MULTIPLEXERS AND DE MULTIPLEXERS

A ***multiplexer*** or *MUX*, is a combinational circuit with more than one input line, one output line and more than one selection line. A multiplexer selects binary information present from one of many input lines, depending upon the logic status of the selection inputs, and routes it to the output line. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. The multiplexer is often labeled as MUX in block diagrams.

A multiplexer is also called a **data selector**, since it selects one of many inputs and steers the binary information to the output line.

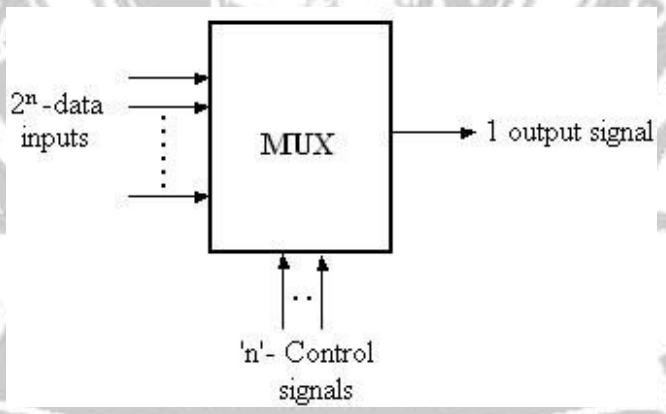


Fig : 2.30 - Block diagram of Multiplexer

2-to-1- line Multiplexer:

The circuit has two data input lines, one output line and one selection line, S. When S= 0, the upper AND gate is enabled and I_0 has a path to the output.

When S=1, the lower AND gate is enabled and I_1 has a path to the output.

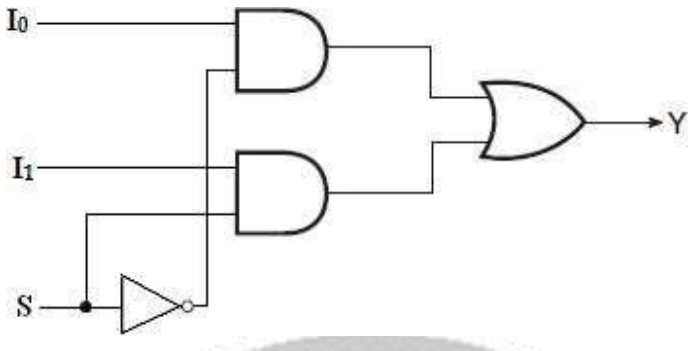


Fig : 2.31 - Logic diagram

The multiplexer acts like an electronic switch that selects one of the two sources.

Truth table:

S	Y
0	I0
1	I1

4-to-1-line Multiplexer:

A 4-to-1-line multiplexer has four (2n) input lines, two (n) select lines and one output line. It is the multiplexer consisting of four input channels and information of one of thechannels can be selected and transmitted to an output line according to the select inputs combinations. Selection of one of the four input channel is possible by two selection inputs.

Each of the four inputs I_0 through I_3, is applied to one input of AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.

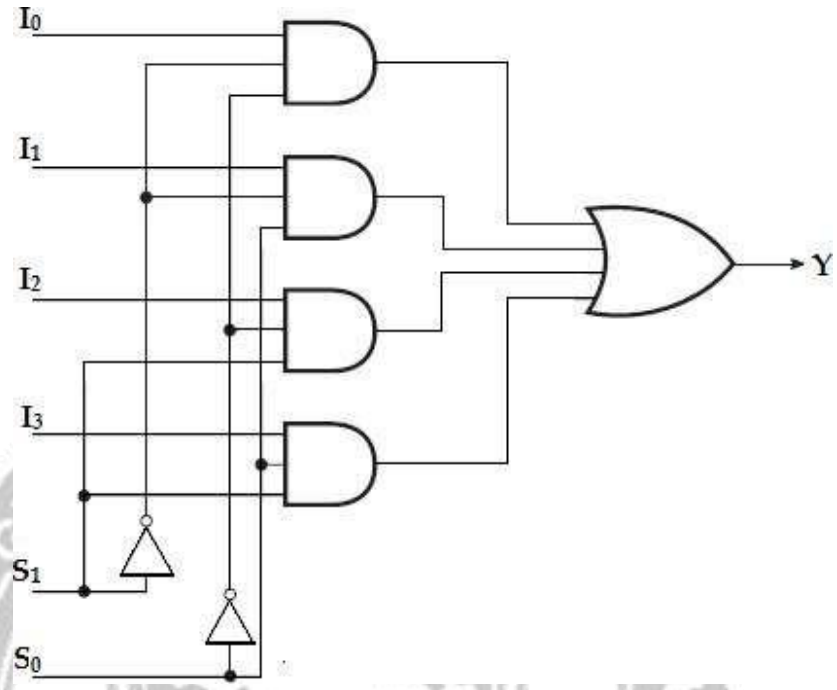


Fig : 2.32 - 4-to-1-Line Multiplexer

Function table:

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

To demonstrate the circuit operation, consider the case when S_1S_0= 10. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I_2, providing a path from the selected input to the output.

The data output is equal to I_0 only if S_1= 0 and S_0= 0; Y= $I_0S_1'S_0'$.

The data output is equal to I_1 only if S_1= 0 and S_0= 1; Y= $I_1S_1'S_0$.

The data output is equal to I_2 only if S_1= 1 and S_0= 0; Y= $I_2S_1S_0'$.

The data output is equal to I_3 only if S_1= 1 and S_0= 1; Y= $I_3S_1S_0$. When these terms are ORed, the total expression for the data output is,

Y= I0S1'S0'+ I1S1'S0 +I2S1S0'+ I3S1S0.

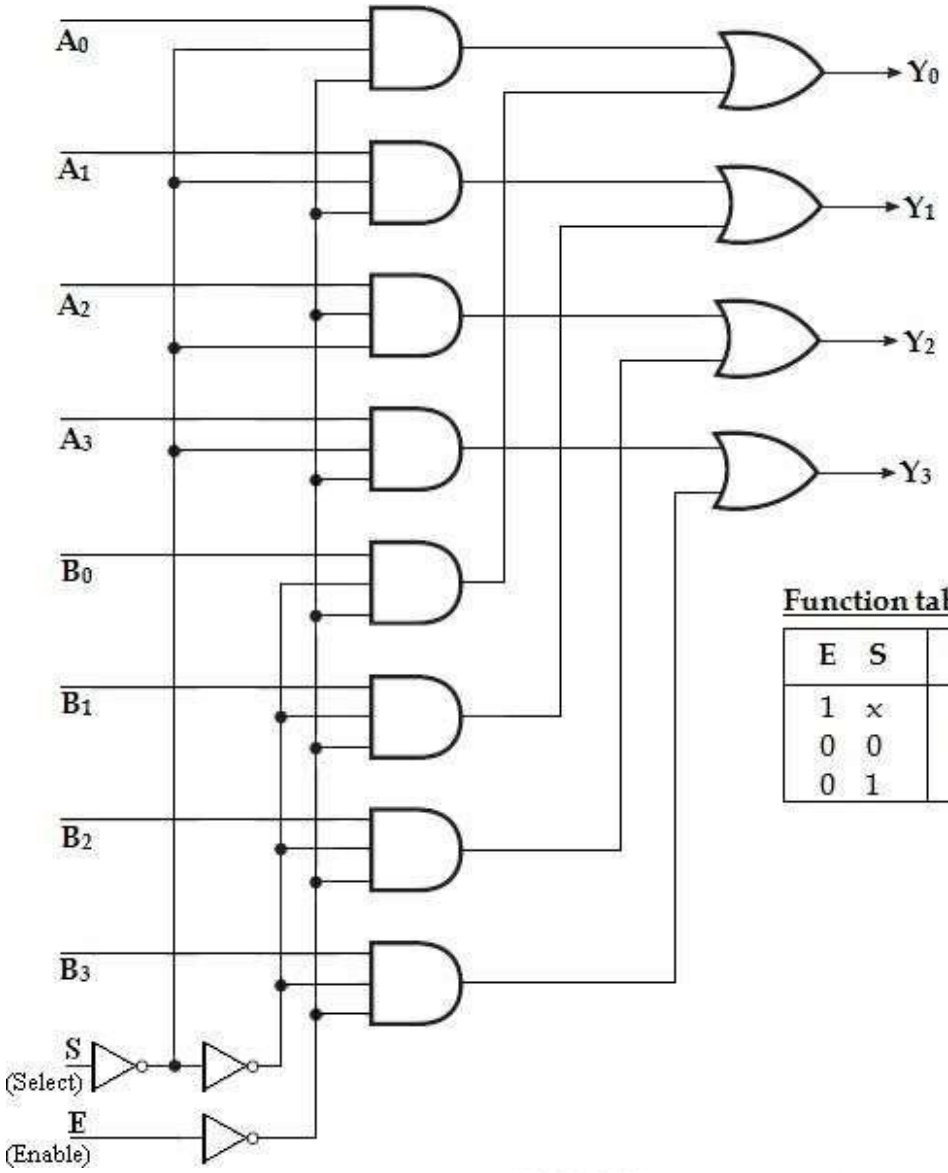
As in decoder, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer.

Quadruple 2-to-1 Line Multiplexer:

This circuit has four multiplexers, each capable of selecting one of two input lines. Output Y_0 can be selected to come from either A0 or B0. Similarly, output Y1 may have the value of A1 or B1, and so on. Input selection line, S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal operation.

Although the circuit contains four 2-to-1-Line multiplexers, it is viewed as a circuit that selects one of two 4-bit sets of data lines. The unit is enabled when E= 0. Then if S= 0, the four A inputs have a path to the four outputs. On the other hand, if

S=1, the four B inputs are applied to the outputs. The outputs have all 0's when E= 1, regardless of the value of S.



Function table:

E	S	Output Y
1	x	All 0's
0	0	Select A
0	1	Select B

Fig : 2.33 – Quadruple 2 to 1 MUX

Application:

The multiplexer is a very useful MSI function and has various ranges of applications in data communication. Signal routing and data communication are the important applications of a multiplexer. It is used for connecting two or more sources to guide to a single destination among computer units and it is useful for constructing a common

bus system. One of the general properties of a multiplexer is that Boolean functionscan be implemented by this device.

Implementation of Boolean Function using MUX:

Any Boolean or logical expression can be easily implemented using a multiplexer. If a Boolean expression has (n+1) variables, then _n' of these variables can be connected to the select lines of the multiplexer. The remaining single variable along with constants 1 and 0 is used as the input of the multiplexer. For example, if C is the single variable, then the inputs of the multiplexers are C, C', 1 and 0. By this method any logicalexpression can be implemented.

In general, a Boolean expression of (n+1) variables can be implemented using a multiplexer with 2^n inputs.

1. **Implement the following boolean function using 4: 1 multiplexer,**

   F (A, B, C) = ∑m (1, 3, 5, 6). Solution:

   Variables, n= 3 (A, B, C) Select lines=

   n-1 = 2 (**S_1, S_0**) 2^{n-1} to MUX i.e., 2^2 to

   1 = 4

   to 1 MUX

   Input lines= $2^{n-1} = 2^2 = 4$ (**D_0, D_1, D_2, D_3**)

Implementation table:

Apply variables A and B to the select lines. The procedures for implementing the function are:

i. List the input of the multiplexer

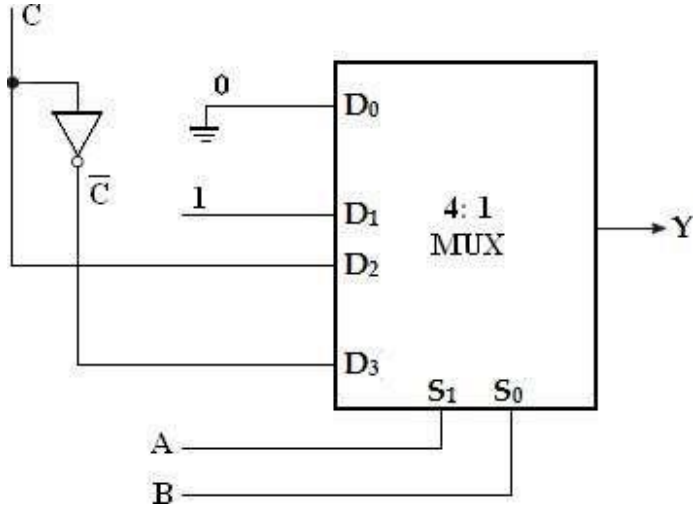
ii. List under them all the minterms in two rows as shown below.

The first half of the minterms is associated with A‘ and the second half with A. The given function is implemented by circling the minterms of the function and applying thefollowing rules to find the values for the inputs of the multiplexer.

1. If both the minterms in the column are not circled, apply 0 to the correspondinginput.
2. If both the minterms in the column are circled, apply 1 to the correspondinginput.
3. If the bottom minterm is circled and the top is not circled, apply C to the input.4.If the top minterm is circled and the bottom is not circled, apply C‘ to the input.

	D_0	D_1	D_2	D_3
$\overline{C}$	0	(1)	2	(3)
C	4	(5)	(6)	7
	0	**1**	C	$\overline{C}$

Multiplexer Implementation:

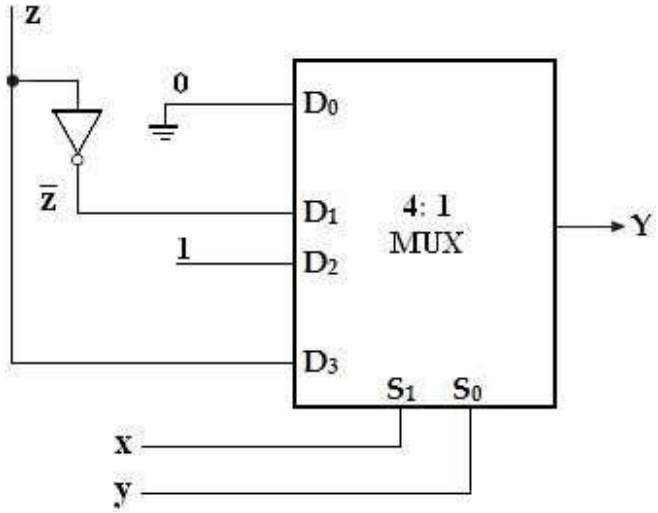


2. F (x, y, z) = ∑m (1, 2, 6, 7) Solution:

Implementation table:

	D_0	D_1	D_2	D_3
$\overline{z}$	0	(1)	(2)	3
z	4	5	(6)	(7)
	0	$\overline{z}$	1	z

Multiplexer Implementation:



3. F (A, B, C) = ∑m (1, 2, 4, 5) Solution:

Variables, n= 3 (A, B, C) Select lines=

n-1 = 2 (**S_1, S_0**) 2^{n-1} to MUX i.e., 2^2 to

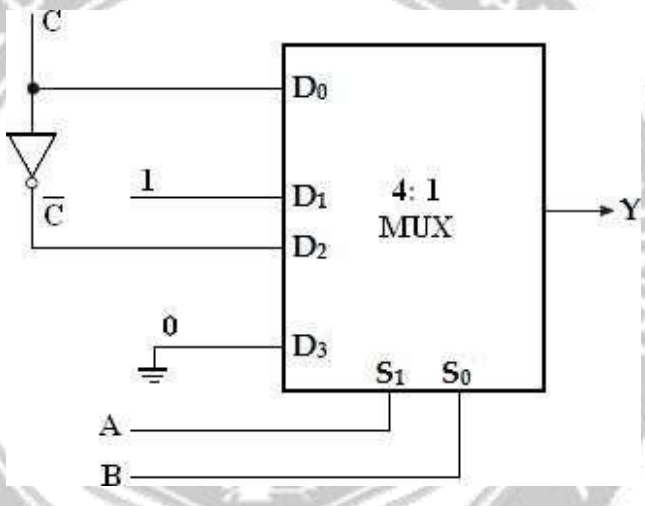
1 = 4

to 1 MUX

Input lines= $2^{n-1} = 2^2 = 4$ (**D_0, D_1, D_2, D_3**) **Implementation table:**

	D_0	D_1	D_2	D_3
$\overline{C}$	0	(1)	(2)	3
C	(4)	(5)	6	7
	C	1	$\overline{C}$	0

Multiplexer Implementation:



4. F(P, Q, R, S)= ∑m (0, 1, 3, 4, 8, 9, 15)

Solution:

Variables, n= 4 (P, Q, R, S) Select

lines= n-1 = 3 (**S_2, S_1, S_0**)

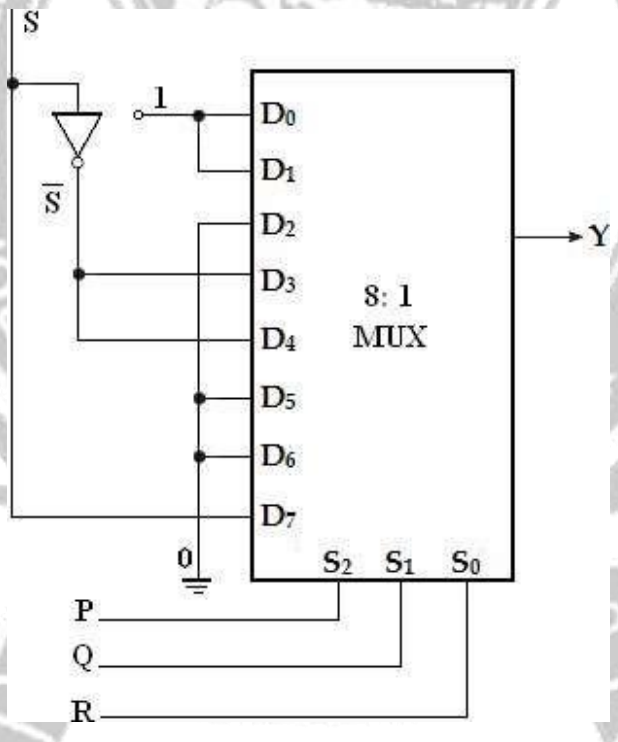
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{S}$	0	1	2	3	4	5	6	7
S	8	9	10	11	12	13	14	15
	1	**1**	**0**	**$\overline{S}$**	**$\overline{S}$**	**0**	**0**	**S**

Multiplexer Implementation:



5. Implement the Boolean function using 8: 1 and also using 4:1 multiplexer F (A, B, C, D) = ∑m (0, 1, 2, 4, 6, 9, 12, 14)

Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

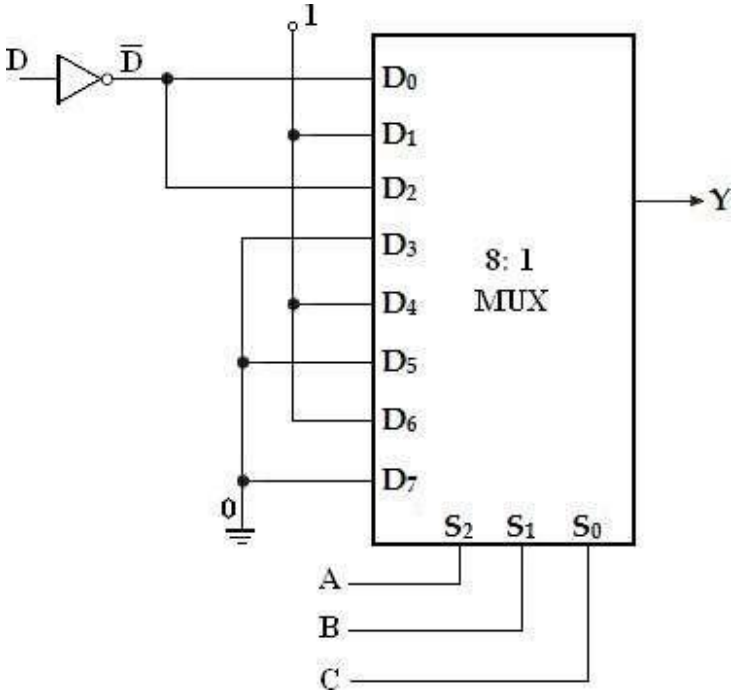
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

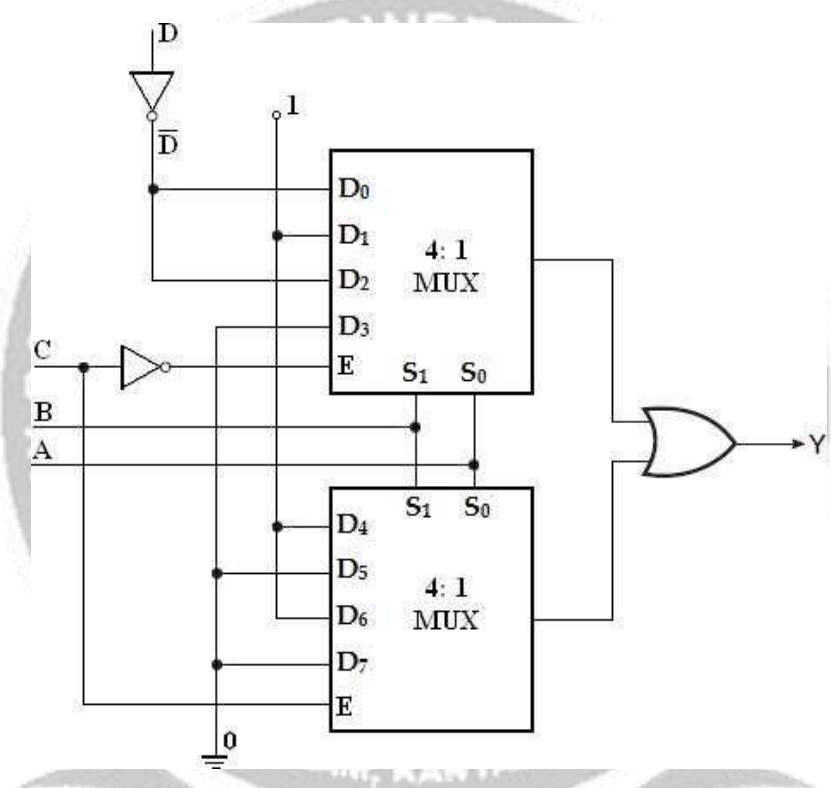
Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	$\overline{D}$	**1**	$\overline{D}$	**0**	**1**	**0**	**1**	**0**

Multiplexer Implementation (Using 8: 1 MUX):



Using 4: 1 MUX:



6. F (A, B, C, D) = ∑m (1, 3, 4, 11, 12, 13, 14, 15)

Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

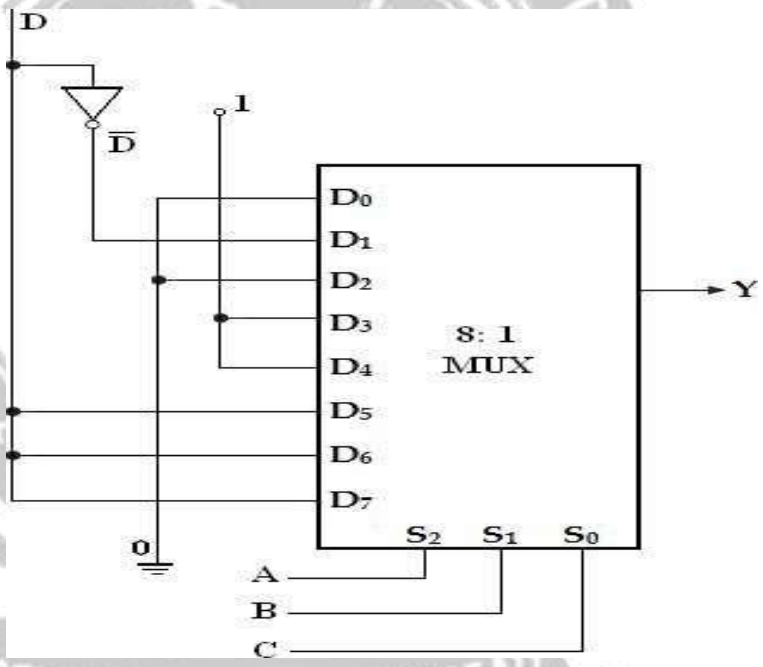
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	(1)	2	(3)	(4)	5	6	7
D	8	9	10	(11)	(12)	(13)	(14)	(15)
	0	$\overline{D}$	**0**	**1**	**1**	D	D	D

Multiplexer Implementation:



7. Implement the Boolean function using 8: 1 multiplexer.

F (A, B, C, D) = A'BD' + ACD + B'CD + A'C'D.

Solution:

Convert into standard SOP form,

= A‘BD‘ (C‘+C) + ACD (B‘+B) + B‘CD (A‘+A) + A‘C‘D (B‘+B)

= A‘BC‘D‘ + A‘BCD‘+ AB‘CD+ ABCD +A‘B‘CD + AB‘CD+A‘B‘C‘D+ A‘BC‘D

= A‘BC‘D‘ + A‘BCD‘+ AB‘CD + ABCD +A‘B‘CD +A‘B‘C‘D+ A‘BC‘D

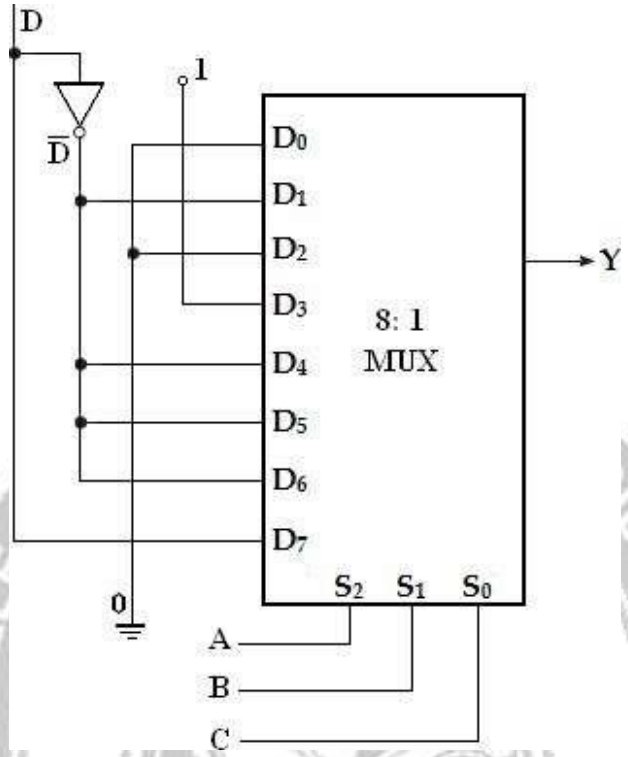
= m4+ m6+ m11+ m15+ m3+ m1+ m5

= ∑m (1, 3, 4, 5, 6, 11, 15)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	(1)	2	(3)	(4)	(5)	(6)	7
D	8	9	10	(11)	12	13	14	(15)
	0	$\overline{D}$	**0**	**1**	$\overline{D}$	$\overline{D}$	$\overline{D}$	D

Multiplexer Implementation:



8. Implement the Boolean function using 8: 1 multiplexer.

F (A, B, C, D) = AB'D + A'C'D + B'CD' + AC'D.

Solution:

Convert into standard SOP form,

= AB‘D (C‘+C) + A‘C‘D (B‘+B) + B‘CD‘ (A‘+A) + AC‘D (B‘+B)

= AB‘C‘D+ AB‘CD+ A‘B‘C‘D + A‘BC‘D +A‘B‘CD‘ + AB‘CD‘ +AB‘C‘D+ ABC‘D= AB‘C‘D

+ AB‘CD+ A‘B‘C‘D + A‘BC‘D +A‘B‘CD‘ + AB‘CD‘+ ABC‘D = m9+

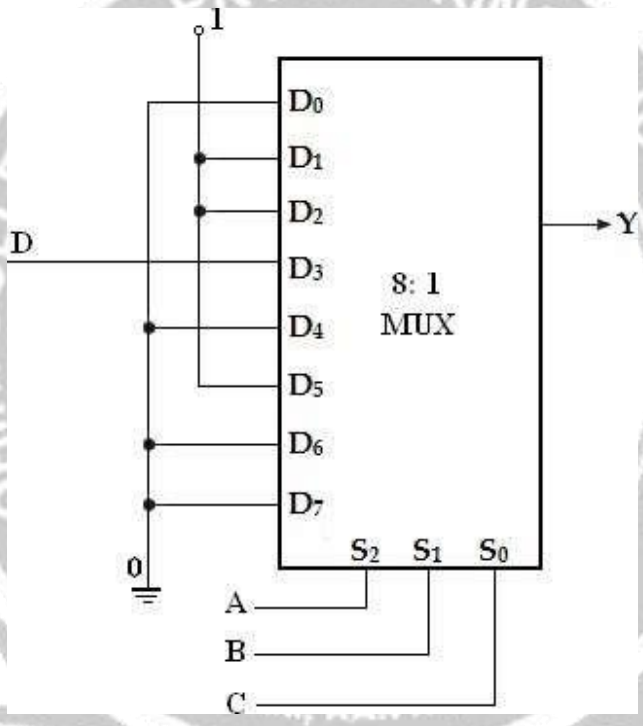
m11+ m1+ m5+m2+ m10+ m13

= ∑m (1, 2, 5, 9, 10, 11, 13).

Implementation Table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	(1)	(2)	3	4	(5)	6	7
D	8	(9)	(10)	(11)	12	(13)	14	15
	0	**1**	**1**	D	**0**	**1**	**0**	**0**

Multiplexer Implementation:



9. Implement the Boolean function using 8: 1 and also using 4:1 multiplexer **F (w, x, y,z) = ∑m (1, 2, 3, 6, 7, 8, 11, 12, 14)**

Solution:

Variables, n= 4 (w, x, y, z)

Select lines= n-1 = 3 (**S_2, S_1, S_0**)

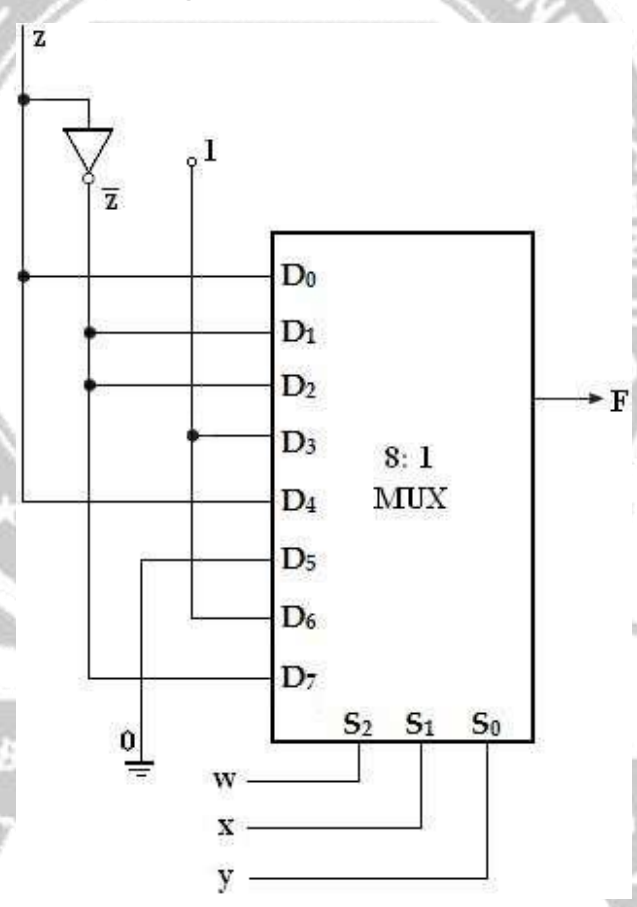
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

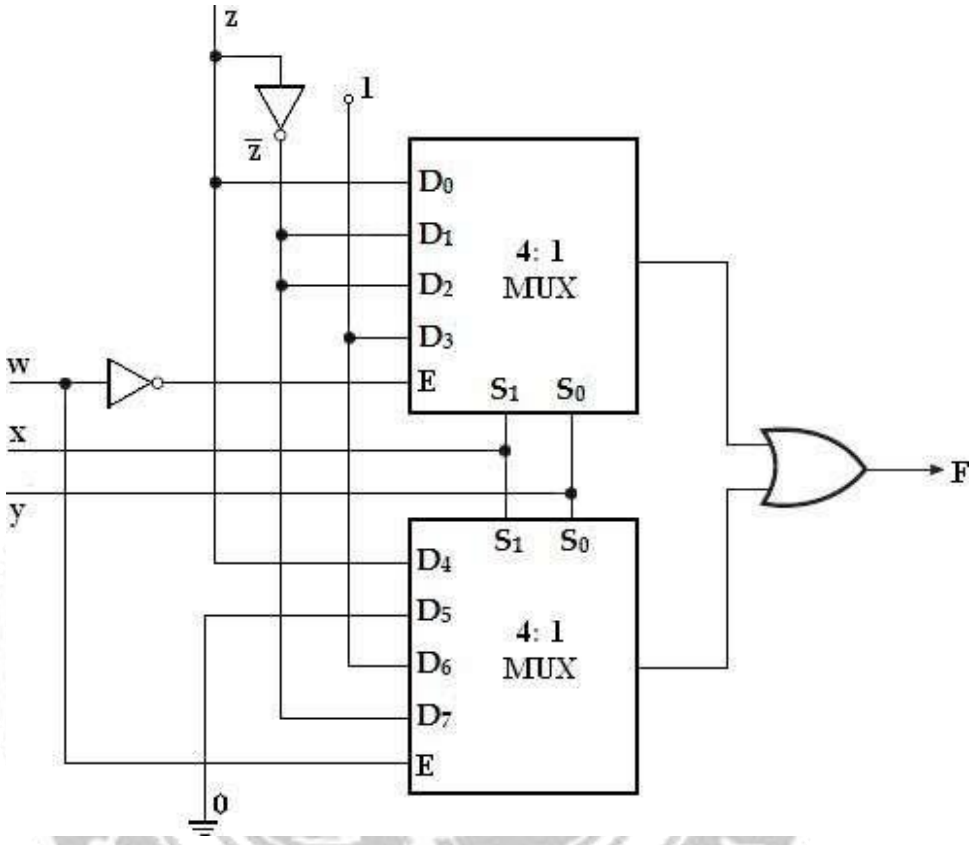
Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{z}$	0	(1)	(2)	(3)	4	5	(6)	(7)
z	(8)	9	10	(11)	(12)	13	(14)	15
	z	$\bar{z}$	$\bar{z}$	**1**	z	**0**	**1**	$\bar{z}$

Multiplexer Implementation (Using 8:1 MUX):



(Using 4:1 MUX):



10. Implement the Boolean function using 8: 1 multiplexer

F(A, B, C, D) = ∏m (0, 3, 5, 8, 9, 10, 12, 14) Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

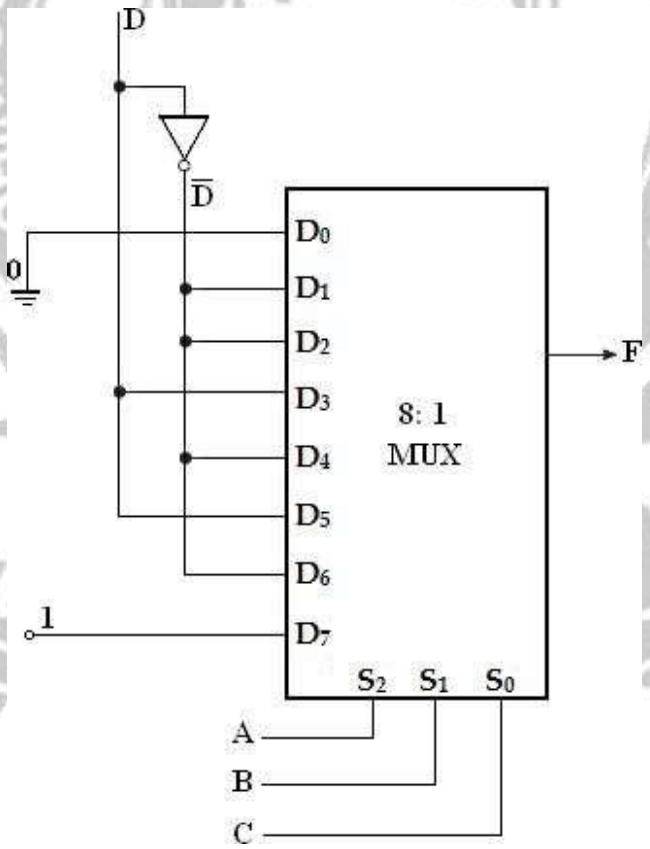
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	(1)	(2)	3	(4)	5	(6)	(7)
D	8	9	10	(11)	12	(13)	14	(15)
	0	$\overline{D}$	$\overline{D}$	D	$\overline{D}$	D	$\overline{D}$	**1**

Multiplexer Implementation:



11. Implement the Boolean function using 8: 1 multiplexer

F(A, B, C, D) = ∑m (0, 2, 6, 10, 11, 12, 13) + d (3, 8, 14)

Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

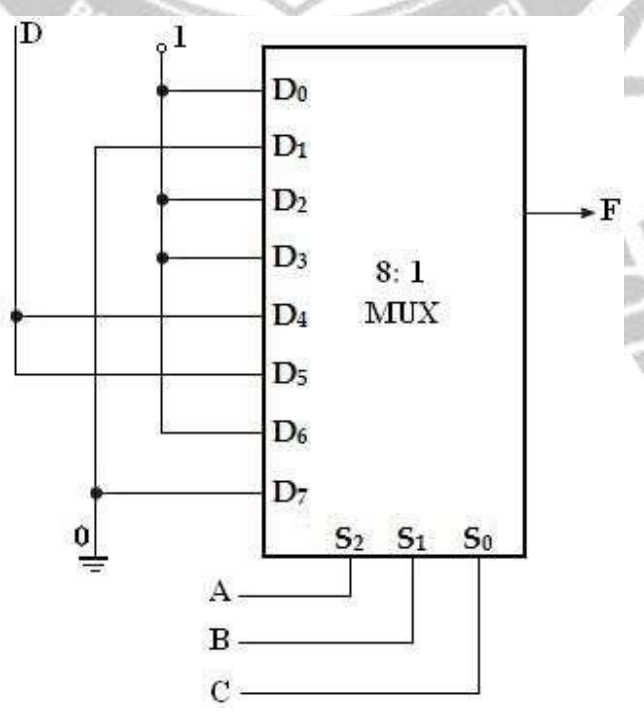
2^{n-1} to MUX i.e., 2^3 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

ImplementationTable:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	1	2	3	4	5	6	7
D	8	9	10	11	12	13	14	15
	1	**0**	**1**	**1**	**D**	**D**	**1**	**0**

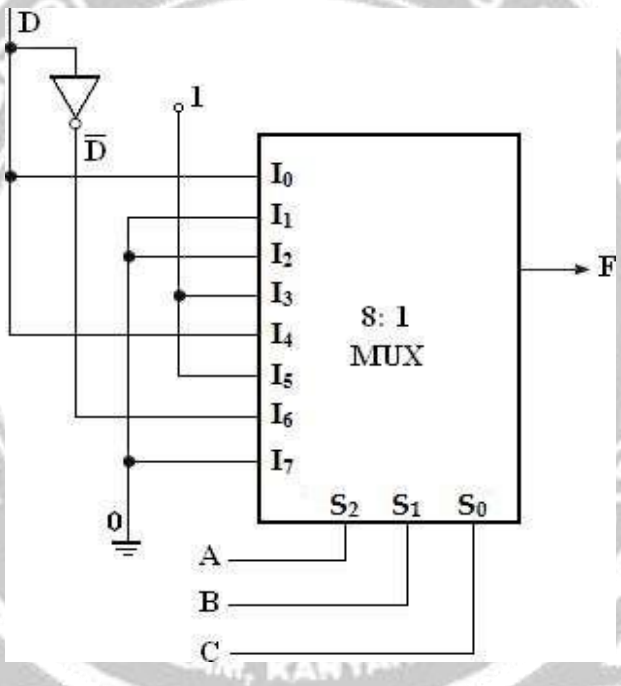
Multiplexer Implementation:



12. An 8×1 multiplexer has inputs A, B and C connected to the selection inputs S_2, S_1, and S_0 respectively. The data inputs I_0 to I_7 are as follows **$I_1=I_2=I_7= 0$; $I_3=I_5= 1$; $I_0=I_4= D$** and **$I_6= D'$.**

Determine the Boolean function that the multiplexer implements.

Multiplexer Implementation:



Implementation table:

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{D}$	0	1	2	(3)	4	(5)	(6)	7
D	(8)	9	10	(11)	(12)	(13)	14	15
	D	0	0	1	D	1	$\overline{D}$	0

F (A, B, C, D) = ∑m (3, 5, 6, 8, 11, 12, 13).

DEMULTIPLEXER:

Demultiplex means one into many. Demultiplexing is the process of taking information from one input and transmitting the same over one of several outputs.

A demultiplexer is a combinational logic circuit that receives information on a single input and transmits the same information over one of several (2n) output lines.

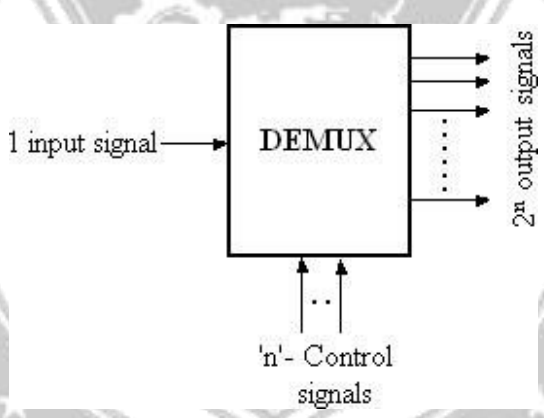


Fig : 2.34 - **Block diagram of demultiplexer**

The block diagram of a demultiplexer which is opposite to a multiplexer in its operation is shown above. The circuit has one input signal, _n' select signals and 2^n output signals. The select inputs determine to which output the data input will be connected. As the serial data is changed to parallel data, i.e., the input caused to appear on one of the n output lines, the demultiplexer is also called a ***—data distributer‖*** or a ***—serial-to- parallel converter‖*** .

1-to-4 Demultiplexer:

A 1-to-4 demultiplexer has a single input, D_{in}, four outputs (Y_0 to Y_3) and two select inputs (S_1 and S_0).

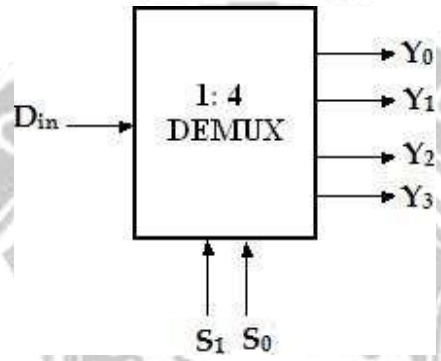


Fig : 2.35 - Logic Symbol

The input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines. The truth table of the 1-to-4 demultiplexer is shown below. Truth table of 1-to-4 demultiplexer

Enable	S1	S0	Din	Y0	Y1	Y2	Y3
0	x	x	x	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

From the truth table, it is clear that the data input, D_{in} is connected to the output Y_0, when S_1= 0 and S_0= 0 and the data input is connected to output Y_1 when S_1= 0 and S_0= 1. Similarly, the data input is connected to output Y_2 and Y_3 when S_1= 1 and S_0= 0 and when S_1= 1 and S_0= 1, respectively. Also, from the truth table, the expression for outputs can be written as follows,

Y0=

S1'S0'Din

Y1=

S1'S0Din

Y2=

S1S0'Din

Y3=

S1S0Din

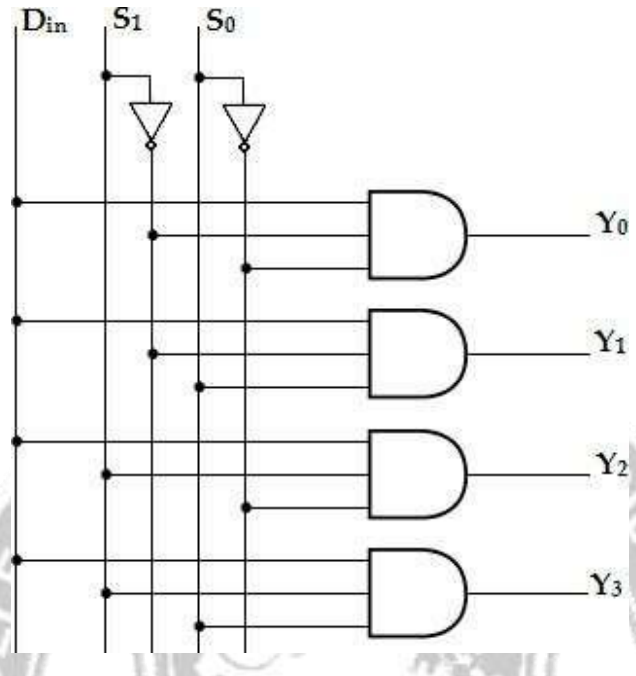


Fig : 2.36 - Logic diagram of 1-to-4 demultiplexer

Now, using the above expressions, a 1-to-4 demultiplexer can be implemented using four 3-input AND gates and two NOT gates. Here, the input data line D_{in}, is connected to all the AND gates. The two select lines S_1, S_0 enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.

1-to-8 Demultiplexer:

A 1-to-8 demultiplexer has a single input, **D_{in}**, eight outputs (**Y_0 to Y_7**) and three select inputs (**S_2, S_1 and S_0**). It distributes one input line to eight output lines based on the select inputs. The truth table of 1-to-8 demultiplexer is shown below.

Din	S2	S1	S0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Truth table of 1-to-8 demultiplexer

From the above truth table, it is clear that the data input is connected with one of the eight outputs based on the select inputs. Now from this truth table, the expression for eight outputs can be written as follows:

Y0= S2‘S1‘S0‘Din Y4= S2 S1‘S0‘Din Y1= S2‘S1‘S0Din

Y5= S2 S1‘S0Din

Y2= S2‘S1S0‘Din Y6= S2 S1S0‘Din Y3= S2‘S1S0Din Y7= S2S1S0Din

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can be drawn as shown below. Here, the single data line, D_{in} is connected to all the eight AND gates, but only one of the eight AND gates will be enabled by the select input lines. For example, if $S_2S_1S_0$= 000, then only AND gate-0 will be enabled and thereby the data input, D_{in} will appear at Y_0. Similarly, the different combinations of the select inputs, the input D_{in} will appear at the respective output.

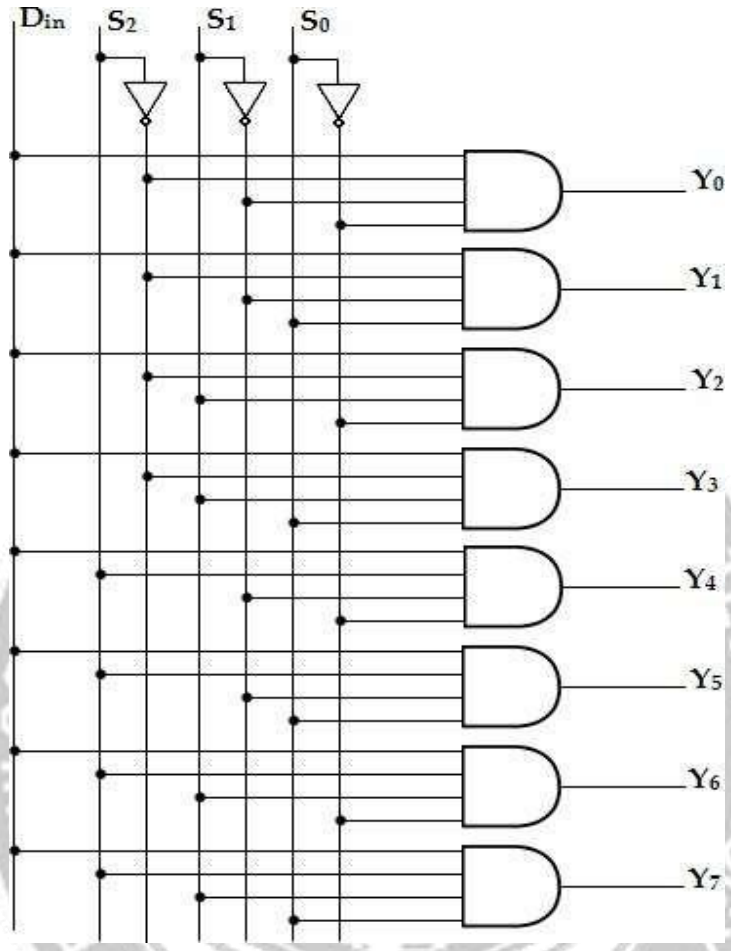
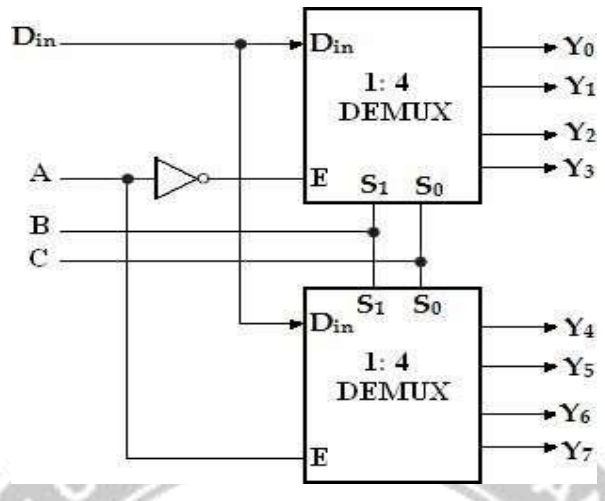


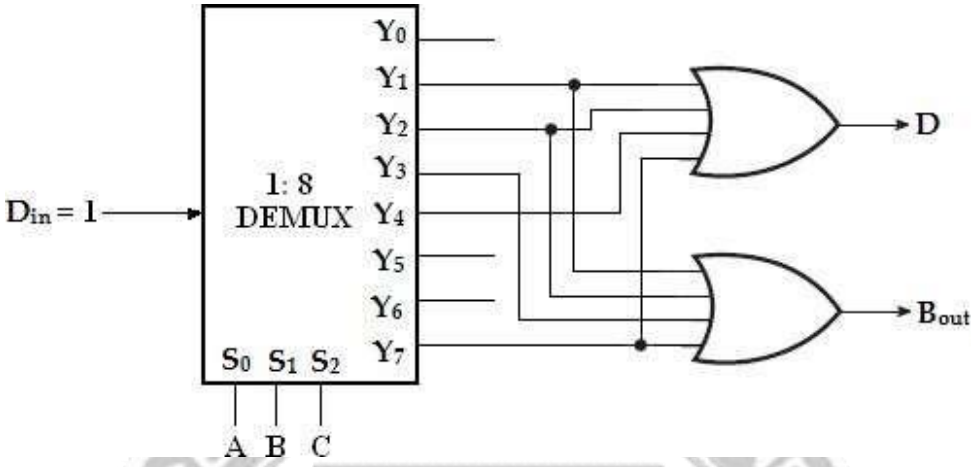
Fig : 2.37 - Logic diagram of 1-to-8 demultiplexer

1. Design 1:8 demultiplexer using two 1:4 DEMUX.



2. Implement full subtractor using demultiplexer.

Inputs			Outputs	
A	**B**	**Bin**	**Difference(D)**	**Borrow(B_{out})**
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


Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
1: 8
DEMUX
Din = 1
S0 S1 S2
A B C
D
Bout