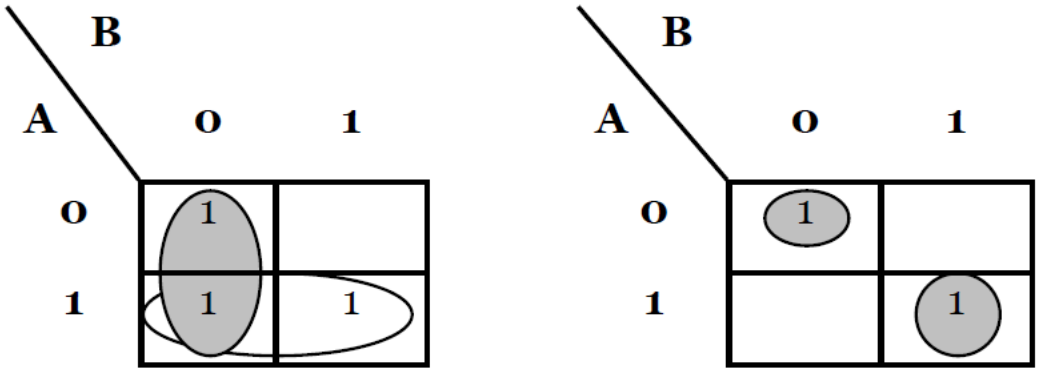
Example: group the 1s in each of the Karnauph maps in the following 2-varible map.

2-variable map



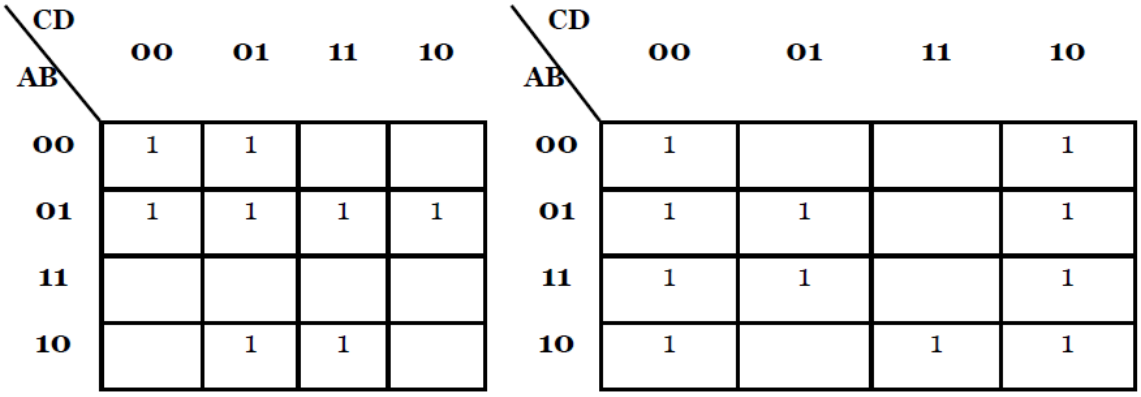
Solution: the grouping are shown the next figure. In some cases, there may be more than one way to group the1s to form maximum grouping.

2-variable map



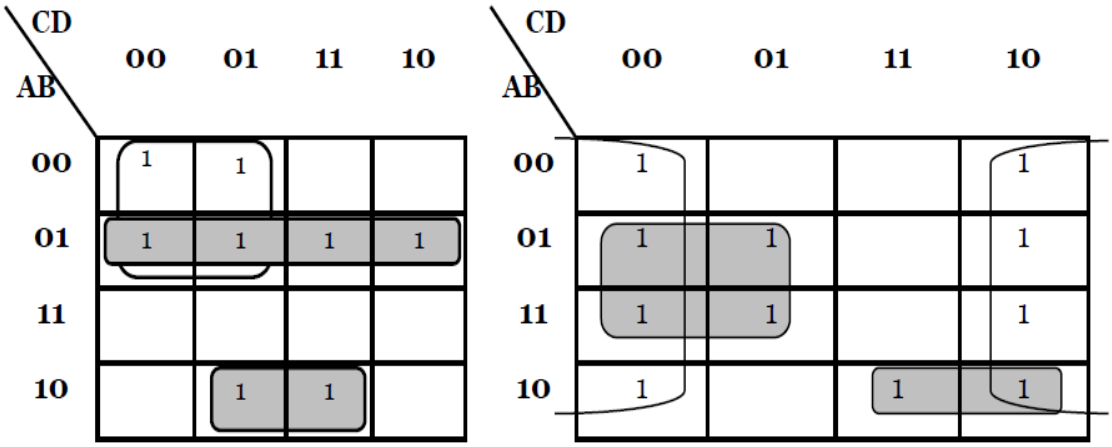
Example: group the 1s in each of the Karnauph maps in the following 4-variable map

4-variable map



Solution: the grouping are shown the next figure. In same cases, there may be more than one way to group the1s to form maximum grouping.

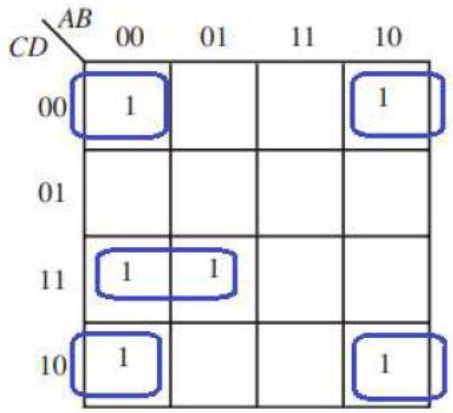
4-variable map



Example: Simplify the following SOP expression on a Karnaugh map:

$F = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}CD + A\bar{B}C\bar{D}$

Solution:

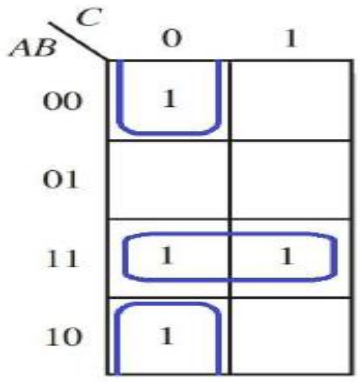


Example: Determine the simply expression by the truth table below using Karnaugh map method.

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

Solution:

$F = AB + \bar{B}\bar{C}$



Determining the minimum SOP Expression from the map

When all the 1s representing the standard product terms in an expression are properly mapped and grouped, the process of determining the resulting minimum SOP expression begins. The following rules ate applied to find the minimum product terms and the minimum SOP expression:

1. Group the cell that have 1's. Each group of cells containing 1's creates one product item composed of all variables that occur in only one form (either uncomplemented or complemented) within the group. ***Variable that occur both uncomplemented and complemented within the group are eliminated . these are called contradictory variables***

2. Determine the minimum product term for each group.

a. For a 2-Variable map

(1) A 1-cell group yields a 2-variable product term.
(2) A 2-cell group yields a 1-variable product term.
(3) An 4-cell group yields a value of 1 for the expression.

b. For a 3-Variable map

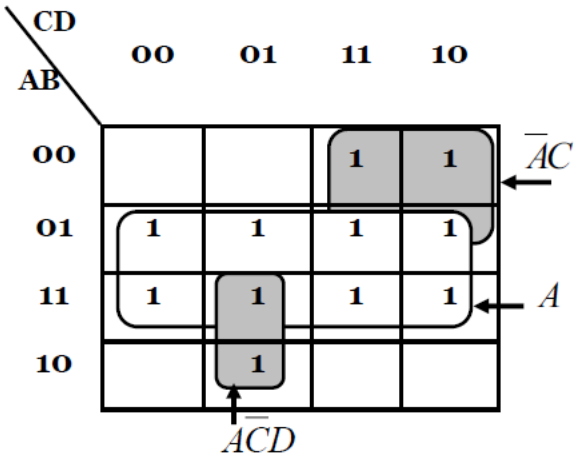
(1) A 1-cell group yields a 3-variable product term .
(2) A 2-cell group yields a 2-variable product term.
(3) A 4-cell group yields a 1-variable product term.
(4) An 8-cell group yields a value of **1** for the expression

c. For a 4-Variable map

(1) A 1-cell group yields a 4-variable product term**.**
(2) A 2-cell group yields a 3-variable product term.
(3) A 4-cell group yields a 2-variable product term.
(4) An 8-cell group yields a 1-variable term.
(5) A 16-cell group yields a value of 1 for the expression.

3. When all the minimum product terms are derived from the Karnaugh map, they are summed to form the minimum SOP expression.

Example: Determine the product terms for the Karnaugh map in figure bellow, and write the resulting minimum SOP expression.



Solution: Eliminate variables that are in a grouping in both complemented and uncomplemented forms. In the above figure,

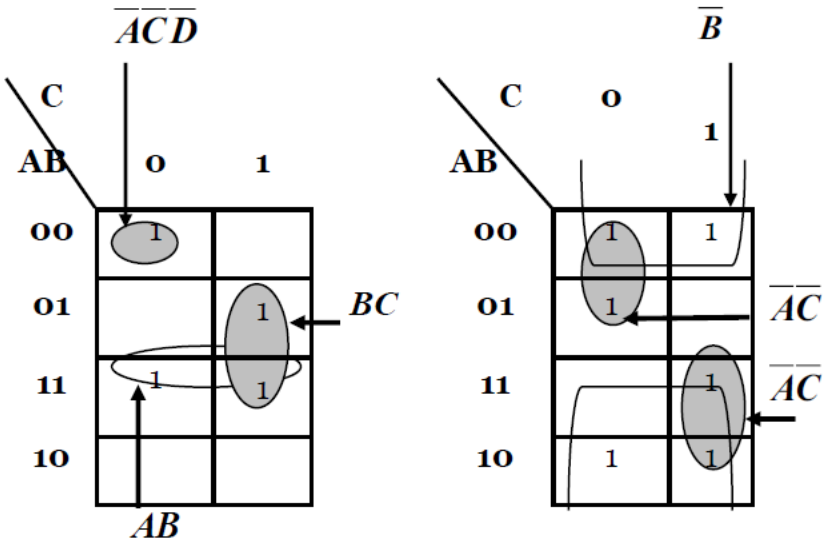
- the product term for the 8-cell group is B because the cells withen that group contain both A and $\overline{A}$, C and $\overline{C}$, and D and $\overline{D}$, which are eliminated.
- The 4-cell group contains $B, \overline{B}, D, and\, \overline{D}$, leaving the variables $\overline{A}$ and C. which form the product term $\overline{A}C$.
- The 2-cell group contains B and $\overline{B}$, leaving variables A , $\overline{C}$, and D which form the product term $A\overline{C}D$.

Notes: how overlapping is used to maximize the size of the groups. The resulting minimum SOP expression is the sum of these product terms:

$$B + \overline{A}C + A\overline{C}D$$

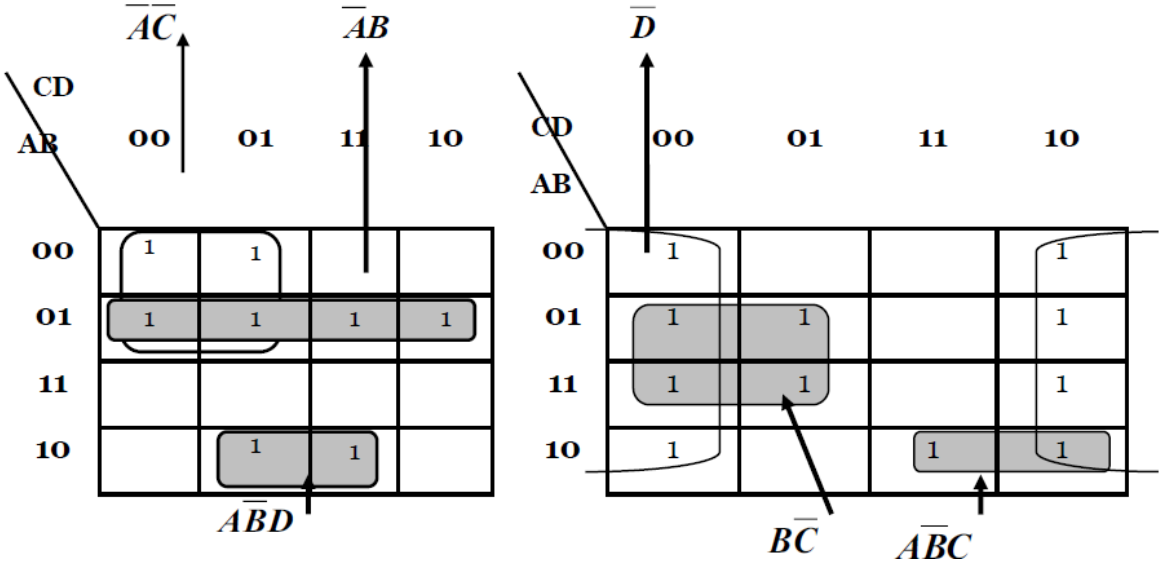
***Example*:** Determine the product terms for the Karnaugh map in two figures bellow, and write the resulting minimum SOP expression.

Figure(a) **Figure(b)**



***Example*:** Determine the product terms for the Karnaugh map in two figures bellow, and write the resulting minimum SOP expression.

Figure (c) **Figure (d)**



Solution: *the resulting minimum product term for each group shown in figure(a) and (b) are:*

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

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

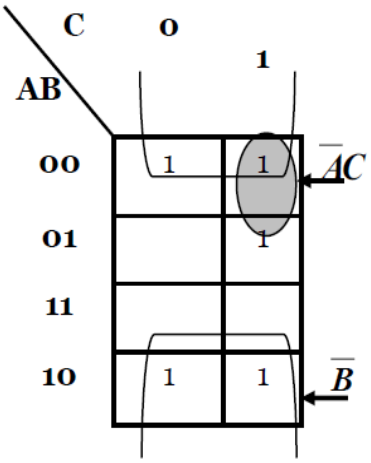
Example: *Use a Karnaugh map to minimize the following standard SOP expression:*

$$A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C}$$

Solution: the binary values of the expression are

101 + 011 + 001 + 000 + 100

Map the standard SOP expression and group the cells are shown in the following figure.



The resulting minimum SOP expression is : $\overline{B} + \overline{A}C$

Simply the following Boolean functions using K –Map?

1 - $F = \overline{X}\,Y\,Z + X\overline{Y}\,\overline{Z} + X\overline{Y}\,Z + \overline{X}\,Y\overline{Z}$

X \ YZ	00	01	11	10
0			1	1
1	1	1		

$F = X\overline{Y} + \overline{X}\,Y$

If the function is simplified using Boolean- algebra

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

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

2 - $F = \overline{X}\,Y\,Z + X\overline{Y}\,\overline{Z} + X\,Y\,Z + X\,Y\overline{Z}$

X \ YZ	00	01	11	10
0			1	
1	1		1	1

$F\,(X,Y,Z) = \sum(0, 2, 4, 5, 6)$

X \ YZ	00	01	11	10
0	1			1
1	1	1		1

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

$$F(X,Y,Z,W) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

XY \ ZW	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

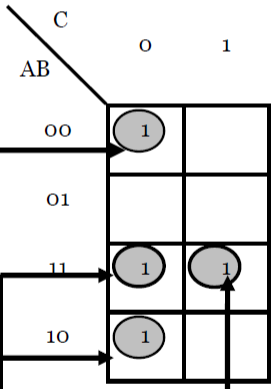
♦ Mapping Directly from the Truth Table

You have seen how to map a Boolean expression; now you will learn how to go directly from a truth table to a Karnauph map. Recall that a truth table gives the output of a Boolean expression for all possible input variable combination.

An example of a Boolean expression and its troth table representation is shown in the next figure. Notice in the truth table that the output X is 1 for four different input variable combinations.

$$X = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$$

Input			Output
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



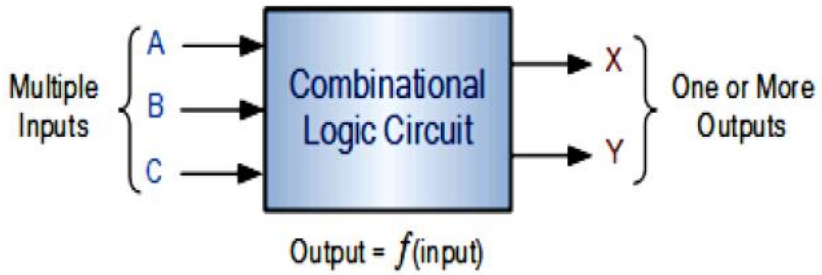
The 1s in the output column of the truth table are mapped directly onto a Karnaugh map into the cells corresponding to the values of the associated input variable combinations.

Chapter two: Combinational Logic Circuits and switching algebra and logic gates

Introduction

Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs.

Or the combinational logic circuits or time-independent logic circuits in digital circuit theory can be defined as a type of digital logic circuit implemented using Boolean circuits, where the output of logic circuit is a pure function of the present inputs only.



Development of a truth table.

What is a Truth Table?

- **Definition:** A truth table is a tabular representation that lists all possible input combinations to a logic gate or circuit, along with their corresponding outputs.
- **Purpose:** It's used for analysing and predicting the behaviour of digital circuits, making it easier to understand how changes in input affect the output.

The operation of the AND, OR, and NOT logic operators can be formally described by using a **truth table**. A truth table is a two-dimensional array where there is one column for each input and one column for each output (a circuit may have more than one output). Since we are dealing with binary values, each input can be either a 0 or a 1. We simply enumerate all possible combinations of 0's and 1's for all the inputs.

Two-Valued Boolean Algebra

Two-valued Boolean algebra is defined on a set of only two elements, S = {0,1}, with rules for two binary operators (+) and (.) and **inversion** or **complement** as shown in the following operator tables at Figures 1, 2, and 3 respectively.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2

A	A′
0	1
1	0

Figure 3

These rules are exactly the same for as the logical **OR**, **AND**, and **NOT** operations, respectively.

Using a truth table is one method to formally describe the operation of a circuit or function. The truth table for any given logic expression (no matter how complex it is) can always be derived. Examples on the use of truth tables to describe digital circuits are given in the following sections. Another method to formally describe the operation of a circuit is by using Boolean expressions or Boolean functions.

The basic building blocks of a computer are called ***logical gates***. Gates are basic circuits that have at least one (and usually more) ***input*** and exactly one output. Input and output values are the logical values ***true*** and ***false***. In computer architecture it is common to use 0 for false and 1 for true. Gates have no memory. The value of the output depends only on the current value of the inputs. A useful way of describing the relationship between the inputs of gates and their output is the truth table. In a truth table, the value of each output is tabulated for every possible combination of the input values. We usually consider three basic kinds of gates, ***AND***-gates, ***OR***-gates, and ***NOT***-gates (or ***Inverters***).

Basic logic Gates

1) AND Gate: -

The AND operation is represented by a dot (**.**) or by the absence of an operator. **E.g.** A.B=F AB=F are all read as A AND B=F. the logical operation AND is interpreted to mean that F=1 if A=1 and B=1 otherwise F=0

Symbol

Truth Table

A	B	F = (A.B)
0	0	0
0	1	0
1	0	0
1	1	1

2) OR Gate: -

The OR operation is represented by a (+) sign for example, A+B=F is interpreted as A OR B=F meaning that F=1 if A=1 or B=1 or if both A=1 and B=1. If both A and B are 0, then F=0.

Symbol

Truth Table

A	B	F = (A+B)
0	0	0
0	1	1
1	0	1
1	1	1

3) NOT Gate or INVERTER: -

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Symbol

Truth Table

A	B
0	1
1	0

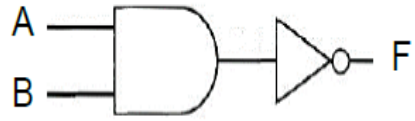
Combined gates

Sometimes, it is practical to combine functions of the basic gates into more complex gates, for instance in order to save space in circuit diagrams. In this section, we show some such combined gates together with their truth tables.

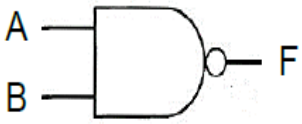
1) NAND Gate: -

The NAND-gate is an AND-gate with an inverter on the output. This operation is represented by $F=\overline{A.B}$. So instead of drawing several gates like this:

Symbol



We draw a single NAND-gate with a little ring on the output like this:



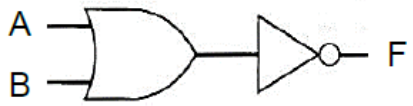
Truth Table

A	B	$F = \overline{(A.B)}$
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

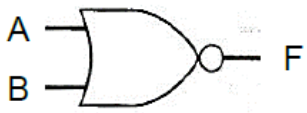
2) NOR Gate:

The NOR-gate is an OR-gate with an inverter on the output. This operation is represented by $F=\overline{A+B}$. So instead of drawing several gates like this:

Symbol



We draw a single NOR-gate with a little ring on the output like this:



Truth Table

A	B	$F = \overline{(A+B)}$
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

3) Exclusive OR Gate(Ex-OR/ XOR): -

The Exclusive-OR-Gate is similar to an OR-gate. It can have an arbitrary number of inputs, and its output value is 1 if exactly one input is 1 (and thus the others 0). Otherwise, the output is 0. This operation is represented by $F=A.\bar{B}+\bar{A}.B = A \oplus B$.

Symbol

Truth Table

A	B	F = A⊕B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

4) Exclusive NOR Gate(Ex-NOR/ XNOR): -

The Exclusive-NOR-Gate is similar to an NOR-gate. It can have an arbitrary number of inputs, and its output value is 1 if the two inputs are of the same values (1 and 1 or 0 and 0). Otherwise, the output is 0. This operation is represented by $F=A.B+\bar{A}.\bar{B} = A \odot B$.

Symbol

Truth Table

A	B	F = A⊙B
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

Digital Logic Gates Summary

The following logic gates truth table compares the logical functions of the 2-input logic gates.

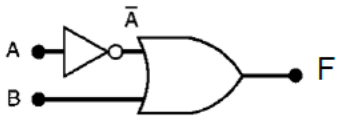
Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Implementing Combinational Logic.
From a Boolean Expression to a Logic Circuit.

Examples: Draw the following functions?

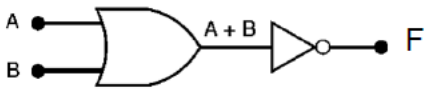
1) $F = \overline{A}+B$

Solution:



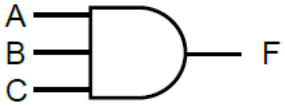
$F = \overline{A+B}$

Solution:



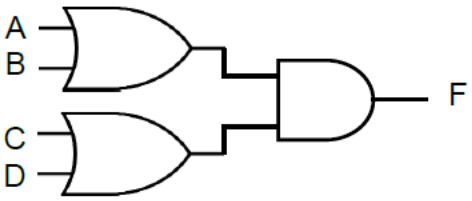
2) $F = A.B.C$

Solution:



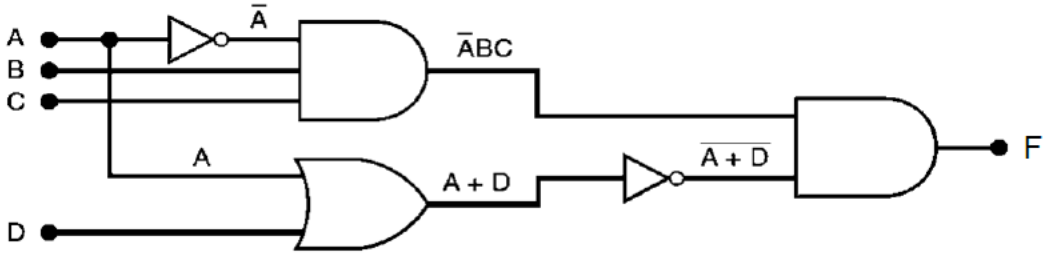
3) $F = A+B.C+D$

Solution:



4) $F = \overline{A}BC(\overline{A + D})$

Solution:



Switching algebra.

Switching algebra is also known as **Boolean Algebra**. It is used to analyze digital gates and circuits It is logical to perform a mathematical operation on binary numbers.

- A Boolean Variable takes the value of either 0 (False) or 1 (True).
- Symbols are used to represent Boolean variables e.g. A, B, C, X, Y, Z

- There are three basic logic operations AND, OR, NOT
- The Boolean Operators are • + ¯
  - A + B means A OR B
  - A • B means A AND B
  - $\overline{A}$ means NOT A
- Nodes in a circuit are represented by Boolean Variables

Properties of switching algebra

These are the simple Boolean postulates. We can verify these postulates easily, by substituting the Boolean variable with '0' or '1'.

Basic Laws of Boolean Algebra

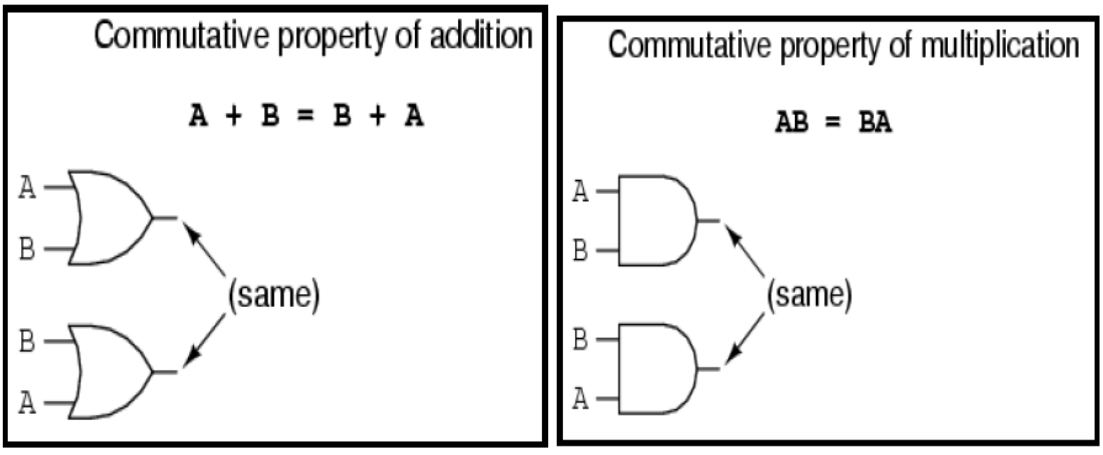
Three basic laws of Boolean Algebra: **Commutative law, Associative law** and **Distributive law.**

• Commutative Law

If any logical operation of two Boolean variables give the same result irrespective of the order of those two variables, then that logical operation is said to be Commutative. The logical OR & logical AND operations of two Boolean variables A & B are shown below

$$A + B = B + A$$
$$A.B = B.A$$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit. Remember, Boolean Algebra as applied to logic circuits, the commutative law can applied to OR and AND gate makes no difference , as show in next figures.



• Associative Law

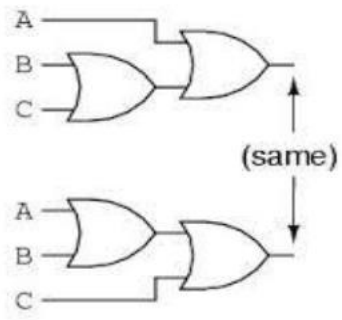
If a logical operation of any two Boolean variables is performed first and then the same operation is performed with the remaining variable gives the same result, then that logical operation is said to be Associative. The logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

$$A + (B + C) = (A + B) + C$$

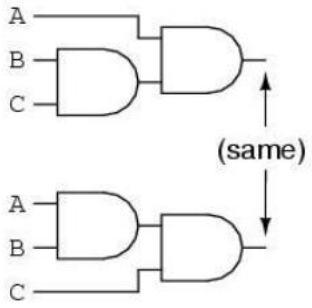
$$A.(B.C) = (A.B).C$$

The follows figures show how to applied the associative low to 2-input OR gates and 2-input And gates.

A + (B + C)=(A +B) +C



A(BC) = (AB)C

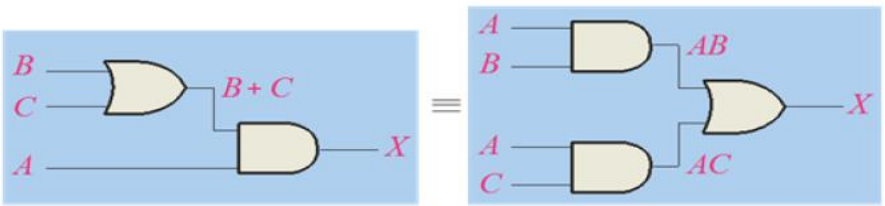


• Distributive Law

If any logical operation can be distributed to all the terms present in the Boolean function, then that logical operation is said to be Distributive. The distribution of logical OR & logical AND operations of three Boolean variables x, y & z are shown below.

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

The follows figures show how to applied the distributive low to 2-input OR gates and 2-input And gates.



$A(B+C)$ $AB + AC$

Where the symbol ≡ mean "equivalent to"

Complement Law

This law states that in case a complement is added to any variable, then it would give one, whereas when we multiply this variable with its own complement, then it would result in '0', i.e.,

- A + A' = 1
- A.A' = 0

Rules of Boolean Algebra

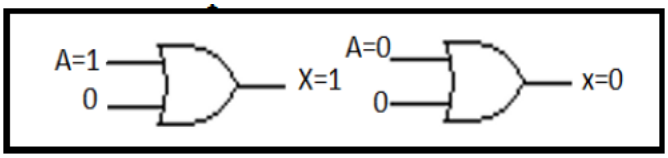
The following table lists the 12 basic rules that are useful in manipulating and simplifying **Boolean expressions**.

No.	Rule	No.	Rule
1	A +0 =A	7	A .A = A
2	A + 1= 1	8	A. $\overline{A}$ = 0
3	A . 0=0	9	$\overline{\overline{A}}$ = A
4	A .1 = A	10	A +AB =A
5	A +A= A	11	A + $\overline{A}$ B=A +B
6	A + $\overline{A}$ =1	12	(A+B)(A+C)=A+BC

Notes: A ,B or C can represent a single variable or a combination of variables

Rule1: A + 0 = A (Identity Law)

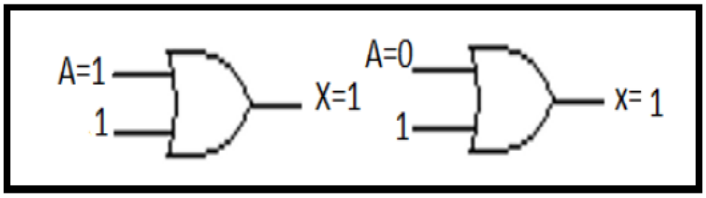
The variable ORed with 0 is always equal to the variable . This rule is illustrated in the following Figure , where the lower input is fixed at 0.



X=A+o=A

Rule 2 : A + 1 = 1 (NULL Elements Law)

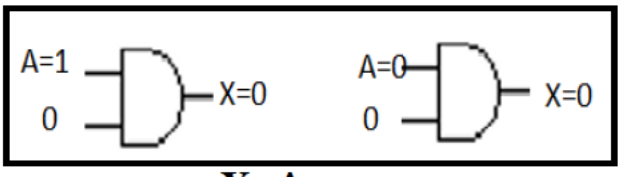
A variable ORed with 1 is always equal to 1. This rule is illustrated in the following Figure , where the lower input is fixed at 1.



X=A+1=1

Rule 3: A .0 = 0 (NULL Elements Law)

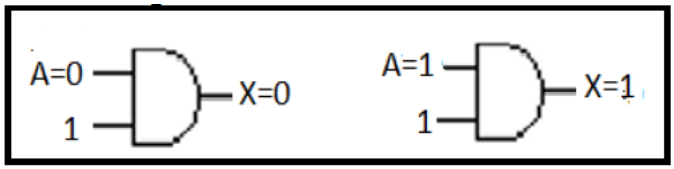
A variable ANDed with 0 is always equal to 0. This rule is illustrated in the following Figure , where the lower input is fixed at 0.



X=A . o=o

Rule 4: A .1 = A (Identity Law)

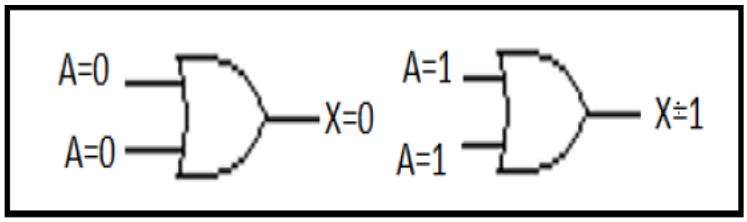
A variable ANDed with 1 is always equal to the variable . This rule is illustrated in the following Figure , where the lower input is fixed at 1.



X=A . 1=A

Rule 5: A + A= A (Idempotent Law)

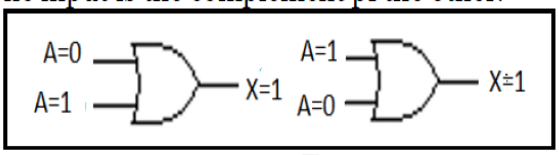
A variable ORed with itself is always equal to the variable . This rule is illustrated in the following Figure, where both inputs are the same variable .



X=A +A=A

Rule 6 : A + $\bar{A}$ = 1

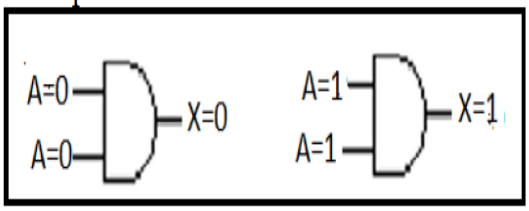
A variable ORed with its complement is always equal to 1. This rule is illustrated in the following Figure , where one input is the complement pf the other.



X=A + $\bar{A}$ =1

Rule 7: A .A = A (Idempotent Law)

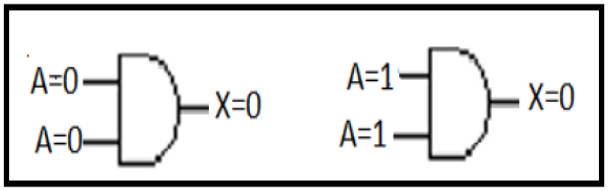
A variable Anded with itself is always equal to the variable . This rule is illustrated in the following Figure , where both inputs are the same variable .



X=A . A= A

Rule 8: A . $\bar{A}$ = 0 (Complement Law)

A variable ANDed with its complement is always equal to 0. This rule is illustrated in the following Figure.



$X = A . \bar{A} = 0$

Rule 9 : $\bar{\bar{A}}$ = A (Complement Law)

The double complement of a variable is always equal to the variable. This rule is illustrated in the following Figure using inverters .

$$\bar{\bar{A}} = A$$

Rule 10: A + AB = A

This rule can be proved by applying the distributive law , rule 2 and rule 4 as follows:

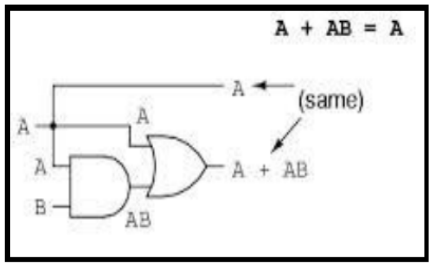
A + AB = A .1 + AB= A(1 + B)	factoring (distributive law)
=A . 1	Rule2 :(1 +b)=1
=A	Rule 4: A . 1 =A

Note : the proof is shown in table bellow , which shows the troth table and the resulting logic circuit simplification .

1- troth table

A	B	AB	A+AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

2- logic circuit



Rule 11: $A + \bar{A}B = A + B$

This rule can be proved as follows :

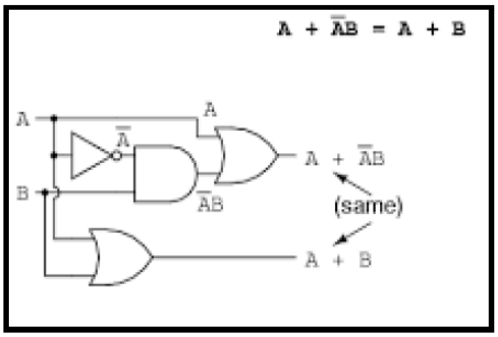
$A + \bar{A}B = (A + AB) + \bar{A}B$	Rule 10: $A = A + AB$
$= (AA + AB) + \bar{A}B$	Rule 7 : $A = AA$
$= AA + AB + A\bar{A} + \bar{A}B$	Rule 8:adding $A\bar{A} = 0$
$= (A + \bar{A})(A + B)$	factoring
$= 1 . (A + B)$	Rule 6: $A + \bar{A} = 1$
$= A + B$	Rule4 :drop the 1

Note : the proof is shown in table bellow , which shows the troth table and the resulting logic circuit simplification

1- troth table

A	B	$\bar{A}B$	$A + \bar{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

2- logic circuit



Rule 12 : (A + B)(A + C) = A+BC

This rule can be proved as follows :

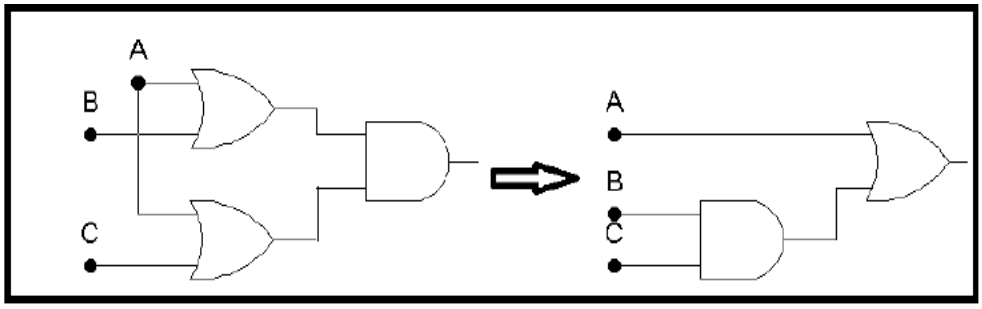
$$(A + B)(A+ C) = AA + AC + AB + BC$$
$$= A + AC + AB + BC$$
$$= A(1+C) + AB + BC$$
$$= A \cdot 1 + AB + BC$$
$$= A(1 + B) + BC$$
$$= A + BC$$

Note : the proof is shown in table bellow , which shows the troth table and the resulting logic circuit simplification

1- Troth Table

A	B	C	A+B	A+C	(A+B) (A +C)	BC	A+BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

logic gates



Manipulating Algebraic Functions

Boolean algebra deals with binary variables and logic operation. A **Boolean Function** is described by an algebraic expression called **Boolean expression** which consists of binary variables, the constants 0 and 1 and the logic operation symbols. Consider the following example:

$$F(A, B, C, D) = A + \overline{BC} + ADC$$

Boolean Function — Boolean Expression

Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result. It is possible to convert the switching equation into a truth table. For example consider the following switching equation.

$$F(A, B, C) = A + BC$$

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Simplification of Boolean Functions

Many times in the application of Boolean algebra, you have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently. the approach taken un this section is to use the basic laws, and theorems of Boolean algebra to manipulate and simplify an expression. This method depends on a thorough knowledge of Boolean algebra and considerable practice in its application. The following two theorems are used in Boolean algebra.

- Duality theorem
- DeMorgan's theorem

DeMorgan's Theorem

This theorem is useful in finding the **complement of Boolean function**. It states that the complement of logical OR of at least two Boolean variables is equal to the logical AND of each complemented variable. The two theorems suggested by De-Morgan which are extremely useful in Boolean Algebra are as following.

Theorem 1

$$\overline{A.B} = \overline{A} + \overline{B}$$

NAND = Bubbled OR

The left hand side (LHS) of this theorem represents a **NAND** gate with input A and B where the right hand side (RHS) of the theorem represents an **OR** gate with inverted inputs.

 This **OR** gate is called as **Bubbled OR**.

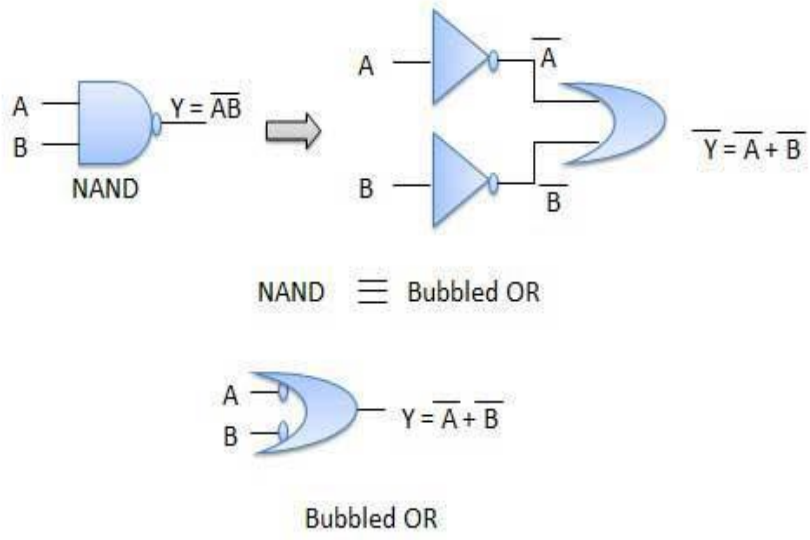


Table showing verification of the De-Morgan's first theorem

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

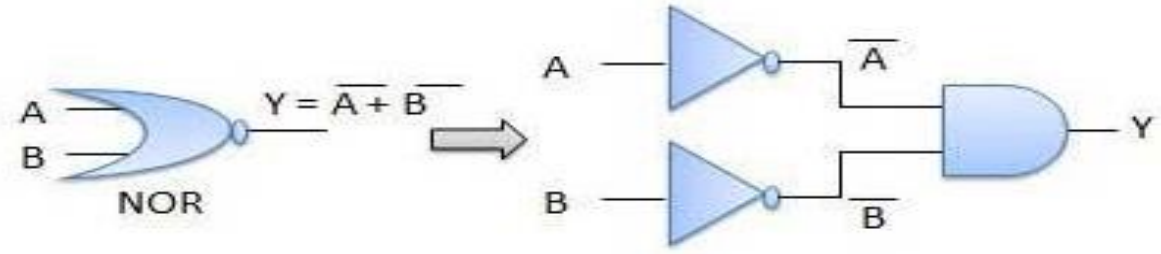
Theorem 2

$$\overline{A + B} = \overline{A}.\overline{B}$$

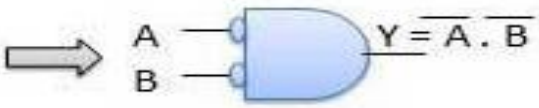
NOR = Bubbled AND

The LHS of this theorem represented a **NOR** gate with input A and B whereas the RHS represented an **AND** gate with inverted inputs.

 This **AND** gate is called as **Bubbled AND**.



NOR ≡ Bubbled AND



Bubbled AND

Table showing verification of the De-Morgan's second theorem

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

Examples: Apply DemMorgan's theorems to the following expression:

$\overline{A+\overline{BC}}$

$=\overline{A+\overline{B}+\overline{C}}$

$=\overline{A}.\overline{\overline{B}}.\overline{\overline{C}}$

$=\overline{A}.B.C$

Ex2: $\overline{\overline{C}+(\overline{A}+B)}$

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

$C.(\overline{\overline{A}}.\overline{B})$

$C.(A.\overline{B})$

$A.\overline{B}.C$

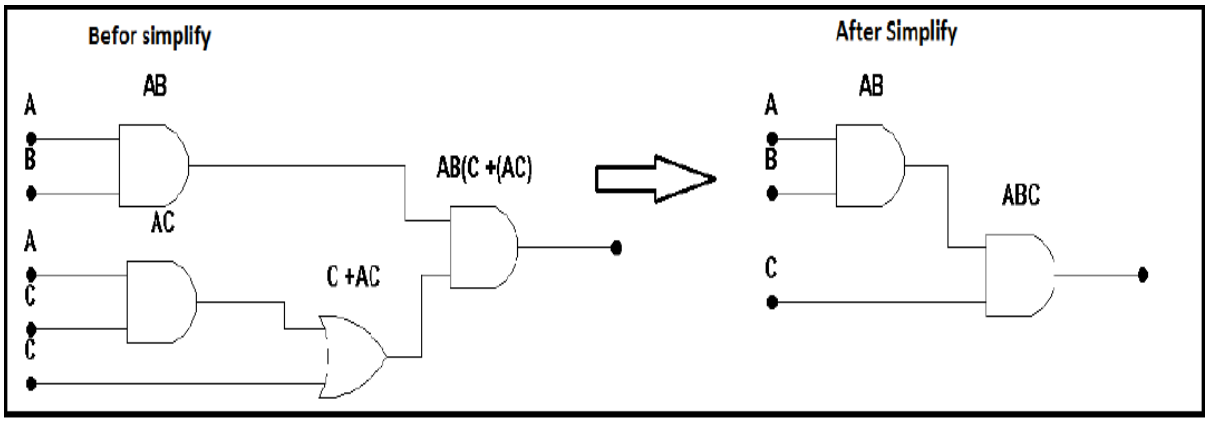
Examples: simplify the following Boolean functions using laws and rules:

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

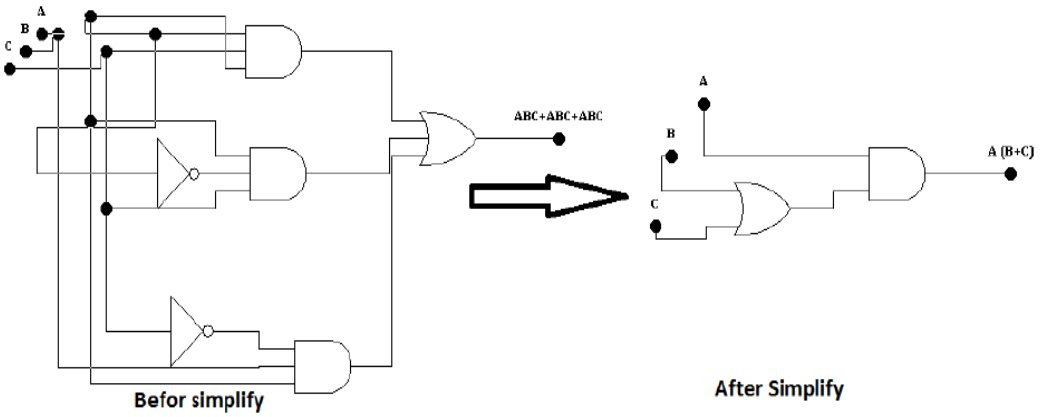
2. $F=(A+B).(A+C)$
   $=AA+AC+AB+BC$
   $=A+AC+AB+BC$
   $=A(1+C)+AB+BC$
   $=A+AB+BC$
   $=A(1+B)+BC$
   $=A+BC$

3. $F=\overline{X}(X+Y)$
   $=X\overline{X}+\overline{X}Y$
   $=0+\overline{X}Y$
   $=\overline{X}Y$

4. $F=AB(C+\overline{A}\overline{C})$
   $=ABC+AB\overline{A}\overline{C}$
   $=ABC+(A\overline{A})B\overline{C}$
   $=ABC+0.B\overline{C}$
   $=ABC$



$$
\begin{aligned}
3 - & ABC + A\overline{B}C + AB\overline{C} \\
& = AC\left(B + \overline{B}\right) + AB\overline{C} \\
& = AC.1 + AB\overline{C} \\
& = AC + AB\overline{C} \\
& = A(C + B\overline{C}) \\
& = A(C + B)
\end{aligned}
$$



3- AB + A(B+C)+ B(B+C)

= AB +AB +AC +BB +BC by applying distributive law

= AB +AB +AC +B +BC by applying rule 7

= AB +AC +B +BC by applying rule 5

=AB+AC+B by applying rule 10

=B+AC by applying rule 10

As this point the expression is simplified as much as possible.

Home work: Draw the logic circuit before and after simplification.

Simplifying Logic Circuits
Two methods for simplifying

- Algebraic method (use Boolean algebra theorems)
- Karnaugh mapping method as explained in chapter three.

Standard form of Boolean Expressions

One way to express Boolean function is in standard form. In this form, the terms form the function may contain one, two, or any number of literals, there are two types of standard form:

1– **Sum of Product:** The Sum of Product expression is equivalent to the logical AND fuction which Sums two or more Products to produce an output.

Some examples are:

$$AB + ABC$$

$$ABC + CDE + \overline{B}\,C\,\overline{D}$$

$$\overline{A}B + \overline{A}B\,\overline{C} + AC$$

In an Pos expression, a single overbar cannot extend over more than one variable in a term can have an overbar. for example, an SOP expression can have the term:

$$\overline{A}\,\overline{B}\,\overline{C} \; but\, not \; \overline{ABC}$$

Domain of a Boolean Expression

The **Domain** of a general Boolean expression is the set of variables contained in the expression in either complement or un complement form. For example, the domain of the expression $\overline{A}B + A\overline{B}C$ is the set of variables A,B ,C and the domain of the expression $AB\overline{C} + C\overline{D}E + \overline{B}C\overline{D}$ is a set of variables A,B,C,D,E.

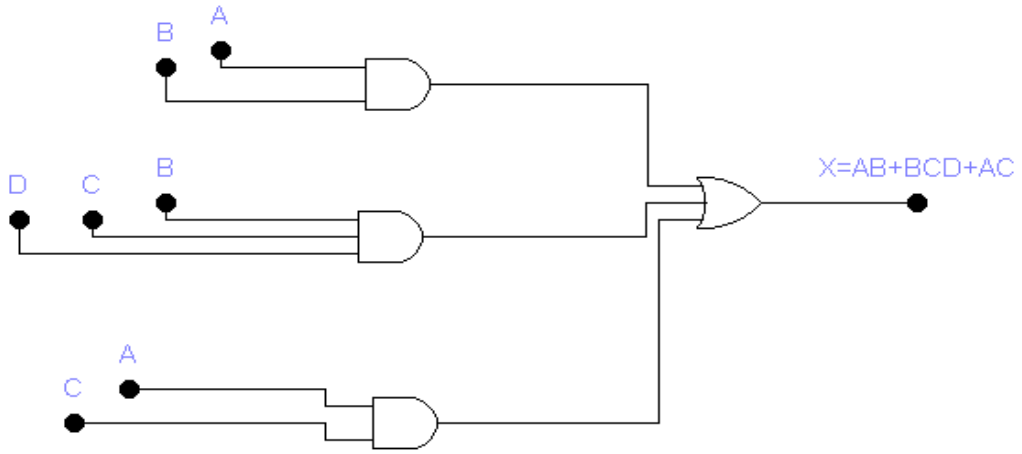
And / Or implementation of an SOP expression

Implementation an SOP expression is simply require

1- ORing the output of two or more AND gates. A product term is produced by an AND operation.
2- And the sum (addition) of two or more product terms is produced by an OR operation.

Note: Therefore, an **SOP** expression can be implemented by **AND-OR** logic in which the outputs of a number (equal to the number of product terms in the expression) of **AND** gates connect to the inputs of an **OR** gate, as show un the

next figure, for the expression **AB+BCD+AC**. the output **X** of the **OR** gate equals the **SOP** expression.



Conversion of a General Expression to SOP Form

Any logic expression can be changed into **SOP** form by applying Boolean algebra techniques. For example, the expression **A(B+CD)** can be converted to SOP form by applying the distributive law: **A (B +CD)= A +ACD**

Example: convert each of the following Boolean expression to the SOP from:

(a) AB + B(CD + EF)

Solution:

AB +BCD +BEF

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

$$Soluation:$$

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

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

$$=A\overline{C}+B\overline{C}$$

Standard SOP

So far, you have been SOP expressions in which some of the product terms do not contain all of variables in the domain of the expression. For example, the expression $\overline{A}B\overline{C}+A\overline{B}D+\overline{A}B\overline{C}D$ variables has a domain made up of the A ,B ,C and D. A ***Standard SOP expression*** is one in which all the variables in the domain appear in each product term in the expression.

For example , $\overline{A}B\overline{C}D+\overline{A}BCD+AB\overline{C}D$ is a standard SOP expression . Standard SOP expressions are important in constructing truth tables, and in the **Karnaugh map** simplification method, which is covered in next chapter.

Converting Product Terms to Standard SOP

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard from using Boolean algebra **Rule 6** $(A+\overline{A}=1)$ **: a variable added to its complement equals 1.**

Step1: Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms. As you know, you can multiply anything by 1 without changing its value.

Step2: Repeat step1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable as shows in following example:

Example: Convert the following expression into standard SOP form:

$$A\overline{B}C+\overline{A}\overline{B}+AB\overline{C}D$$

The first term $A\overline{B}C$, *is missing variable D or* $\overline{D}$ *,so multiply the first term by* $D + \overline{D}$ as follows :

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

In this case, two standard product terms are the result .

The second term $\overline{A}\overline{B}$ **,** *is missing variables C or* $\overline{C}$ *and D or* $\overline{D}$ **,** *so first multiply the second term by* $C+\overline{C}$ *as follows:*

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

The two resulting terms are missing variable D or $\overline{D}$ *, so multiply both terms by* $D+\overline{D}$ *as follows :*

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

In this case , four standard product term are result .

The third term, $\overline{A}B\overline{C}\overline{D}$ is already in standard form. The complete standard SOP form of the original expression is as follows:

$$A\overline{B}C + \overline{A}\overline{B} + AB\overline{C}D =$$

$$A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D}$$

Binary Representation of a Standard Product Term

A standard product term is equal to 1 for only one combination of variable values.

Example: $A\overline{B}C\overline{D}$ is equal to 1 when A=1, B=0, C=1 and D=0 as shown below:

$A\overline{B}C\overline{D}= 1.\overline{0}.1.\overline{0} =1.1.1.1=1$

In this case, the product term has a binary value of 1010 (decimal ten).

A product term is implement with an AND gate whose output is 1 only if each of its inputs is 1. inverters are used to produce the complements of the variables as required.

Example: determine the binary value for which the following Standard SOP expression is equal 1: $ABCD+A\overline{B}\overline{C}D+\overline{A}\overline{B}C\overline{D}$

Solution:

The first term *ABCD is equal to 1 when A=1, B=1, C=1 and D=1, so*

A.B.C. D=1.1.1.1=1

The second term $A\overline{B}\overline{C}D$ is equal to 1 when A=1 , B=0 ,C=0 and D=1 , so

$$A\overline{B}\overline{C}D=1\,.\,\overline{0}\,.\overline{0}.1=1.\,1\,.\,1\,.\,\,1=1$$

The third term $\overline{A}\overline{B}\overline{C}\overline{D}$ is equal to 1 when A=0 , B=0 ,C=0 and D=0 , so

$$\overline{A}\overline{B}\overline{C}\overline{D}=\overline{0}\,.\,\overline{0}\,.\overline{0}\,.\overline{0}=1.\,1\,.\,1\,.\,\,1=1$$

Note. The Sop expression equals 1 when any or all of the three product terms is 1.

2– **Product of Sum:** When two or more sum terms are multiplied, the resulting expression is a product-of-sums (POS).

Examples

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

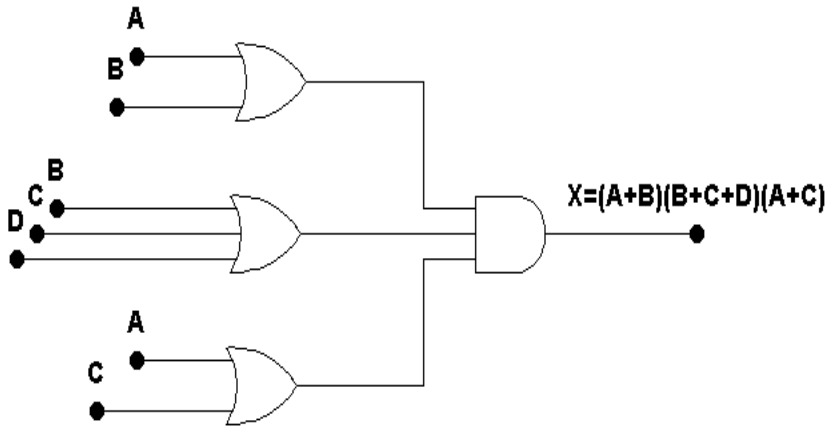
$$(\overline{A}+\overline{B}+\overline{C})(C+\overline{D}+E)(\overline{B}+C+D)$$

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

The Product of Sum expression is equivalent to the logical OR-AND function which gives the AND Product of two or more OR Sums to produce an output.

Implementation of a POS expression

Implementing a POS expression simply require ANDing the outputs of two or more OR gates. The next figure shows the expression (A+B)(B+C+D)(A+C). the output X of the AND gate equals the POS expression.



The Standard POS Form

So far, you have seen POS expression in which some of the sum terms do not contain all of the variables in the domain of the expression. For example , the expression $(A+\overline{B}+C)(A+B+\overline{D})(A+\overline{B}+\overline{C}+D)$, has the domain made up of the variables ,A ,B ,C and D .

Notes that the complete set of variables in the domain is not represented in the first two terms of the expression ; that is , D or $\overline{D}$ is missing from the first term and C or $\overline{C}$ is missing from the second term.

A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression , for example ,

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

Is a standard POS expression , Any nonstandard POS expression (referred to simply as POS) can be converted to the standard form using Boolean algebra .

Converting a Sum Term to Standard POS

Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard POS expression is converted into standard form using Boolean algebra **Rule8** $(A.\overline{A}=0)$ **:A variable multiplied by its complement equal 0.**

Step1: Add to each nonstandard product term a term made up of the product of ,the missing variable and its complement. This results in two sum terms. As you know ,you can add 0 to anything without changing its value.

Step2: Apply **rule12 : A +BC =(A+B)(A+C)**
Step3: Repeat Step 1 until al resulting sum terms contain all variables in the domain in either complement or un complemented form.
***Example*: Convert the following Boolean expression into Standard** POS form.

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

Solution: *the domain of this POS expression is A,B,C,D. take one term at a time,*
The first term $A+\overline{B}+C$ *D or* $\overline{D}$ *, so add* $D\overline{D}$ *and apply* ***rule12*** *as follows:*

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

The Second term $\overline{B}+C+\overline{D}$ *is missing variable A or* $\overline{A}$ *,so add* $A\overline{A}$ *and apply rule 12 as follows:*

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

The third term , $A+\overline{B}+\overline{C}+D$ *, is already in standard form , so, the standard POS form of the original expression is as follows:*

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

Binary representation of SOP

A standard sum term is equal to 0 for only one combination of variable values. For example ,the sum term $A+\overline{B}+C+\overline{D}$ is 0 when A=0 , B=1 , C=0 , and D=1 , as shown below , and 1 for all for all other combinations of values for the variables ,

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

In this case ,the sum term has a binary value of 0101 (decimal 5) .

Remember , a sum term is implemented with an OR gate whose output is 0 only if each its inputs is 0. Inverters are used to produce the complements of the variables as required .

A POS expression is equal to 0 only if one or more of the sum terms in the expression is equal to 0.

Example: Determine the binary values of the variables for which the following standard POS expression is equal to 0:

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

The first term $(A+B+C+D)$ *is equal to o when A=o , B=o , C=o and D=o.*

$$A+B+C+D=0+0+0+0=0$$

The second term $A+\overline{B}+\overline{C}+D$ *is equal to o when A=o , B=1, C=1 and D=1*

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

The third term $\overline{A}+\overline{B}+\overline{C}+\overline{D}$ *is equal to o when A=1 ,B=1 , C=1 and D=1*

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

The POS expression equals o when any of the three terms eaual o.

Designing Combinational system.

Design 1-bit and 2-bits full adder design 1-bit subtractor

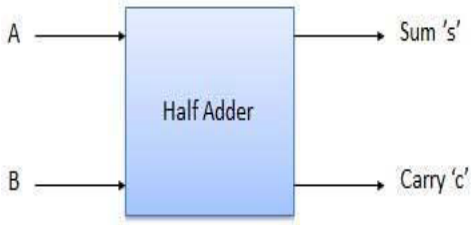
Basic Adders

Adders are important in computers and also in other types of digital systems in which numerical data are processed. An understanding of basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full adder are introduced.

The One-bit Full-Adder (FA) is used widely in systems with operations such as counter, addition, subtraction, multiplication and division etc. It is the basic core component of Arithmetic-Logic-Unit (ALU).

Half adder (HA)

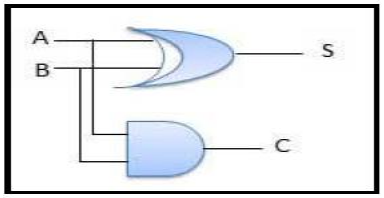
Half adder is a combinational logic circuit with two input and two output. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.



Half Adder Truth Table

Input		Outputs	
A	B	$\Sigma(s)$	C_o
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1
Binary digits to be added		Sum	Carry out
		XOR	AND

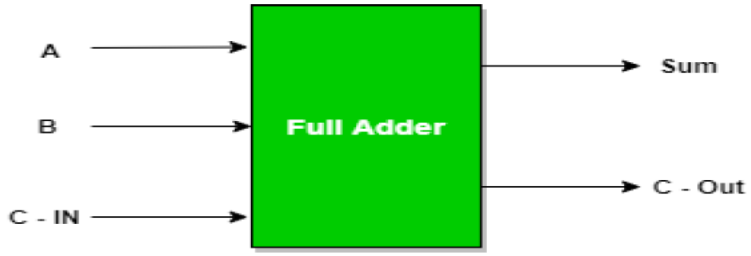
Half Adder Logic circuit



The Boolean expression *for the (C0) output* is **C0=A.B** which represent AND gate

The Boolean expression I s $\sum(s)=\overline{A}.B+A.\overline{B}$ *output* which represent XOR gate. We Can also simplify HA function using Karnough Map.

Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to the another.



Full Adder Truth Table:

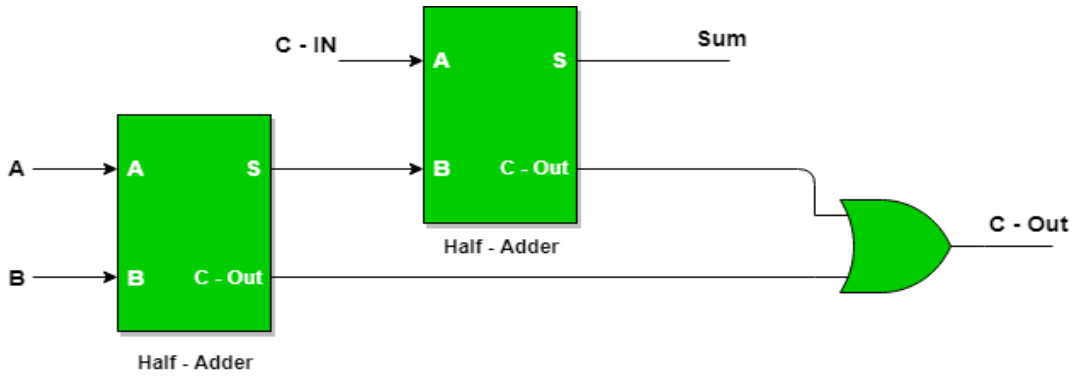
The full adder must be used when it possible to have an extra Carry input ,then the full adder has three inputs: **A , B , and** C_{in}. These three inputs must be added to get the $\sum(s)$ and C_o output .

Input			Outputs	
A	B	C_{in}	$\sum(s)$	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
A + B + C_{in}			Sum	Carry out

$\sum(s) = \sum(1,2,4,7)$

$C_o = \sum(3,5,6,7)$

Implementation of Full Adder using Half Adders 2 Half Adders and a OR gate is required to implement a Full Adder.



Subtractor

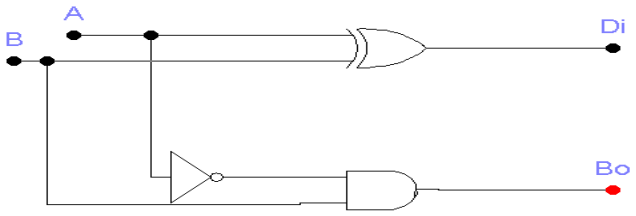
We can construct a one-bit subtractor circuit similar to the method used for constructing the full adder.

half subtractors

You will find that adders and suntractors are very similar. You use half subtractors and full subtractors just as you use half and full adders. Converting the rules to truth table from as in bellow.

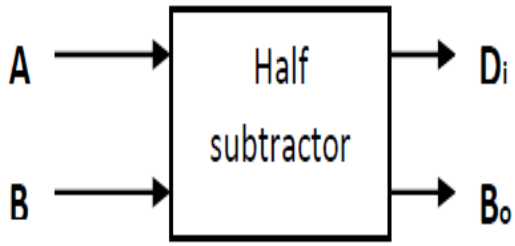
Inputs		Outputs	
A	B	D_i	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
A - B		Difference	Borrow out

On the input side,(B) is subtracted from(A) to give output Di(Diffrence). If B is larger then A , we need a borrow , which is shown in the column labeled Bo (borrow out).
* form the truth table, we can determine the Boolean expression for the half-subtractor.



The expression for the Di column is : $\mathbf{Di = A \oplus B}$, this is the same as for the half adder .

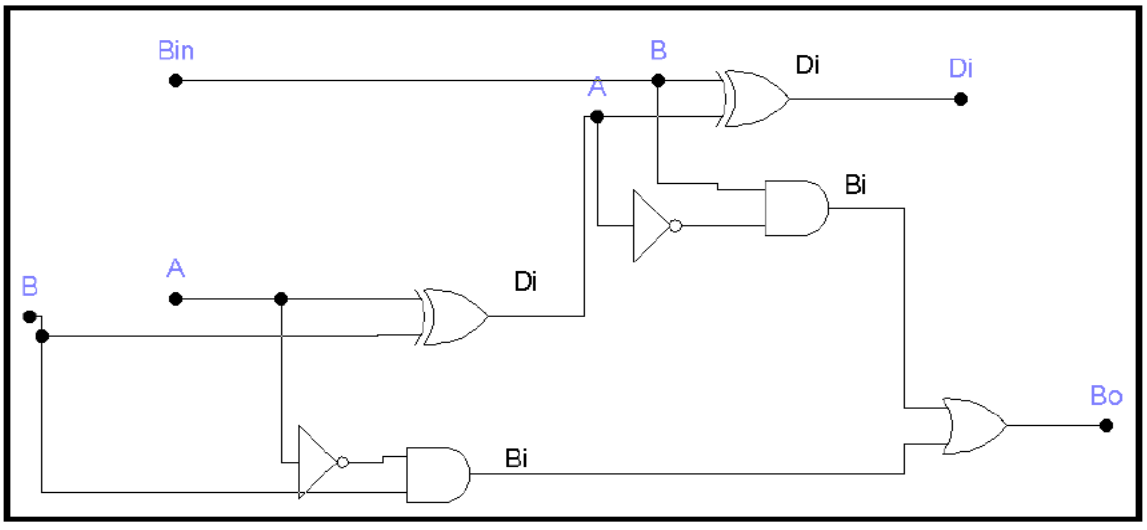
The Boolean expression for the B_o column is $B_o = \overline{A}, B$, combining these two expression in a logic diagram gives the logic circuit for a half subtractor.



A full subtractor is a **combinational circuit** that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit. This circuit **has three inputs and two outputs**. The three inputs A, B and Bin, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and Bout represent the difference and output borrow, respectively.

Truth table

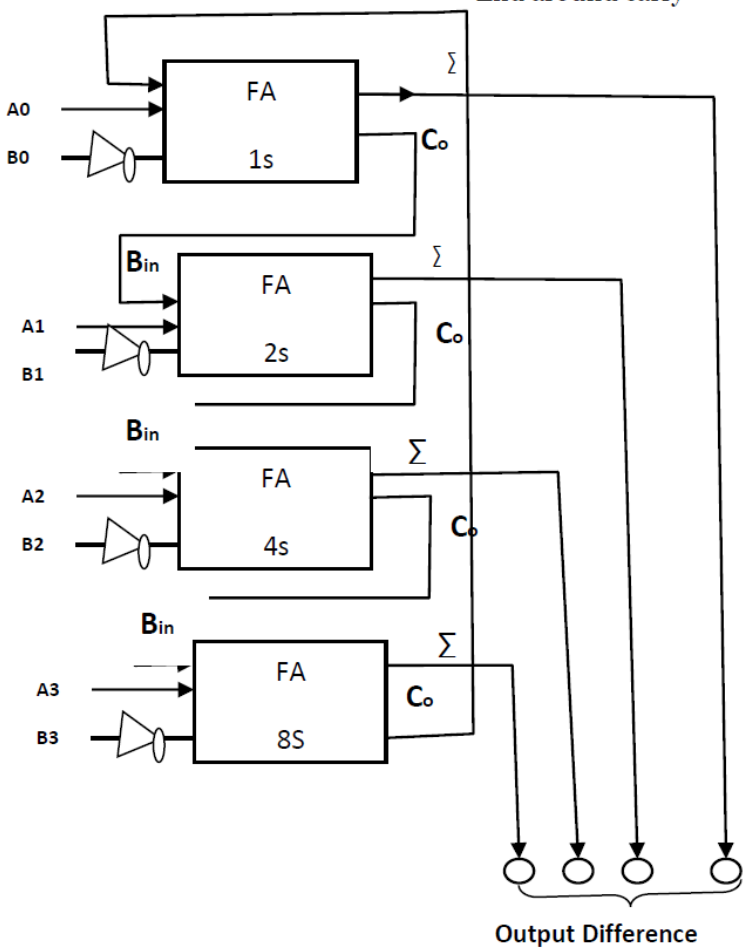
Input			Outputs	
A	B	Bin	Di	Bo
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1
A - B - Bin			Di	Bo



Using Adders For Subtractor

With A few a little thinks we can use as adder to also do subtraction. There is a mathematical technique that helps us use an adder to do binary subtraction.

Exp:	Decimal subtraction	Binary Subtraction		
	10	1010		1010
	- 6	- 0110	→ 1's complement and add	+1001
	4	100		1 0011
			End around carry	+ 1
				100



In this special technique the steps are first to first wire the 1's complement of the number being subtracted (change all 1's to 0's and all 0'sto 1's) and then add.

Now let us us adders to do binary subtraction in this example, the temporary answer to this addition is shown as (10011). Next, the last carry on the left I carried around to the 1's place. This is called an end-around carry. When the end around carry is added to the rest of the number, the result is the difference between the original binary numbers, (1010) and (0110). The an swer to this problem is 0100.

Comparators

The comparison of two numbers is an operations that determines if one number is greater then, less than, or equal to other number.

A Magnitude comparator: is a combinational circuit that compares two numbers A and B , and determines there relative . The outcome of the comparison is specified by three binary variables that indicate whether A > B , A=B or A <B .

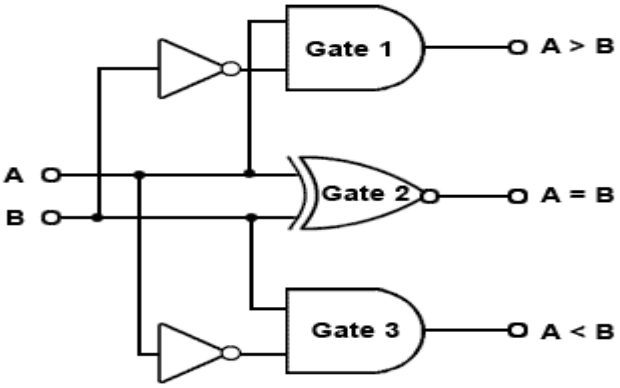
1-Bit Magnitude Comparator

A comparator used to compare two bits is called a single-bit comparator. It consists of two inputs each for two single-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 1-bit comparator is given below.

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

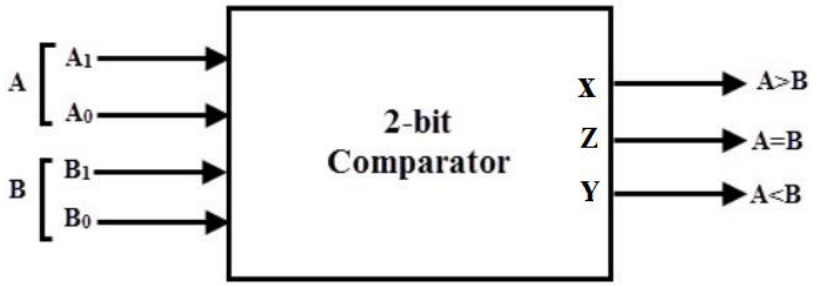
by using logic circuit



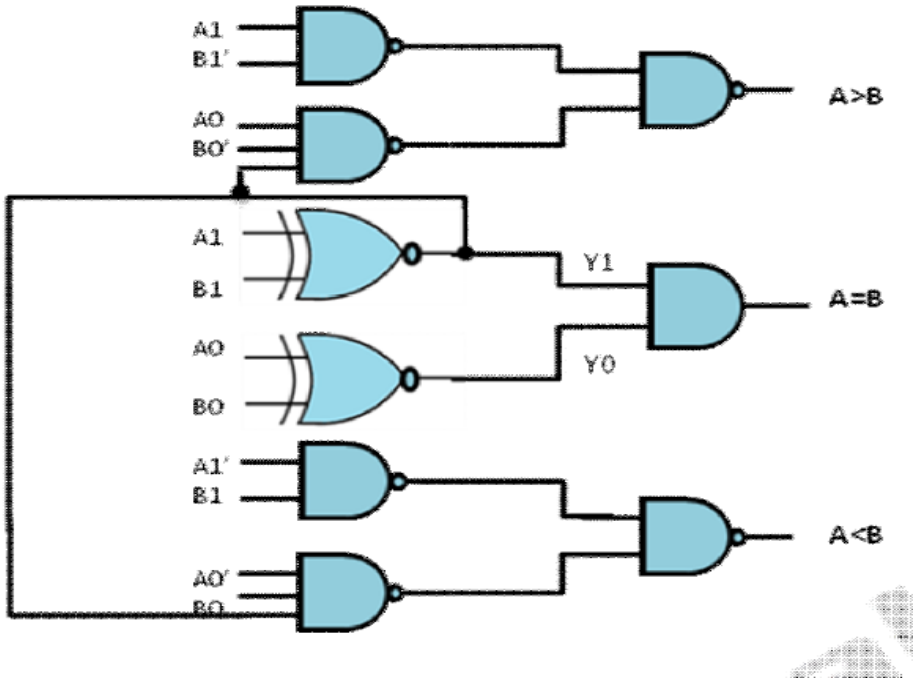
2-Bit Magnitude Comparator

A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.

The truth table for a 2-bit comparator is given below.



InPuts				OutPuts		
A1	A0	B1	B0	X (A>B)	Y (A<B)	Z (A=B)
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1



Equality Relation

We say that if the number A equal to B we are implement an XNOR gate to do this state, but with one bit number, then now about numbers with n- bits ?

Note: the two numbers are equal if all pairs of significant digit are equal:

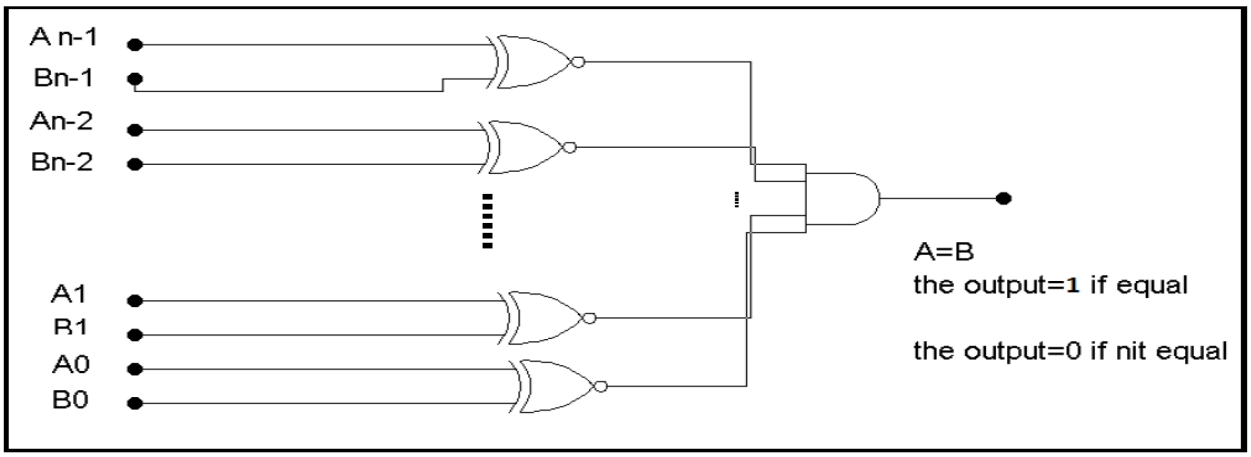
$$a_{n-1}=b_{n-1}\ ,\ \ldots\ldots\ldots\ldots\ldots\ ,\ a_0=b_0$$

$$\longrightarrow$$

That means for the equality condition being true (equal 1) if all equality relation of each pair must equal to 1, this dictates **AND** gate to combine the outputs to gather to get the final output 1.

Therefore Equality Relation:-

$$A=B\ \ if\ \ (\overline{a_{n-1}\oplus b_{n-1}}).(\overline{a_{n-2}\oplus b_{n-2}})\ \ldots\ldots\ldots\ldots\ldots\ ,(\overline{a\ \oplus b_0})=1$$



Greater than relation

If the corresponding digit of is (1) and that of B is (0) , we conclude that A >B.

Then How can we implement a comparator circuit to do this comparison:

1- if N= 2

$$Let\ \ A=a_1a_0$$

$$Let\ \ B=b_1b_0$$

$$to\ prof\ that\ \ A>B\ \ if\ \ \ a_1a_0>b_1b_0$$

$$=(a_1>b_1)+(a_1=b_1).(a_0>b_0)$$

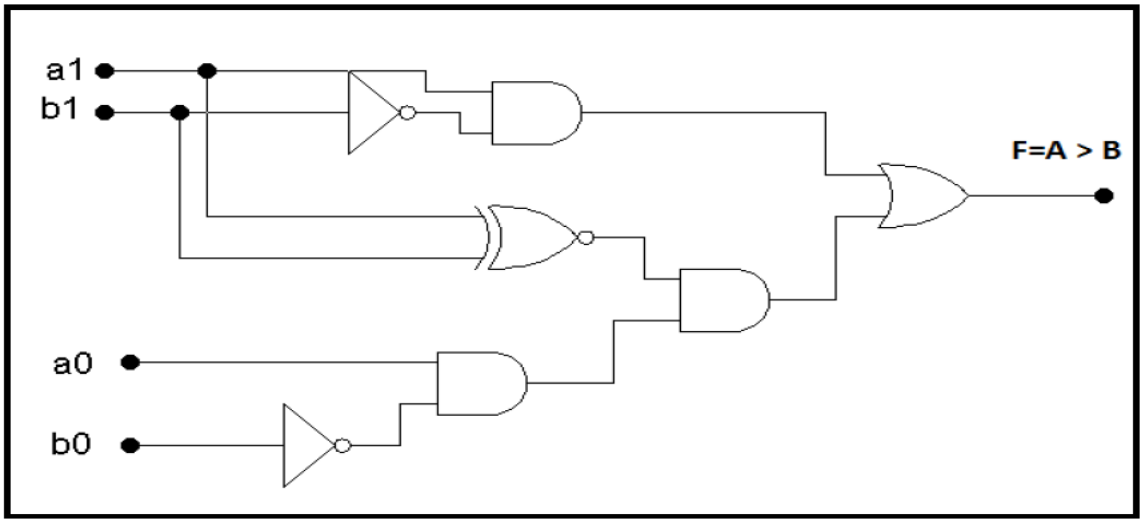
$$and\ \ from\ \ truth\ table:$$

$$(a_1>b_1)=a_1\overline{b_1}$$

$$(a_1=b_1)=(\overline{a_1\oplus b_1})$$

$$(a_0>b_0)=a_0\overline{b_0}$$

$$then\ \ F(A>B)=a_1\overline{b_1}+(\overline{a_1\oplus b_1}).a_0\overline{b_0}$$



2- if N= 3

$$Let \quad A = a_2 a_1 a_0$$

$$Let \quad B = b_2 b_1 b_0$$

then

$$F(A>B) = (a_2 > b_2) + (a_2 = b_2).(a_1 > b_1) + (a_2 = b_2).(a_1 = b_1).(a_0 > b_0)$$

$$\because (a_2 > b_2) = a_2 \overline{b_2}$$

$$(a_2 = b_2) = (\overline{a_2 \oplus b_2})$$

$$(a_1 > b_1) = a_1 \overline{b_1}$$

$$(a_1 = b_1) = (\overline{a_1 \oplus b_1})$$

$$(a_0 > b_0) = a_0 \overline{b_0}$$

$$\therefore A > B = (a_2 \overline{b_2}) + (\overline{a_2 \oplus b_2}).(a_1 \overline{b_1}) + (\overline{a_2 \oplus b_2}).(\overline{a_1 \oplus b_1}).(a_0 \overline{b_0})$$

(A) *Decoder:*

A decoder is a combinational circuit that converts coded information, such as binary, into a recognizable form, such as decimal.

In its general form : a decoder has **n** input lines to handle n bits and from one to $\mathbf{2^n}$ output lines to indicate the presence of one or more n-bit combinations.

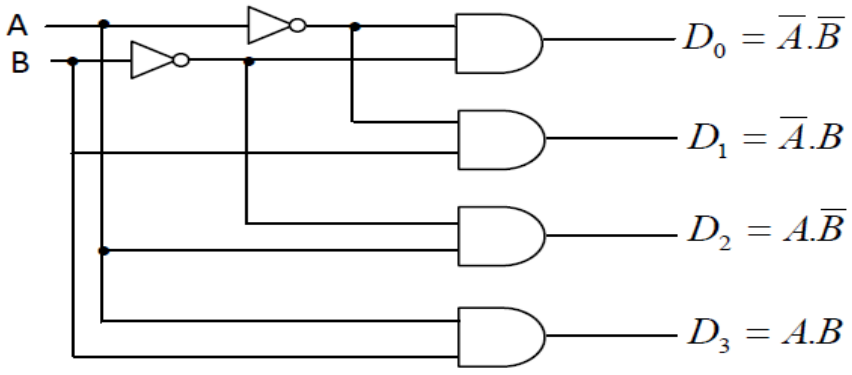
Block diagram

2-to-4 line decoder

The block diagram of 2 to 4 line decoder is shown in the figure A and B are the two inputs where D_0 through D_3 are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

We can design 2 to 4 line decoder by using the following truth table:

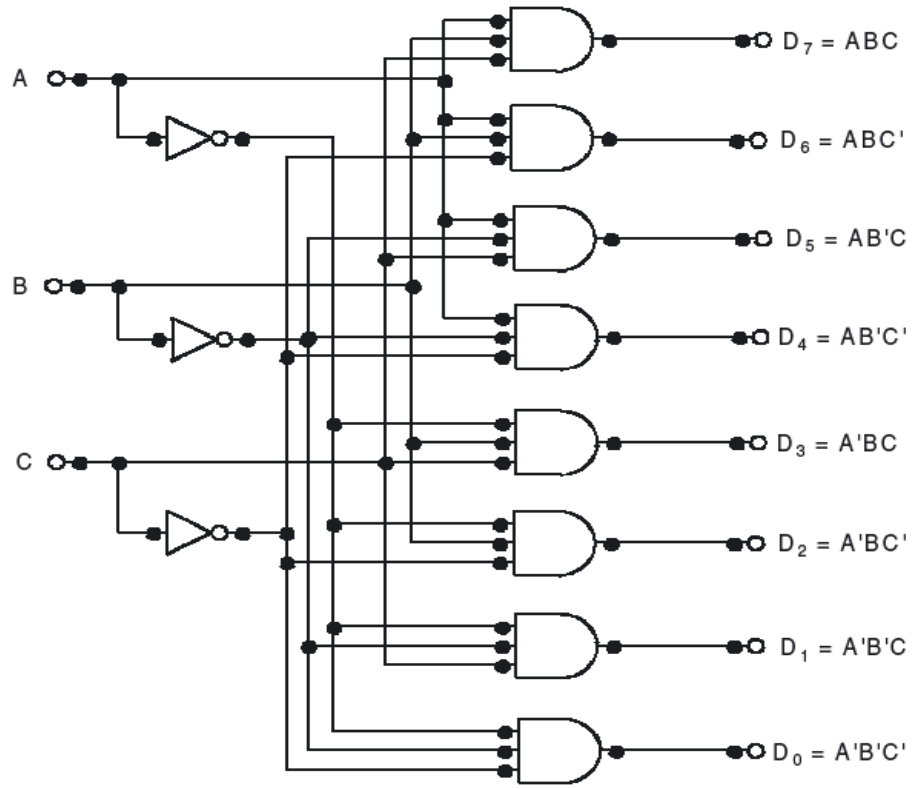
INPUTS		OUTPUTS			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



A 3-to-8 line decoder

Input variables			*Outputs*							
A	*B*	*C*	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

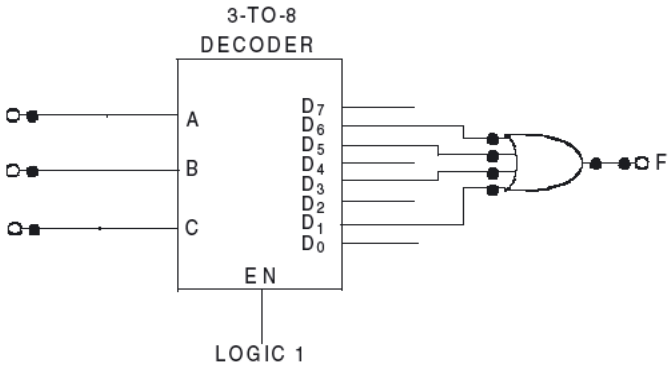
The 3-to-8 line decoder consists of three input variables and eight output lines. Note that each of the output lines represents one of the minterms generated from three variables. The internal combinational circuit is realized with the help of INVERTER gates and AND gates. The operation of the decoder circuit may be further illustrated from the input output relationship as given in the above table. Note that the output variables are mutually exclusive to each other, as only one output is possible to be logic 1 at any one time.



Application of decoder

Example 1. Implement the function F (A,B,C) = Σ (1,3,5,6).

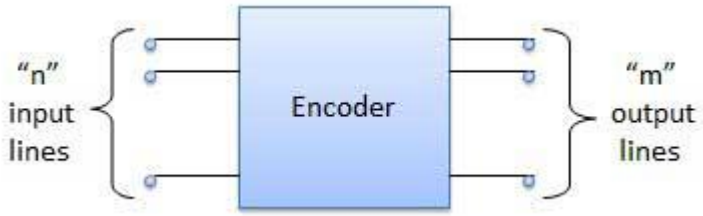
Solution. Since the above function has three input variables, a 3-to-8 line decoder may be employed. It is in the sum of the products of the minterms *m*1, *m*3, *m*5, and *m*6, and so decoder output D1, D3, D5, and D6 may be OR-gated to achieve the desired function. The combinational circuit of the above functions is shown in the following Figure .



Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. The encoder is also a combinational logic circuit; it converts information, such as a decimal number or an alphabetic character, into some coded form such as binary or BCD.

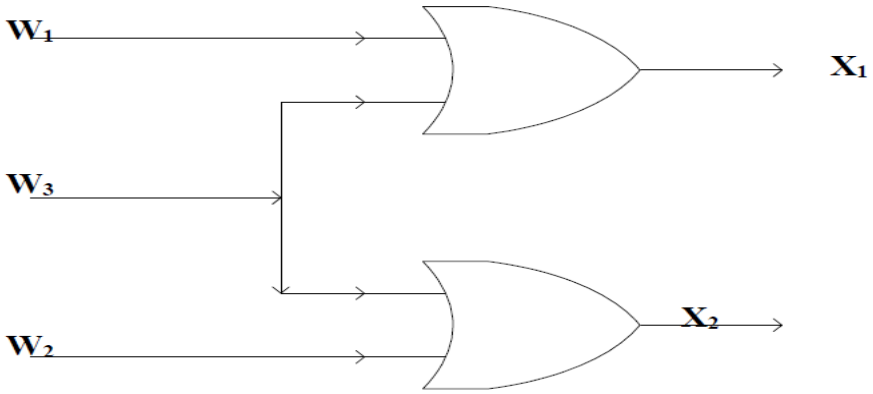
Block diagram



- **four to two line encoder and its truth table.**

INPUTS				OUTPUTS	
W_3	W_2	W_1	W_0	X_2	X_1
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Truth Table of Four to Two Line Encoder.



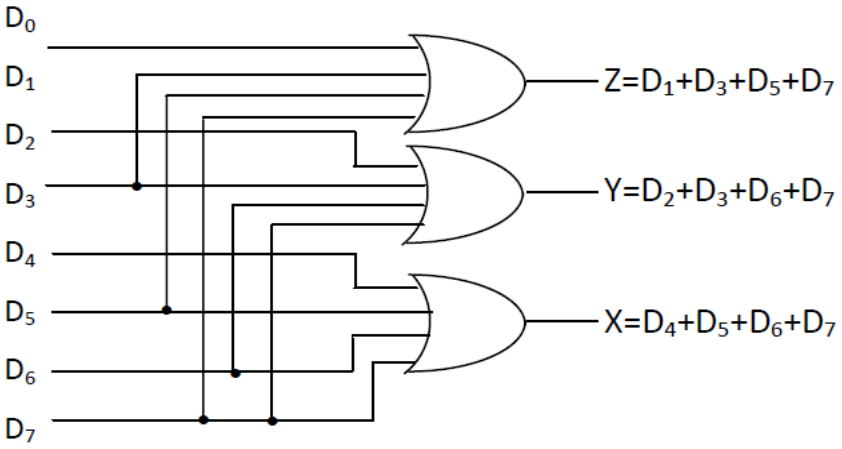
Four to Two line Encoder

Eight to binary encoder

The octal-to-binary encoder consists of eight inputs, one for each of the eight digits, and three outputs that generate the corresponding binary number. It is constructed with OR gates whose inputs can be determined from the truth table. The lower-order output bit Z is 1 if the input octal digit is odd. Output X is 1 for octal digits 4, 5, 6 or 7. Note that D_0 is not connected to any OR gate, the binary inputs are all 0's.

INPUTS								OUTPUTS		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Truth table of an Octal-to-binary encoder



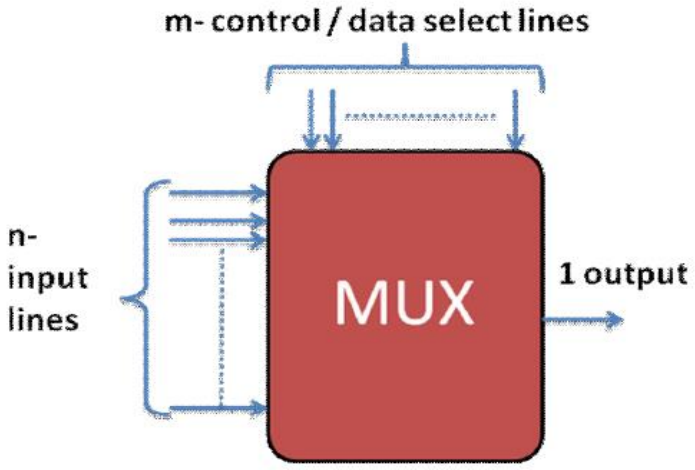
Logic diagram of Octal-to-binary encoder

Dr. suhad muhajer kareem

Multiplexer and Demultiplexer

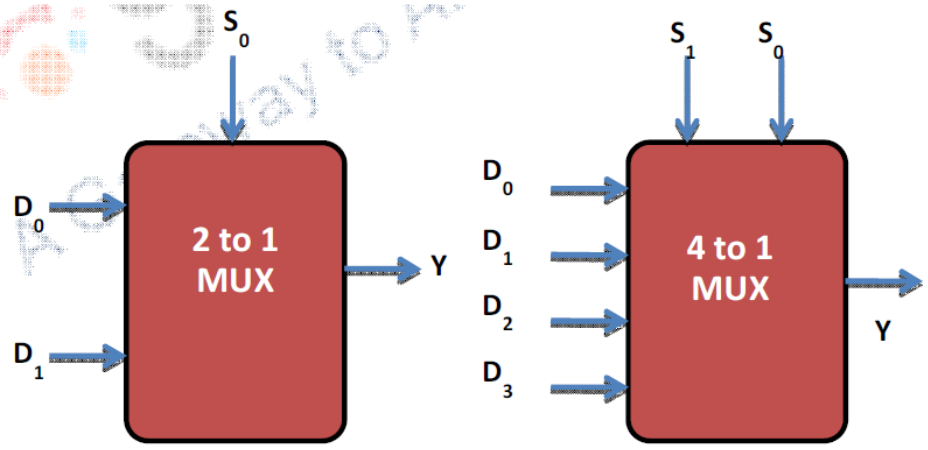
1. Multiplexers

- A Multiplexers (**MUX**) is a combinational logic component that has several inputs and only one output.
- **MUX** directs one of the inputs to its output line by using a control bit word (***selection line***) to its select lines.
- Multiplexer contains the followings:
  - 2^n data inputs
  - n selection inputs
  - a ***single*** output
  - Selection input determines the input that should be connected to the output.
- The multiplexer sometime is called data selector.
- The multiplexer acts like an electronic switch that selects one from different.
- A multiplexer may have an enable input to control the operation of the unit.

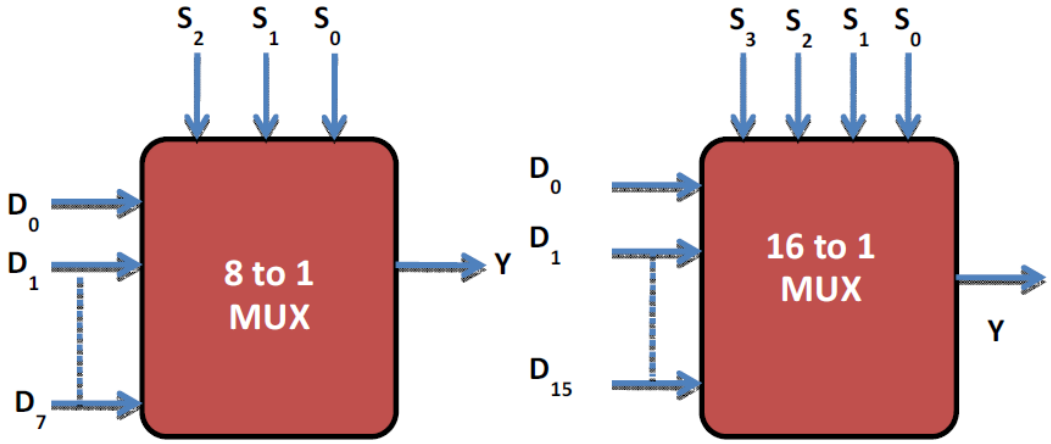


Logic symbol of Multiplexer

Types of multiplexer



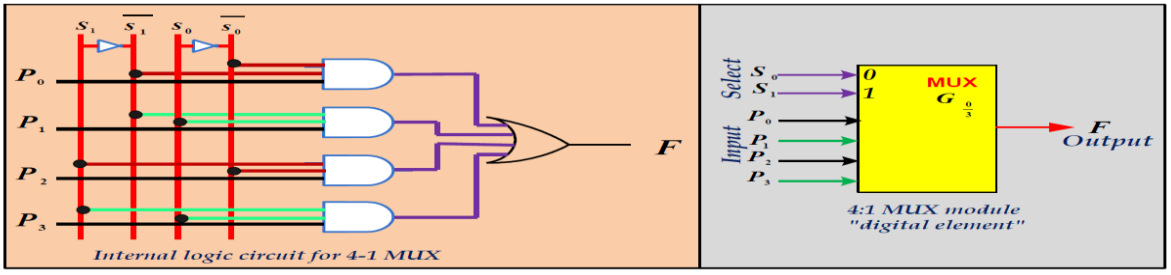
Logic symbols of 2 to 1 and 4 to 1 multiplexers



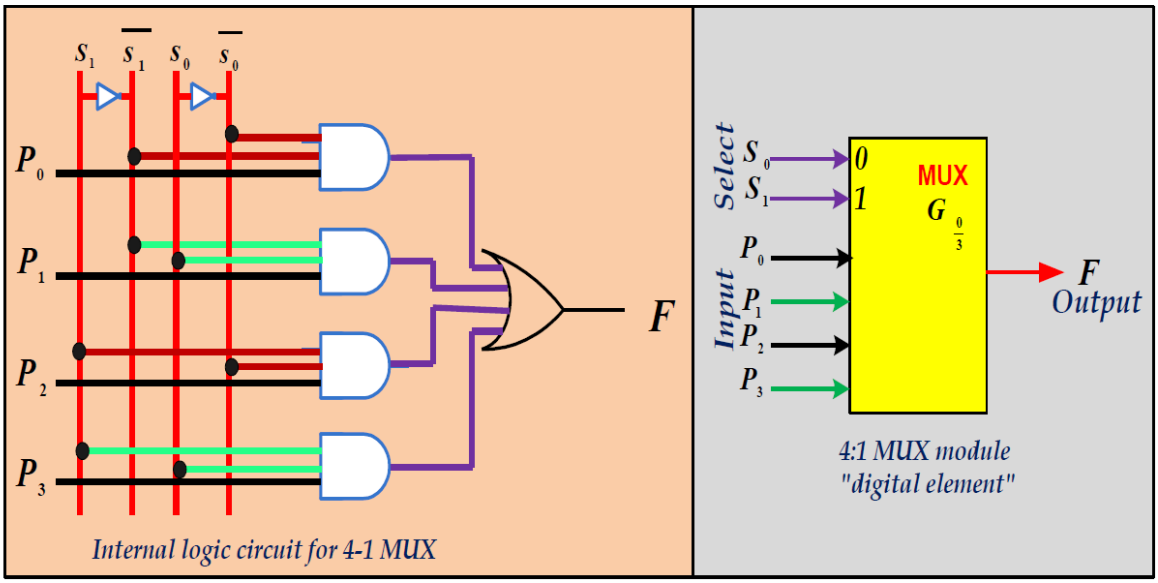
Logic symbols of 8 to 1 and 16 to 1 multiplexers

a) 4-to-1 Multiplexers

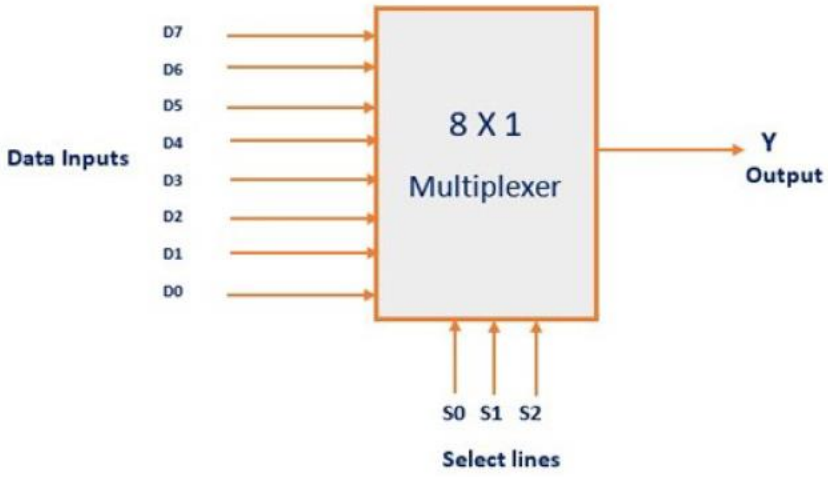
- ✓ 4-data input MUX
- ✓ s_1, s_0 - Select lines.
- ✓ p_0, p_2, p_3, p_1 - Input lines.
- ✓ F- Single output line.



$$F = \overline{S_1}\,\overline{S_0}P_0 + \overline{S_1}S_0P_1 + S_1\overline{S_0}P_2 + S_1S_2P_3$$



b. 8 t0 1 Multiplexers

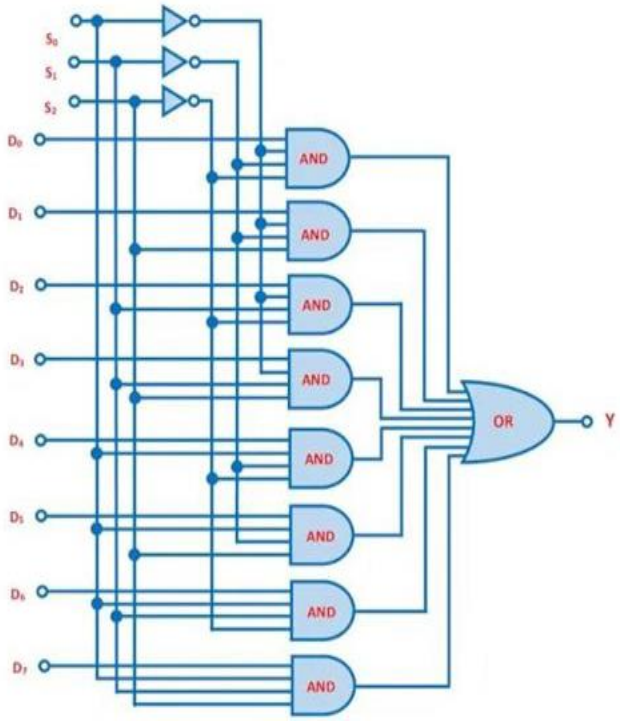


Block diagram of 8 to 1 Multiplexer

Multiplexer is a combinational circuit that has most of 2^n data inputs, 'n' selection lines with a single output. One of these data inputs will be connected to the output Y based on the values of selection lines. 8 X 1 Multiplexer has 8 data inputs D0, D1, D2, D3, D4, D5, D6 & D7, 3 select lines S0, S1, & S2 and one output Y.

INPUTS			OUTPUTS
S0	***S1***	***S2***	***Y***
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

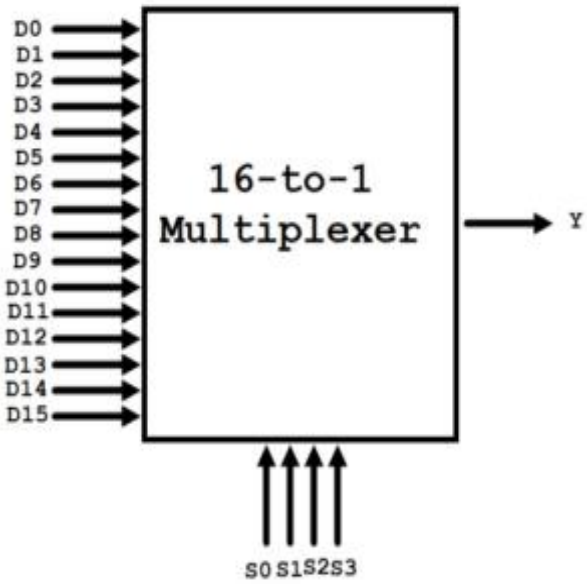
Truth Table of 8 to 1 MUX



Logical Circuit Diagram of 8X1 MUX

c. 16 to 1 multiplexers

The 16 to 1 multiplexer has 16 inputs and 4 control signals.



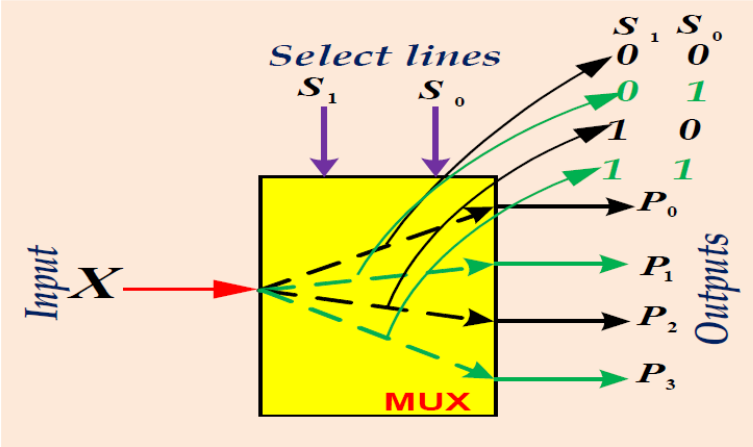
16-to-1 Multiplexer

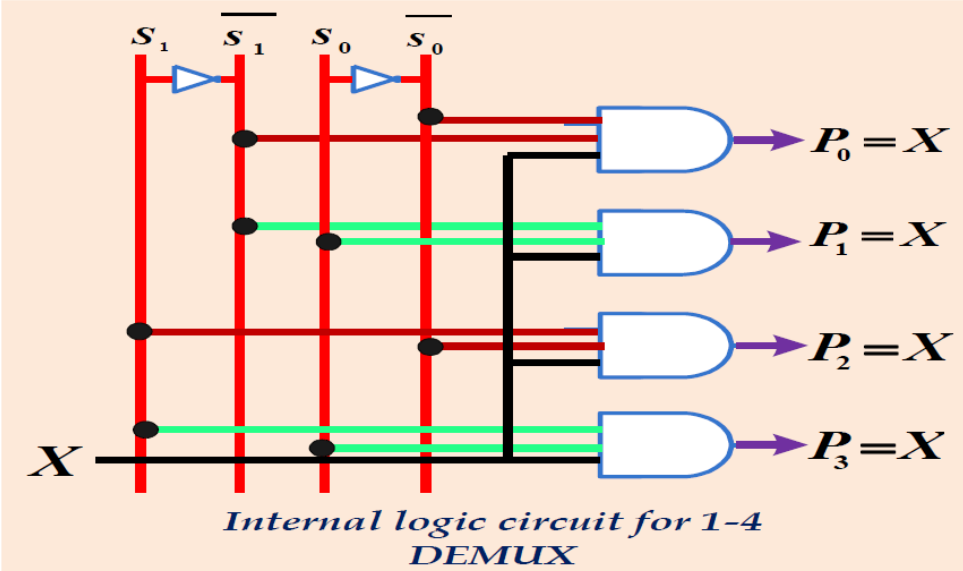
- Logic circuit
- Truth Table

A	B	C	D	Y
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7
1	0	0	0	D8
1	0	0	1	D9
1	0	1	0	D10
1	0	1	1	D11
1	1	0	0	D12
1	1	0	1	D13
1	1	1	0	D14
1	1	1	1	D15

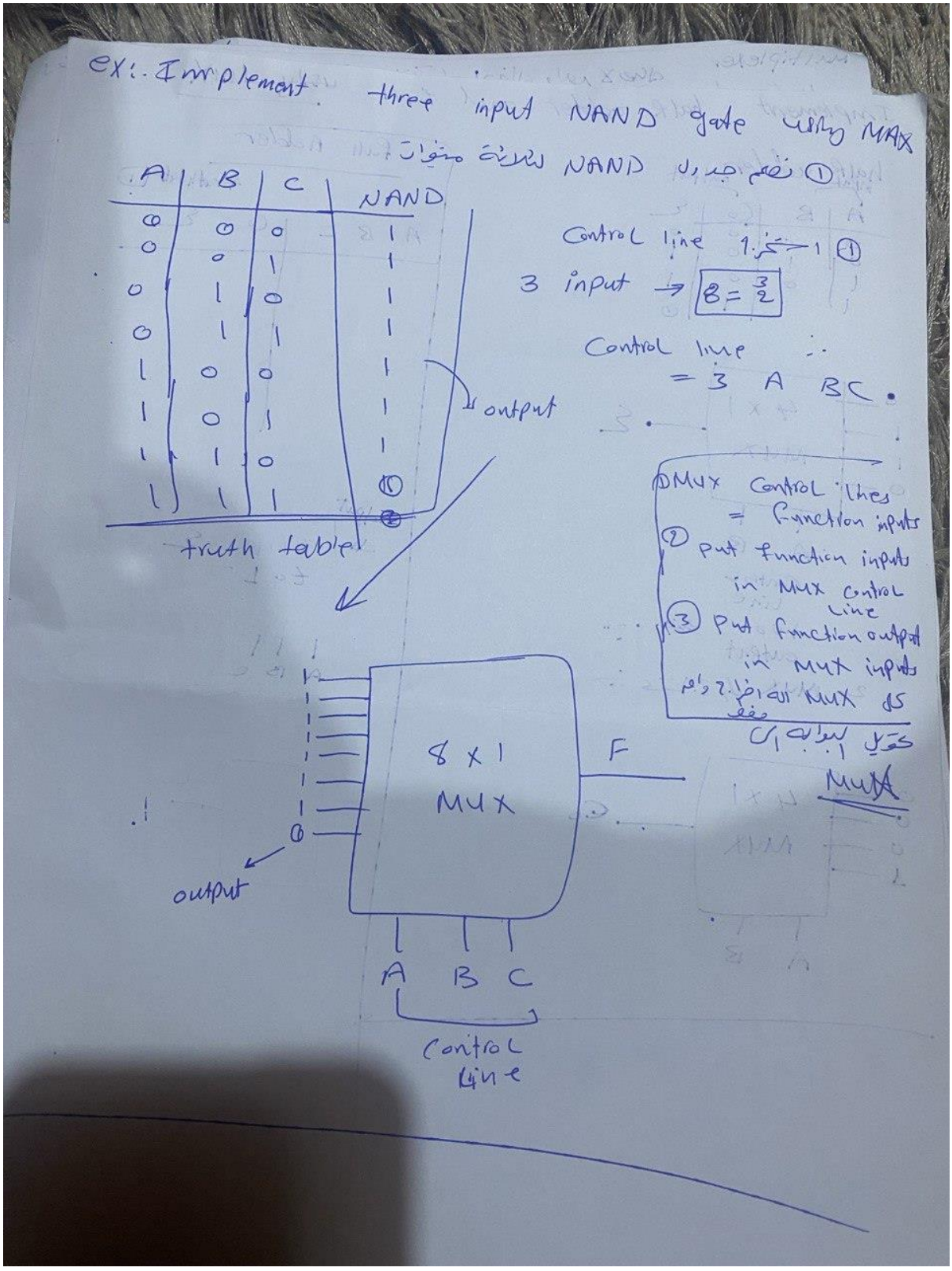
2. Demultiplexers

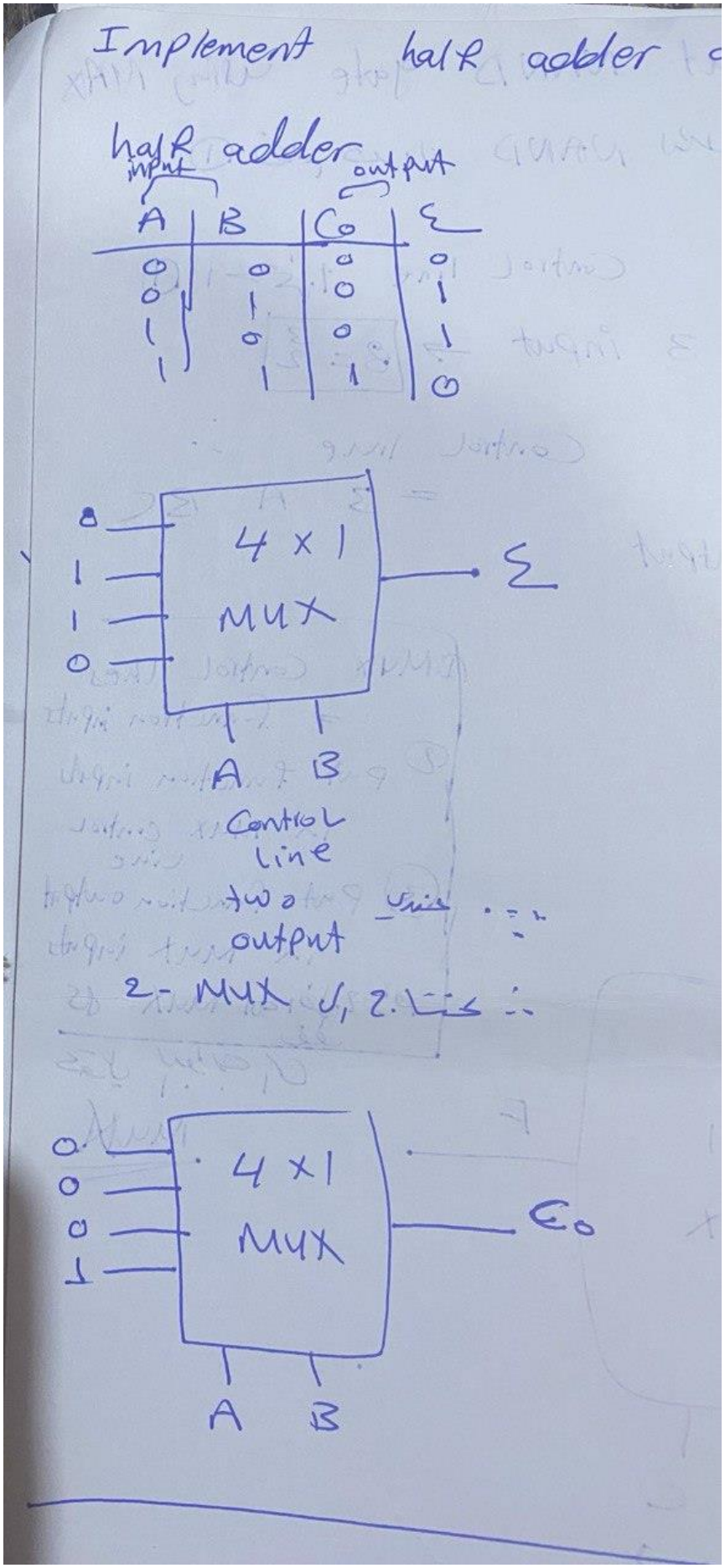
- The demultiplexer is a combinational logic circuit that performs the reverse operation of multiplexer (Several output lines, one input line).

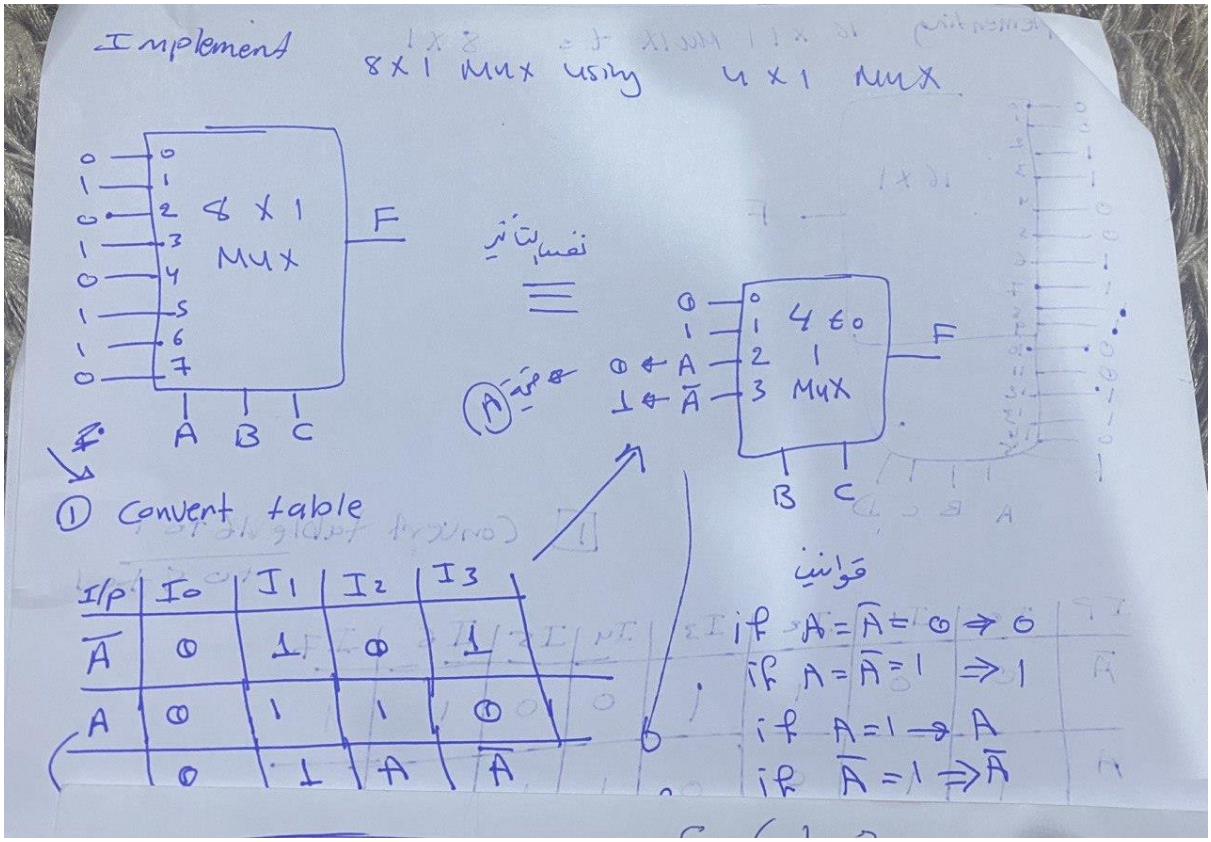


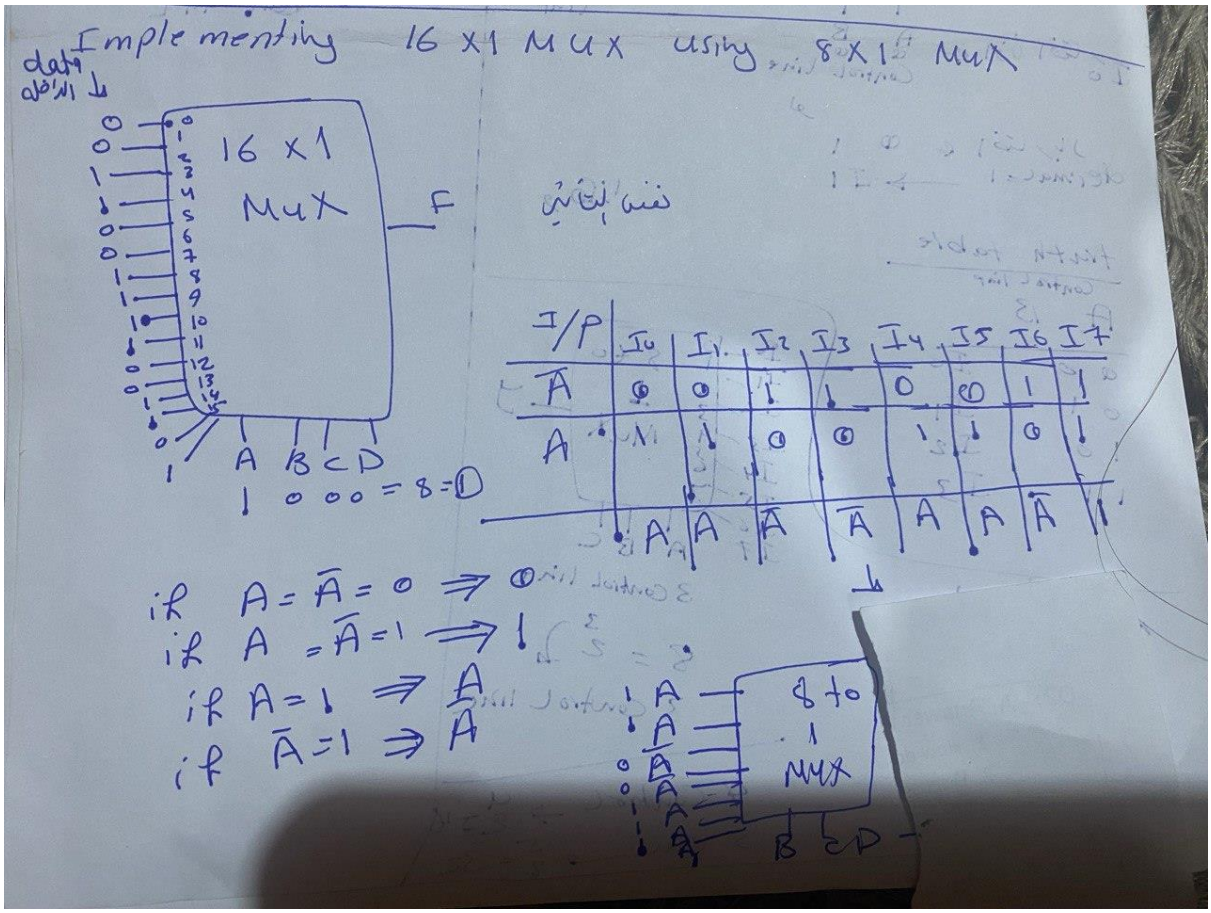


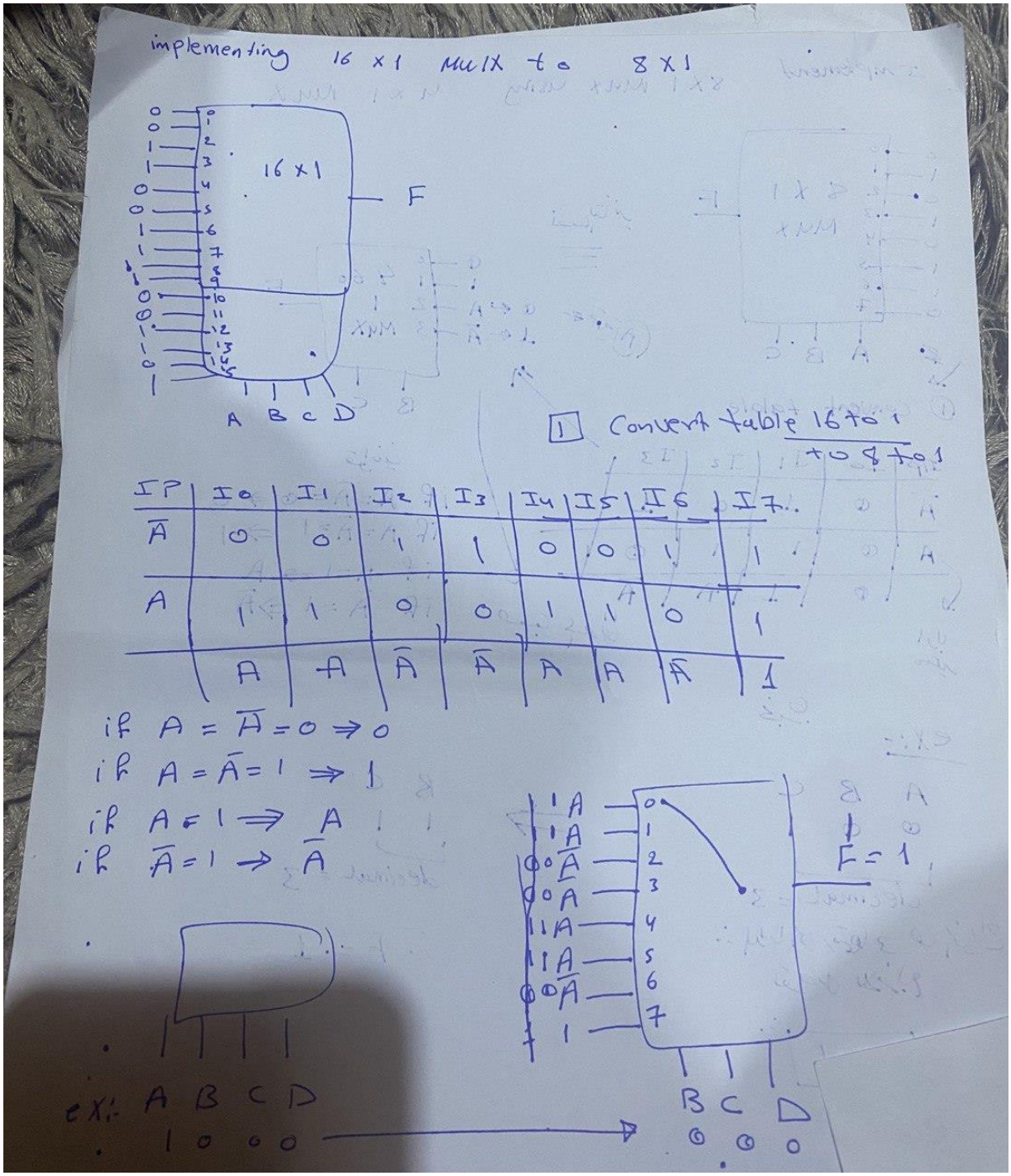
Internal logic circuit for 1-4 DEMUX


Ex:- Implement three input NAND gate using MUX
A B C NAND
truth table
output
Control line
3 input → B = 3/2
Control line = 3 A B C
MUX Control lines = Function inputs
Put function inputs in MUX control line
Put function output in MUX inputs
8 x 1 MUX
F
output
A B C
Control Line


Implement half adder
half adder
input
output
A B Co S
0 0 0 0
0 1 0 1
1 0 0 1
1 1 1 0
4 x 1
MUX
S
A B
Control line
two output
2- MUX
4 x 1
MUX
Co
A B


Implement 8x1 MUX using 4x1 MUX
8 x 1 MUX
F
A B C
نفس الشيء
4 to 1 MUX
① Convert table
I/P | I0 | I1 | I2 | I3
A | 0 | 1 | 0 | 1
A | 0 | 1 | 1 | 0
0 | 1 | A | Ā
قوانين
if A = Ā = 0 ⇒ 0
if A = Ā = 1 ⇒ 1
if A = 1 ⇒ A
if Ā = 1 ⇒ Ā


Implementing 16 x 1 MUX using 8x1 MUX
data الداخل
16 x 1 MUX
F
A B C D
1 0 0 0 = 8 = ①
نفس الشيء
I/P | I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7
A | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1
A | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1
A A Ā Ā A A Ā 1
if A = Ā = 0 ⇒ 0
if A = Ā = 1 ⇒ 1
if A = 1 ⇒ A
if Ā = 1 ⇒ Ā
8 to 1 MUX
B C D


implementing 16 x 1 MUX to 8 X 1
16 x 1
F
A B C D
1 Convert table 16 to 1
to 8 to 1
IP | I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7
Ā | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1
A | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1
A | A | Ā | Ā | A | A | Ā | 1
if A = Ā = 0 ⇒ 0
if A = Ā = 1 ⇒ 1
if A = 1 ⇒ A
if Ā = 1 ⇒ Ā
F = 1
ex: A B C D
1 0 0 0
B C D
0 0 0