MODULE - 2

CHAPTER - 1

DATA PROCESSING CIRCUITS

MULTIPLEXERS -

Multiplexer is a circuit with many inputs but only one output. By applying control signals, we can steer any input to the output. Thus, its also called a data selector. and the control inputs are termed as select inputs.

The figure below illustrates the general idea. (fig a).

The circuit has n input signals, m control signals and 1 output signal. Note that, m control signals can select at the most 2^m input signals thus, $n \leq 2^m$.

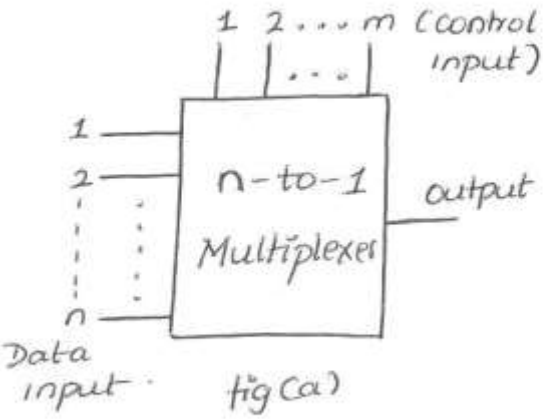


fig (a)

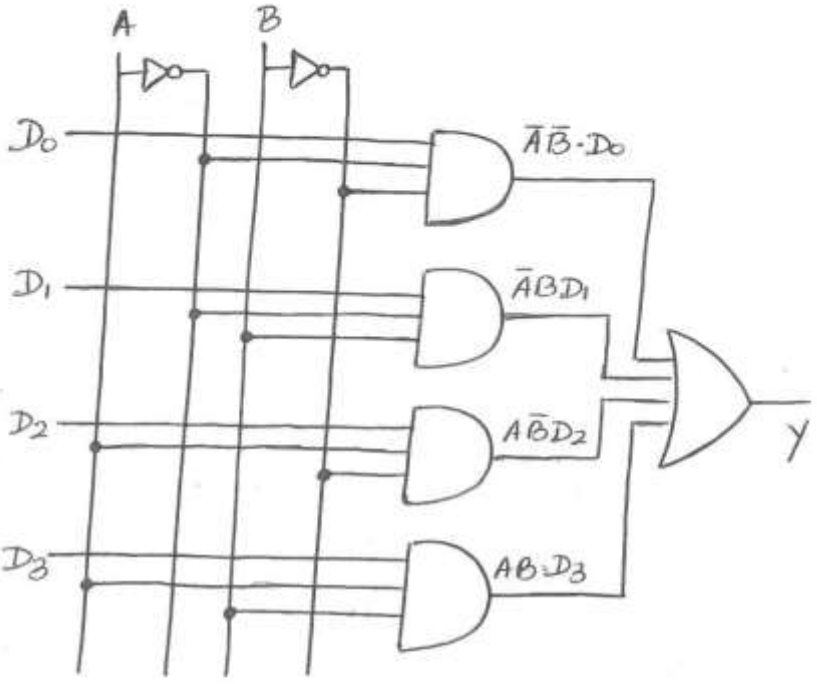


fig (c)

A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

fig (b)

fig (a) Multiplexer block diagram

fig (b) 4-to-1 multiplexer truth table

fig (c) its logic circuit.

The circuit diagram of a 4-to-1 multiplexer is shown in fig (c), and its truth table in fig (b).

Depending on the inputs A, B one of the four inputs D_0 to D_3 is steered to the output Y.

The logic eqn of this 4-to-1 mux will give a SOP representation each AND gate generating a product term, which finally are summed by OR gate.

Thus,

If $A=0$, $B=0$.

$$Y = A'B'D_0 + A'BD_1 + AB'D_2 + ABD_3$$

$$Y = 0'0'D_0 + 0'0D_1 + 00'D_2 + 00D_3$$

$$Y = 1\cdot1D_0 + 1\cdot0D_1 + 0\cdot1D_2 + 0\cdot0D_3$$

$$Y = D_0$$

In other words, for $AB=00$, the first AND gate to which D_0 is connected remains active & equal to D_0 and all other AND gates are inactive with output held at logic 0.

Thus, multiplexer output Y is same as D_0.

If $D_0=0$, $Y=0$. and if $D_0=1$, $Y=1$.

Similarly, for $AB=01$, the second AND gate will be active and all other AND gates remain inactive.

Thus $Y=D_1$. Following the same procedure we can complete the truth table of Fig(b).

16-to-1 Multiplexer.

The figure below shows a 16-to-1 multiplexer. The input bits are labeled D_0 to D_{15}. Only one of these is transmitted to the output. But which one depends on the value of ABCD, the control input. For eg, when

$$ABCD = 0000$$

the upper AND gate is enabled while all other AND gates are disabled. Therefore, data bit D_0 is transmitted to the output, giving $Y=D_0$.

If D_0 is low, Y is low; if D_0 is high, Y is also high. The point is that Y depends only on the value of D_0. If the control nibble is changed to $ABCD = 1111$

all gates are disabled except the bottom AND gate. In this case D_{15} is the only bit transmitted to the output and

thus $Y=D_{15}$.

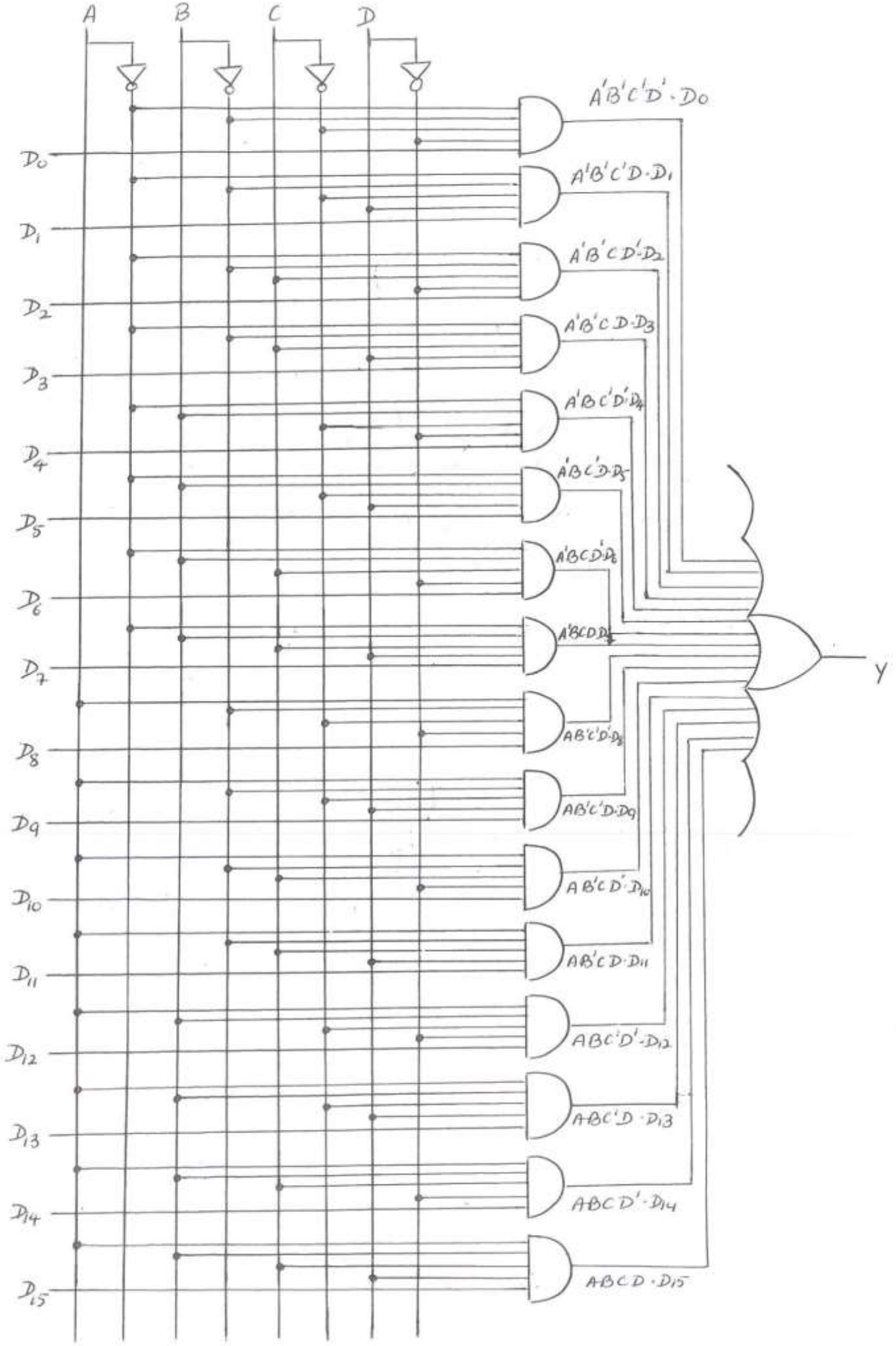


Fig: 16-to-1 Multiplexer.

As you can see, the control nibble determines which of the input data bit is transmitted to the output.

Thus we can write the output as,

$$Y = A'B'C'D' \cdot D_0 + A'B'C'D \cdot D_1 + A'B'C D' \cdot D_2 + \cdots + ABCD' \cdot D_{14} + ABCD \cdot D_{15}$$

The 74150

Try to visualize the 16-input OR gate of the above fig changed to a NOR gate. what effect does this have on the operation of the circuit? All that happens is we get the complement of the selected data bit rather than the data bit itself.

for eg, when ABCD = 0111, the output is

$$Y = \overline{D_7}$$

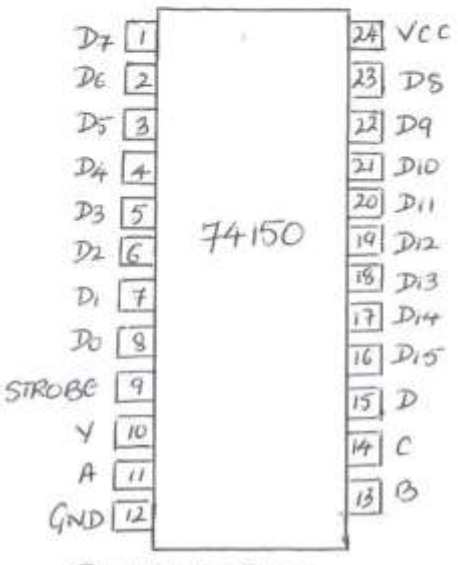
This is the boolean eqn for a typical transistor-transistor logic (TTL) multiplexer bcoz it has an inverter on the output that produces the complement of the selected data bit.

The 74150 is a 16-to-1 TTL multiplexer with the pin diagram shown below. Pins 1 to 8 and 16 to 23 are for the input data bits D_0 to D_{15}. Pins 11, 13, 14, 15 are for the control bits ABCD. Pin 10 is the output; and it equals the complement of the selected data bit. Pin 9 is for the STROBE, an input signal that disables or enables the multiplexer.

A low strobe enables the multiplexer, so that output Y equals the complement of the input data bit.

$$Y = \overline{D_n}$$ where n is the decimal equivalent of ABCD

on the other hand, a high strobe disables the multiplexer & forces the output into the high state. with a high strobe, the value of ABCD doesn't matter.



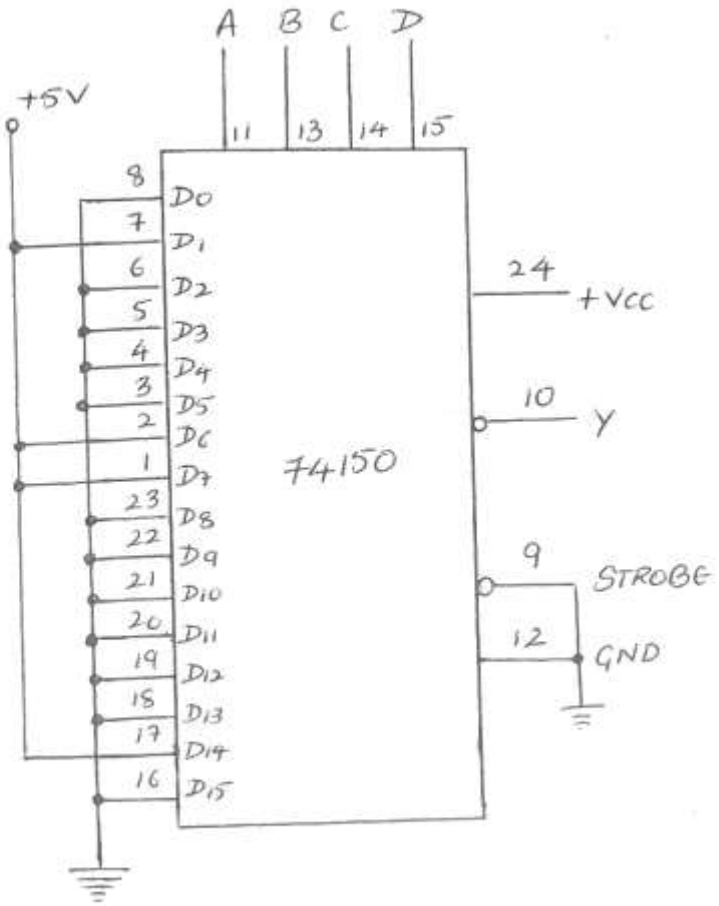
PIN DIAGRAM

Strobe	A	B	C	D	Y
L	L	L	L	L	$\overline{D_0}$
L	L	L	L	H	$\overline{D_1}$
L	L	L	H	L	$\overline{D_2}$
L	L	L	H	H	$\overline{D_3}$
L	L	H	L	L	$\overline{D_4}$
L	L	H	L	H	$\overline{D_5}$
L	L	H	H	L	$\overline{D_6}$
L	L	H	H	H	$\overline{D_7}$
L	H	L	L	L	$\overline{D_8}$
L	H	L	L	H	$\overline{D_9}$
L	H	L	H	L	$\overline{D_{10}}$
L	H	L	H	H	$\overline{D_{11}}$
L	H	H	L	L	$\overline{D_{12}}$
L	H	H	L	H	$\overline{D_{13}}$
L	H	H	H	L	$\overline{D_{14}}$
L	H	H	H	H	$\overline{D_{15}}$
H	X	X	X	X	H

Truth table.

* Implement the Truth Table given below using a 74150 IC ⑤

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



The fig is the multiplexer design soln.

When ABCD = 0000, D_0 is the selected input. In the fig - Since D_0 is Low, Y is high.

When ABCD = 0001, D_1 is selected. Since D_1 is high, Y is low

and so on

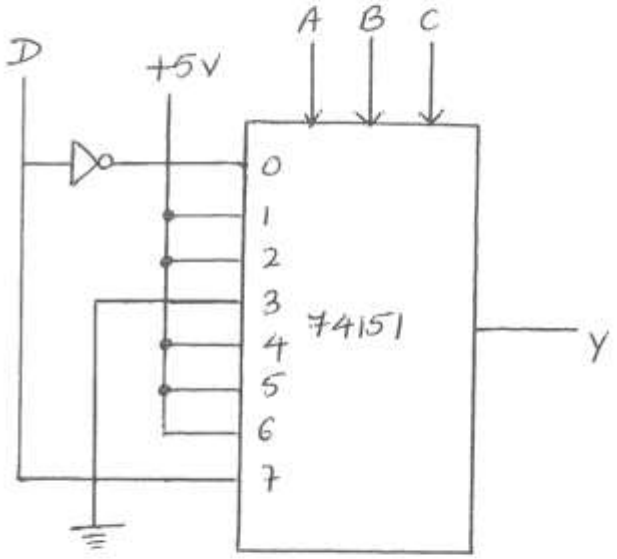
Universal Logic Circuit -

Multiplexer is sometimes called as universal logic circuit bcoz a 2^n-to-1 multiplexer can be used as a design soln for any n variable truth table.

* Implement the following eqn using a 8:1 mux

$F(A,B,C,D) = \Sigma m(0,2,3,4,5,8,9,10,11,12,13,15)$

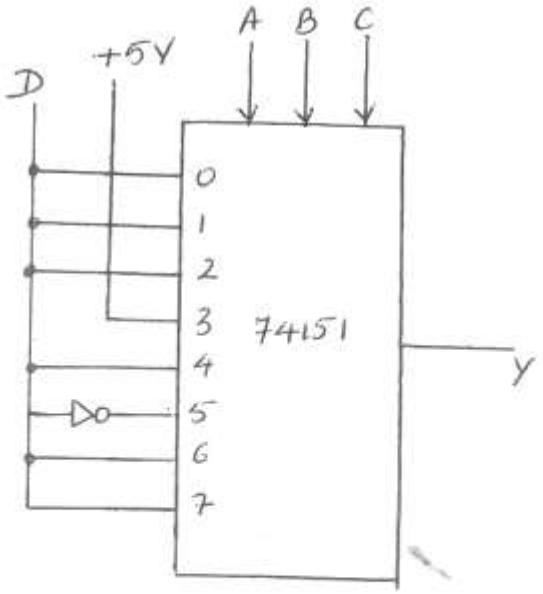
A	B	C	D	Y	
0	0	0	0	1	$\bar{D} = D_0$
0	0	0	1	0	
0	0	1	0	1	$1 = D_1$
0	0	1	1	1	
0	1	0	0	1	$1 = D_2$
0	1	0	1	1	
0	1	1	0	0	$0 = D_3$
0	1	1	1	0	
1	0	0	0	1	$1 = D_4$
1	0	0	1	1	
1	0	1	0	1	$1 = D_5$
1	0	1	1	1	
1	1	0	0	1	$1 = D_6$
1	1	0	1	1	
1	1	1	0	0	$D = D_7$
1	1	1	1	1	



* Implement the following eqn using a 8:1 mux

$F(A,B,C,D) = \Sigma m(1,3,5,6,7,9,10,13,15)$

A	B	C	D	Y	
0	0	0	0	0	$D = D_0$
0	0	0	1	1	
0	0	1	0	0	$D = D_1$
0	0	1	1	1	
0	1	0	0	0	$D = D_2$
0	1	0	1	1	
0	1	1	0	1	$1 = D_3$
0	1	1	1	1	
1	0	0	0	0	$D = D_4$
1	0	0	1	1	
1	0	1	0	1	$\bar{D} = D_5$
1	0	1	1	0	
1	1	0	0	0	$D = D_6$
1	1	0	1	1	
1	1	1	0	0	$D = D_7$
1	1	1	1	1	



NIBBLE MULTIPLEXERS -

Sometimes we want to select one of the two input nibbles. In this case, we can use a nibble multiplexer like the one shown in the fig below.

The input nibble on the left is $A_3A_2A_1A_0$ and the one on the right is $B_3B_2B_1B_0$.

The control signal labeled SELECT determines which input nibble is transmitted to the output. When SELECT is LOW, the four NAND gates on the left are activated; therefore,

$$Y_3Y_2Y_1Y_0 = A_3A_2A_1A_0$$

When SELECT is high, the four NAND gates on the right are active, and $Y_3Y_2Y_1Y_0 = B_3B_2B_1B_0$.

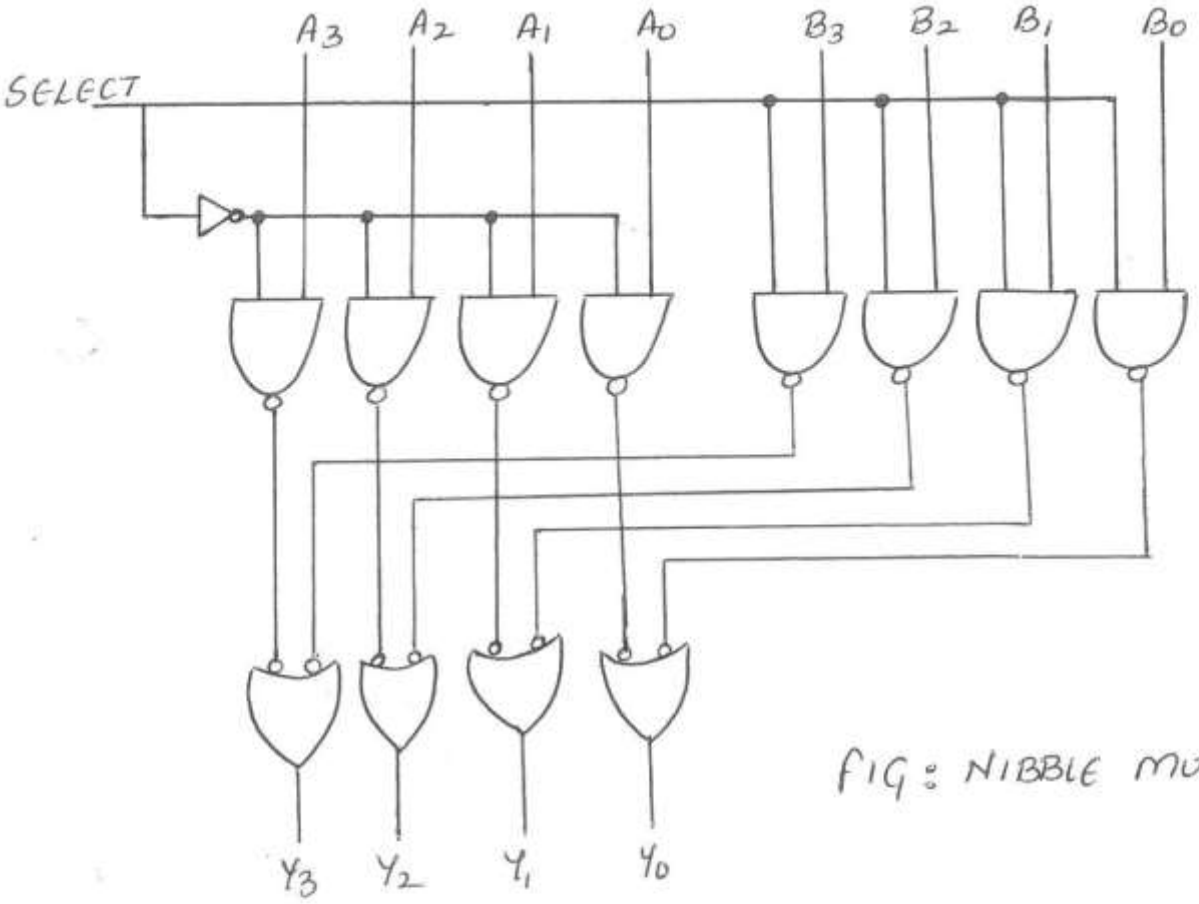
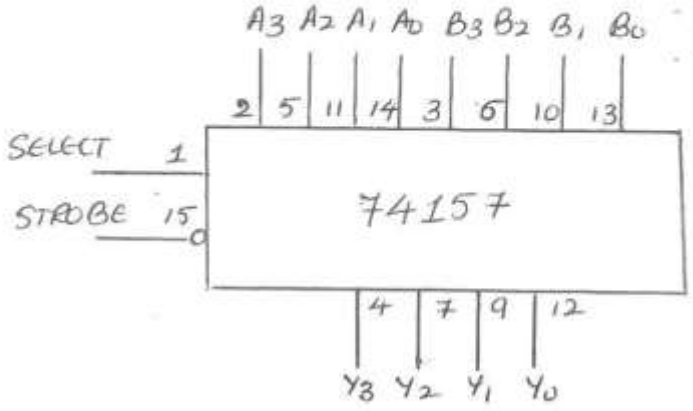


FIG: NIBBLE MULTIPLEXER.



PIN DIAGRAM of 74157

PROBLEMS -

① Show how 4-to-1 multiplexer can be obtained using only 2-to-1 multiplexer.

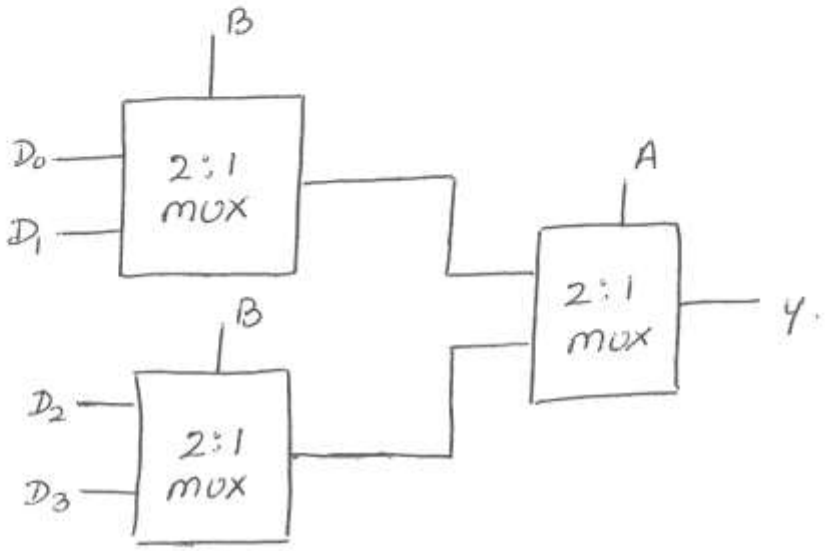
Logic eqn of 2-to-1 multiplexer :

$$Y = A'D_0 + AD_1$$

Logic eqn of 4-to-1 multiplexer :

$$Y = A'B'D_0 + A'BD_1 + AB'D_2 + ABD_3$$

can be rewritten as $Y = A'(B'\cdot D_0 + BD_1) + A(B'\cdot D_2 + BD_3)$.



② Realize $Y = A'B + B'C' + ABC$ using an 8-to-1 multiplexer.

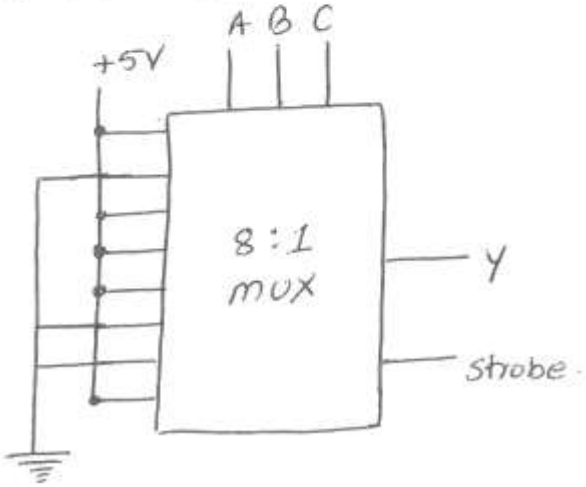
$$Y = A'B + B'C' + ABC.$$

$$Y = A'B(C'+C) + (A'+A)B'C' + ABC$$

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

Comparing this with the eqn of 8 to 1 multiplexer, we find $D_0 = D_2 = D_3 = D_4 = D_7 = 1$ and

$$D_1 = D_5 = D_6 = 0.$$



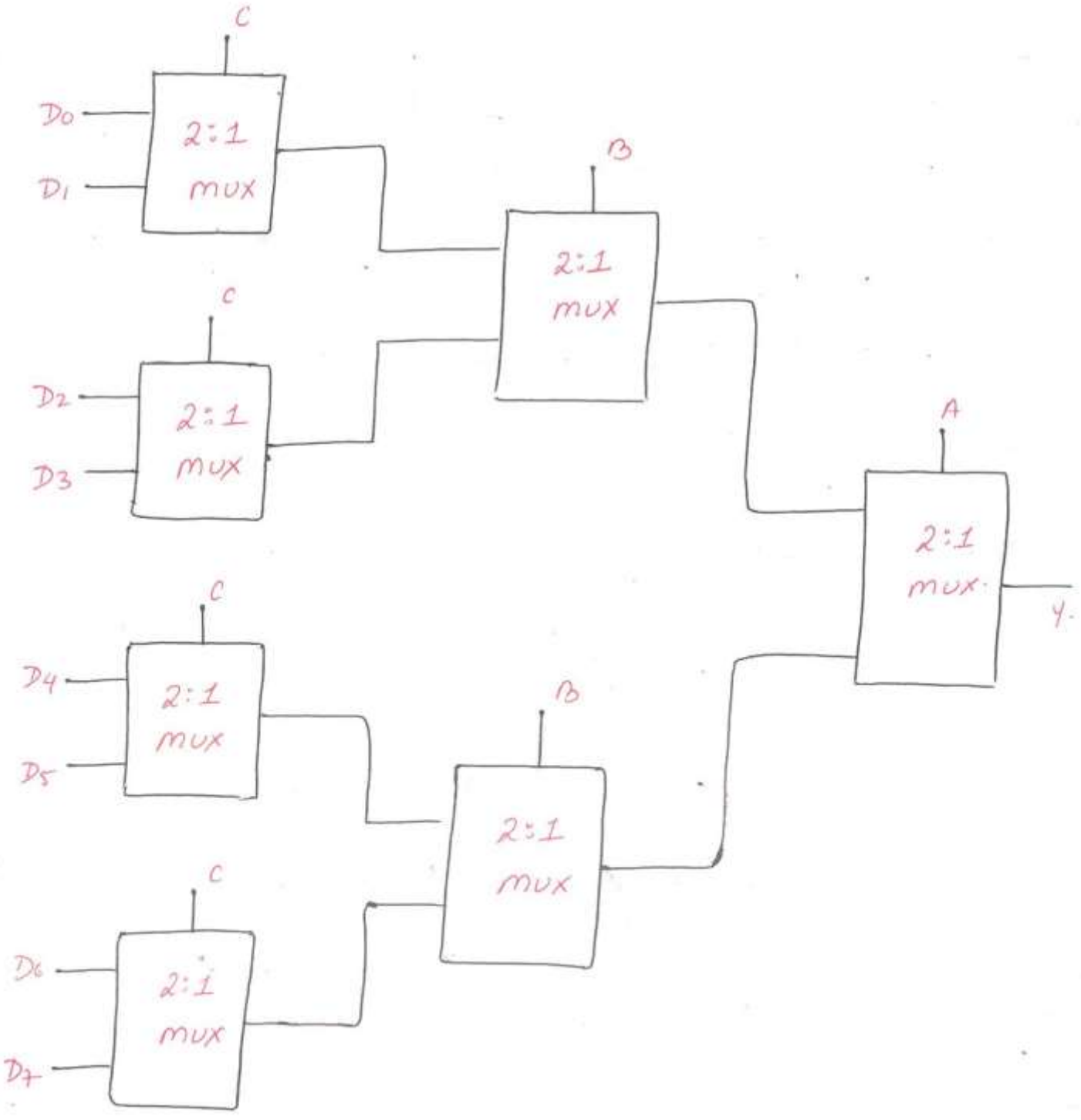
③ Show how 8-to-1 multiplexer can be obtained using only 2-to-1 multiplexer.

Logic eqn of 8-to-1 mux:

$$Y = A'B'C'D_0 + A'B'C D_1 + A'BC'D_2 + A'BCD_3 + AB'C'D_4 + AB'C D_5 + ABC'D_6 + ABCD_7.$$

$$Y = A'(B'C'D_0 + B'C D_1 + BC'D_2 + BCD_3) + A(B'C'D_4 + B'C D_5 + BC'D_6 + BCD_7)$$

$$Y = A'(B'(C'D_0 + CD_1) + B(C'D_2 + CD_3)) + A(B'(C'D_4 + CD_5) + BC'D_6 + CD_7))$$

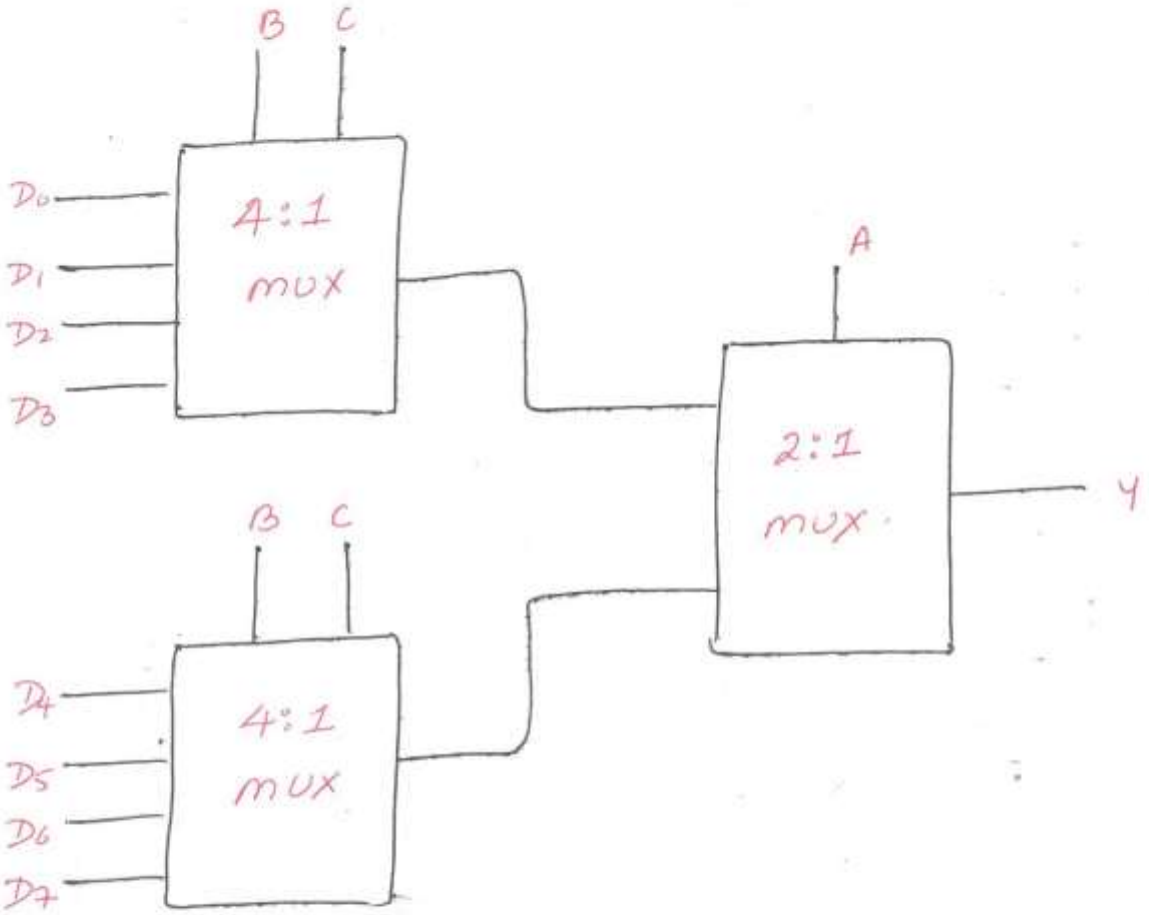


④ Show how 8-to-1 multiplexer can be obtained using 4 to 1 multiplexer and 2:1 multiplexer.

Logic eqn of 8 to 1 mux.

$$Y = A'B'C'D_0 + A'B'CD_1 + A'BC'D_2 + A'BCD_3 + AB'C'D_4 + AB'CD_5 + ABC'D_6 + ABCD_7.$$

$$Y = A'(B'C'D_0 + B'CD_1 + BC'D_2 + BCD_3) + A(B'C'D_4 + B'CD_5 + BC'D_6 + BCD_7)$$



ASSIGNMENT -

→ Design 16-to-1 mux using only 2 to 1 mux.

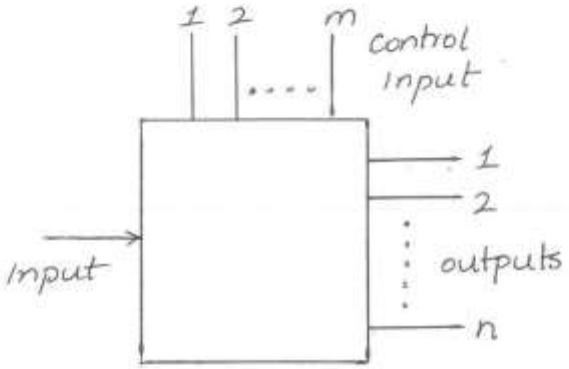
→ Design 16-to-1 mux using 4 to 1 mux and 2 to 1 mux

→ Design 16-to-1 mux using 8 to 1 mux and 2 to 1 mux

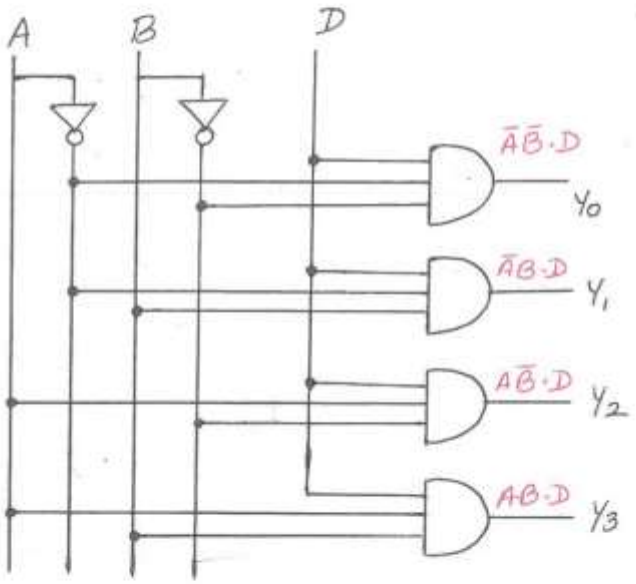
→ Design 32-to-1 mux using 8 to 1 mux and 4 to 1 mux.

DEMULTIPLEXERS-

Demultiplexer means one into many. A demultiplexer is a logic circuit with one input and many outputs. By applying the control signals, we can steer the input signal to one of the output lines. The figure below illustrates the general idea. The circuit has one input signal, m control or select signals and n output signals where $n \leq 2^m$.



(a) Demultiplexer Block Diagram



(b) Logic Circuit of 1-to-4 demultiplexer.

D	A	B	Output
D	0	0	$Y_0 = D$
D	0	1	$Y_1 = D$
D	1	0	$Y_2 = D$
D	1	1	$Y_3 = D$

(c) Truth Table of 1-to-4 Demultiplexer.

1-to-4 Demultiplexer-

The above figure (b) shows the circuit diagram of a 1-to-4 demultiplexer, that consists of single input D (data input) and two select inputs A and B and four outputs from Y_0 to Y_3.

From the table, the output logic can be expressed as min terms which is given below,

$$Y_0 = \bar{A}\bar{B}\cdot D$$
$$Y_1 = \bar{A}B\cdot D$$
$$Y_2 = A\bar{B}\cdot D$$
$$Y_3 = AB\cdot D$$

From the above Boolean expression, a 1-to-4 demultiplexer can be implemented by using four 3-input AND gates and two NOT gates, as shown in the above figure (b).

The two select lines enable the particular gate at a time. So, depending upon the combination of the select inputs, input data is passed through the selected gate to the associated output.

1 to 16 Demultiplexer

The figure below shows a 1-to-16 Demultiplexer. The input bit is labeled D. This data bit (D) is transmitted to the data bit of the output lines.

To which output line is the data bit transmitted from depends on the value of ABCD, the control inputs.

When ABCD = 0000, the upper AND gate is enabled, while all other AND gates are disabled.

Therefore, the data bit D is transmitted only to the Y_0 output, giving $Y_0 = D$. If D is low, Y_0 is also low. If D is high, Y_0 is high. All the other outputs are in the low state.

If the control nibble is changed to ABCD = 1111, all the AND gates are disabled except the bottom AND gate. Then, D is transmitted only to the Y_{15} output, and $Y_{15} = D$.

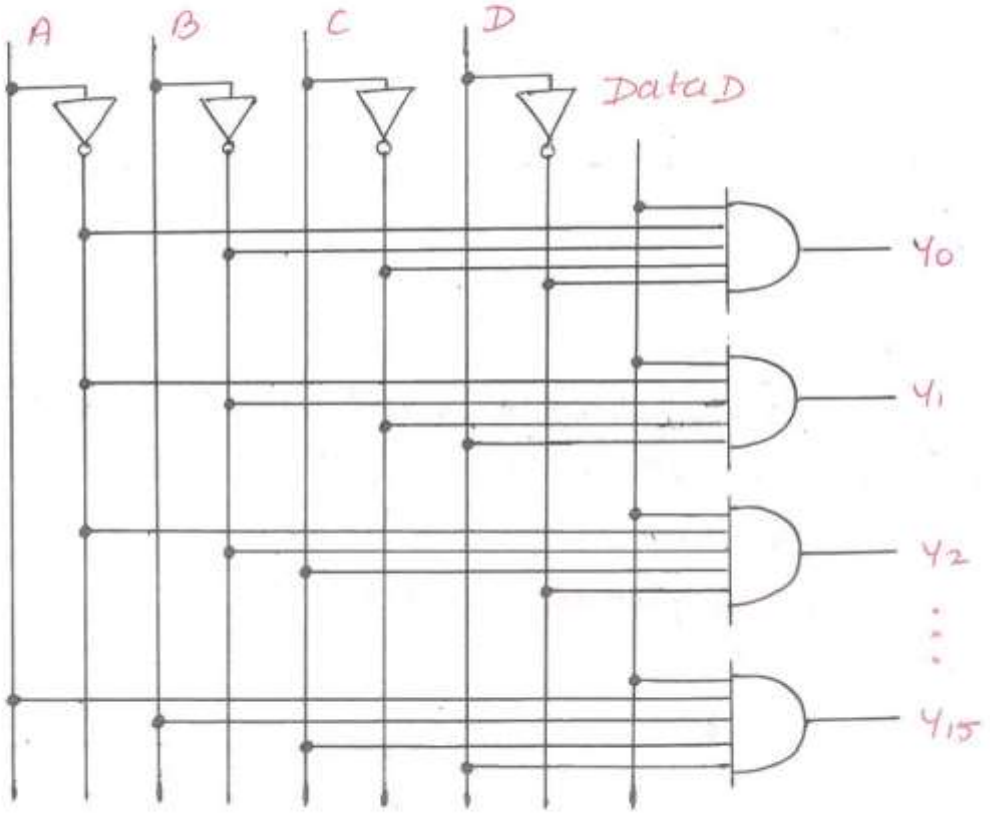
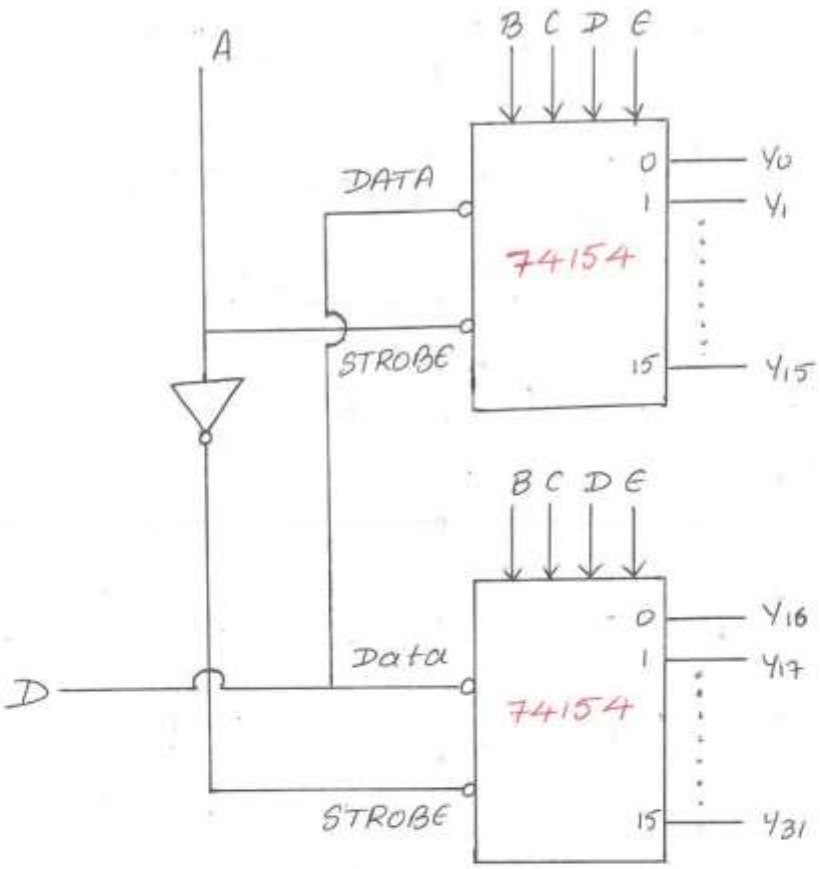


Fig: 1-to-16 Demultiplexer

PROBLEM -

* Show how two 1-to 16 demultiplexers can be connected to get a 1-to-32 demultiplexer.

Soln:



1-of-16 DECODER

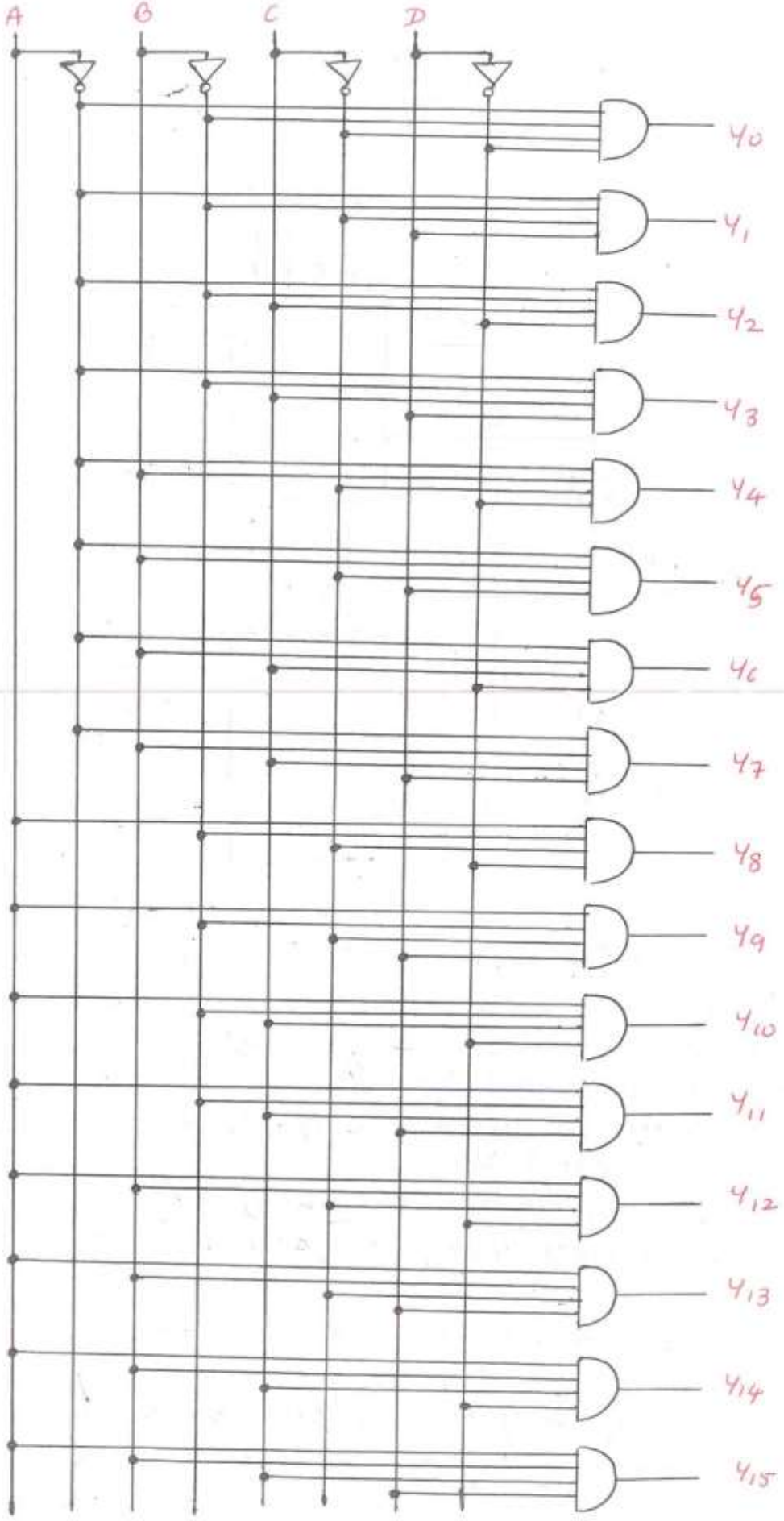
A decoder is similar to a demultiplexer, with one exception - there is no data input. The only inputs are the control bits ABCD, which are shown in the fig below.

This logic circuit is called a 1-of-16 decoder bcoz only 1 of the 16 output lines are high.

For example, when ABCD is 0001, only the Y_1 AND gate has all the output high. If ABCD changes to 0100, the output is generated through Y_4.

If you check the other ABCD possibilities (0000 to 1111), you will find that the subscript of the high output always equals the decimal equivalent of ABCD. For this reason, the circuit is also known as a binary-to-decimal decoder.

Because it has 4 input lines and 16 output lines, the circuit is also known as a 4-line to 16-line decoder.



1-of-16 Decoder

PROBLEMS-

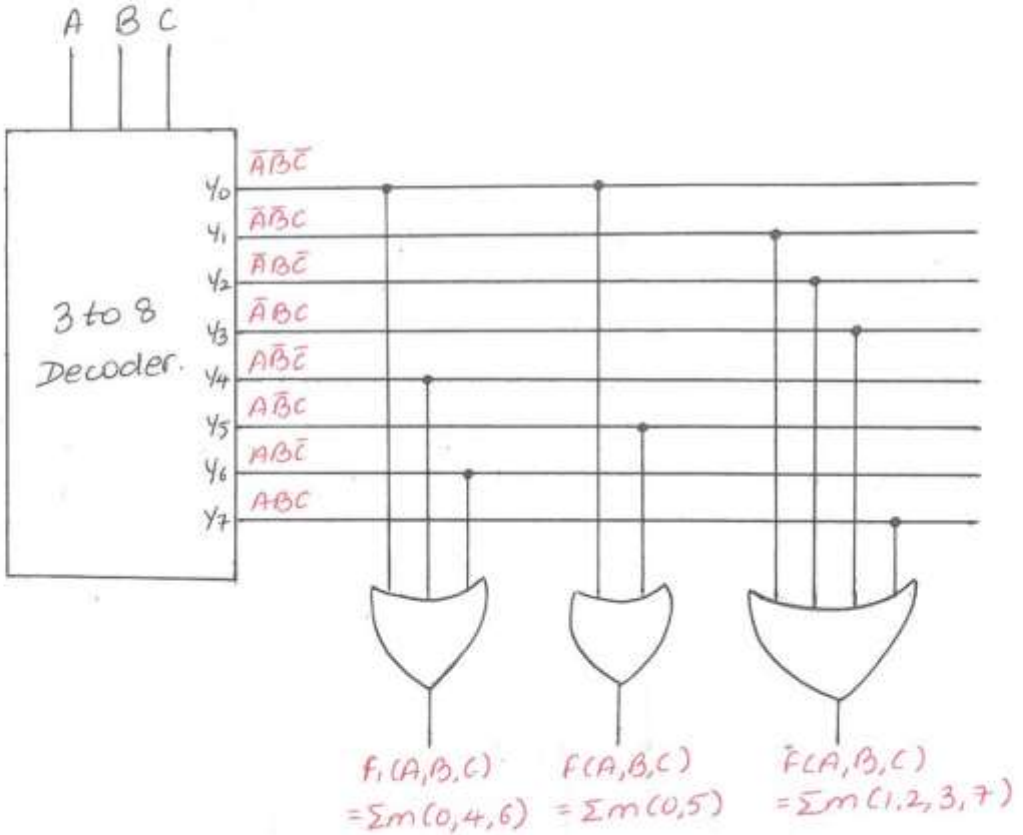
① Show how using a 3-to-8 decoder and multi-input OR gates, the following Boolean expressions can be realized simultaneously.

$F_1(A,B,C) = \Sigma m(0,4,6)$

$F_2(A,B,C) = \Sigma m(0,5)$

$F_3(A,B,C) = \Sigma m(1,2,3,7)$

Soln.



② Implement a Full Adder using 3-to-8 Decoder.

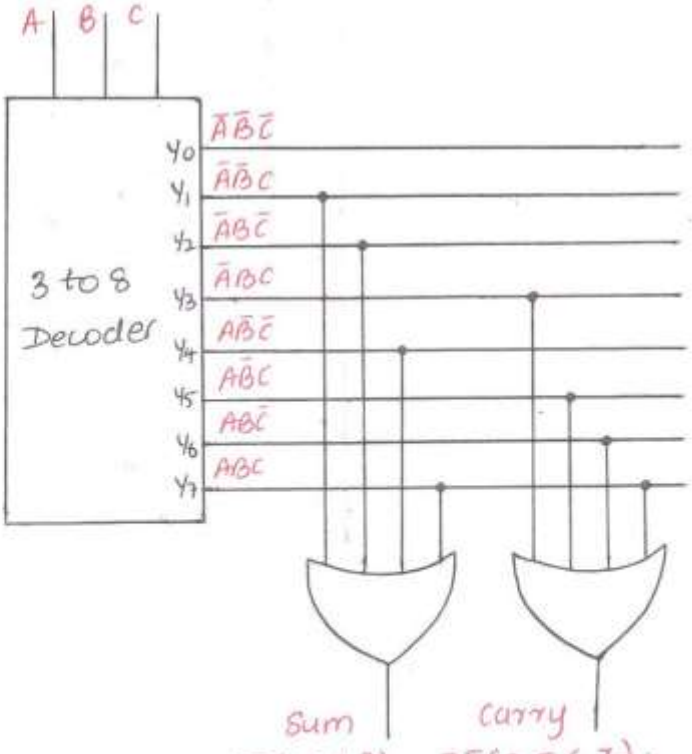
Soln:

A	B	C	Sum	Carry
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

Truth Table of a Full Adder.

$Sum = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$Carry = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$



BCD - to - Decimal Decoders

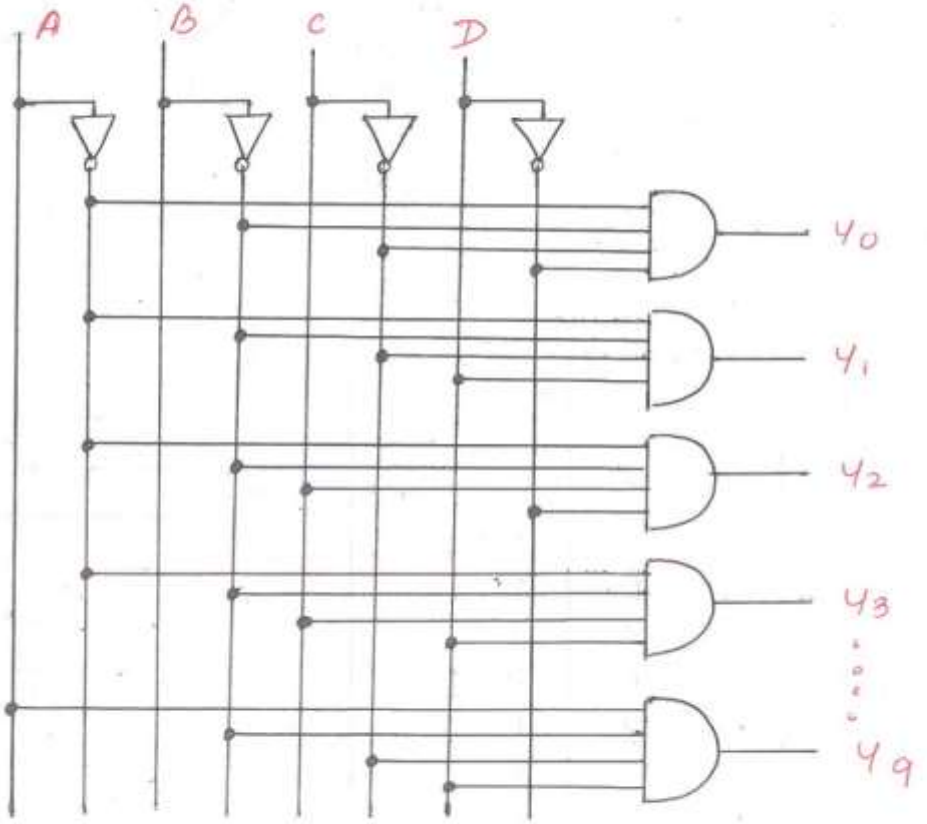
BCD is an abbreviation for "Binary Coded Decimal".
The BCD code expresses each digit in a decimal number by its nibble equivalent.
For eg, decimal number 429 is changed to its BCD form as follows:

4	2	9
↓	↓	↓
0100	0010	1001

To anyone using the BCD code, 0100 0010 1001 is equivalent to 429.
As another example, here is how to convert the decimal number 8963 to its BCD form.

8	9	6	3
↓	↓	↓	↓
1000	1001	0110	0011

Some early computers processed BCD numbers. This means that the decimal numbers were changed into BCD numbers, which the computer then added, subtracted, etc.
The final answer was converted from BCD back to decimal numbers.
The BCD digits are from 0000 to 1001. All combinations above this cannot exist in BCD code because the highest decimal digit being coded is 9.



1-of-10 Decoder.

The above circuit is called a 1-of-10 decoder because only 1 out of the 10 output lines is high.

For eg, when ABCD is 0011, only the Y3 AND gate has all high inputs; therefore, only the Y3 output is high.

If ABCD changes to 1000, only the Y8 AND gate has all high inputs; as a result, only the Y8 output goes high.

If you check the other ABCD possibilities (0000 to 1001), you will find that the subscript of the high output always equals the decimal equivalent of the input BCD digit. For this reason, the circuit is also called a BCD-to-Decimal Converter.

SEVEN-SEGMENT DECODERS-

A LED emits radiation when forward biased.

The fig (a) shows a seven segment indicator, ie seven LED's labelled 'a' through 'g'. By forward biasing different LED's we can display the digits 0 through 9.

For instance, to display a 0, we need to light up the segments a, b, c, d, e and f.

To light up 5, we need the segments a, c, d, f and g.

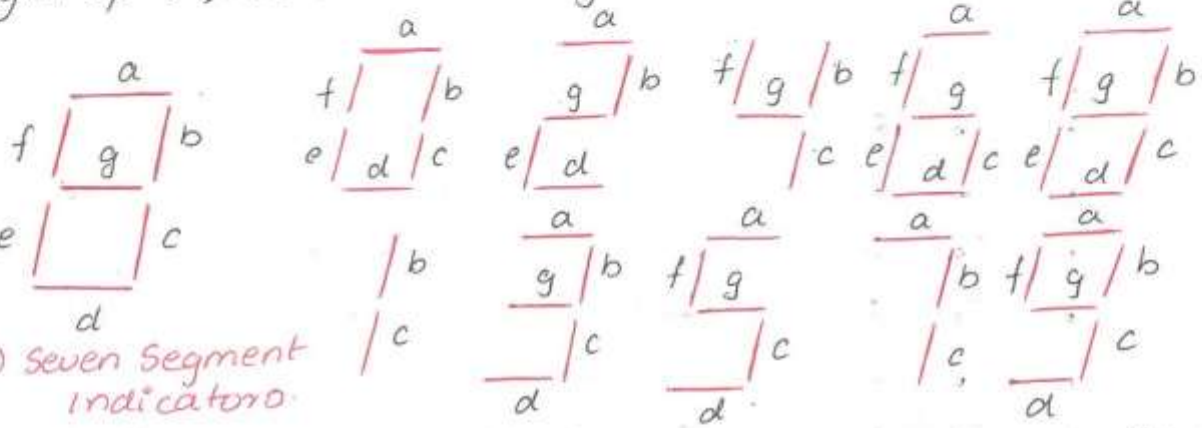
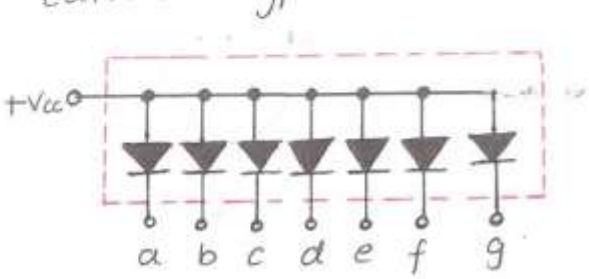


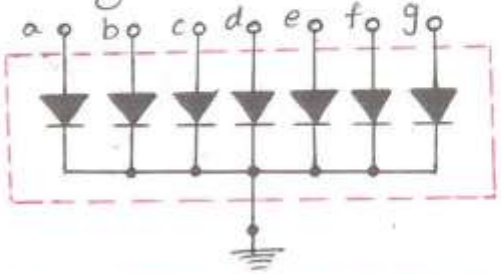
fig (a) Seven Segment Indicators

Seven segment indicators may be the common anode type where all anodes are connected togeather or common cathode type, where all cathodes are connected togeather.

With the common anode type, you have to connect a current limiting resistor between each LED and ground. The common cathode type uses a current limiting resistor b/w each LED & +Vcc



Common Anode Type



Common Cathode Type

ENCODERS -

An encoder converts an active input signal into a coded output signal. There are n input lines, only one of which is active. Internal logic within the encoder converts this active input to a coded ~~el~~ binary output with m bits.

Decimal - to - BCD Encoder.

The fig(b) shows a common type of encoder. – the decimal to BCD encoder. The switches are push button switches like those of a pocket calculator. when the button 3 is pressed, the C and D OR gates have high inputs; therefore, the output is.

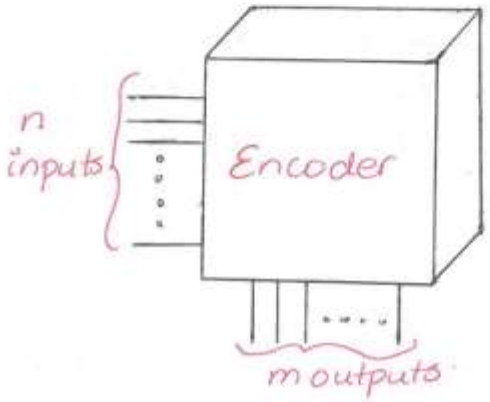
$$ABCD = 0011.$$

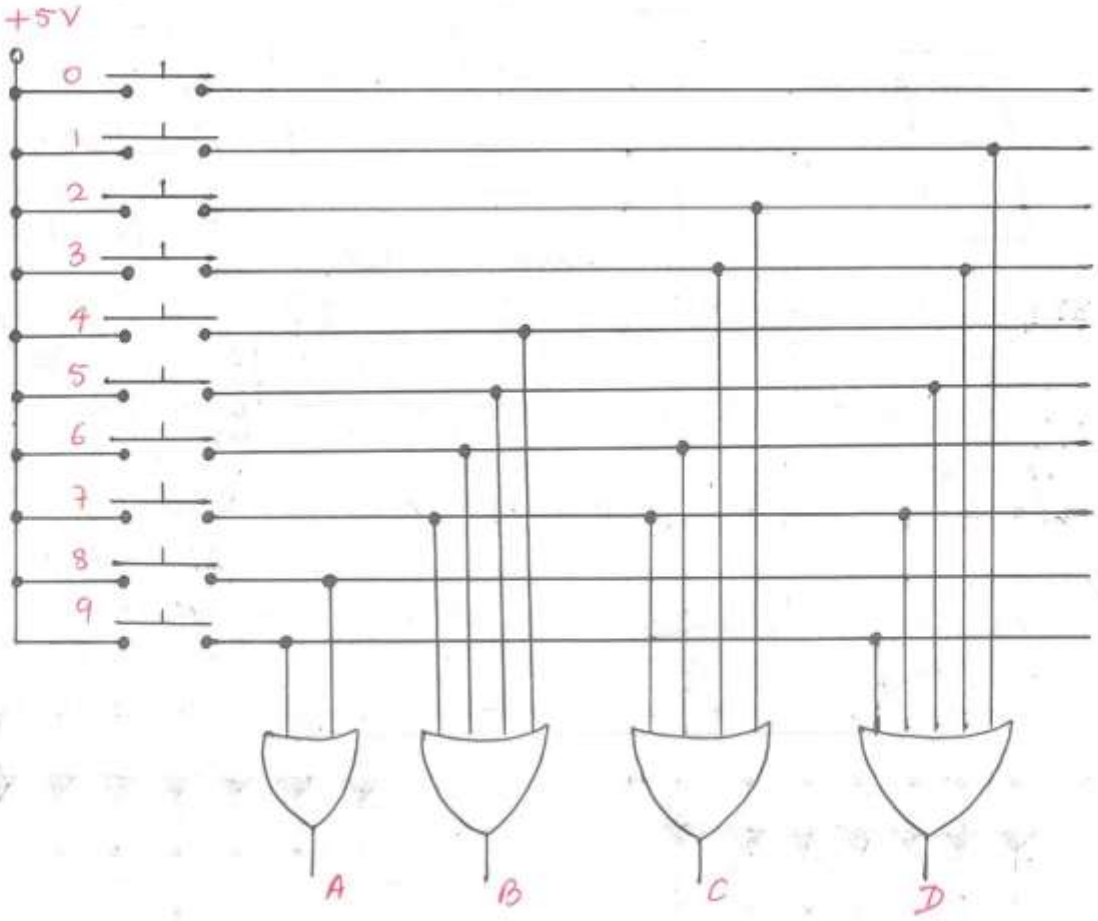
If button 5 is pressed, the output becomes

$$ABCD = 0101$$

when switch 9 is pressed,

$$ABCD = 1001.$$





Decimal - to - BCD Encoder.

EXCLUSIVE - OR GATES.

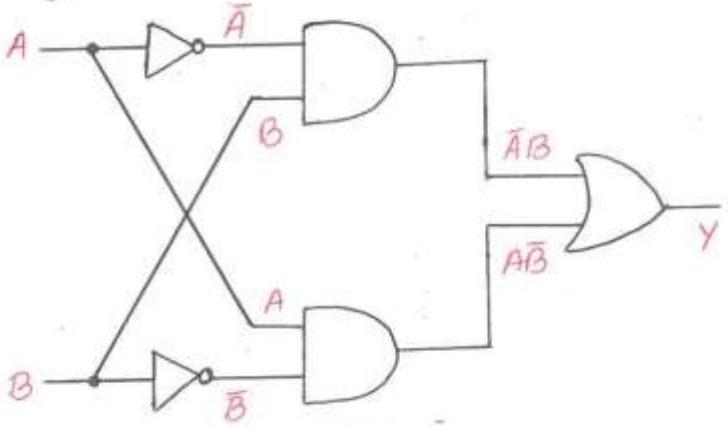
The exclusive OR gate has a high output only when an odd number of inputs is high.

The figure below shows how to build an exclusive OR gate. The upper AND gate forms the product $\bar{A}B$, while the lower one produces $A\bar{B}$. Therefore, the output of the OR gate is,

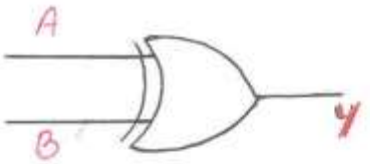
$$Y = \bar{A}B + A\bar{B}$$

Here is what happens for different inputs. When A and B are low, both AND gates have low outputs; therefore, the final output Y is low. If A is low, and B is high, the upper AND gate has a high output, so the OR gate has high output. Likewise, a high A and a low B results in a final output that is high. If both the inputs are high, both the AND gates have low outputs, and the final output is low.

The output of an exclusive OR gate is high when either A or B is high, but not when both are high.



Exclusive-OR gate.



Logic Symbol for Exclusive OR gate.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

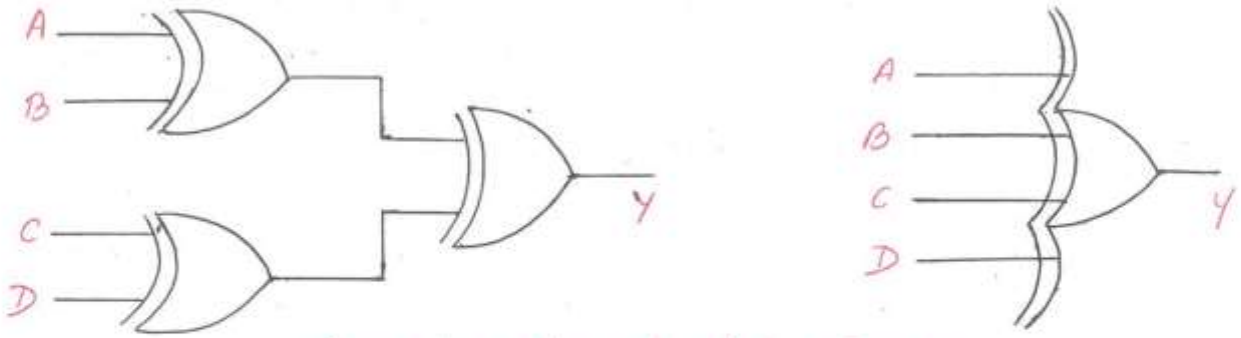
Exclusive - OR Truth Table

Four Inputs.

The fig below shows a pair of exclusive - OR gates driving an exclusive - OR gate. If all inputs (A to D) are low, the input gates have low outputs, so the final gate has a low output.

If A to C are low and D is high, the upper gate has low output, the lower gate has a high output, & the output gate has a high output.

If we continue analyzing the circuit operation for the remaining input possibilities, we get the below table.
Here is an important property of this truth table. Each ABCD input with an odd number of 1's produces an output 1.

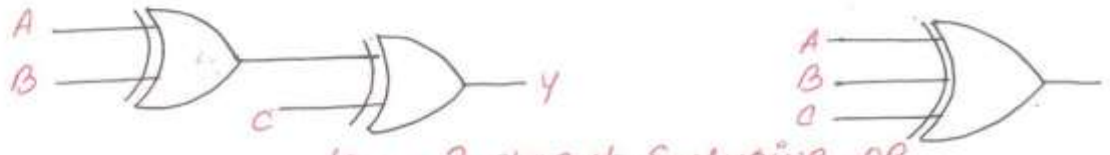


four Input exclusive OR gate.

Comment	A	B	C	D	Y
Even	0	0	0	0	0
odd	0	0	0	1	1
odd	0	0	1	0	1
Even	0	0	1	1	0
odd	0	1	0	0	1
Even	0	1	0	1	0
Even	0	1	1	0	0
odd	0	1	1	1	1
Odd	1	0	0	0	1
Even	1	0	0	1	0
Even	1	0	1	0	0
odd	1	0	1	1	1
Even	1	1	0	0	0
odd	1	1	0	1	1
Odd	1	1	1	0	1
Even.	1	1	1	1	0

4-Input Exclusive OR Gate Truth Table.

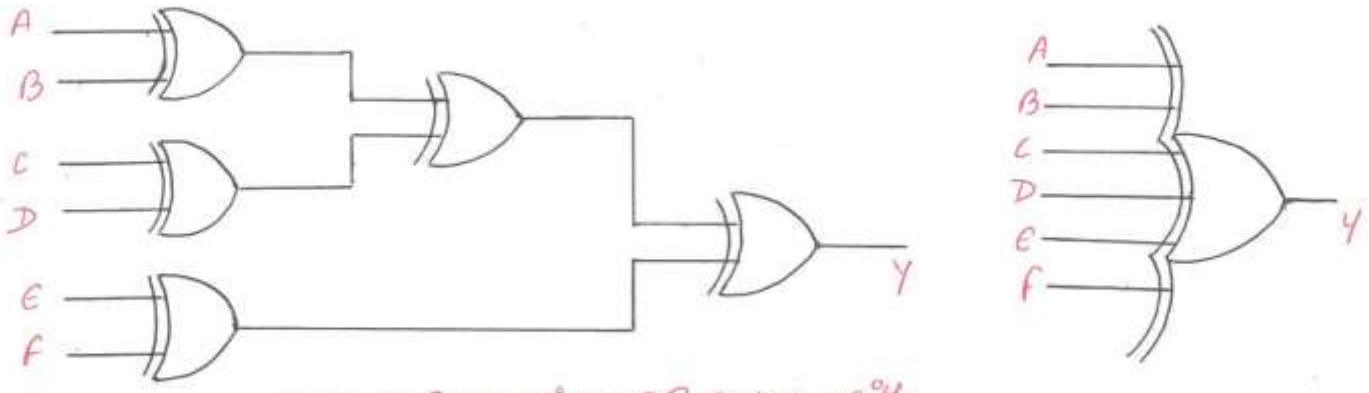
Any Number of Inputs -
Using 2-input exclusive OR gates as building blocks, you can produce exclusive-OR gates with any number of inputs.
The fig(a) shows a pair of exclusive-OR gates. There are 3 inputs and 1 output. If you analyze this circuit, you will find, it produces an output 1 only when the 3-bit input has an odd number of 1's.



fig(a) 3-Input Exclusive-OR gate.

The fig(b) shows a circuit with 6 inputs and 1 output. Analysis of the circuit shows that it produces an output 1 only when the 6-bit input has an odd number of 1's.

In general, you can build an exclusive-OR gate with any number of inputs. Such a gate always produces an output 1 only when the n-bit input has an odd number of 1's.



Fig(b) Exclusive-OR Gates with Several Inputs.

PARITY GENERATORS AND CHECKERS -

Even Parity means an n-bit input has even no. of 1's: for instance, 110011 has an even parity because it contains four 1's.

Odd Parity means an n-bit input has odd no. of 1's. for example, 110001 has an odd parity because it contains three 1's.

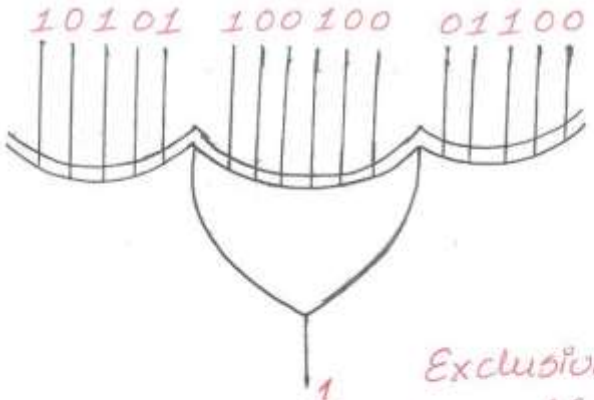
Here are two more examples,

1111 0000 1111 0011 even parity.
1111 0000 1111 0111 odd parity.

Parity Checker.

Exclusive-OR gates are ideal for checking the parity of a binary number; because they produce an output 1 when the input has odd number of 1's.

Therefore, an even parity input to an exclusive OR gate produces a low output, while an odd parity input produces a high output for eg, the fig below shows a 16 ~~bit~~ input exclusive OR gate. The exclusive OR gate produces an output 1 because the input has odd parity. If the 16-bit input changes to another value, the output becomes 0 for even parity numbers & 1 for odd-parity numbers.



Exclusive -OR Gate with 16 inputs.

Parity Generation.

In a computer, a binary number may represent an instruction that tells the computer to add, subtract, and so on; or the binary number may represent data to be processed like a number, letter, etc. In either case, you sometimes will see an extra bit added to the original binary number to produce a new binary number with even or odd parity.

For instance, The figure below shows the 8-bit binary number.

$$X_7 X_6 X_5 X_4 \; X_3 X_2 X_1 X_0.$$

Suppose this number equals 0100 0001. Then the number has even parity, which means the Exclusive OR gate produces an output 0. Because of the inverter,

$$X_8 = 1$$

and the final 9-bit output is 1 0100 0001. Notice that this has an odd parity.

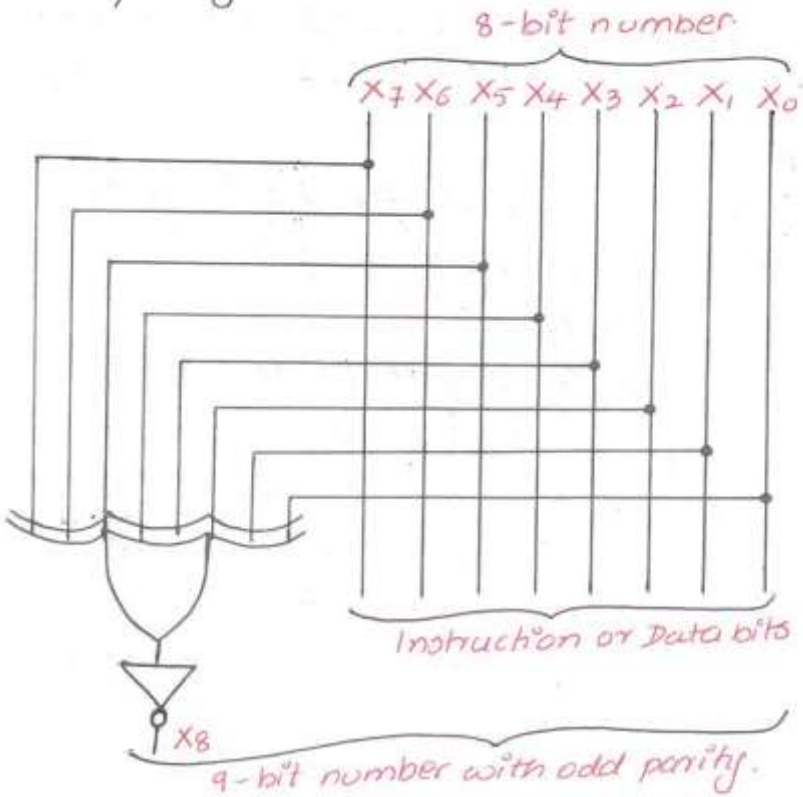


FIG: odd Parity Generation

The circuit given in the above figure is called an odd Parity generator because it always produces a 9-bit output number with odd parity. If the 8-bit input has even parity, a 1 comes out of the inverter to produce a final output with odd parity. on the other hand, if the 8-bit input has odd parity, a '0' comes out of the inverter, and the final 9-bit output again has odd parity.

Application -

What is the practical application of parity generation & checking? Because of transients, noise, and disturbances, 1 bit errors sometimes occur when binary data is transmitted over telephone lines or other communication paths.

one way to check for errors is to use an odd-parity generator at the transmitting end and an odd parity checker at the receiving end. If no 1-bit error occurs in transmission, the received data will have odd parity. But if one of the transmitted bits is changed by noise or any other disturbances, then the received data will have even parity.

For instance, suppose we want to send 0100 0011. with an odd parity generator, the data to be transmitted will be 0 0100 0011. This data can be sent over telephone lines to some destination. If no error occurs in transmission, the odd parity checker at the receiving end will produce a high output, meaning that the received number has an odd parity.

On the other hand, if there is a one bit error during transmiss-ion, then the odd parity ~~will~~ checker will have a low output, indicating that the received data is invalid.

Errors are rare to begin with. when they do occur, they are usually 1-bit errors.

MAGNITUDE COMPARATOR -

Magnitude comparator is a combinational circuit capable of comparing the relative magnitude of two binary numbers.

Magnitude comparator accepts two n-bit binary numbers, say A & B as inputs and produces one of the outputs:

$A > B$, $A = B$ and $A < B$.

The output $A > B$ will be high if A is greater than B,

The output $A = B$ will be high if A equals to B, and

The output $A < B$ will be high if A is lesser than B.

Magnitude comparators are used in CPU's & microcontrollers

The figure (a) presents the block diagram of such a comparator, figure (b) presents the truth table when two 1-bit numbers are compared, and its circuit diagram is shown in figure (c).

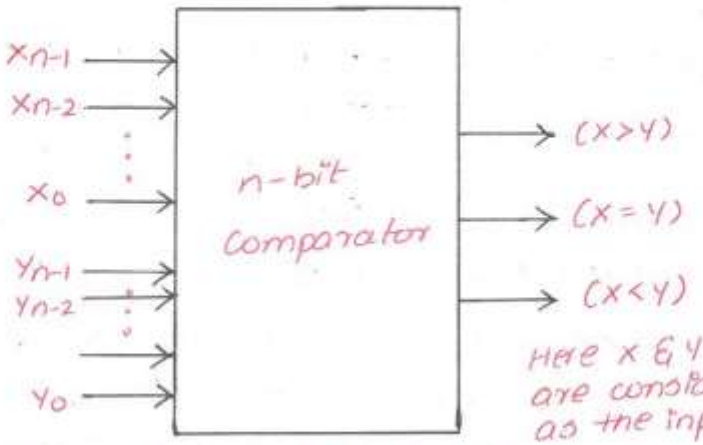


Fig (a) Block Diagram of magnitude comparator.

Input		Output		
A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Fig (b) Truth Table of 1-bit comparator.

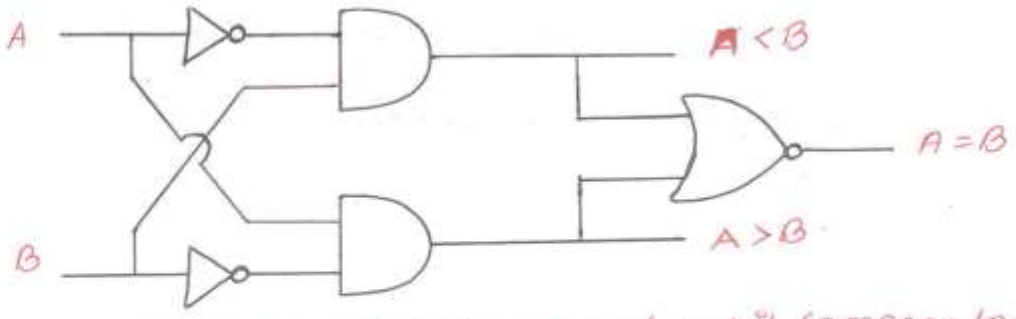


Fig (c) Circuit Diagram of 1-bit comparator.

The logic eqns for the outputs $A > B$; $A = B$ and $A < B$ can be written as follows:

$(A > B) : G = A\bar{B}$ $\quad (A < B) : L = \bar{A}B$ $\quad (A = B) : E = \bar{A}\bar{B} + AB$

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

$= \overline{(G + L)}$

PROGRAMMABLE ARRAY LOGIC -

Programmable Array Logic (PAL) is a programmable array of logic gates on a single chip. PAL's are another design solution, similar to a sum of product solution, product of sums solutions and multiplexer logic.

Programming a PAL

A PAL has a programmable AND array and a fixed OR array. For eg, the fig below shows a PAL with 4 inputs and 4 outputs.

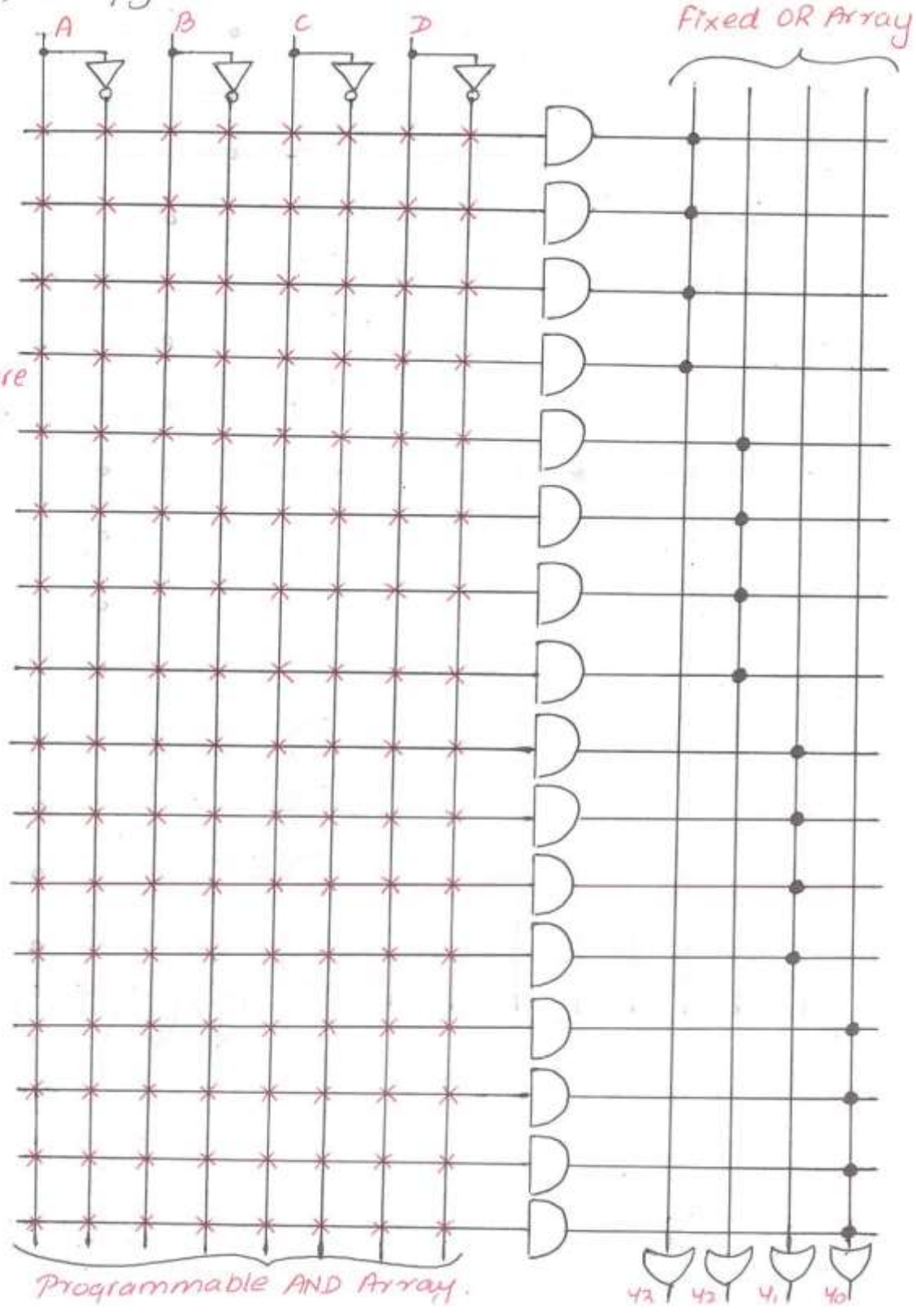


FIG: Structure of PAL

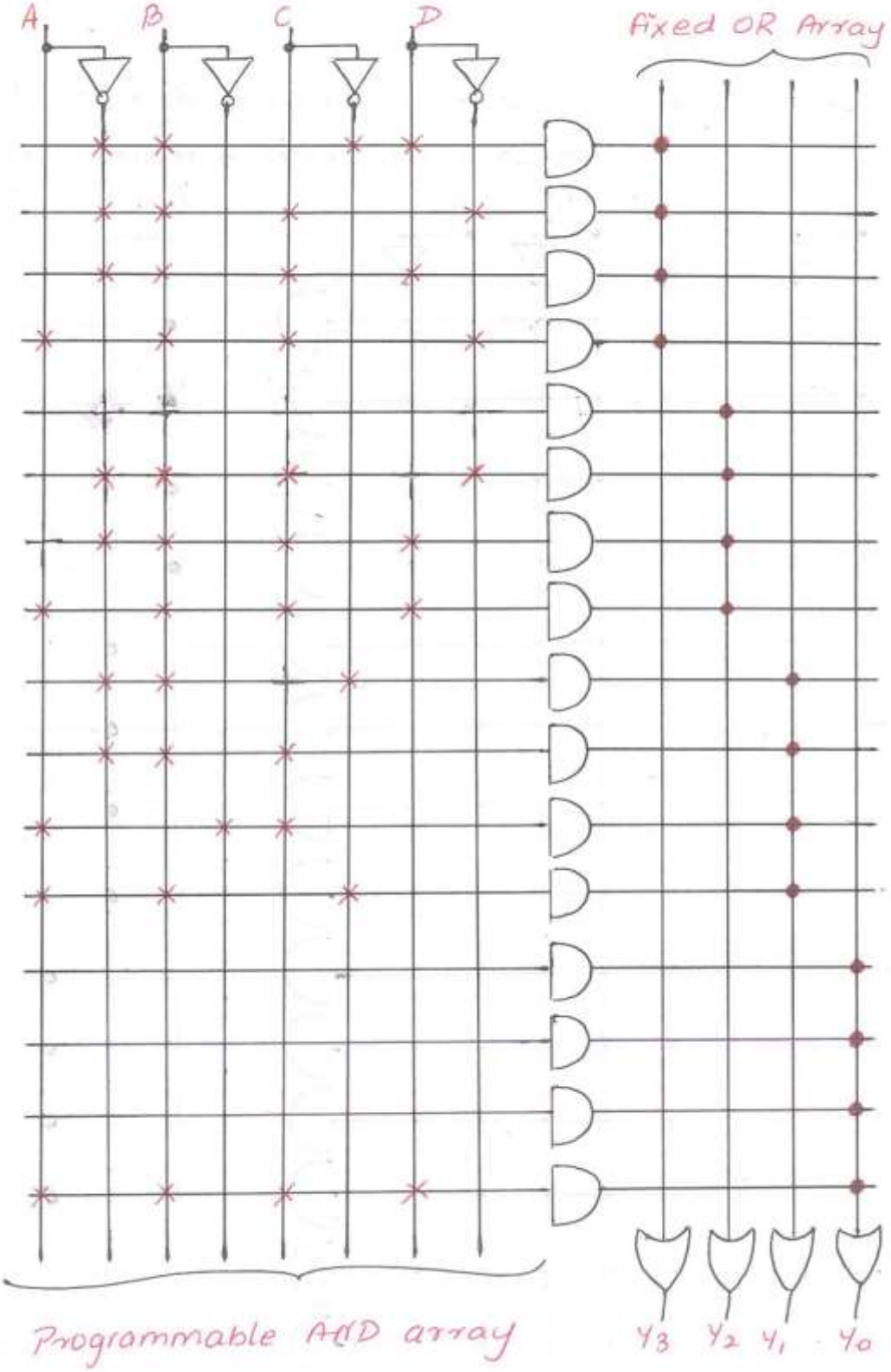
Here is an eg. of how to program a PAL. Suppose we want to generate the following Boolean functions:

$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D}$$

$$Y_2 = \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$$

$$Y_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$

$$Y_0 = ABCD$$



PROGRAMMABLE LOGIC ARRAY -

Programmable Logic Arrays (PLA's) are included in the more general classification of IC's called as Programmable Logic Devices (PLD's).

The PLA is much more versatile than the PAL, since both its AND gate array & its OR-gate array are fusible linked and are programmable.

It is also more complicated to utilize since the number of fusible links are doubled.

A PLA having 3 input variables (ABC) and 3 output variables (XYZ) is illustrated in the figure below.

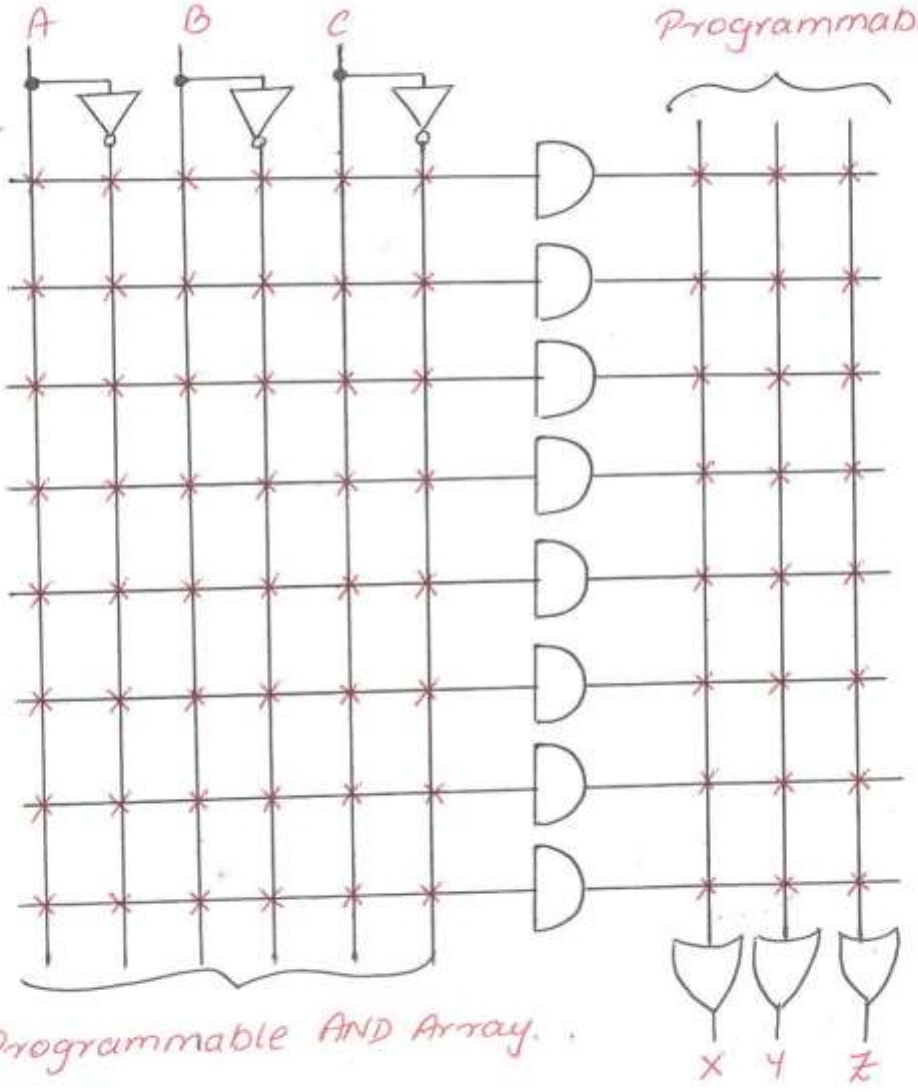
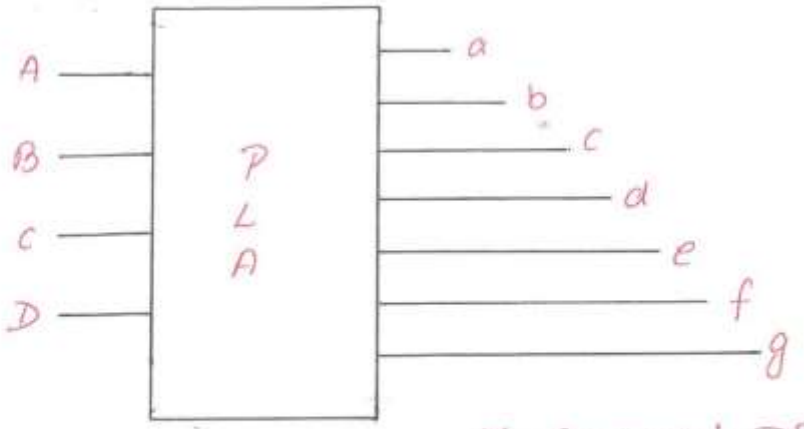
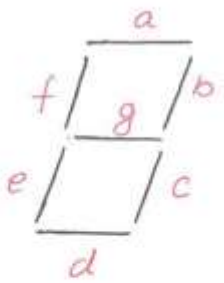


FIG: Sample PLA

Eight AND gates are required to decode the 8 possible input states. In this case there are 3 OR gates that can be used to generate logic functions at the output.

Implement a 7-segment display using PLA.





7-Segment Display.

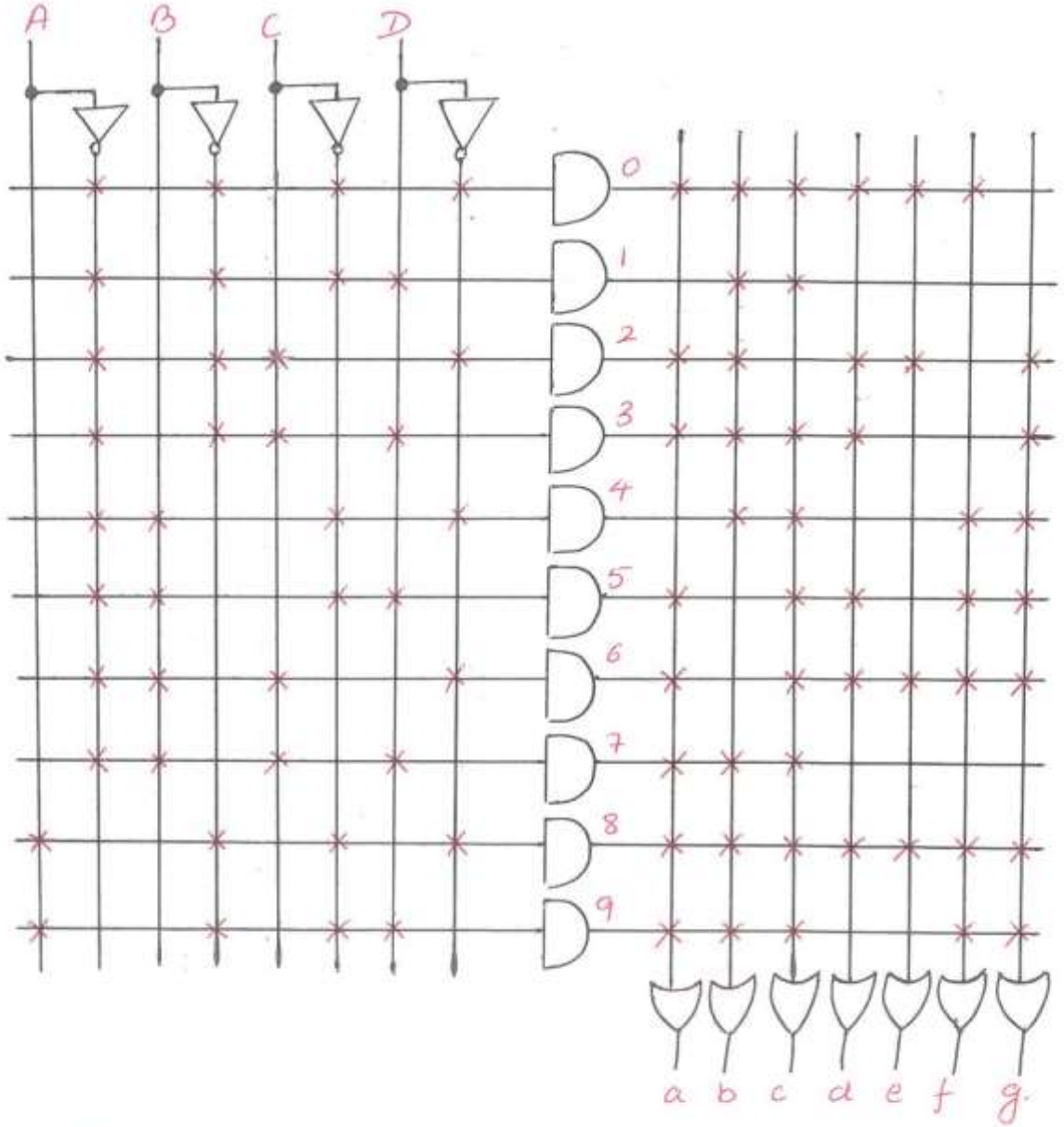


FIG: 7-Segment Decoder using PLA

HDL Implementation of Data Processing Circuits

① write a HDL verilog code to realize a 2 to 1 multiplexer.

```
module mux 2 to 1 (A, D0, D1, Y);
input A, D0, D1;
output Y;
assign Y = (~A & D0) | (A & D1);
endmodule
```

OR

```
module mux 2 to 1 (A, D0, D1, Y);
input A, D0, D1;
output Y;
assign Y = A ? D1 : D0; /* conditional assignment */
endmodule
```

② write a verilog code to realize a 2-to-1 mux using Behavioral model.

```
module mux 2 to 1 (A, D0, D1, Y);
input A, D0, D1;
output Y;
reg Y;
always @ (A or D0 or D1)
if (A == 1) Y = D1;
else Y = D0;
endmodule
```

OR

```
module mux 2 to 1 (A, D0, D1, Y);
input A, D0, D1;
output Y;
reg Y;
always @ (A or D0 or D1)
   case (A)
      0 : Y = D0;
      1 : Y = D1;
   endcase
endmodule
```

③ Design a 4 to 1 multiplexer using conditional assign and case statements.

i> Using Conditional Statements

```
module mux4to1 (A,B,D0,D1,D2,D3,Y);
input A,B,D0,D1,D2,D3;
output Y;
assign Y = A ? (B ? D3 : D2) : (B ? D1 : D0);
endmodule
```

ii> Using Case Statements.

```
module mux 4 to 1 (A,B,D0,D1,D2,D3,Y);
input A,B,D0,D1,D2,D3;
output Y;
reg Y;
always @ (A or B or D0 or D1 or D2 or D3)
case ({A,B})
    0: Y = D0;
    1: Y = D1;
    2: Y = D2;
    3: Y = D3;
endcase
endmodule
```