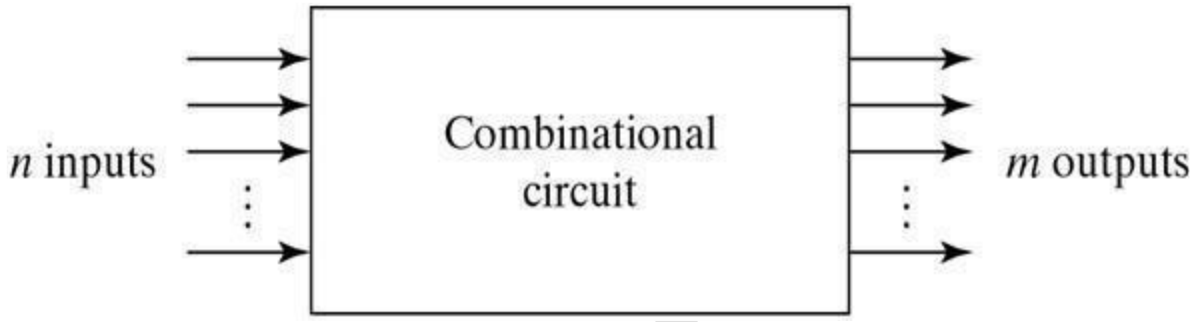
COMBINATIONAL CIRCUITS

Combinational Logic

• Logic circuits for digital systems may be combinational or sequential.

• A combinational circuit consists of input variables, logic gates, and output variables.



For n input variables, there are 2n possible combinations of binary input variables. For each possible input Combination, there is one and only one possible output combination. A combinational circuit can be described by m Boolean functions one for each output variables.

Analysis procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

A. Adders:

In electronics, an adder or summer is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as Binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where twos complement or one's complement is being used to represent negative numbers; it is trivial to modify an adder into an adder-subtractor. Other signed number representations require a more complex adder.

Digital computers perform variety of information processing tasks, the one is arithmetic operations. And the most basic arithmetic operation is the addition of two binary digits. i.e, 4 basic possible operations are:

0 + 0 = 0, **0 + 1 = 1,** **1 + 0 =1,** **1 + 1= 10**

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. A combinational circuit that performs the addition of two bits is called a half- adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder & two half adder can employ as a full-adder.

1. The Half Adder:

A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

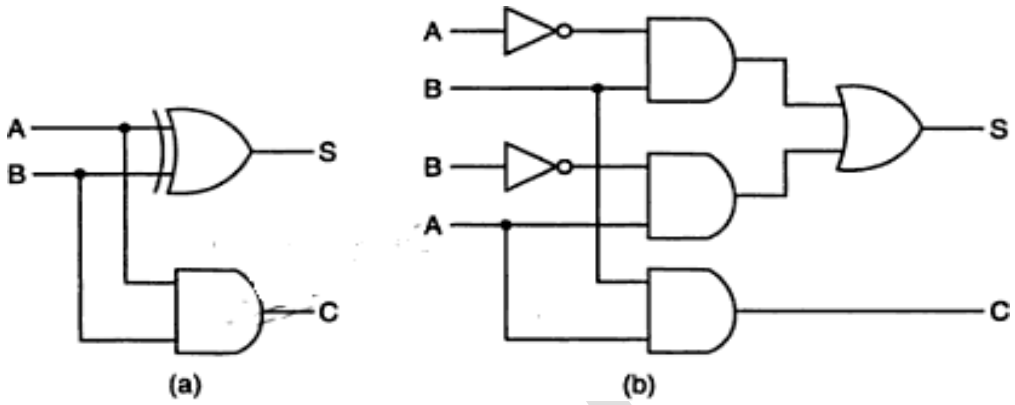
(a) Truth table

(b) Block diagram

The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B. Therefore,

S = A⊕B = A´B + AB´

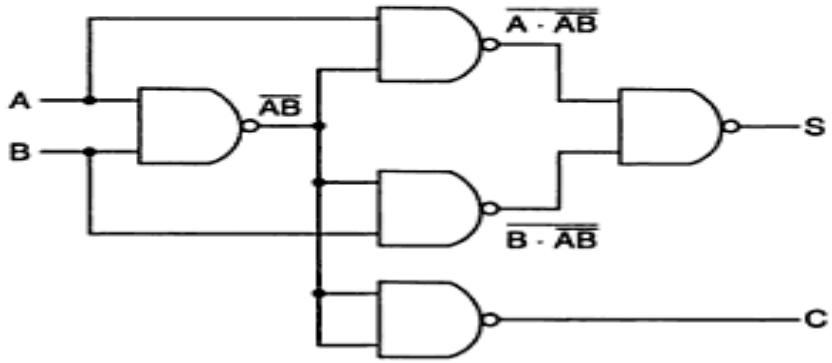
The carry (C) is the AND of A and B. Therefore, **C=AB**



Logic diagrams of half-adder

NAND LOGIC:

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



Logic diagram of a half-adder using only 2-input NAND gates.

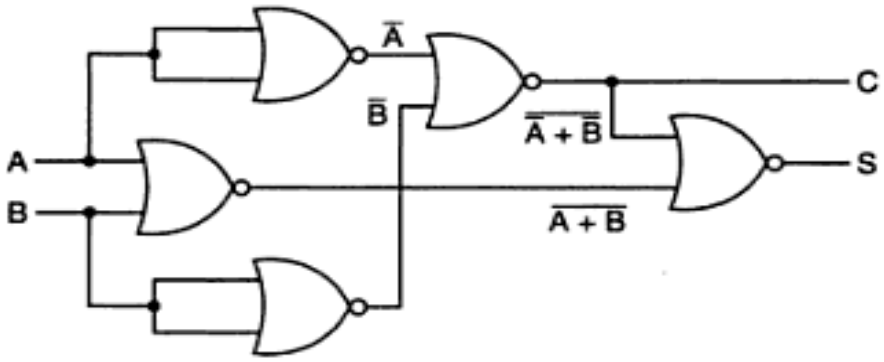
NOR Logic:

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

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

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

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



Logic diagram of a half-adder using only 2-input NOR gates.

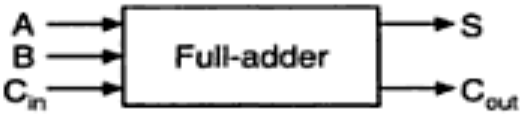
2. The Full Adder:

A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in (Cin) and outputs the sum bit (S) and the carry bit called the carry-out (Cout). The variable S gives the value of the least significant bit of the sum. The variable Cout gives the output carry. The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits.
When all the bits are 0s, the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The Cout has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
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) Truth table



(b) Block diagram

Full-adder.

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A, B and C_{in} is described by

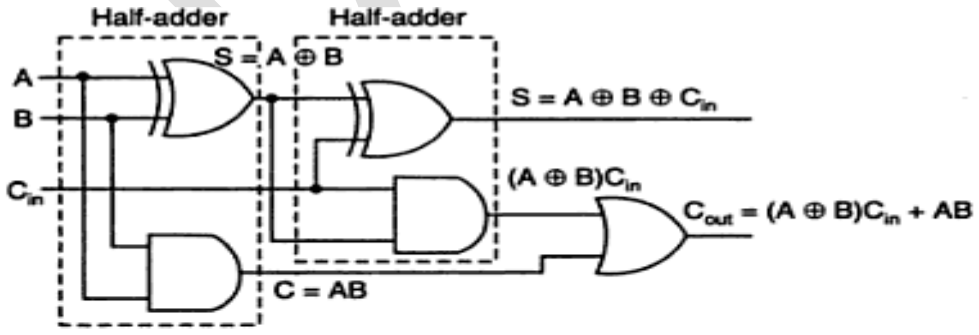
S = A´B´Cin + A´BC´in + AB´C´in + ABCin

Cout = A´BCin + AB´Cin + ABC´in + ABCin

$$S = A \oplus B \oplus C_{in}$$

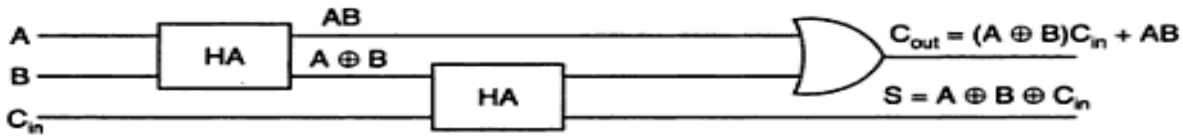
$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A, B and Cin, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



Block diagram of a full-adder using two half-adders.

The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as -

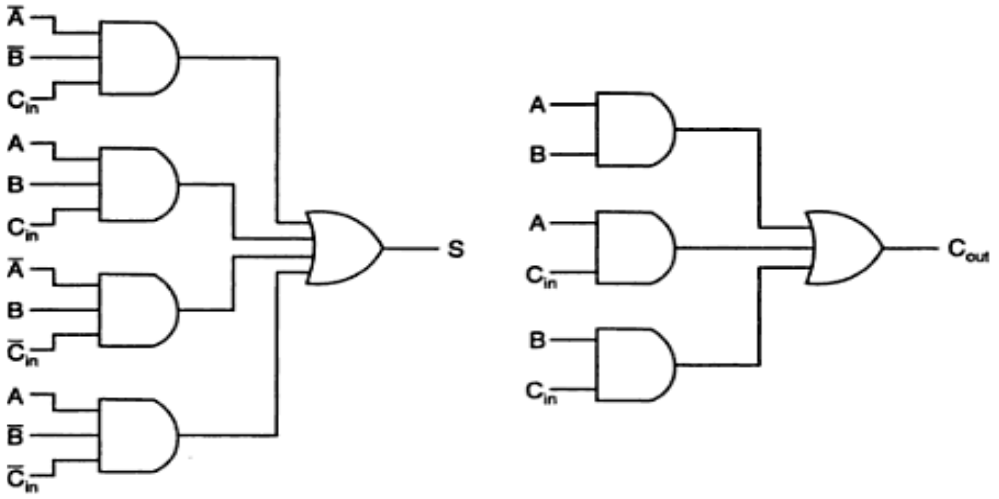
NAND Logic

$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

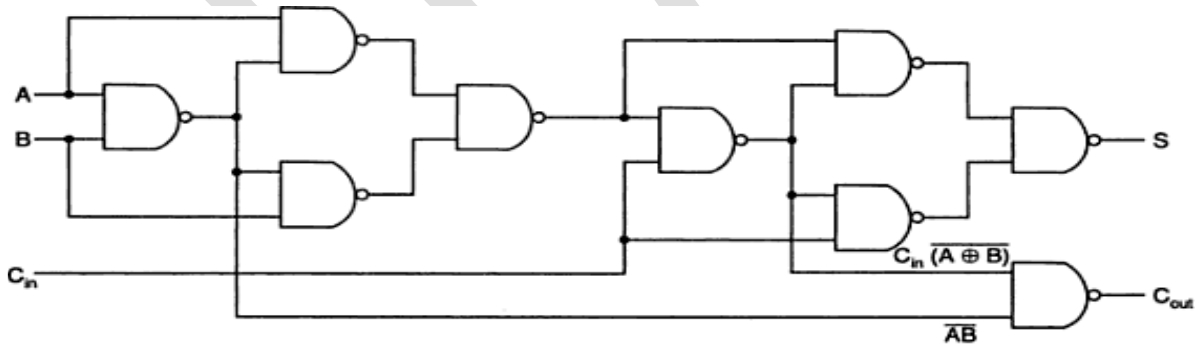
Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)C_{in}}}}$$

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$



Sum and carry bits of a full-adder using AOI logic.



Logic diagram of a full-adder using only 2-input NAND gates.

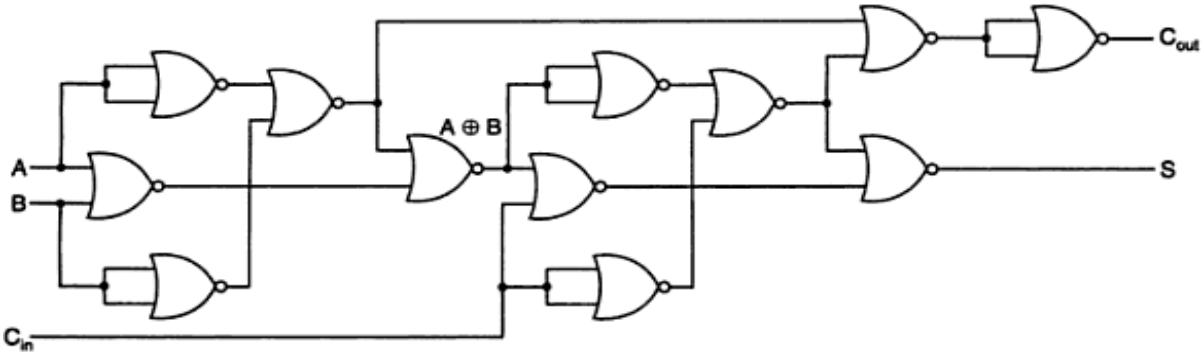
NOR Logic:

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

Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) + C_{in}} + \overline{(A \oplus B) + \overline{C}_{in}}}$$

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{A} + \overline{B}} + \overline{\overline{C}_{in} + \overline{A \oplus B}}$$



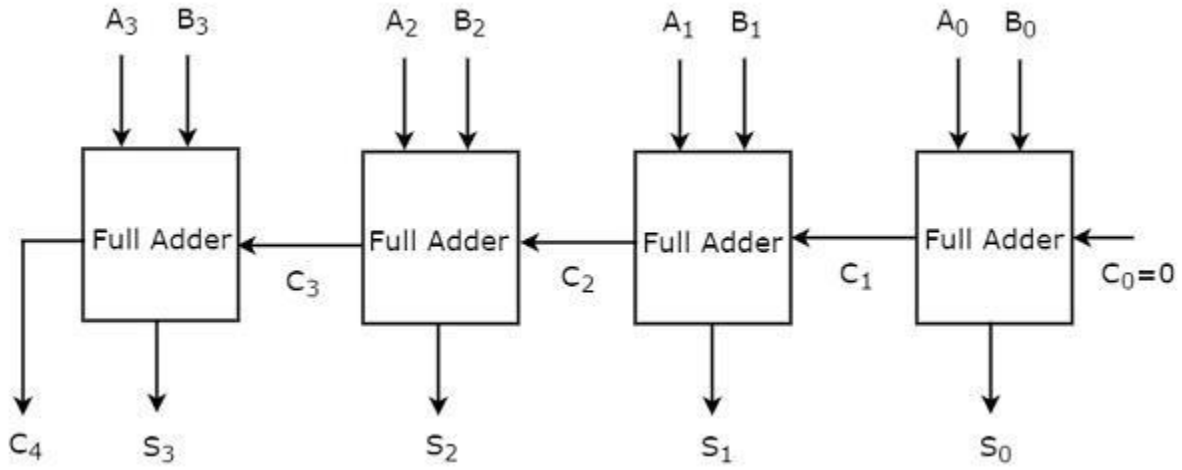
Logic diagram of a full-adder using only 2-input NOR gates.

3. 4-bit Binary Adder

The 4-bit binary adder performs the **addition of two 4-bit numbers**.
Let the 4-bit binary numbers, A = A_3 A_2 A_1 A_0 and B =B_3 B_2 B_1 B_0
We can implement 4-bit binary adder in one of the two following ways.
 Use one Half adder for doing the addition of two Least significant bits and three Full adders for doing the addition of three higher significant bits.

 Use four Full adders for uniformity. Since, initial carry Cinis zero, the Full adder which is used for adding the least significant bits becomes Half adder.

For the time being, we considered second approach. The **block diagram** of 4-bit binary adder is shown in the following figure.



Here, the 4 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs A & B. The carry output of one Full adder will be the carry input of subsequent higher order Full adder. This 4-bit binary adder produces the

resultant sum having at most 5 bits. So, carry out of last stage Full adder will be the MSB.
In this way, we can implement any higher order binary adder just by cascading the required number of Full adders. This binary adder is also called as **ripple carry (binary) adder** because the carry propagates (ripples) from one stage to the next stage.

B. Subtractors:

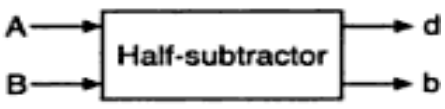
The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

1. The Half-Subtractor

A Half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.
A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



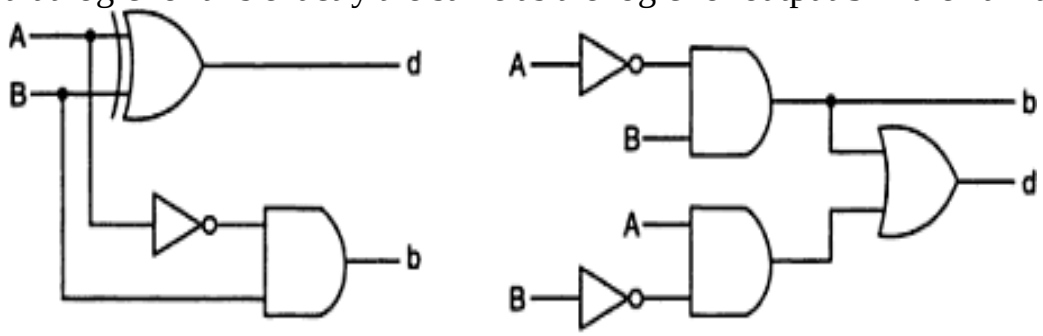
(b) Block diagram

Half-subtractor.

The output borrow b is a 0 as long as A≥B. It is a 1 for A=0 and B=1. The d output is the result of the arithmetic operation2b+A-B.
A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is, therefore,

Diff = A´B+ AB´ and
borr = $\bar{A}$ B

That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend. Note that logic for this exactly the same as the logic for output S in the half-adder.



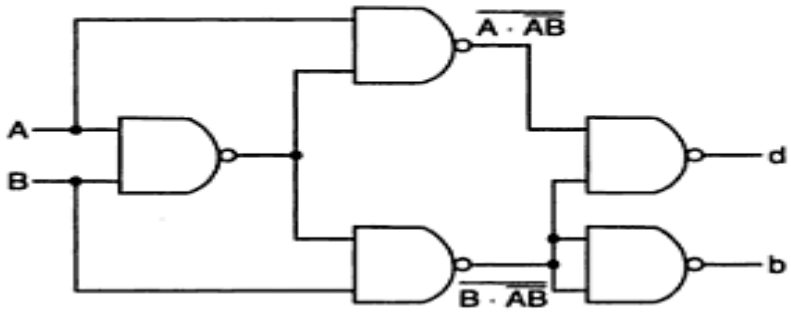
Logic diagrams of a half-subtractor.

A half-subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:

$$d = A \oplus B = \overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}$$

$$b = \bar{A}B = B(\bar{A} + \bar{B}) = B(\overline{AB}) = \overline{B \cdot \overline{AB}}$$



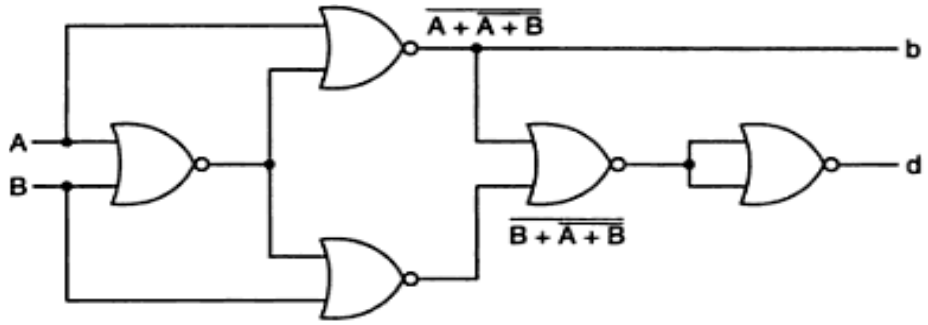
Logic diagram of a half-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B = A\bar{B} + \bar{A}B = A\bar{B} + B\bar{B} + \bar{A}B + A\bar{A}$$

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

$$d = \bar{A}B = \bar{A}(A + B) = \overline{\bar{A}(A + B)} = \overline{A + (\overline{A + B})}$$



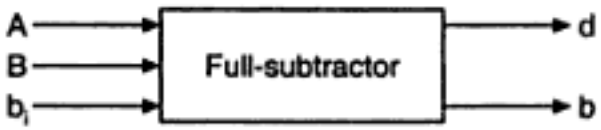
Logic diagram of a half-subtractor using only 2-input NOR gates.

2. The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A), when already there is a borrow bi from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of A-B-bi.

Inputs			Difference	Borrow
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-subtractor.

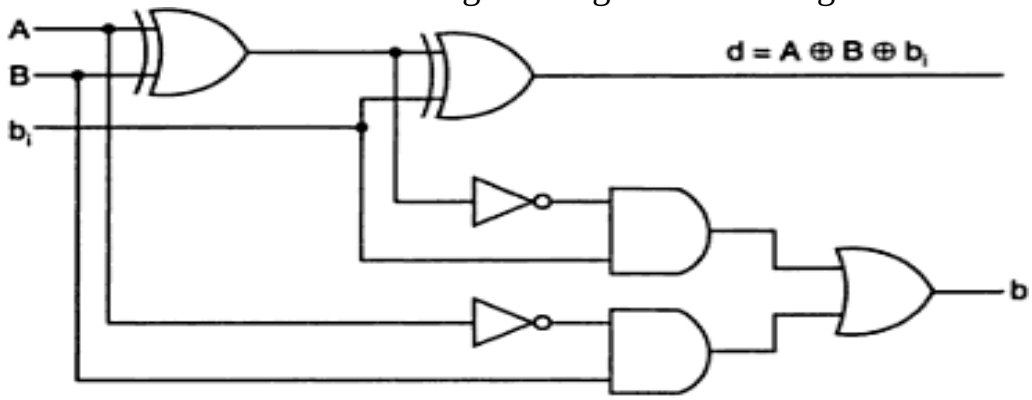
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combinations of A, B and bi is

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B)$$
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i$$

and

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$
$$= \bar{A}B + (\overline{A \oplus B})b_i$$

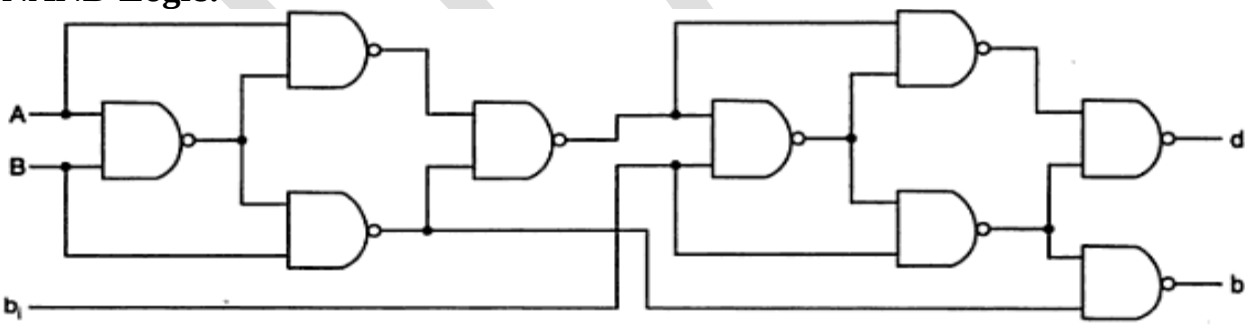
A full-subtractor can be realized using X-OR gates and AOI gates as



Logic diagram of a full-subtractor.

The full subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:



Logic diagram of a full-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}}$$
$$= \overline{\overline{(A \oplus B)b_i} + \overline{(\overline{A \oplus B})\bar{b}_i}}$$
$$= \overline{\overline{[(A \oplus B) + \overline{(\overline{A \oplus B})\bar{b}_i}][b_i + \overline{(\overline{A \oplus B})\bar{b}_i}]}}$$

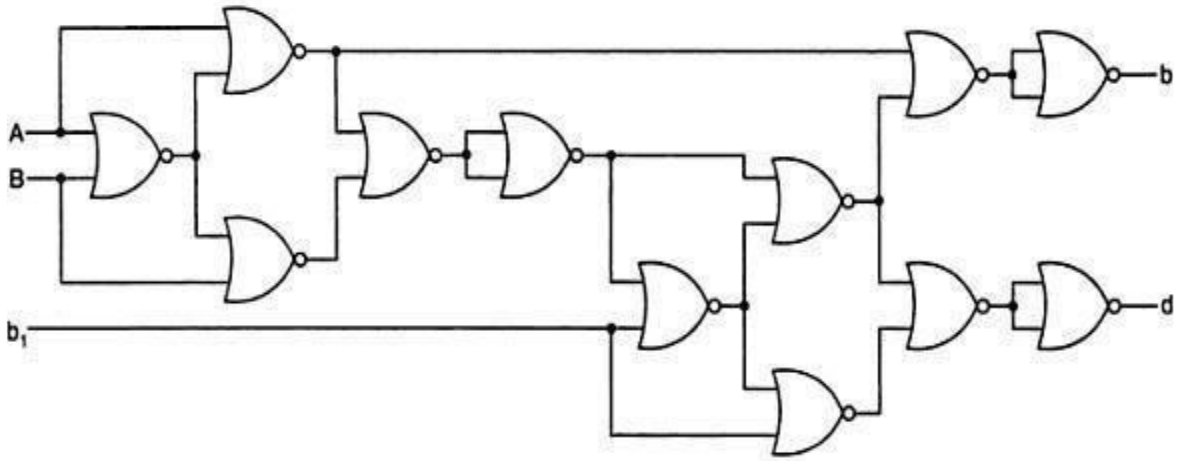
$$= \overline{\overline{(A \oplus B) + \overline{(A \oplus B) + b_i}} + \overline{b_i + \overline{(A \oplus B) + b_i}}}$$

$$= \overline{\overline{(A \oplus B) + \overline{(A \oplus B) + b_i}} + \overline{b_i + \overline{(A \oplus B) + b_i}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B})$$

$$= \bar{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + b_i]$$

$$= \overline{\overline{A + \overline{A + B}} + \overline{(A \oplus B) + \overline{(A \oplus B) + b_i}}}$$



Logic diagram of a full subtractor using only 2-input NOR gates.

C. Magnitude Comparator

A magnitude comparator compares two numbers A and B and determines their relative magnitudes. The results of comparison between two numbers are: A > B, A = B, A < B

Design Approaches: The truth table for two n-bit numbers comparison» 22*n* entries - too cumbersome for large *n* use inherent regularity of the problem (*algorithm* approach); *algorithm*— a procedure which specifies a finite set of steps, reduce design efforts; reduce human errors.

1. 1-bit Magnitude Comparator:

Block diagram of a 1-bit comparator

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\bar{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

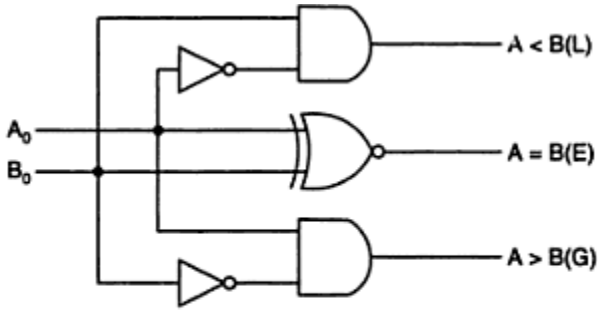
$$A < B: L = \bar{A}_0 B_0$$

If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

1-bit comparator.

2. 2-bit Magnitude Comparator

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

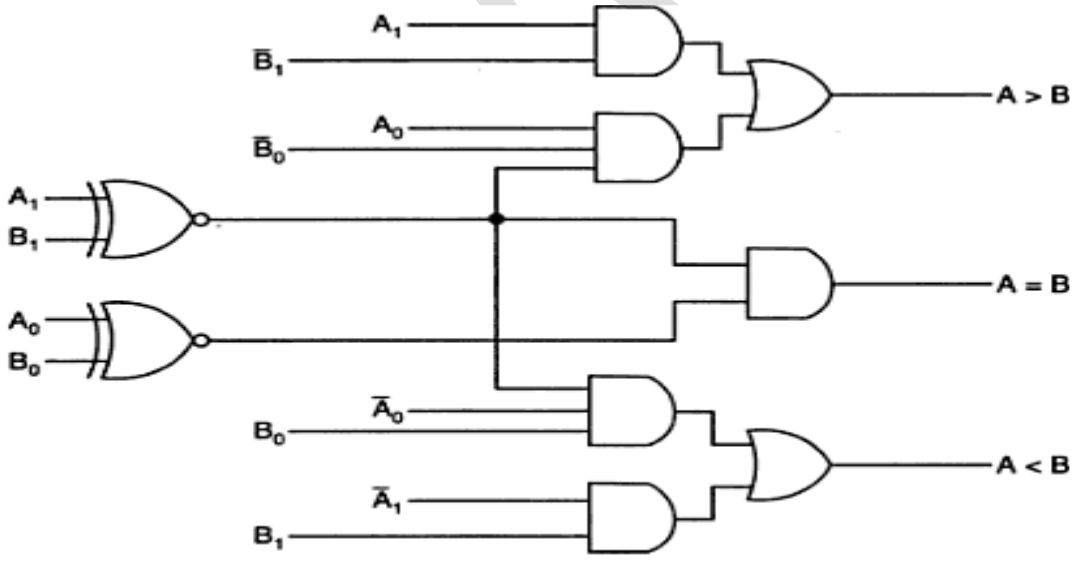
$$A > B : G = A_1\overline{B}_1 + (A_1 \odot B_1)A_0\overline{B}_0$$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$. So the expression for $A < B$ is

$$A < B : L = \overline{A}_1 B_1 + (A_1 \odot B_1)\overline{A}_0 B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$



Logic diagram of a 2-bit magnitude comparator.

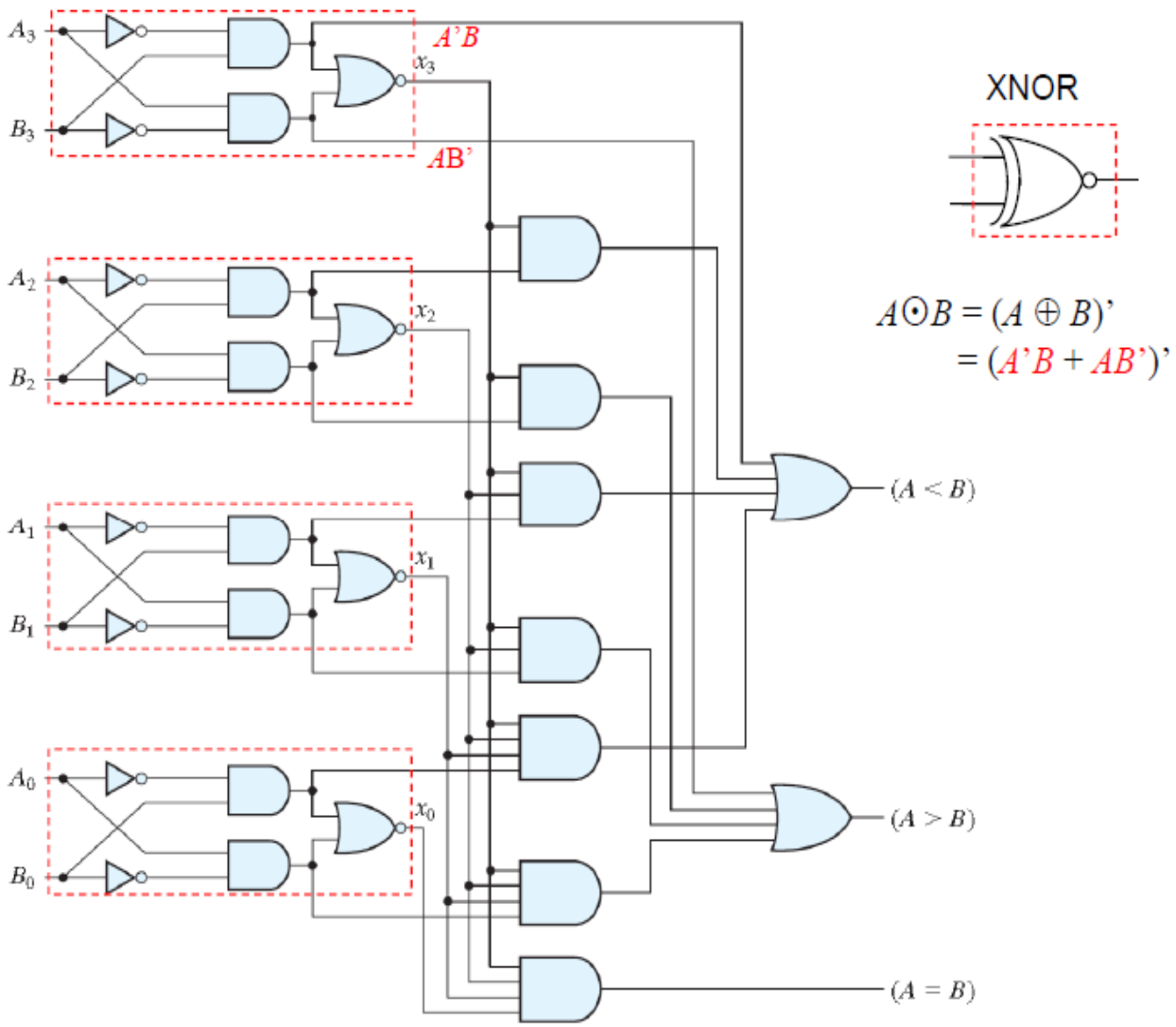
Consider two 4-bit numbers, $A = A_3 A_2 A_1 A_0$, $B = B_3 B_2 B_1 B_0$

- A and B are equal (A = B) if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, and $A_0 = B_0$.
- The equality of each pair of bits can be expressed with an exclusive-NOR function as:
  - xi = Ai Bi + Ai' Bi' for i = 0, 1, 2, 3; xi = (Ai' Bi + Ai Bi')'; xi = 1 only if the pair of bits in position i are equal (both are 1 or both are 0). For

equality to exist (A = B), all xi variables must be equal to 1: (A = B) = x3x2x1x0;To determine whether (A > B) or (A < B), starting from the MSB, if the two bits are equal, then compare the next lower significant pair of bits until a pair of unequal bits is reached.

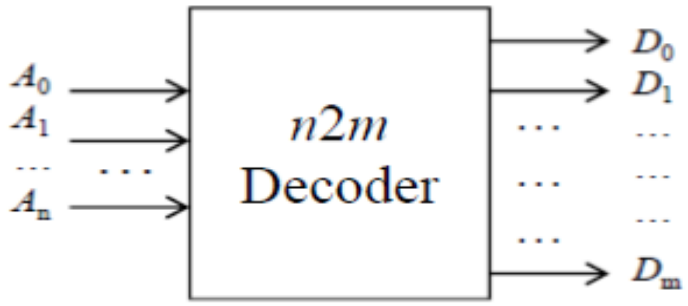
- If the corresponding bit of A is 1 and that of B is 0, we conclude that A > B.
- If the corresponding digit of A is 0 and that of B is 1, we have A < B.
- The sequential comparison can be expressed by the two Boolean functions

(A > B) = A3B3' + x3A2B2' + x3x2A1B1' + x3x2x1A0B0'
(A < B) = A3'B3 + x3A2'B2 + x3x2A1'B1 + x3x2x1A0'B0



D. DECODER

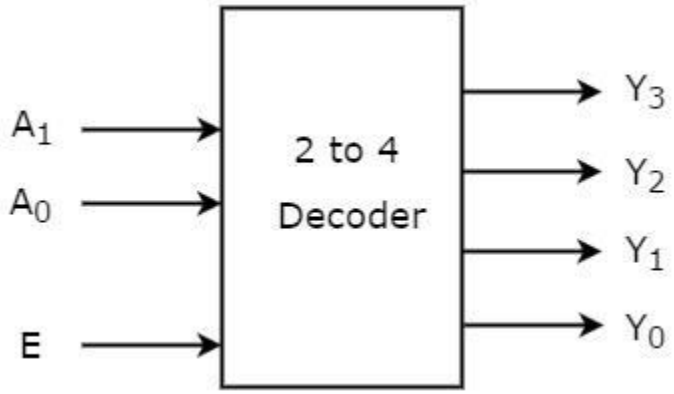
The output of a digital system is binary coded. A decoder is a circuit that energies a particular output line or lines depending on the binary code at the input. Thus, Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables (lines), when it is enabled.



Logic Diagram of Decoder

a. 2 to 4 Decoder:

Let 2 to 4 Decoder has two inputs A1 & A0 and four outputs Y3, Y2, Y1 & Y0. The **block diagram** of 2 to 4 decoder is shown in the following figure –



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below –

Enable	Inputs		Outputs			
E	**A_1**	**A_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

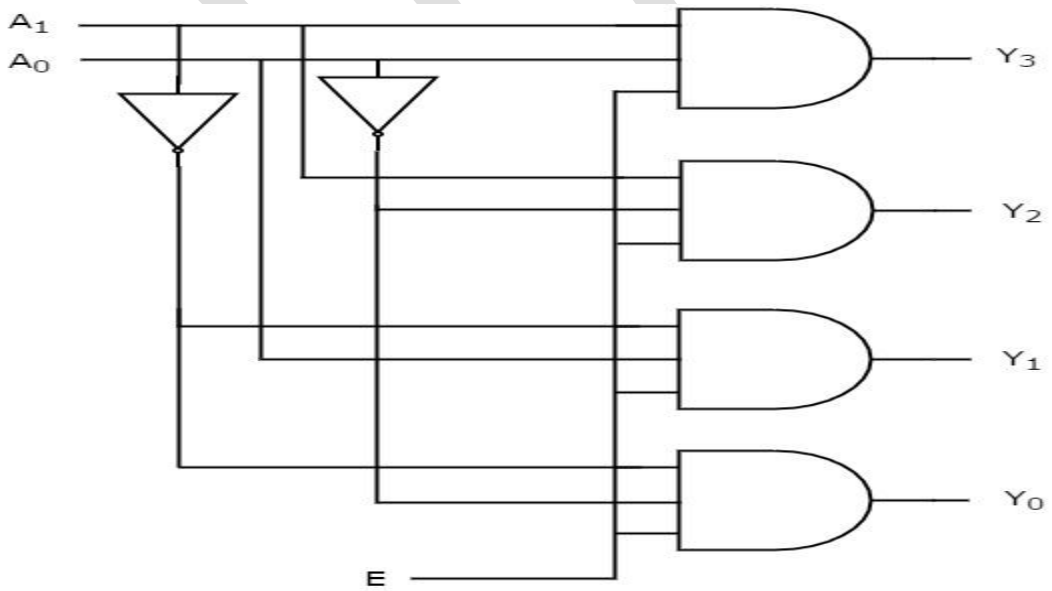
$Y3=E.A_1.A_0$

$Y2=E.A1.A_0'$

$Y1=E.A_1'.A_0$

$Y0=E.A_1'.A_0'$

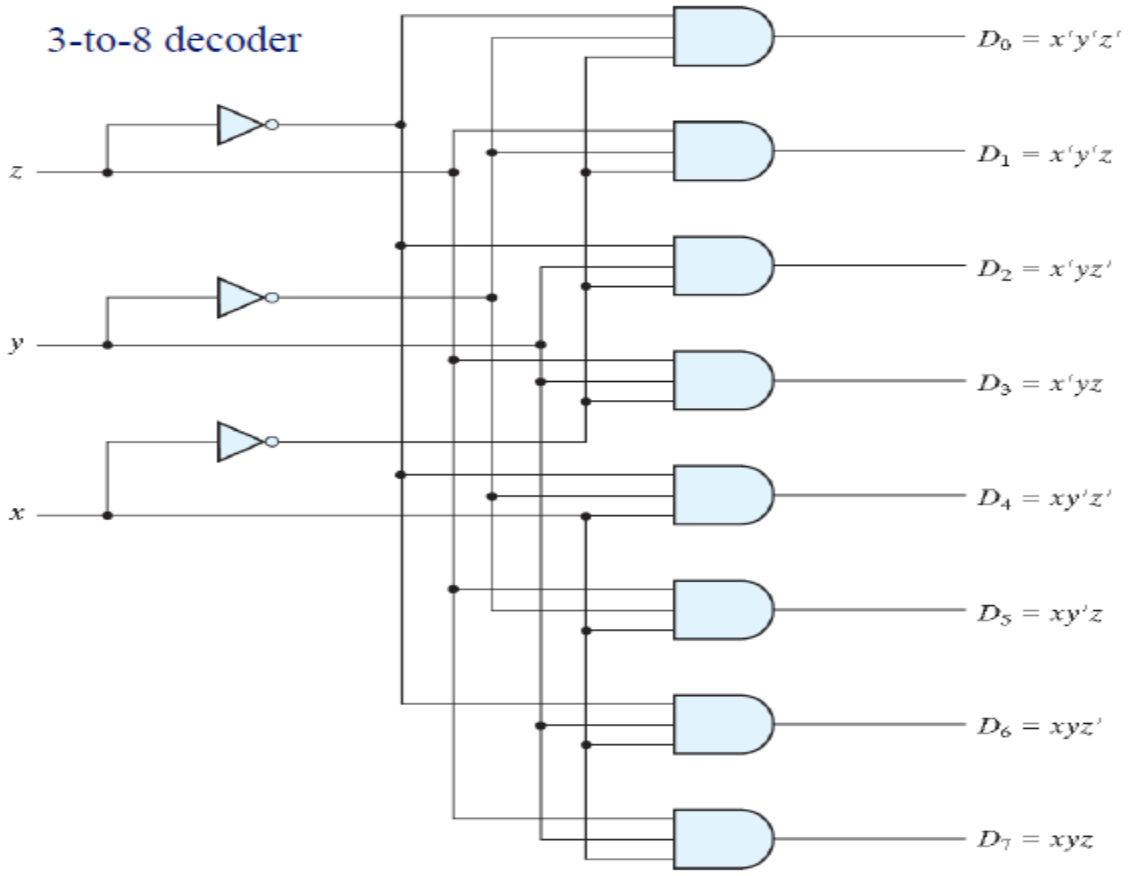
The **circuit diagram** of 2 to 4 decoder is shown in the following figure –



Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.
Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

Truth Table of a Three-to-Eight-Line Decoder

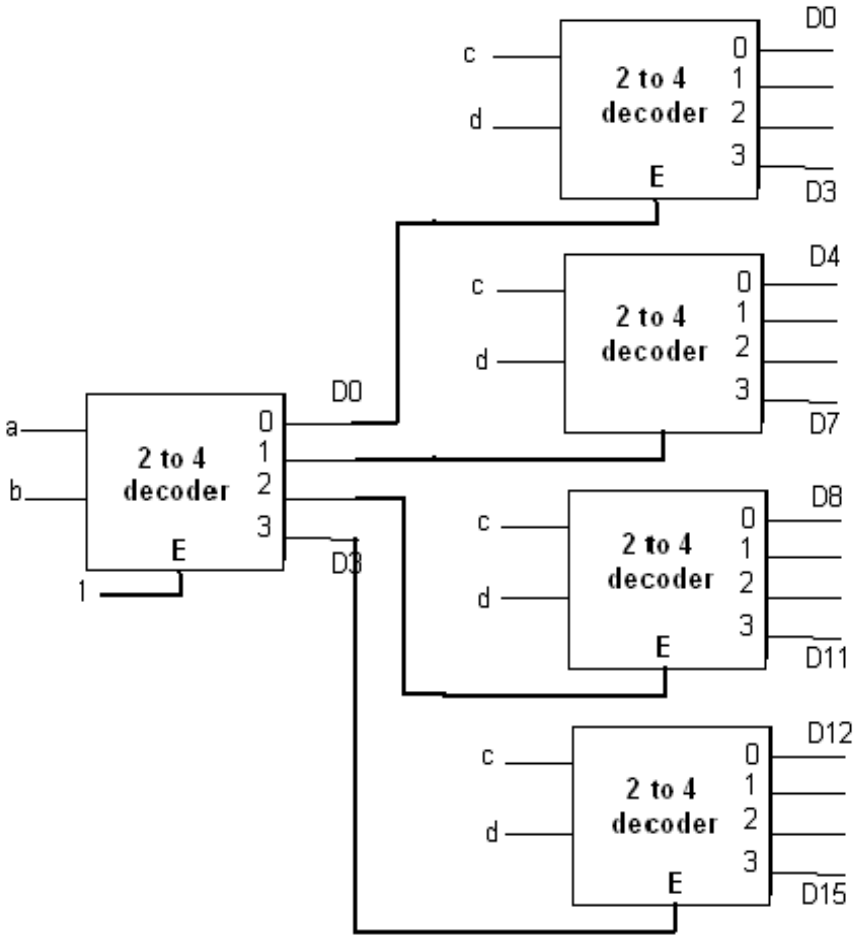
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



HIGHER DECODER FROM LOWER DECODERS

Example: Obtain a 4 to 16 decoder using a) 2 to 4 decoder (b) 3 to 8 decoder

a) we take $abcd_2$ as the input to the decoder. Following is the diagram to design 4 to 16 decoder using 2 to 4 decoders



When we have a = 0 & b=0 then top most decoder is enabled and 1 is placed on the output line out of 0 to 3 based on the value of cd.

When we have a = 0 & b=1 then 2nd decoder from top is enabled and 1 is placed on the output line out of 4 to 7 based on the value of cd.

When we have a=1 b=0 then 3rd decoder is enabled and 1 is placed on the output line out of 8 to 11 based on the value of cd.

When we have a=1 b=1 then bottom most decoder is enabled and 1 is placed on the output line out of 12 to 15 based on the value of cd.

Hence top 4 outputs generate min terms 0000 to 0011, next 4 generates min terms 0100 to 0111, next generates 1000 to 1011 and the last 4 outputs generate min terms 1100 to 1111.

b) **4 to 16 Decoder**

In this section, let us implement **4 to 16 decoder using 3 to 8 decoders**. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0

We know the following formula for finding the number of lower order decoders required.

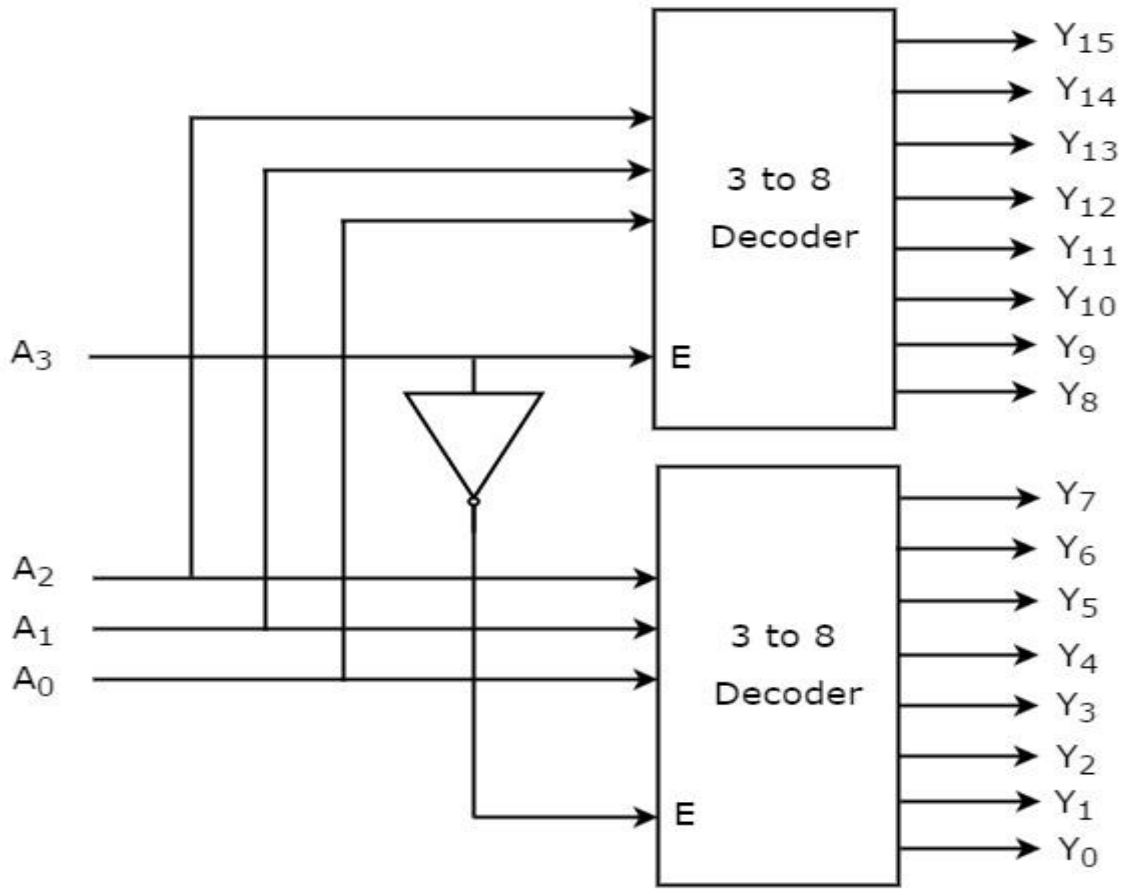
$$\text{Required number of lower order decoders} = \frac{m_2}{m_1}$$

Where, m_1 is the number of outputs of lower order decoder and m_2 is the number of outputs of higher order decoder.

Here, $m_1 = 8$ and $m_2 = 16$. Substitute, these two values in the above formula.

Required number of 2to4 decoders = 16/8 = 2

Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder. The **block diagram** of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure.



c) 3 to 8 Decoder

In this section, let us implement **3 to 8 decoder using 2 to 4 decoders**. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.

We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

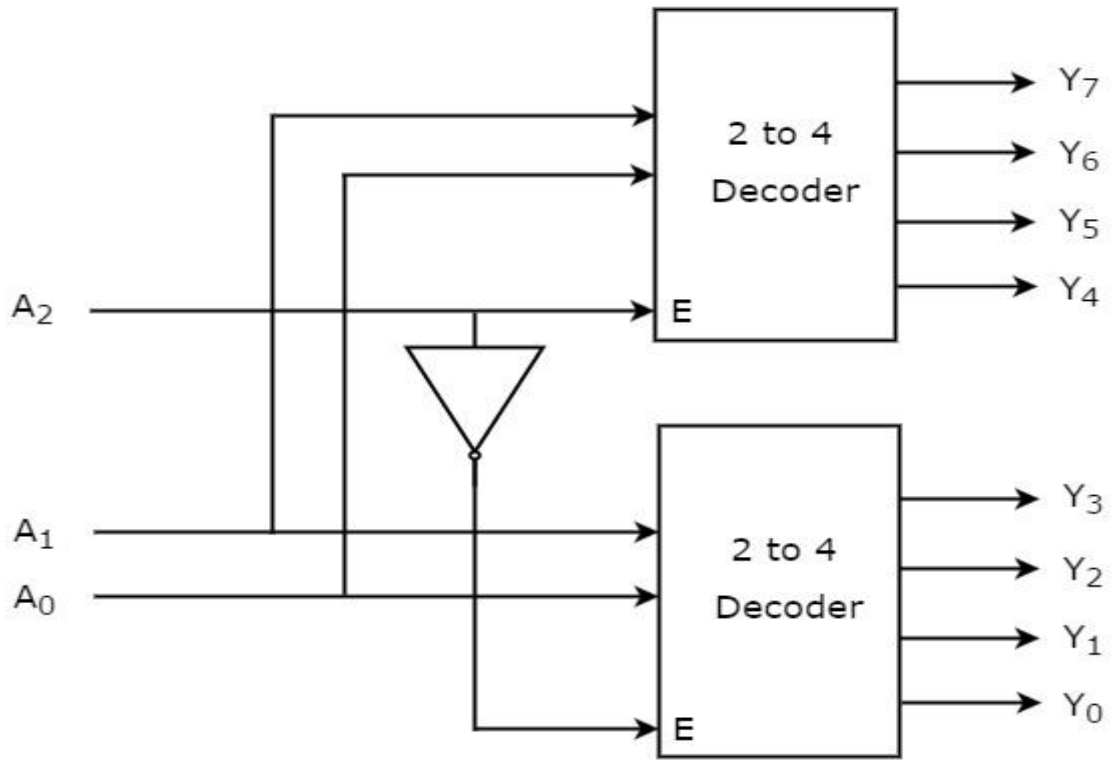
$$\text{Required number of lower order decoders} = \frac{m_2}{m_1}$$

Where, m_1 is the number of outputs of lower order decoder and m_2 is the number of outputs of higher order decoder.

Here, $m_1 = 4$ and $m_2 = 8$. Substitute, these two values in the above formula.

Required number of 2to4 decoders = 8/4 = 2

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.



d. Implementation the Full adder using 3 to 8 decoder.

For full adder, the equation for sum & carry are -

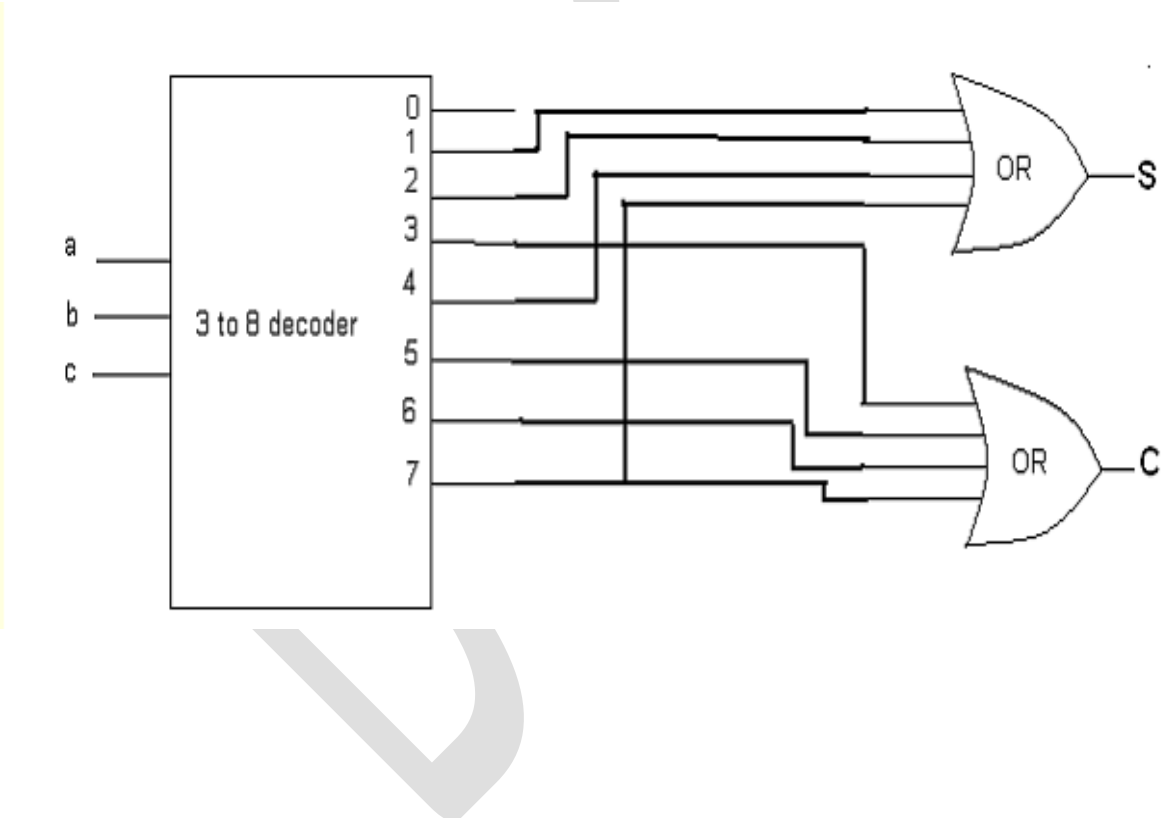
Sum = ab'c' + a'b'c + a'bc' + abc = Σ m (1,2,4,7)

Carry = ab + ac + bc = ab (c + c') + ac (b + b') + bc (a + a')

= abc + abc' + abc + ab'c + abc + a'bc

= abc +a'bc +ab'c+abc' = Σ m (3, 5, 6, 7)

So, we can implement it from decoder using OR gates as follow:

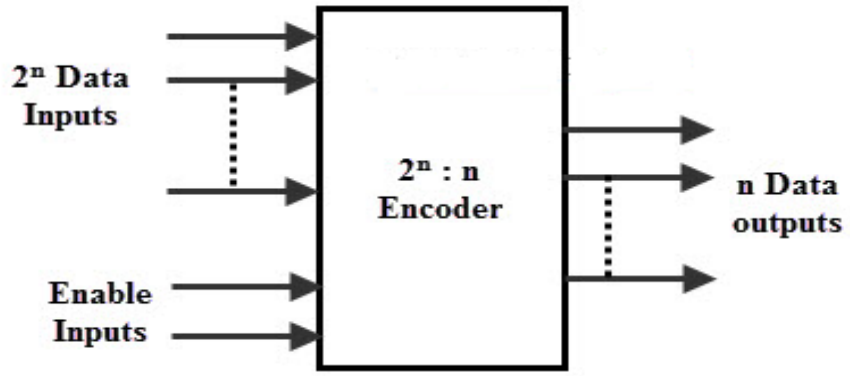


E. Encoder:

Digital circuits operate in a binary manner. So, the information available in the form of decimal numerals, alphabets or special characters is required to be converted into suitable binary form before it can be processed digital circuits. For this a process of coding is employed whereby each numerals, alphabets or special character is coced in a unique combination of 0s and 1s. The device that can be used to perform such coding is known as encoder.

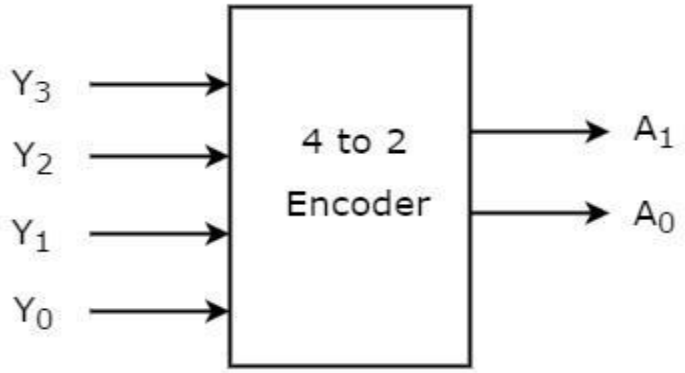
An encoder is basically multi inputs and multi outputs digital logic circuit, which has as many inputs as the number of character to be encoded and as many outputs as the number of bits in encoded form of characters.

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.



a. 4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The **block diagram** of 4 to 2 Encoder is shown in the following figure.



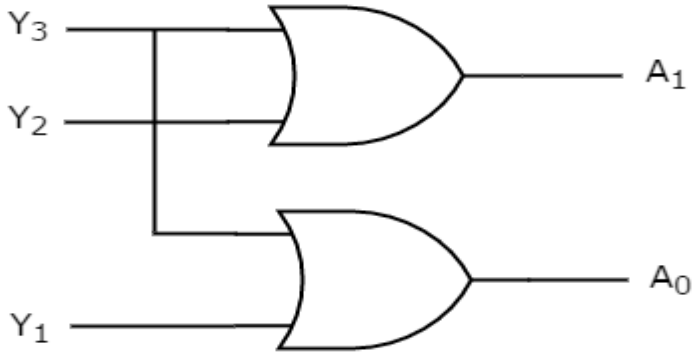
The **Truth table** of 4 to 2 encoder is shown below –

Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$
$$A_0 = Y_3 + Y_1$$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure –



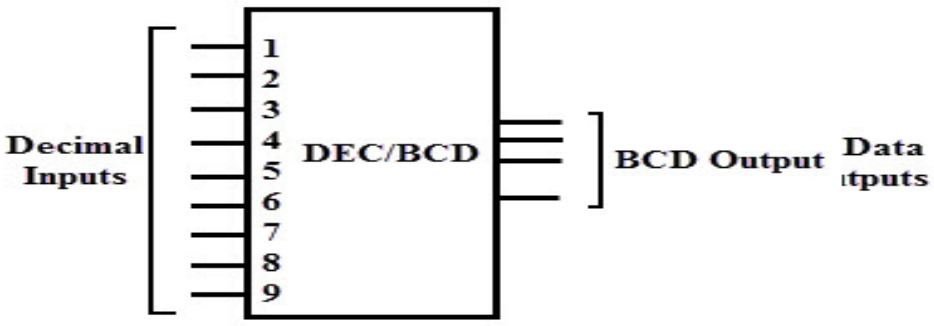
b. <u>Decimal to BCD Encoder</u>

This type of encoder usually consists of ten input lines and 4 output lines. Each input line corresponds to each decimal digit and 4 outputs correspond to the BCD code.

This encoder accepts the decoded decimal data as an input and encodes it to the BCD output which is available on the output lines.

The figure below shows the basic logic symbol of decimal to BCD encoder along with its truth table. The truth table represents the BCD code for each decimal digit.

From this we can formulate the relationship between the BCD bit and decimal digit. It is important to note that there is no explicit input line for decimal zero. When this condition occurs, i.e., decimal inputs 1 to 9 all are zero, then the BCD output is 0000.

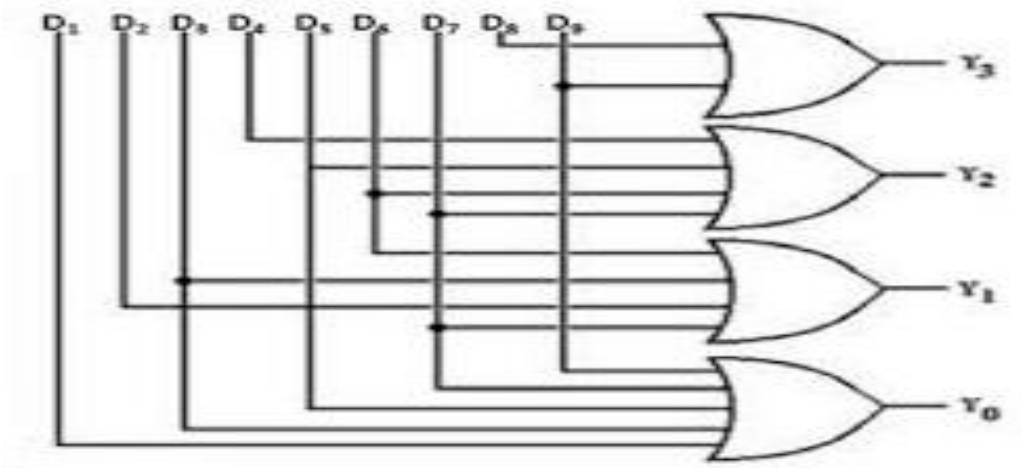


Logic Symbol

From the above table, we get the expressions as

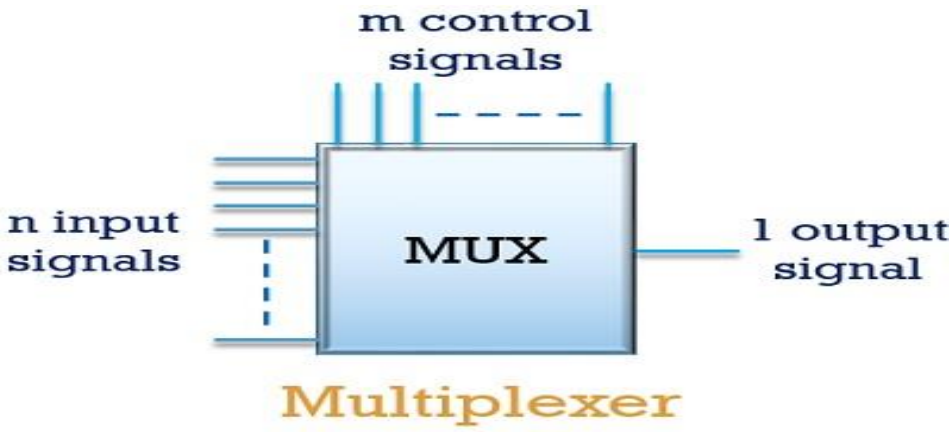
$$Y_3 = D_8 + D_9$$
$$Y_2 = D_4 + D_5 + D_6 + D_7$$
$$Y_1 = D_2 + D_3 + D_6 + D_7$$
$$Y_0 = D_1 + D_3 + D_5 + D_7 + D_9$$

From the above expressions, the decimal to BCD encoder logic circuit can be implemented by using set of OR gates as shown in below figure –

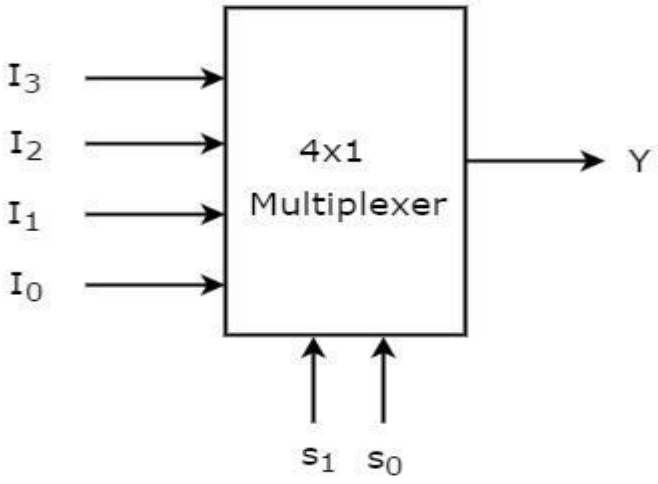


F. Multiplexer

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.
Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input.



4x1 Multiplexer:
4x1 Multiplexer has four data inputs I3, I2, I1 & I0, two selection lines s1 & s0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure –



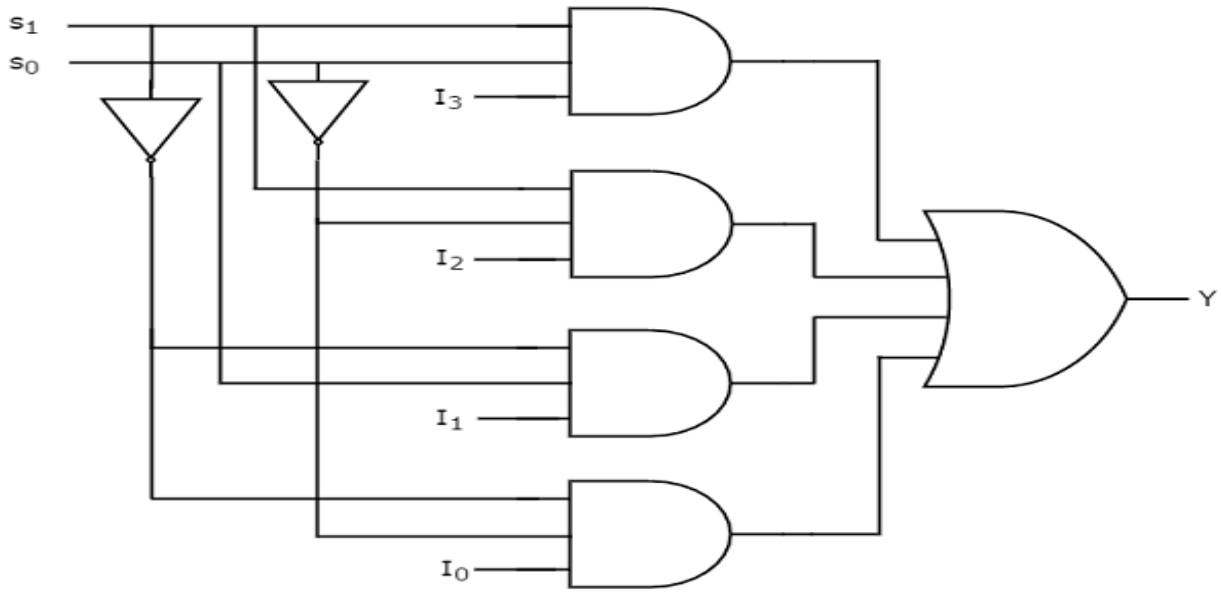
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below –

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y = S_1' S_0' I_0 + S_1' S_0 I_0 + S_1 S_0' I_2 + S_1 S_0 I_2$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure –

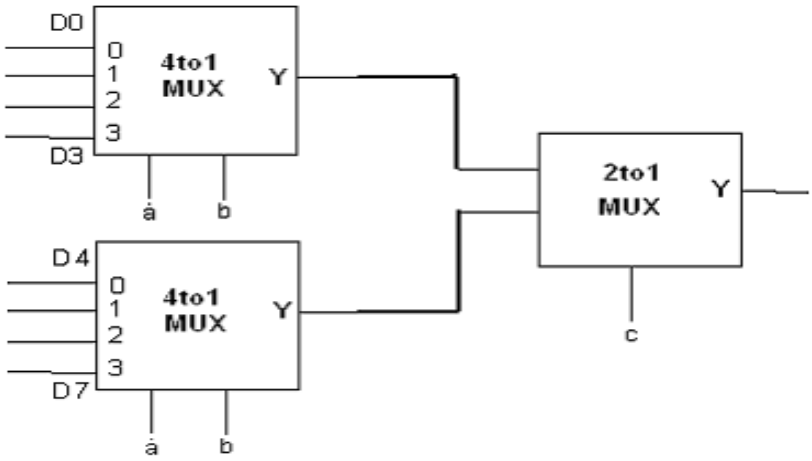


HIGHER MUXes FROM LOWER MUXes

Example 1: Implement - a) 8 to 1 MUX and b) 16 to 1 MUX using 4 to 1 MUX.

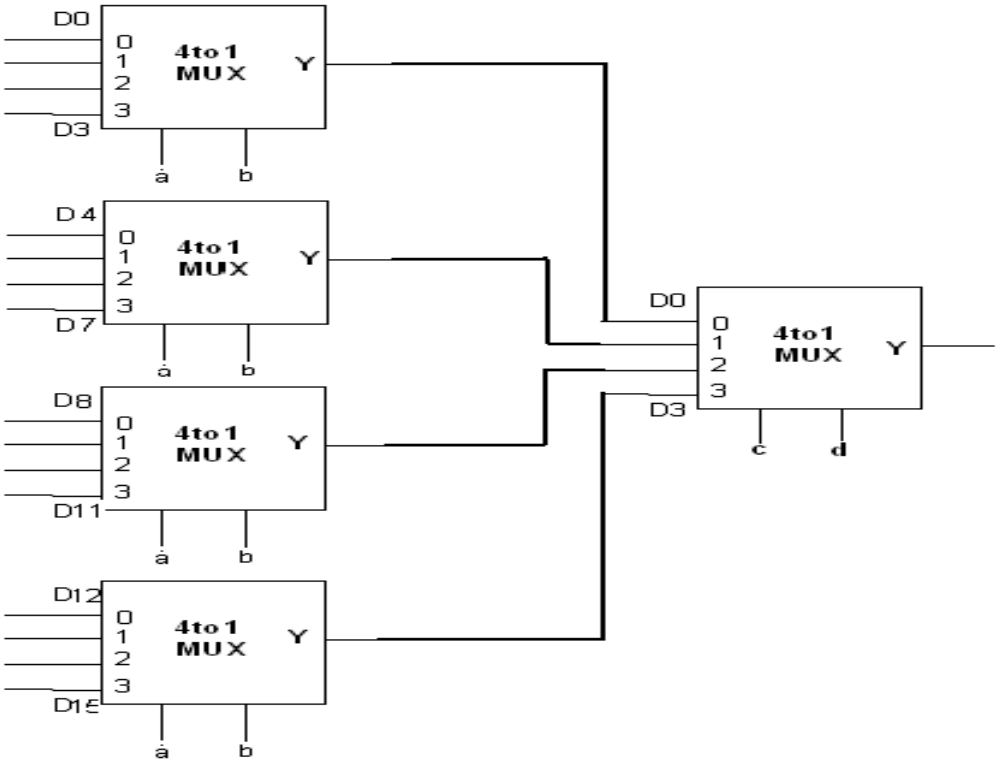
Ans: a) Select lines are $\mathbf{abc}_2$

Following is the 8 to 1 multiplexer from 4 to 1 multiplexer

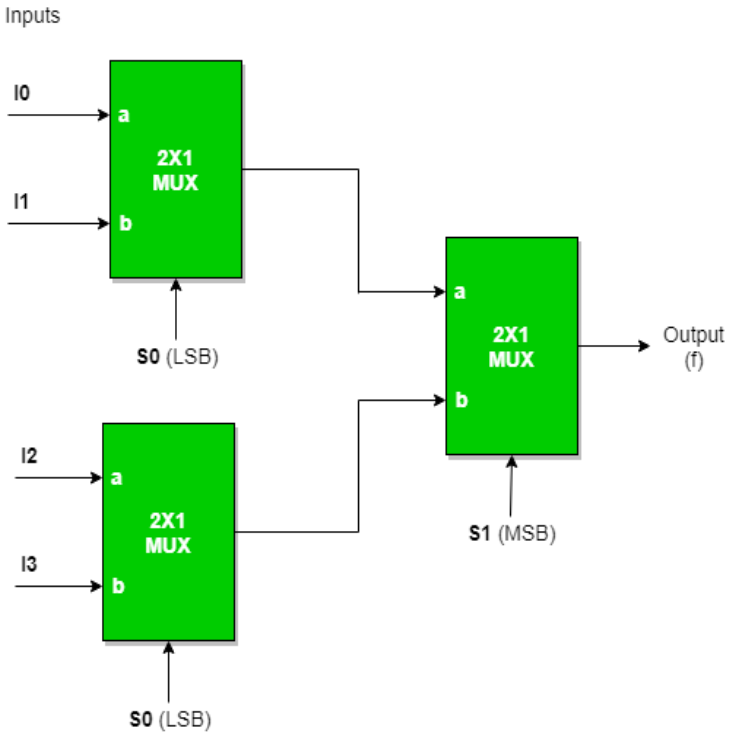


b) Select lines are $\mathbf{abcd}_2$

Following is the circuit for **16 to 1 MUX**



Example 2: 4: 1 MUX using 2: 1 MUX



Truth Table

S0	S1	f
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Implementation any Boolean function using MUX

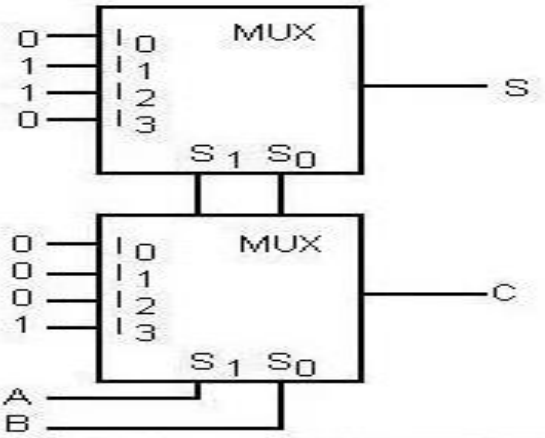
While implementing any function using MUX, if we have N variables in the function then we take (N-1) variables on the selection lines and 1 variable is used for inputs of MUX. As we have N-1 variables on selection lines we need to have 2^{N-1} to 1 MUX. We just have to connect A, A', 0 or 1 to different input lines.

1. Half Adder using 4 to 1 Multiplexer:

Hare, A & B are the inputs and S & C are the outputs. Now implementation function for sum and carry out are as followes.

Sum (A, B) = $\sum$ m (1,2)

Carry (A, B) = $\sum$ m (3)



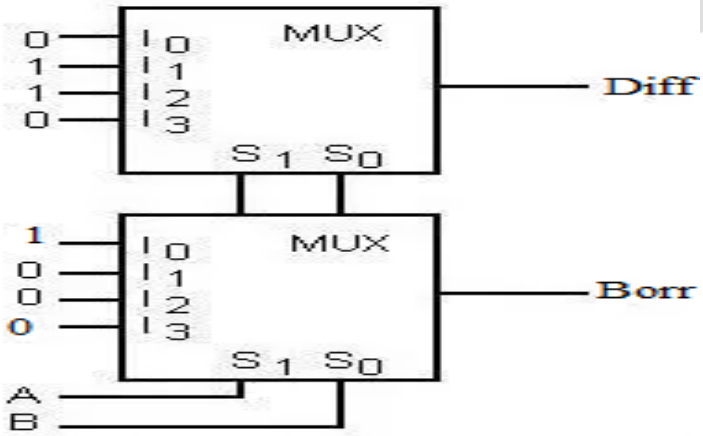
LOGIC DIAGRAM OF HALF-ADDER USING 4X1 MUX

2. Half Subtractor using 4 to 1 Multiplexer:

Hare, A & B are the inputs and Diff & Borr are the outputs. Now implementation function for difference and borrow are as follows -

Diff (A, B) = ∑ m (1,2)

Borr (A, B) = ∑ m (1)



Half Subtractor using MUX

3. Full Adder using 4 to 1 Multiplexer:

Multiplexer is also called a data selector, whose single output can be connected to anyone of N different inputs. A 4 to 1-line multiplexer has 4 inputs and 1 output line.

Hare, A,B,Cin are the inputs and S & Cout are the outputs. Now implementation function for sum and carry out are as follows

S(A,B,Cin)= ∑(1,2,4,7)

Cout(A,B,Cin)= ∑(3,5,6,7)

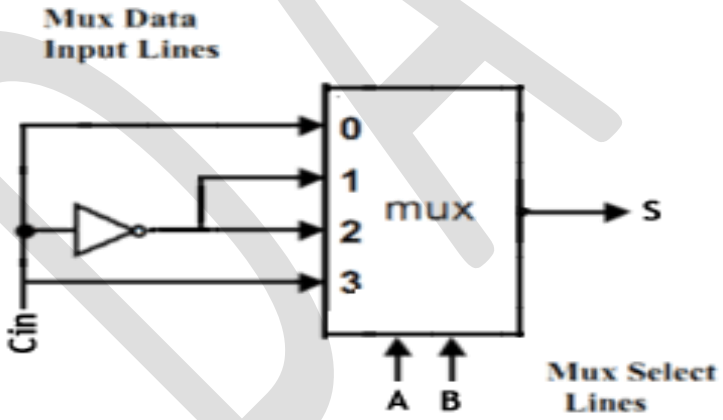
For sum out:

Cin \ AB	00	01	10	11
0	0	1	1	0
1	1	0	0	1
	Cin	C'in	C'in	Cin

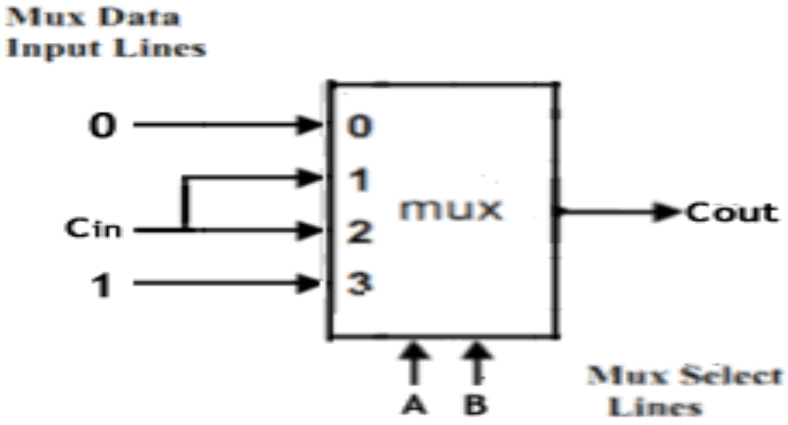
For carry out:

Cin \ AB	00	01	10	11
0	0	0	0	1
1	0	1	1	1
	0	Cin	Cin	1

Logic Diagram for Sum out using MUX –

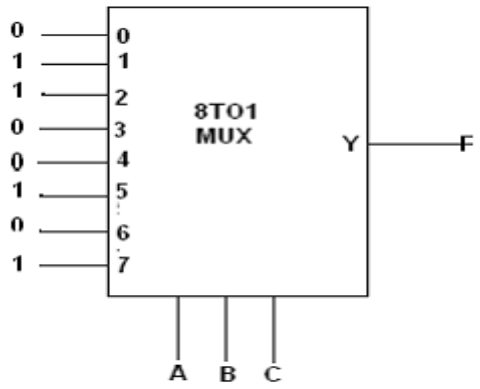


Logic Diagram for Carry out using MUX –



Example 1: To implement the function F(A, B, C)= Σ (1, 2, 5, 7) using (a)8 to 1 MUX (b)4 to 1 MUX

Ans: We can implement it using all three variables at selection lines. We put 1 on the min term lines which are present in functions and 0 on the rest.



Example 2: F= A'B'C + A'BC' + AB'C + ABC

N=3 so we use $2^{N-1} = 2^2 = 4$ to 1 MUX.

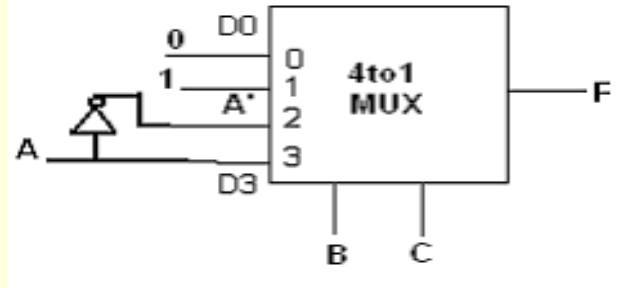
Suppose we have B, C on the selection lines. So, when we have BC = 00, put B = 0, C = 0 in the function and we see output of the function should be 0 hence we connect 0 to 0th input line.

When BC = 01, then output of the function should be A'+ A = 1. Hence, we connect 1 to 1st line.

When BC =10, then output of the function should be A'. Hence we connect A' to 2nd line.

When BC = 11, then output of the function should be A. Hence, we connect A to 3rd line.

Hence, we have the circuit as:



Another procedure to implement the function using MUX

- Take one variable for input lines and rest of the term for selection lines.
- Then list the min terms with the variable selected in complimented form in 1st row and list the
- The min terms with variable selected in un-complimented form in 2nd row.
- Then encircle the min terms which are present in the function.

  - If we have no circled variable in the column, then we put 0 on the corresponding line
  - If we have both circled variables, then we put 1 on the line
  - If bottom variable is circled and top is not circled, apply A to input line
  - If bottom variable is not circled and top is circled, apply A' to input line

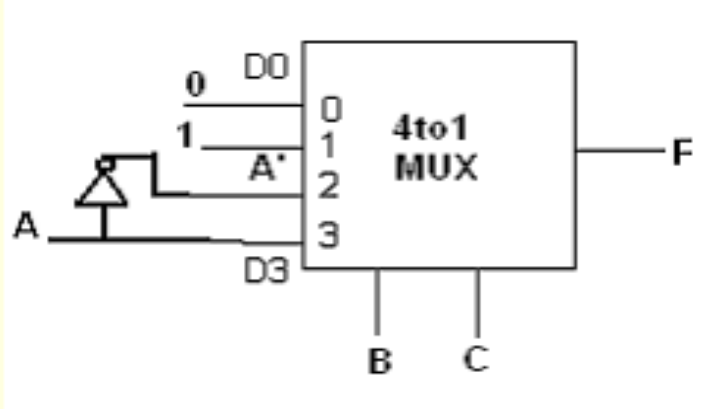
Example 3: To implement the function F (A, B, C) = Σ (1, 2, 5, 7) using MUX.

Let's now take the variable A for input lines and B & C for selection lines.

So, we list the min terms as follow:

	D0	D1	D2	D3
A'	0	(1)	(2)	3
A	4	(5)	6	(7)
	0	1	A'	A

So, the circuit is -



Example 4: To implement the function F(A, B, C, D)= Σ (1, 2, 5, 7, 9, 14) using MUX using different variable as selection variable.

Let's now take the **variable A** for input lines and **B, C & D** for selection lines.

N=4 so MUX is $2^{N-1} = 2^3 = 8$ to 1

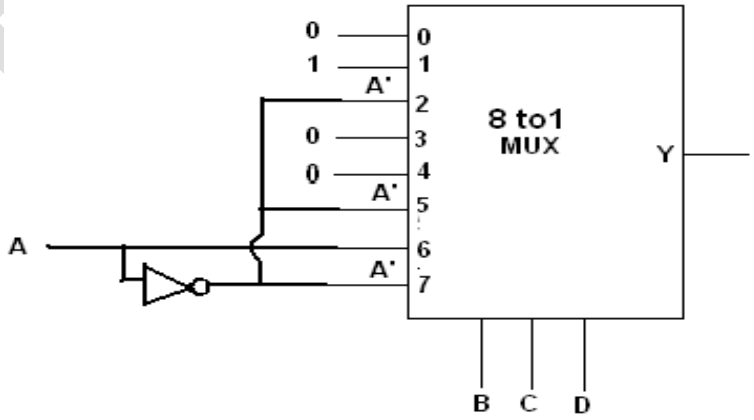
So, min terms with A in compliment form are 0 – 7

So, min terms with A in un-compliment form are 8 – 15

So, we list the MIN TERMS as:

	D0	D1	D2	D3	D4	D5	D6	D7
A'	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	0	1	A'	0	0	A'	A	A'

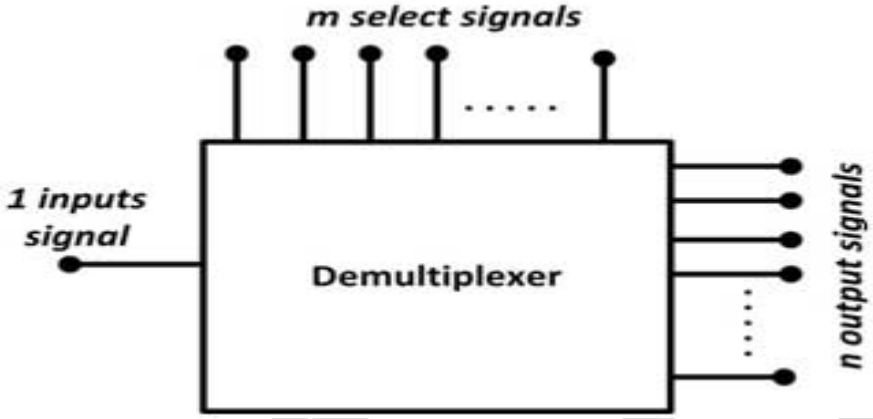
And the circuit diagram is shown below:



G. De-Multiplexer:

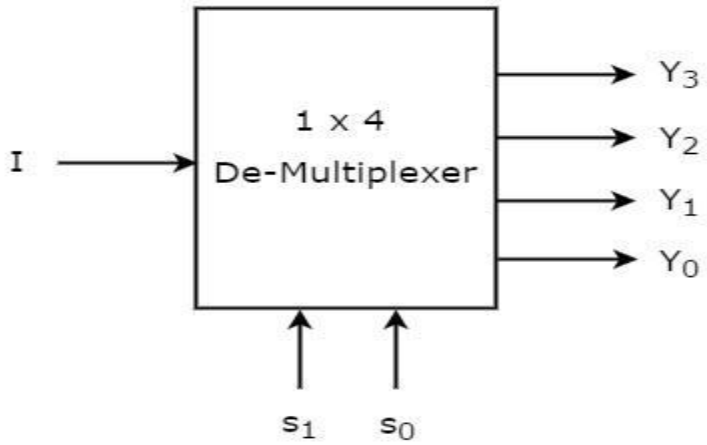
The Demultiplexer is a combinational circuit having a single input and many outputs. It performs the reverse operation of a Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.



1x4 De-Multiplexer :

1x4 De-Multiplexer has one input I, two selection lines, s1 & s0and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of (1 x 4) De-Multiplexer is shown in the following figure



The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines S_1 & S_0. The **Truth table** of 1x4 De-Multiplexer is shown below –

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output as

$$Y3 = S_1\ S_0\ I$$
$$Y2 = S_1\ S_0'\ I$$
$$Y1 = S_1'\ S_0\ I$$
$$Y0 = S_1'\ S_0'\ I$$

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure -