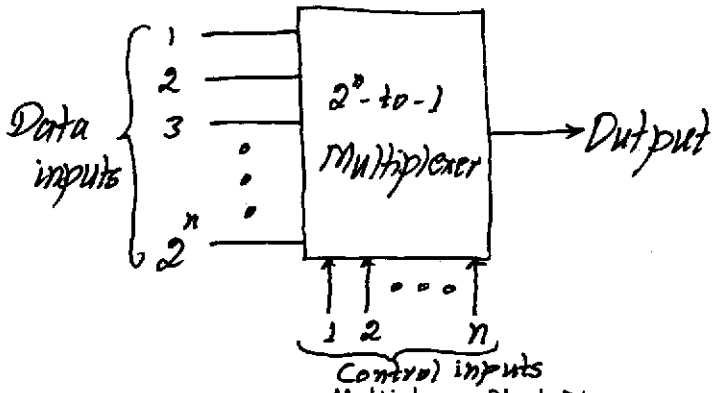
MODULE – 3

DATA PROCESSING CIRCUITS & FLIP-FLOPS

DATA PROCESSING CIRCUITS

MULTIPLEXERS (MUX):

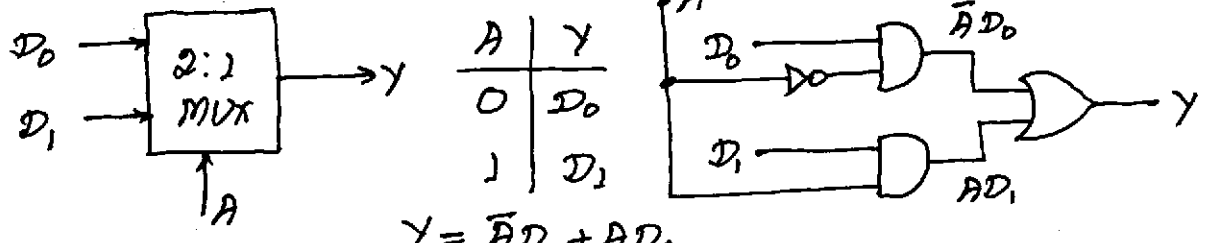
Multiplex means *many into one. A multiplexer is a circuit with many inputs but only one output. By applying control signals, we can steer any input to the output*. Thus, it is also called a *data selector* and the control inputs are termed *select inputs*. The following Fig illustrates the general idea.



Multiplexer Block Diagram

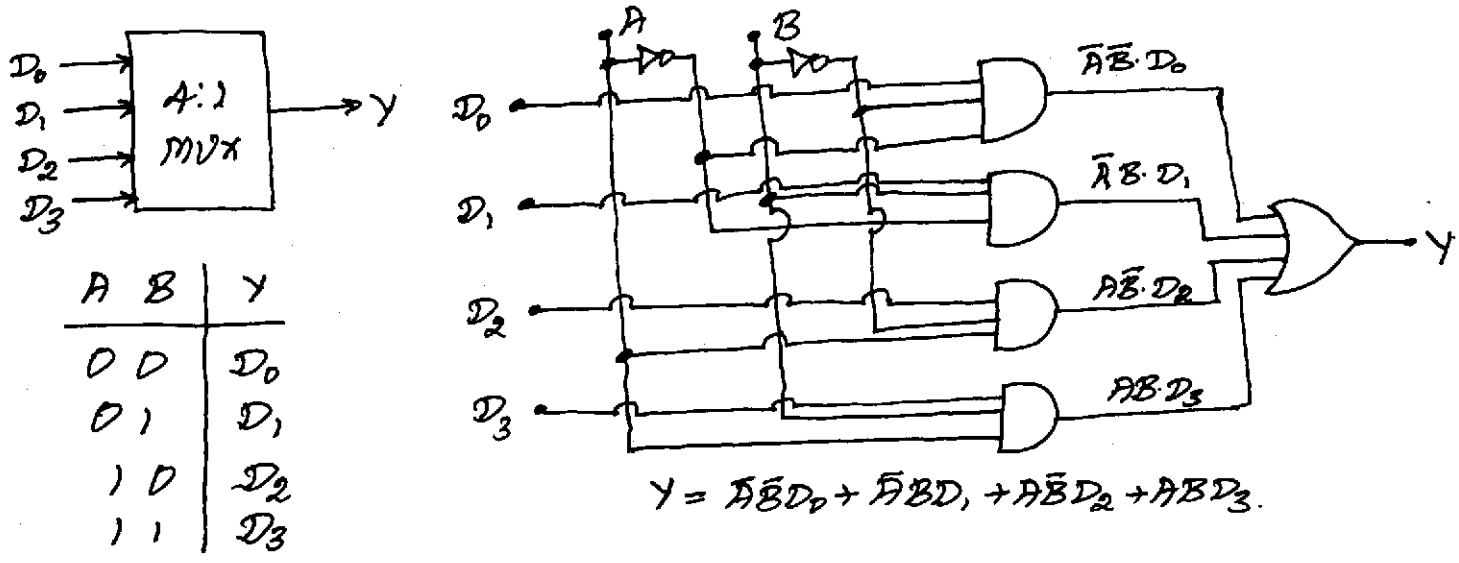
The block diagram has 2^n input signals, n control signals, and *1* output signal.

2-to-1 MUX:



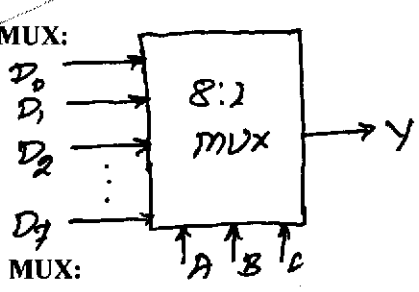
$Y = \bar{A}D_0 + AD_1$.

4-to-1 MUX:



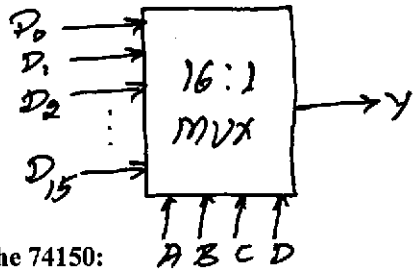
$Y = \bar{A}\bar{B}D_0 + \bar{A}BD_1 + A\bar{B}D_2 + ABD_3$.

8-to-1 MUX:



$$Y = \bar{A}\bar{B}\bar{C}\cdot D_0 + \bar{A}\bar{B}C\cdot D_1 + \bar{A}B\bar{C}\cdot D_2 + \bar{A}BC\cdot D_3 + A\bar{B}\bar{C}\cdot D_4 + A\bar{B}C\cdot D_5 + AB\bar{C}\cdot D_6 + ABC\cdot D_7.$$

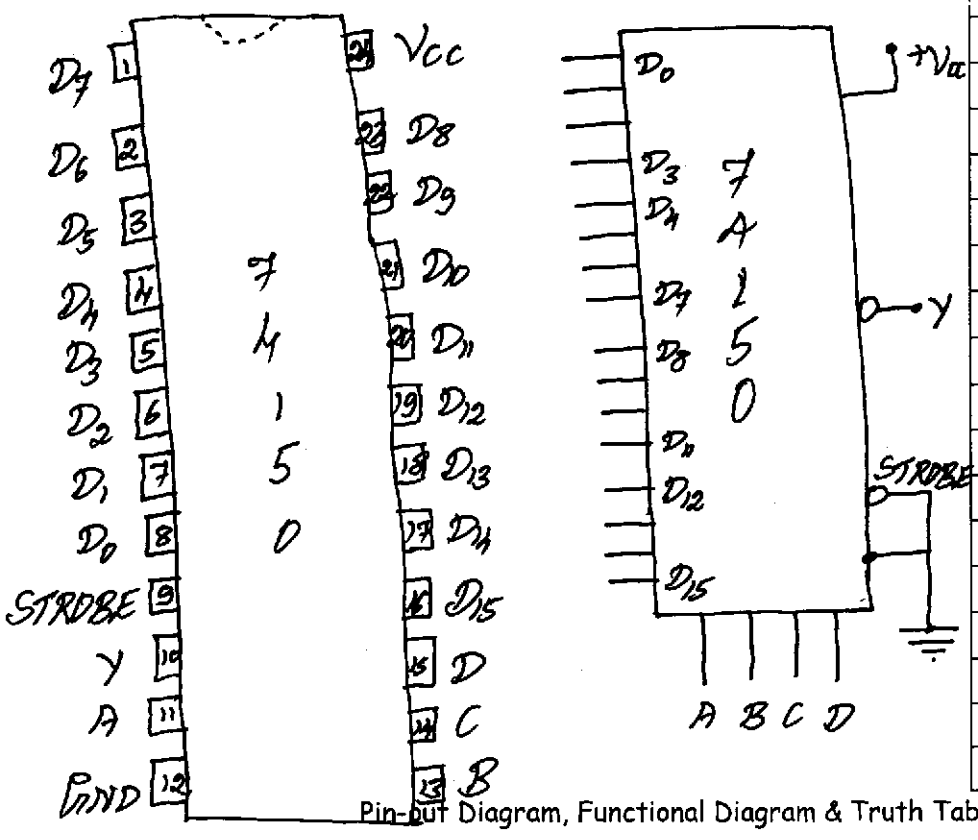
16-to-1 MUX:



$$Y = \bar{A}\bar{B}\bar{C}\bar{D}\cdot D_0 + \bar{A}\bar{B}\bar{C}D\cdot D_1 + \cdots + ABCD\cdot D_{15}.$$

The 74150:

The 74150 is a 16-to-1 TTL multiplexer with the pin diagram as shown below. Pins 1 to 8 and 16 to 23 are input data bits D_0 to D_{15}.Pins 11, 13, 14, and 15 are control bits ABCD. Pin 10 is the output; and it equals the complement of the selected data bit. Pin 9 is for STROBE, an input signal that enables or disables the multiplexer. A low STROBE enables the multiplexer, so that, the output Y equals the complement of input data bit; $Y = \bar{D}_n$, where n is the decimal equivalent of ABCD.



STROBE	A	B	C	D	Y
L	L	L	L	L	$\bar{D}_0$
L	L	L	L	H	$\bar{D}_1$
L	L	L	H	L	$\bar{D}_2$
L	L	L	H	H	$\bar{D}_3$
L	L	H	L	L	$\bar{D}_4$
L	L	H	L	H	$\bar{D}_5$
L	L	H	H	L	$\bar{D}_6$
L	L	H	H	H	$\bar{D}_7$
L	H	L	L	L	$\bar{D}_8$
L	H	L	L	H	$\bar{D}_9$
L	H	L	H	L	$\bar{D}_{10}$
L	H	L	H	H	$\bar{D}_{11}$
L	H	H	L	L	$\bar{D}_{12}$
L	H	H	L	H	$\bar{D}_{13}$
L	H	H	H	L	$\bar{D}_{14}$
L	H	H	H	H	$\bar{D}_{15}$
H	X	X	X	X	H

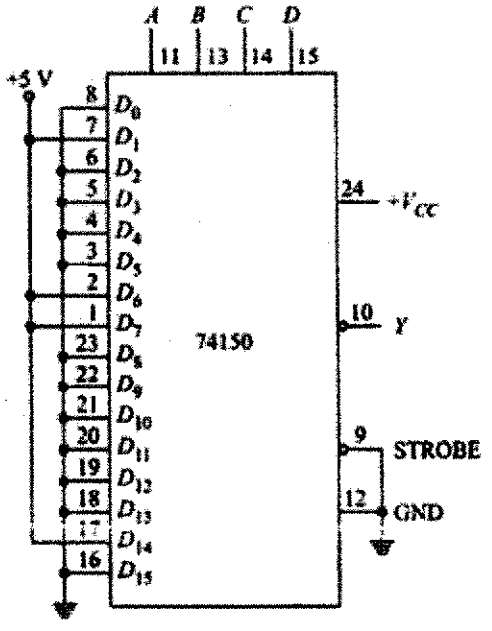
Pin-out Diagram, Functional Diagram & Truth Table of 74150

Multiplexer Logic: A digital design usually begins with a truth table. The problem is to come up with a logic circuit that has the same truth table. We have two standard methods for implementing a truth table – the SOP and the POS solution. The third method is the *multiplexer solution.*

Problem: *Implement Y (A, B, C, D) = ∑m (0, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15) using 74150 multiplexer.*

Solution:

STROBE	A	B	C	D	Y
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	1
1	X	X	X	X	1



Bubbles on Signal Lines:

Data sheets often show inversion bubbles on some of the signal lines. (Notice the bubble on Pin 10 of 74150. This bubble is reminder that the output is the complement of the selected data bit). Notice the bubble on the Pin 9, STROBE input. The multiplexer is active (enabled) when the STROBE is low, and is inactive (disabled) when it is high. Because of this, the STROBE is called an *active-low signal*; it causes something to happen when it is low rather than when it is high.

Universal Logic Circuit:

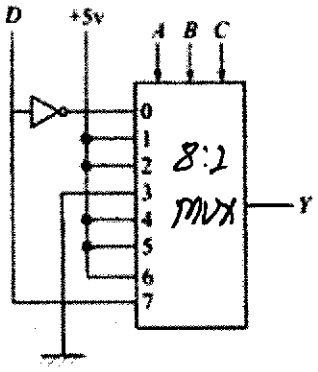
Multiplexer, sometimes, is called *universal logic circuit*; because a 2^n*-to-1* multiplexer can be used as a design solution for any *n* variable truth table. We have seen the realization of 4 variable truth table by 16-to1 multiplexer; now we will see the realization of same expression using 8-to-1 multiplexer.

Problem: *Implement Y (A, B, C, D) = ∑m (0, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15) using 8-to-1 multiplexer.*

Solution: We follow a procedure that is similar to the one that we adopted in Entered Variable Map method.

A	B	C	D	Y	8-to-1 MUX Data Inputs
0	0	0	0	1	$\bar{D}$
0	0	0	1	0	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	1	1
0	1	0	1	1	
0	1	1	0	0	0
0	1	1	1	0	
1	0	0	0	1	1
1	0	0	1	1	
1	0	1	0	1	1
1	0	1	1	1	
1	1	0	0	1	1
1	1	0	1	1	
1	1	1	0	0	D
1	1	1	1	1	

A	B	C	8-to-1 MUX Data Inputs
0	0	0	$\bar{D}$
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	D



$D_0 = \bar{D}$

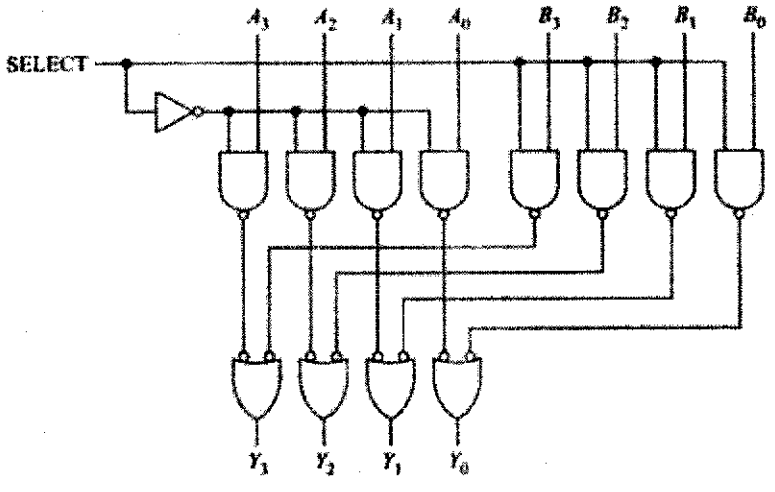
$D_1 = D_2 = D_4 = D_5 = D_6 = 1$

$D_3 = 0$

$D_7 = D.$

Nibble Multiplexer:

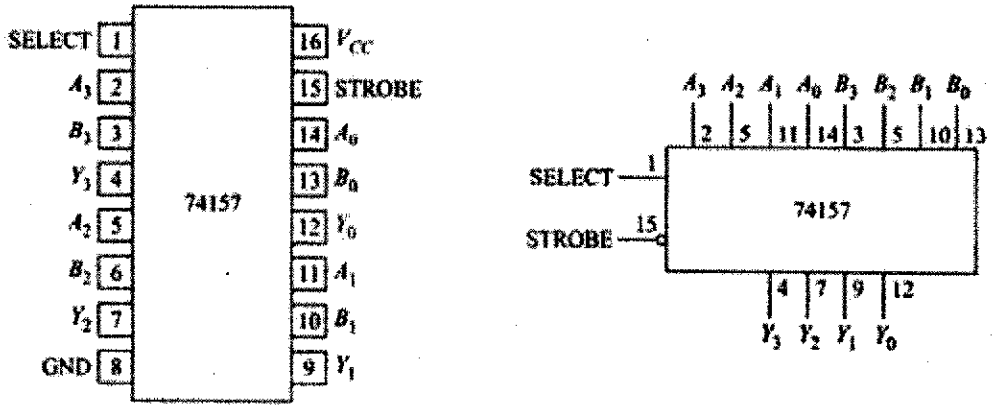
Sometimes, we may want to select one of two input nibbles. In this case, we can use a nibble multiplexer, as shown below:



Nibble Multiplexer

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 activated; therefore, $Y_3Y_2Y_1Y_0 = B_3B_2B_1B_0$.

The 74157: The following Fig shows the pin-out diagram and functional diagram of a 74157, a nibble multiplexer. The STROBE input must be low for the multiplexer to properly operate. When the STROBE is high, the multiplexer is inoperative.



Pin-out Diagram, Functional Diagram of 74157

Problem: *Show how 4-to-1 multiplexer can be obtained using only 2-to-1 multiplexers.*

Solution:

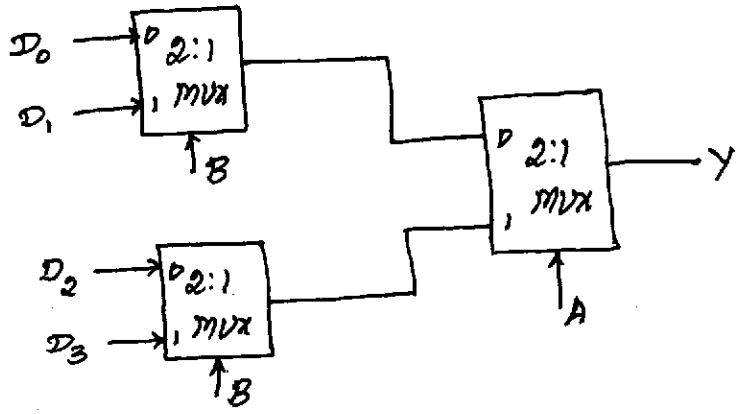
Logic equation for 2-to-1 multiplexer: $Y = \bar{A}D_0 + AD_1$ ⟶ (1)

Logic equation for 4-to-1 multiplexer: $Y = \bar{A}\bar{B}D_0 + \bar{A}BD_1 + A\bar{B}D_2 + ABD_3$

$Y = \bar{A}[\bar{B}D_0 + BD_1] + A[\bar{B}D_2 + BD_3]$ ⟶ (2)

Comparing above equations, we need two 2-to-1 multiplexer to realize two bracketed terms, where B serves as select input. The output of these two multiplexers can be sent to a third multiplexer as data inputs, where A serves as select input and we get the 4-to-1 multiplexer.

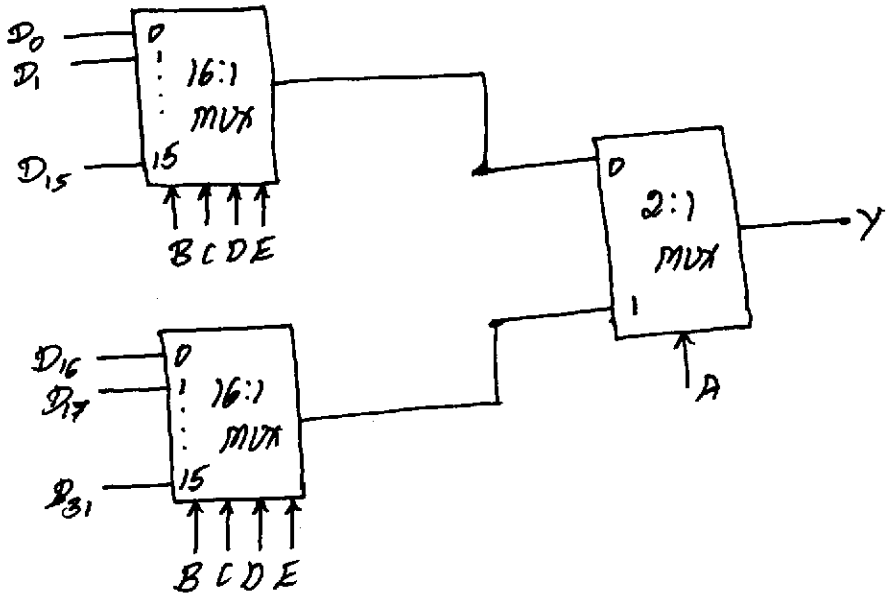
The following Fig shows the circuit diagram:



Problem: *Design a 32-to-1 multiplexer using two 16-to-1 multiplexer and one 2-to-1 multiplexer.*

Solution:

The circuit diagram is shown in the following Fig. A 32-to-1 multiplexer required 5 ($\log_2 32$) select lines (say, ABCDE). The lower four select lines (BCDE) chose 16-to-1 multiplexer outputs. The 2-to-1 multiplexer chooses one of the output of two 16-to-1 multiplexers, depending on the 5th select line (A).



Problem: *Realize* $Y = \bar{A}B + \bar{B}\bar{C} + ABC$ *using an 8-to-1 multiplexer. Also, realize the same with a 4-to-1 multiplexer.*

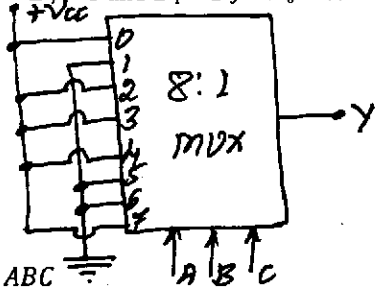
Solution: Given, $Y = \bar{A}B + \bar{B}\bar{C} + ABC$

$Y = \bar{A}B(\bar{C}+C) + \bar{B}\bar{C}(\bar{A}+A) + ABC = \bar{A}B\bar{C} + \bar{A}BC + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + ABC$

$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC = \Sigma m(0, 2, 3, 4, 7)$

Hence, to generate the given logic function, using *8-to-1 multiplexer*,

we find $D_0 = D_2 = D_3 = D_4 = D_7 = 1$ and $D_1 = D_5 = D_6 = 0$.



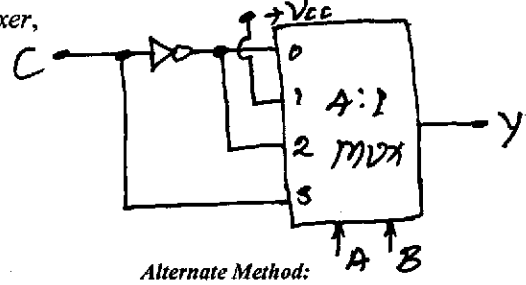
Now, $Y = \bar{A}B + \bar{B}\bar{C} + ABC$

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

$Y = \bar{A}\bar{B}\cdot\bar{C} + \bar{A}B\cdot 1 + A\bar{B}\cdot\bar{C} + AB\cdot C$

Hence, for a *4-to-1 multiplexer*,

we find $D_0 = C'$, $D_1 = 1$, $D_2 = C'$, and $D_3 = C$ generates the given function.

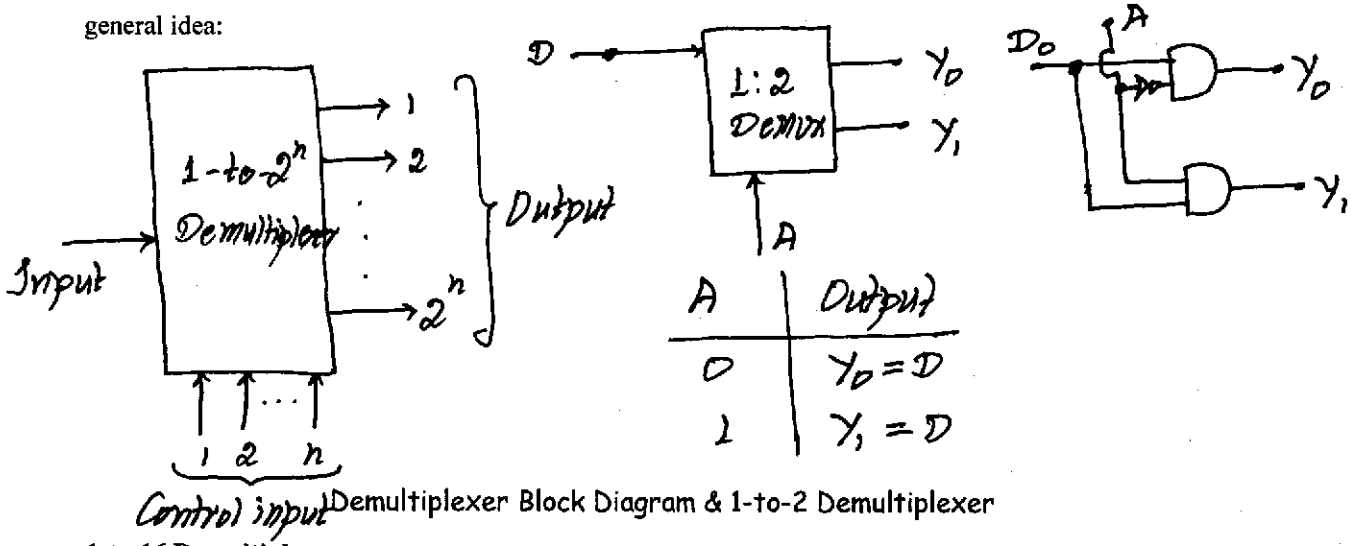


Alternate Method:

A	B	C	8-to-1 MUX Data Inputs	4-to-1 MUX Data Inputs
0	0	0	$1 = D_0$	$\bar{C} = D_0$
0	0	1	$0 = D_1$	
0	1	0	$1 = D_2$	$1 = D_1$
0	1	1	$1 = D_3$	
1	0	0	$1 = D_4$	$\bar{C} = D_2$
1	0	1	$0 = D_5$	
1	1	0	$0 = D_6$	$C = D_3$
1	1	1	$1 = D_7$	

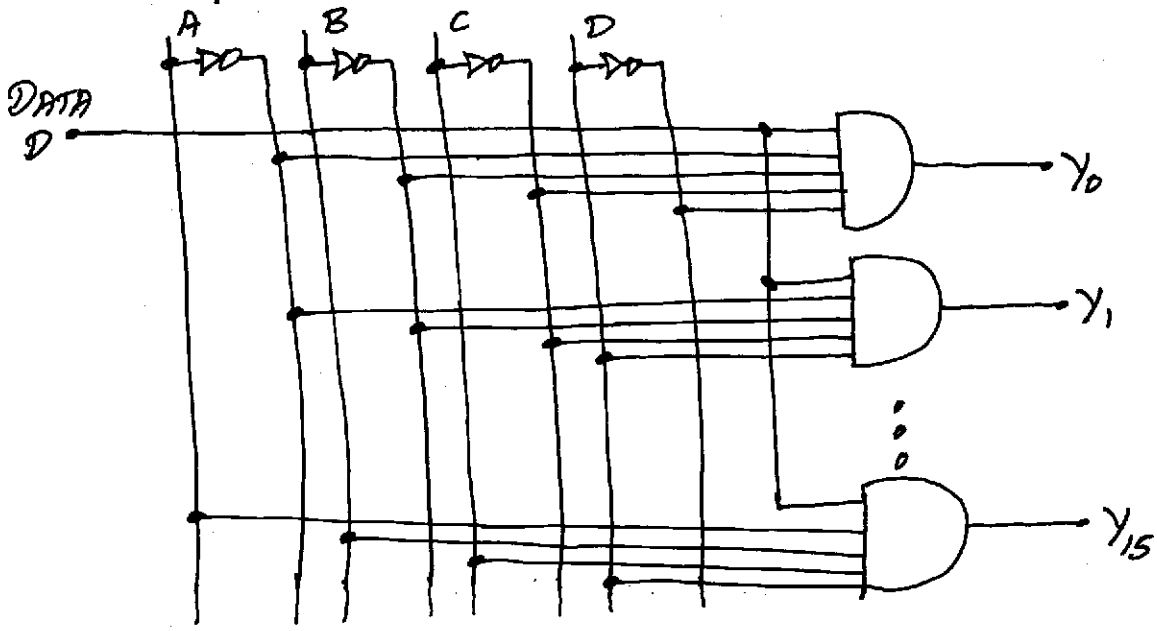
DEMULTIPLEXERS (DE-MUX):

Demultiplexer means *one into many. A Demultiplexer is a logic circuit with one input and many outputs.* By applying control signals, we can steer the input signal to one of the output lines. A Demultiplexer circuit has *1* input signal, *n* control or select signals, and 2^n output signals. The following Fig shows the general idea:



Demultiplexer Block Diagram & 1-to-2 Demultiplexer

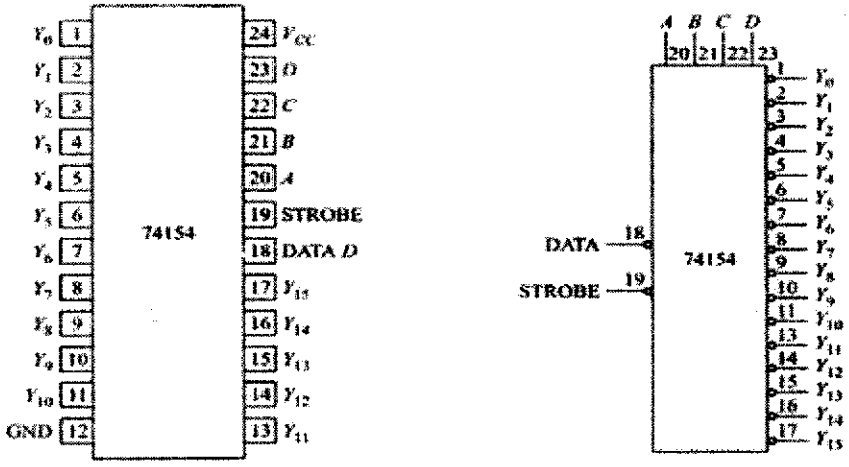
1-to-16 Demultiplexer:



1-to-16 Demultiplexer

The input bit is labeled D. This input data bit is transmitted to the data bit of the output line. When ABCD = 0000, the upper AND gate is enabled, while all other AND gates are disabled. Hence, data bit D is transmitted only to the Y_0 output, giving $Y_0 = D$. If D is low, Y_0 is low; if D is high, Y_0 is high. All other outputs are in the low state. If the control nibble is changed to ABCD = 1111, all gates are disabled, except the bottom AND gate. Then, D is transmitted only to the Y_{15} output, hence, $Y_{15} = D$.

The 74154: The 74154 is a 1-to-16 Demultiplexer with the pin diagram and functional diagram as shown below. STROBE is active-low signal and must be low to activate the 74154. When STROBE is high, all the output lines are high. When STROBE is low, the control input ABCD determines the output line; the DATA bit is steered to the output line, whose subscript is the decimal equivalent of ABCD. Notice that, DATA is inverted at the input (the bubble on pin 18) and again on any output (the bubble on each output line). With this double inversion, DATA passes through the 74154 unchanged.



Pin-out Diagram & Functional Diagram of 74154

ST	DA	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9	Y_{10}	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
L	*L*	L	L	L	L	*L*	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	*L*	L	L	L	H	H	*L*	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	*L*	L	L	H	L	H	H	*L*	H	H	H	H	H	H	H	H	H	H	H	H	H
L	*L*	L	L	H	H	H	H	H	*L*	H	H	H	H	H	H	H	H	H	H	H	H
L	*L*	L	H	L	L	H	H	H	H	*L*	H	H	H	H	H	H	H	H	H	H	H
L	*L*	L	H	L	H	H	H	H	H	H	*L*	H	H	H	H	H	H	H	H	H	H
L	*L*	L	H	H	L	H	H	H	H	H	H	*L*	H	H	H	H	H	H	H	H	H
L	*L*	L	H	H	H	H	H	H	H	H	H	H	*L*	H	H	H	H	H	H	H	H
L	*L*	H	L	L	L	H	H	H	H	H	H	H	H	*L*	H	H	H	H	H	H	H
L	*L*	H	L	L	H	H	H	H	H	H	H	H	H	H	*L*	H	H	H	H	H	H
L	*L*	H	L	H	L	H	H	H	H	H	H	H	H	H	H	*L*	H	H	H	H	H
L	*L*	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	*L*	H	H	H	H
L	*L*	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	*L*	H	H	H
L	*L*	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	*L*	H	H
L	*L*	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	*L*	H
L	*L*	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	*L*
L	*L*	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	*L*	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	*H*	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

74154 Truth Table

Problem: *In the following Fig, what does Y_{12} output equal for the following conditions:*

a) R is high, T is high, ABCD = 0110

b) R is low, T is high, ABCD = 1100

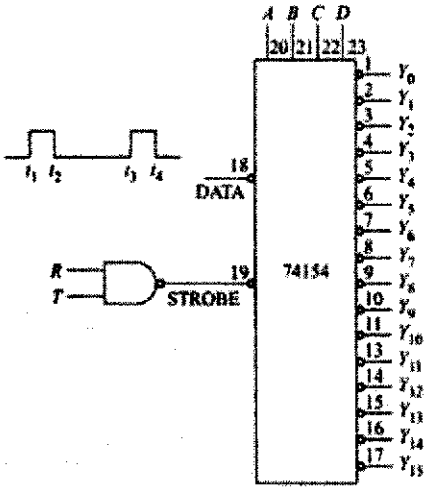
c) R is high, T is high, ABCD = 1100.

Solution:

a) Since R and T both are high, the STROBE is low and the 74154 is active. Since, ABCD = 0110, the input data is steered to the Y_6 output line. The Y_{12} output remains in the high state.

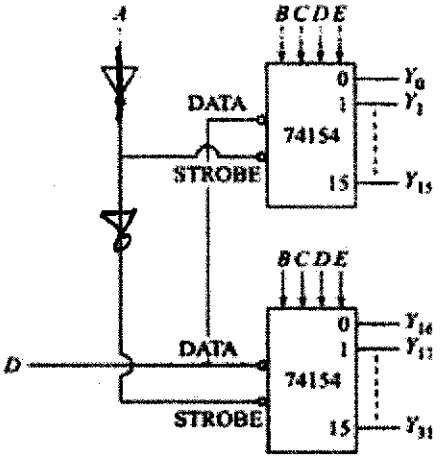
b) Since, the STROBE will be high, the 74154 will be inactive. The Y_{12} output will be high.

c) Since, the STROBE is ~~high~~ low, the 74154 is active. Since, ABCD = 1100, the two pulses (from DATA input) are steered to the Y_{12} output.



Problem: *Show how two 1-to16 Demultiplexers can be connected to get a 1-to-32 Demultiplexer.*

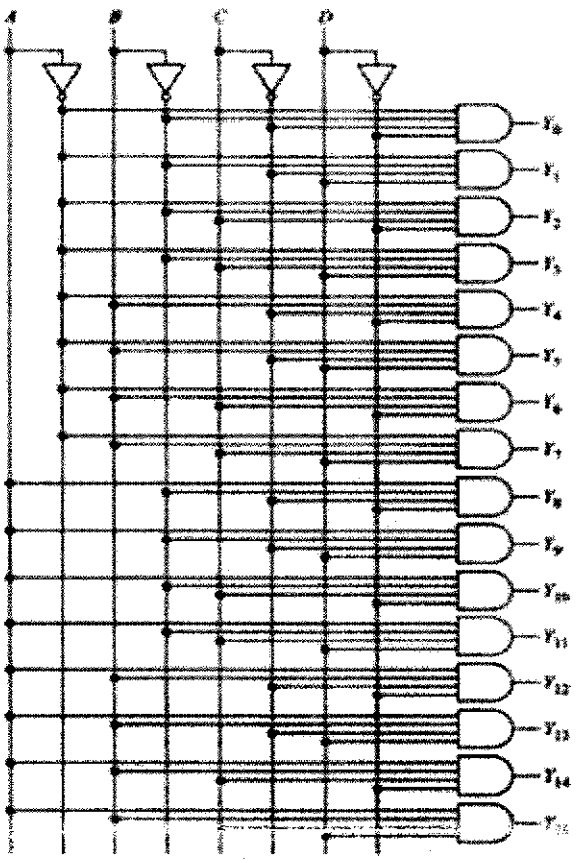
Solution: The following Fig shows the circuit diagram. A 1-to-32 Demultiplexer has 5 select lines, ABCDE. Four of them (BCDE) are fed to two 1-to-16 Demultiplexer; and the fifth (A) is used to select one of these two Demultiplexer through STROBE input. If A = 0, the top 74154 is chosen and BCDE directs DATA to one of the 15 outputs of that IC. If A = 1, the bottom 74154 is chosen and BCDE directs DATA to one of the 15 outputs of this IC.



1-OF-16 DECODER:

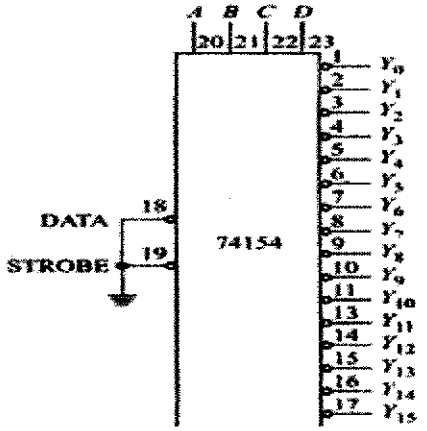
A *decoder* is similar to demultiplexer, with one exception – there is no data input. The only inputs are the control bits ABCD, which are shown in the following Fig. This logic circuit is called a *1-of-16 decoder*, because only 1 of the 16 output lines is high. For example, when ABCD is 0001, only the Y_1 AND gate has all inputs high; therefore, only the Y_1 output is high. If ABCD changes to 0100, only the Y_4 AND gate has all the inputs high; hence, only Y_4 output goes high.

The subscript of the high output always equals the decimal equivalent of ABCD. For this reason, the circuit is also called as *binary-to-decimal decoder*. Since, it has 4 input lines and 16 output lines, the circuit is also known as a *4-line to 16-line decoder*.



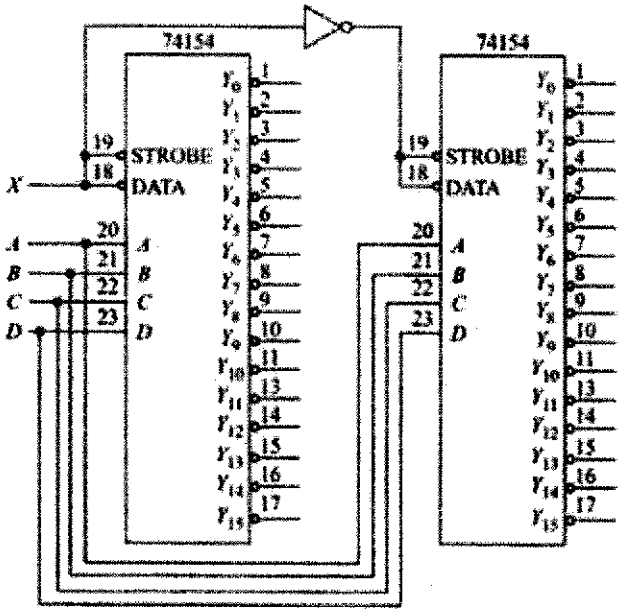
1-of-16 Decoder

The 74154 is called a decoder-demultiplexer, because it can be used either as a decoder or as a demultiplexer. To use the same IC as a decoder, all you have do is ground the DATA and STROBE inputs as shown in the following Fig. Then the selected output line is in the low state. For example, if the binary input is ABCD = 0111, then the Y_7 output is low, while all other outputs are high.



Using 74154 as Decoder

Chip Expansion: The following Fig illustrates *chip expansion*. Here, we have expanded two 74154s to get a 1-of-32 decoder. Bit X drives the first 74154, and the complement of X drives the second 74154. When X is low, the first 74154 is active and the second is inactive. The ABCD input drives both decoders.

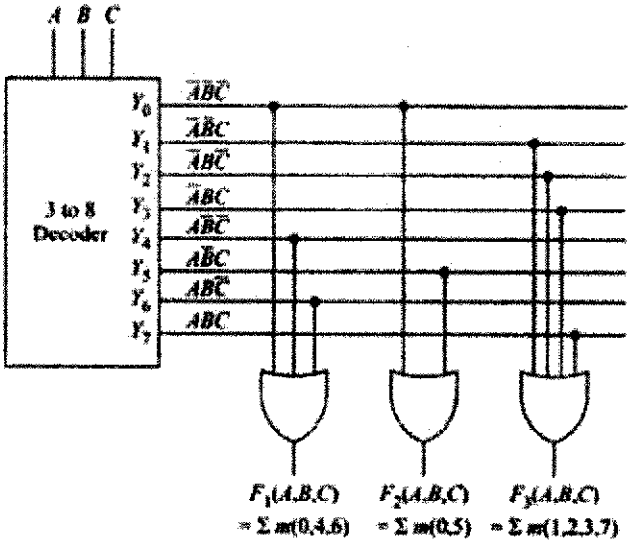


Chip Expansion

Problem: *Show how using a 3-to8 decoder and multi-input OR gates following Boolean expression can be realized simultaneously.*

$F_1 (A, B, C) = \sum m (0, 4, 6)$ $F_2 (A, B, C) = \sum m (0, 5)$ $F_3 (A, B, C) = \sum m (1, 2, 3, 7)$.

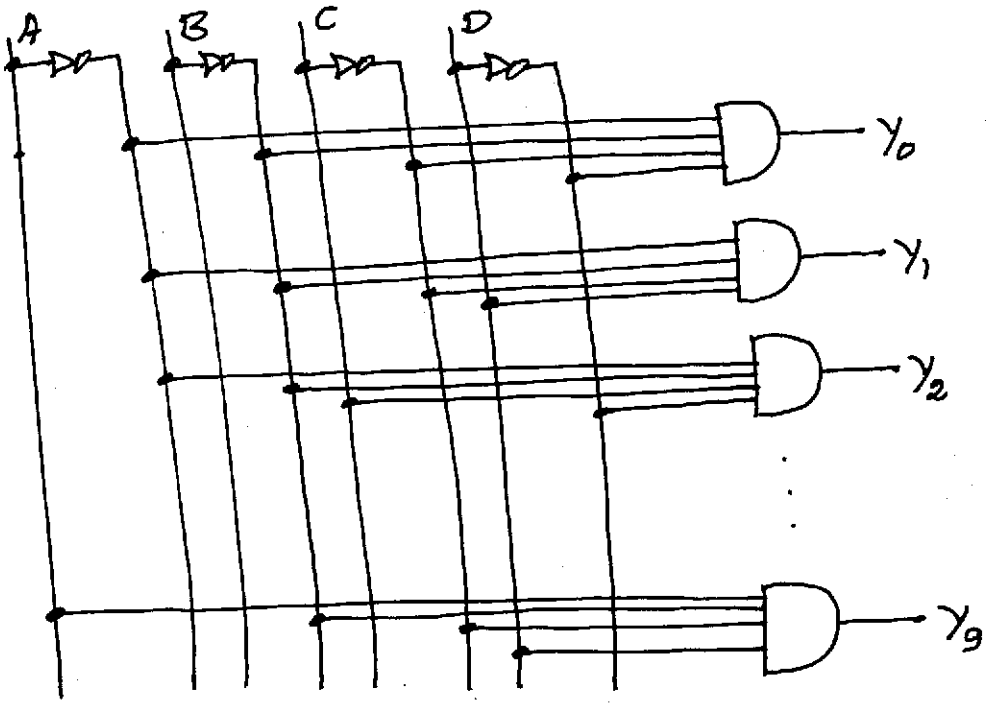
Solution: Since, at the decoder output, we get all the min-terms, we use them as shown in the following Fig, to get the required Boolean expression:



BCD-TO-DECIMAL DECODER:

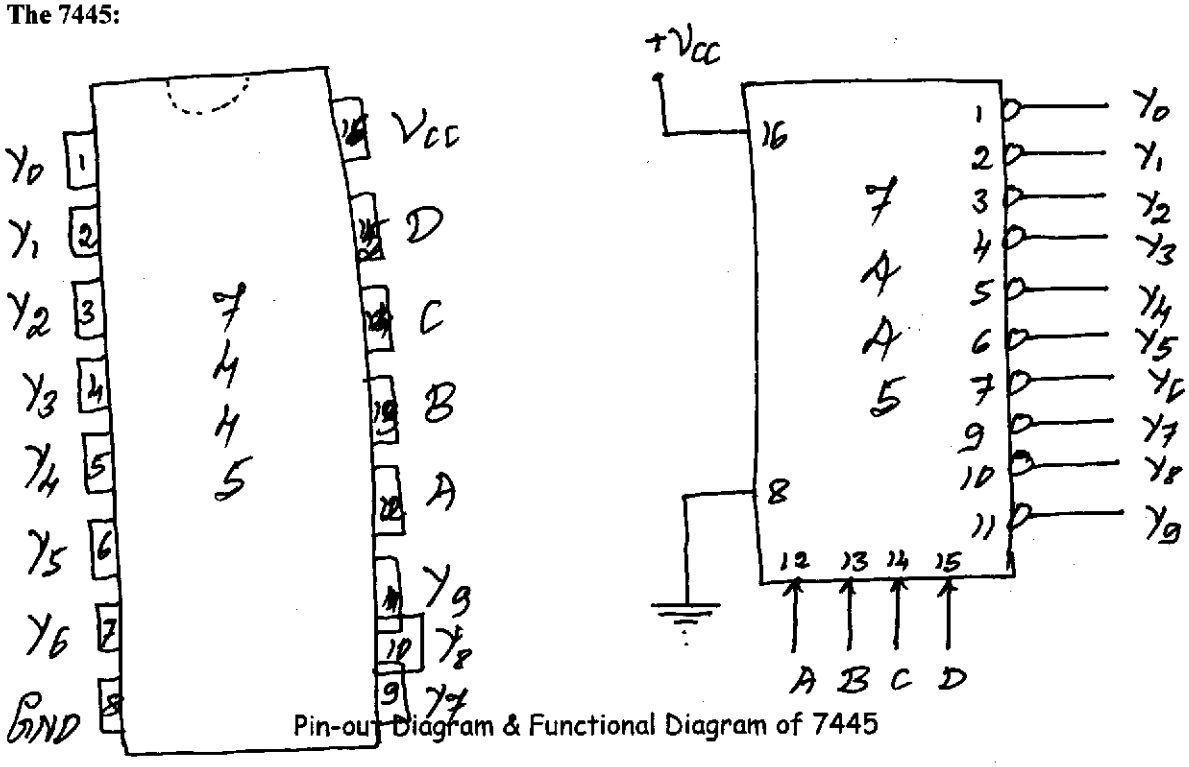
The circuit of the following Fig is called a *1-of-10 decoder*, because, only 1 of the 10 output lines is high. For example, when ABCD is 0011, only the Y_3 AND gate has all high inputs; therefore, only the Y_3 output is high.

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*.



1-of-10 decoder

The 7445:



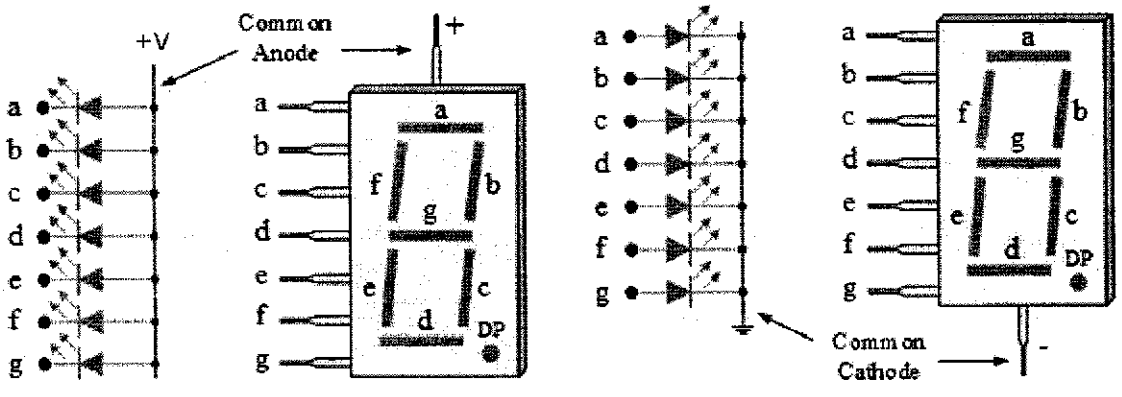
Pin-out Diagram & Functional Diagram of 7445

No.	Inputs				Outputs									
	A	B	C	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	Y_8	Y_9
0	L	L	L	L	*L*	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	*L*	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	*L*	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	*L*	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	*L*	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	*L*	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	*L*	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	*L*	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	*L*	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	*L*
	H	L	H	L	H	H	H	H	H	H	H	H	H	H
	H	L	H	H	H	H	H	H	H	H	H	H	H	H
	H	H	L	L	H	H	H	H	H	H	H	H	H	H
	H	H	L	H	H	H	H	H	H	H	H	H	H	H
	H	H	H	L	H	H	H	H	H	H	H	H	H	H
	H	H	H	H	H	H	H	H	H	H	H	H	H	H

7445 Truth Table

SEVEN-SEGMENT DECODERS:

The following Fig shows a *seven-segment indicator*, i.e. seven LEDs labeled *a* through *g* (actually, eight LEDs labeled through *a* through *h*). By forward biasing the LEDs, we can display the digits 0 through 9. For example, to display the digit 0, we need to light-up the segments *a*, *b*, *c*, *d*, *e*, and *f*. Similarly, to light-up the digit 5, we need segments *a*, *c*, *d*, *f*, and *g*.



Seven-Segment Indicator: Common Anode Type & Common Cathode Type

Seven-segment indicators may be *common-anode* type; where all anodes are connected together (as shown above) or *common-cathode* type; where all cathodes are connected together (as shown above).

CA: Low input for bright LED.

dp g f e d c b a

0: 1 1 0 0 0 0 0 0 = C0H

3: 1 0 1 1 0 0 0 0 = B0H

8: 1 0 0 0 0 0 0 0 = 80H

CC: High input for bright LED.

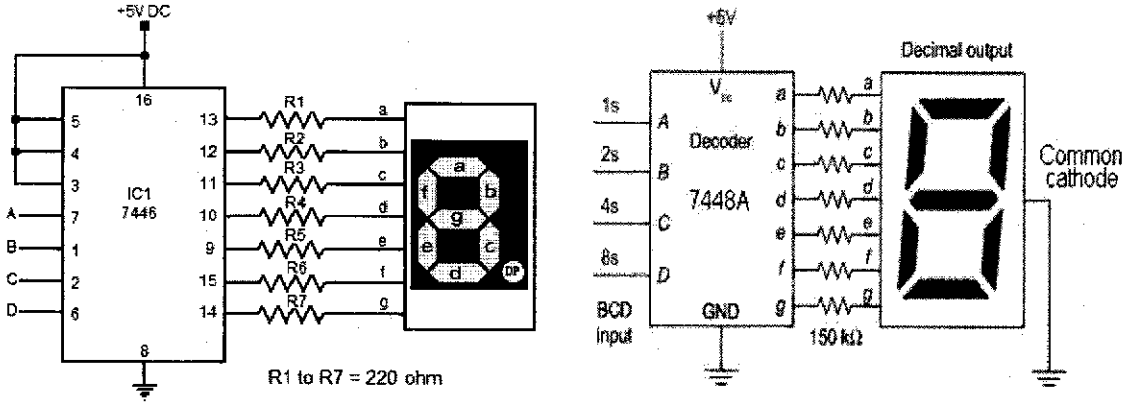
dp g f e d c b a

0: 0 0 1 1 1 1 1 1 = 3FH

3: 0 1 0 0 1 1 1 1 = 4FH

8: 0 1 1 1 1 1 1 1 = 7FH

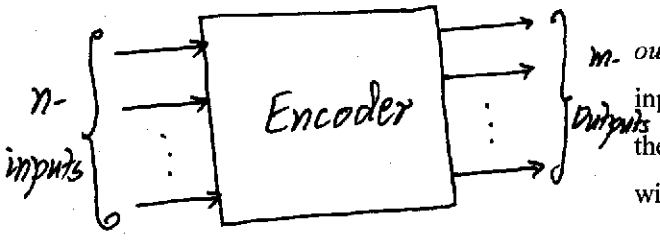
The 7446 & The 7448: A *seven-segment decoder-driver* is an IC decoder that can be used to drive a seven-segment indicator. There are two types of decoder-drivers, corresponding to common-anode (IC 7446) and common cathode (IC 7448) indicators. Each decoder driver has 4 input pins (the BCD input) and 7 output pins (*a* through *h* segments), as shown in the following Fig.



7446 Decoder-driver & 7448 Decoder-driver

The logic circuits inside 7446 / 7448 convert the BCD input to the required output. For Example, if the BCD input is 0111, the internal logic of the 7446 / 7448 will force segments *a*, *b*, and *c* to conduct. As a result, digit 7 will appear on the seven-segment indicator.

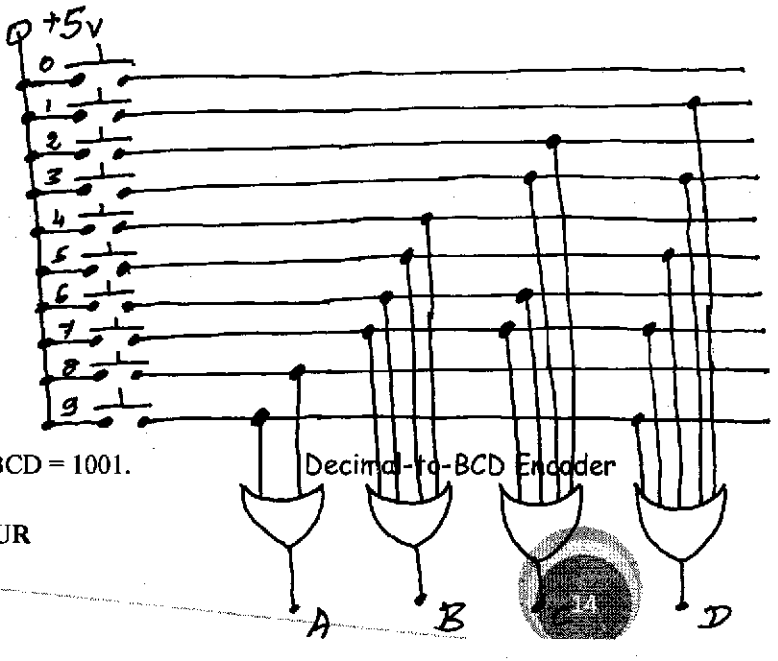
ENCODERS:



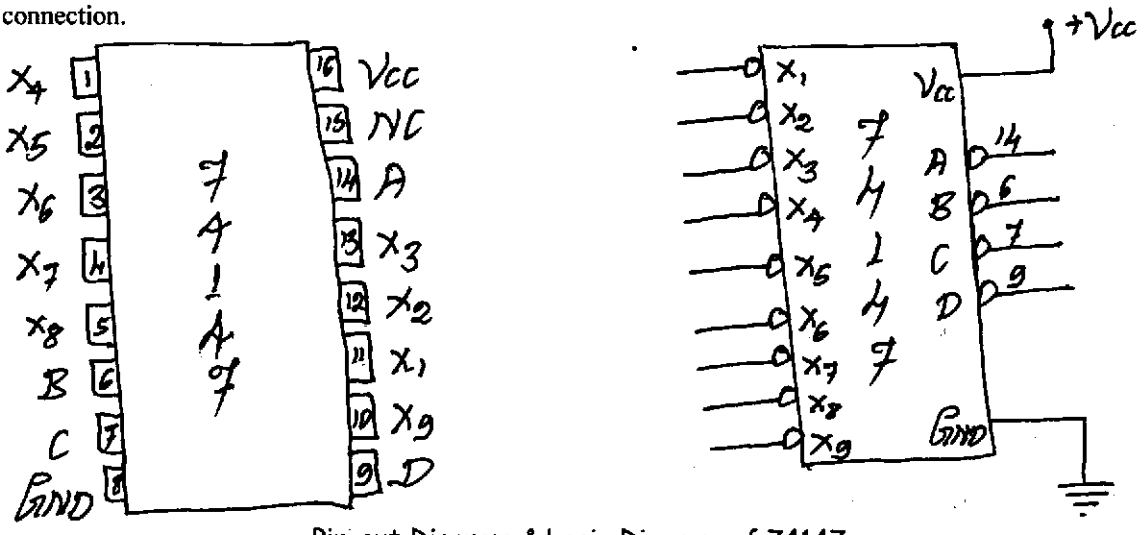
An encoder converts an active input signal to a coded output signal. The Fig illustrates the general idea. There are *n* input lines, only one of which is active. The internal logic within the encoder converts this active input to a coded binary output with *m* bits.

Decimal-to-BCD Encoder:

The following Fig shows a common type of encoder – the decimal-to-BCD encoder. The switches are push-button switches. When 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.



74147: The following Fig shows the pin-out and the logic diagram of a 74147- a decimal-to-BCD encoder. The decimal inputs, X_1 to X_9, connect to pins 1 to 5, and 10 to 13. The BCD output comes from pins 14, 6, 7, and 9. Pin 16 is for supply voltage, and pin 8 is grounded. The label NC on pin 15 means no connection.



Pin-out Diagram & Logic Diagram of 74147

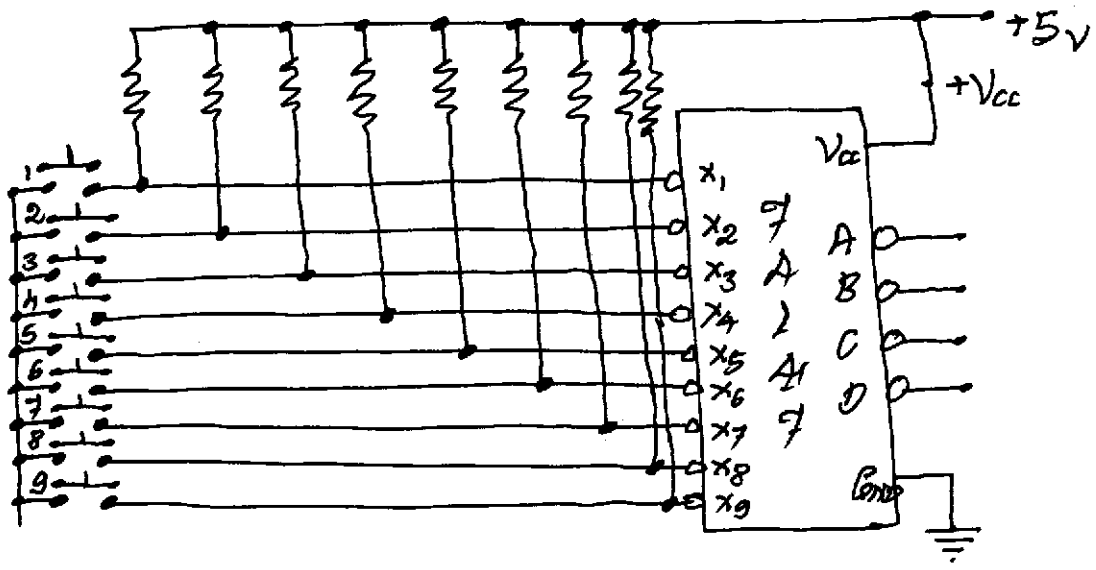
The bubbles indicate active-low inputs and outputs. The following Table is the truth table of 74147. When all X inputs are high, all outputs are high. When X_9 is low, the ABCD output is LHHL (equivalent to 9 if you complement the bits). When X_8 is the only low input, ABCD is LHHH (equivalent to 8 if the bits are complemented).

Inputs									Outputs			
X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	A	B	C	D
H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	*L*	*L*	H	H	*L*
H	H	H	H	H	H	H	*L*	H	*L*	H	H	H
H	H	H	H	H	H	*L*	H	H	H	*L*	*L*	*L*
H	H	H	H	H	*L*	H	H	H	H	*L*	*L*	H
H	H	H	H	*L*	H	H	H	H	H	*L*	H	*L*
H	H	H	*L*	H	H	H	H	H	H	*L*	H	H
H	H	*L*	H	H	H	H	H	H	H	H	*L*	*L*
H	*L*	H	H	H	H	H	H	H	H	H	*L*	H
L	H	H	H	H	H	H	H	H	H	H	H	*L*

74147 Truth Table

The 74147 is called a *priority encoder*, because it gives priority to the highest-order input. If all inputs X_1 through X_9 are low, the highest of these, X_9, will be encoded to get an output LHHL. In other words, X_9 has priority over all others. When X_9 is high, X_8 is the next inline of priority and gets encoded if it is low.

Problem: *What is the ABCD output of the following Fig, when button 6 is pressed?*



Solution:

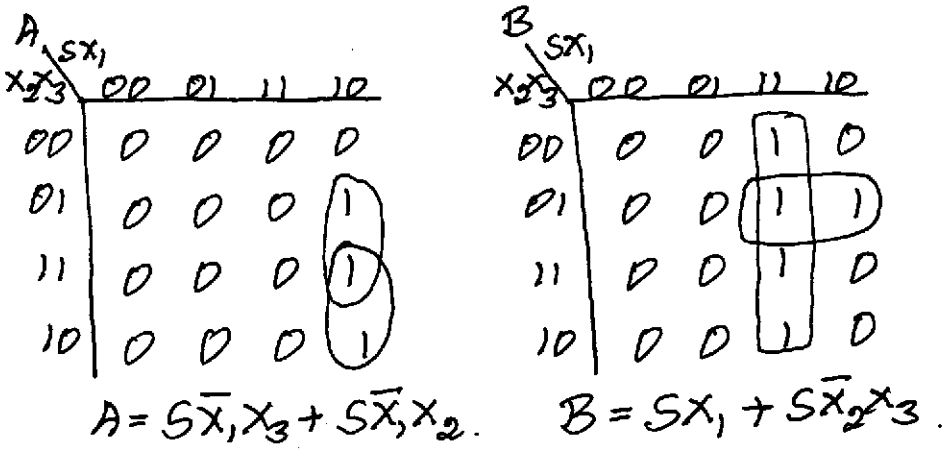
When all switches are open, the X_1 to X_9 inputs are pulled up to the high state (+5 V). Hence, ABCD output is HHHH at this time.

When switch 6 is pressed, the X_6 input is grounded. Therefore, all X inputs are high, except for X_6. Hence, the ABCD output is HLLH, which is equivalent to 6 when the output bits are complemented.

Problem: *Design a priority encoder, the truth table of which is shown in the following Fig. The order of priority for these inputs is $X_1 > X_2 > X_3$. However, if the encoder is not enabled by S or all the inputs are inactive, the output AB = 00.*

Solution:

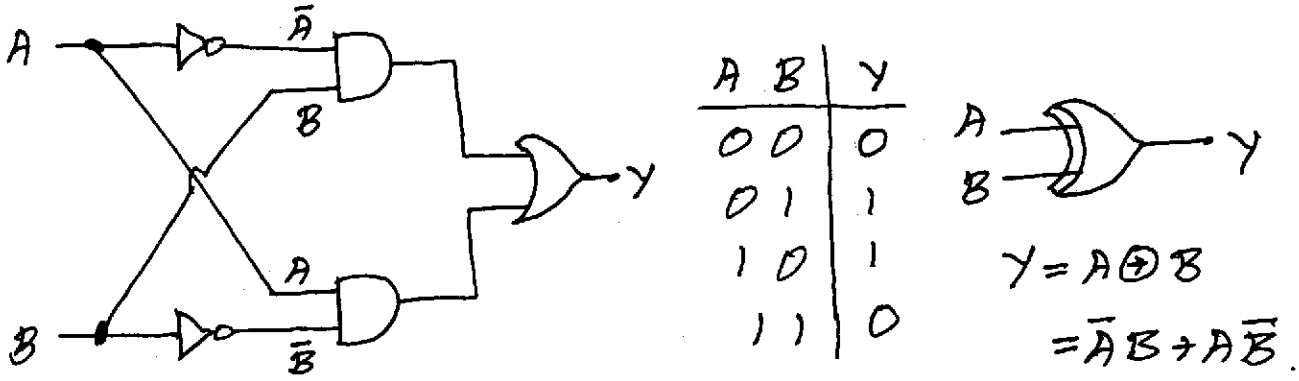
Inputs				Outputs	
S	X_1	X_2	X_3	A	B
0	x	x	x	0	0
1	1	x	x	0	1
1	0	1	x	1	0
1	0	0	1	1	1
1	0	0	0	0	0



Construct Karnaugh map for output A and B. Then, taking groups of 1s, we get the design equations, as shown above. The logic circuits for input A and B can be directly drawn from the equations.

EXCLUSIVE-OR (Ex-OR) GATES:

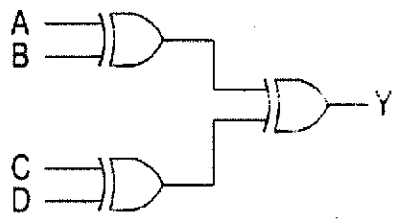
The *exclusive-OR gate* has a high output, when an odd number of inputs are high. The following Fig shows how to build an exclusive-OR gate. The upper AND gate forms the product A'B, while the lower one produces AB'. Therefore, the output of the OR gate is $Y = \bar{A}B + A\bar{B}$.



Exclusive-OR Gate, Its Logic Symbol & Truth Table

Problem: *Construct the truth table for a 4 input XOR gate.*

Solution:



The gate produces an output 1, when the ABCD input has an odd number of 1s.

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

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

PARITY GENERATORS AND CHECKERS:

Parity is the number of 1s in an *n*-bit input. *Even parity* means an *n*-bit input has an even number of 1s. *Odd parity* means an *n*-bit input has an odd number of 1s.

1111 0000 1111 0011 even parity

1111 0000 1111 0111 odd parity

Parity Checker:

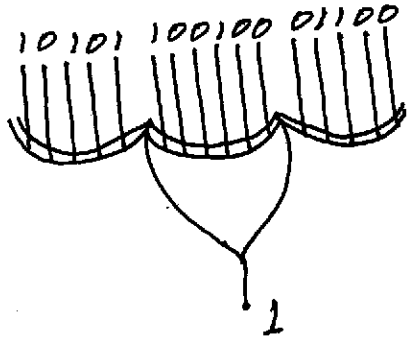
Exclusive-OR gates are ideal for checking the parity of a binary number, because they produce an output 1 when the input has an odd number of 1s. Therefore, an even-parity input to an exclusive-OR gate produces a low output, while an odd-parity input produces a high output.

The following Fig shows a 16-input exclusive-OR gate,

which produces an output 1, because the input has odd parity.

1111 0000 1111 0011 : even parity

1111 0000 1111 0111 : odd parity



Exclusive-OR gate with 16 Inputs

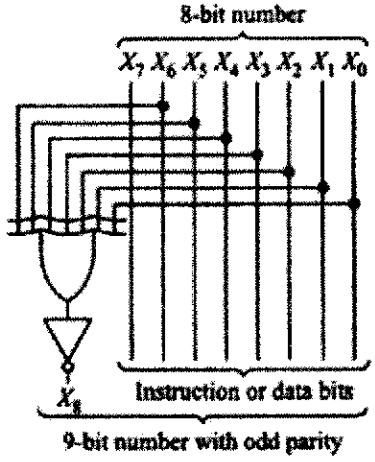
Parity Generation

In a computer, a binary number may represent –

1. An instruction that tells the computer to add, subtract, and so on or
2. A data to be processed like number, letter, etc.

In both the cases, sometimes, an extra bit will be added to the original binary number to produce a new binary number with even or odd parity.

Consider the following Fig:



Odd-Parity Generation

The Fig shows an 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 of 0. Because of the inverter, $X_8 = 1$. Hence, the final 9-bit output is 1 0100 0001. Notice that, this has odd-parity.

Suppose, we change the 8-bit input to 0110 0001. Now, it has odd-parity. In this case, the exclusive-OR gate produces an output 1. But the inverter produces a 0, so that the final 9-bit output is 0 0110 0001. Again, the final output has odd-parity.

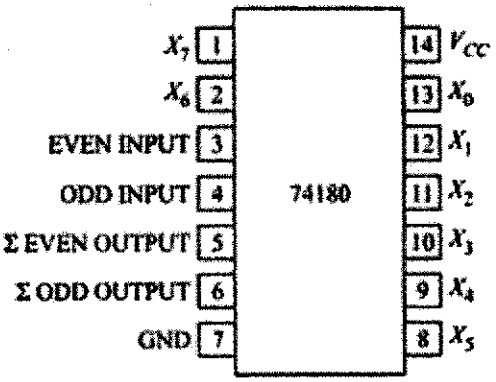
Hence, the circuit given above is called an *odd-parity generator*, because it always produces a 9-bit output number with odd-parity. If the 8-bit has even-parity, a 1 comes out of the inverter to produce the final output with odd-parity. On the other hand, if the 8-bit input has odd-parity, a 0 comes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

NOTE: To get an even-parity generator, delete the inverter.

Applications of parity generation and checking: When binary data is transmitted over telephone lines, sometimes, 1-bit error occurs due to noise and other disturbances. 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 the transmission, the received data will have odd parity. But, if one of the transmitted bits is changed by noise or other disturbance, the received data will have even-parity.

For example; 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. If no error occurs in transmission, the odd-parity checker at the receiving end will produce a high output, meaning the received number has odd-parity. On the other hand, if a 1-bit error does creep into the transmitted data, the odd-parity checker will have a low output, indicating the received data is invalid.

The 74180: The following Fig shows the pin-out diagram and truth table for a 74180, a TTL parity generator-checker. The input data bits are X_7 to X_0; these bits may have even or odd parity. The even input (pin 3) and the odd input (pin 4) control the operation of the chip. The symbol $\sum$ stands for summation. In the left input column of the table, $\sum$ of H's (highs) refers to the parity of the input data X_7 to X_0. Depending on how you set up the values of the even and odd inputs, the $\sum$ even and $\sum$ odd outputs may be low or high.



Inputs			Outputs	
Sum of H's of X_7 to X_0	Even	Odd	$\sum$ even	$\sum$ odd
Even	H	L	H	L
Odd	H	L	L	H
Even	L	H	L	H
Odd	L	H	H	L
X	H	H	L	L
X	L	L	H	H

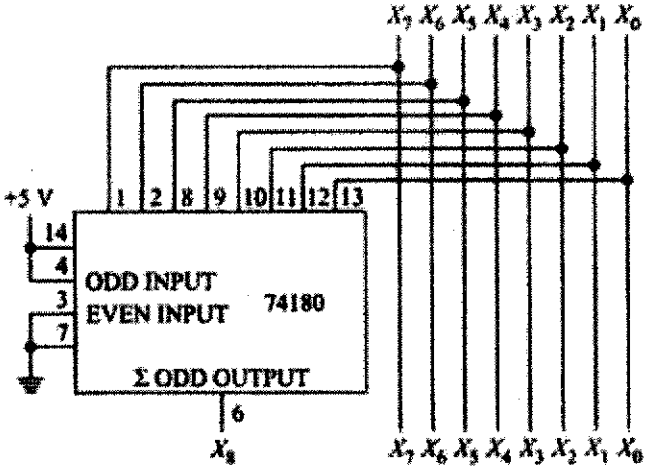
Pin-out Diagram & Truth Table of 74180

example, suppose even input is high and odd input is low. When the input data has even-parity, the ∑ even output is high and ∑ odd output is low. When the input data has odd-parity, the ∑ even output is low and the ∑ odd output is high.

The 74180 can be used to detect even or odd parity. It can also be set up to generate even or odd parity.

Problem: *Show how to connect a 74180 to generate a 9-bit output with odd parity.*

Solution:



Using a 74180 to Generate Odd-Parity

The ODD INPUT (pin 4) is connected to +5 V, and the EVEN INPUT (pin 3) is grounded. Suppose the input data $X_7 \ldots X_0$ has even-parity. Then, the 3rd entry of the above Table, tells us, the ∑ ODD OUTPUT (pin 6) is high. Therefore, the 9-bit number $X_8 \ldots X_0$ coming out of the circuit has odd parity.

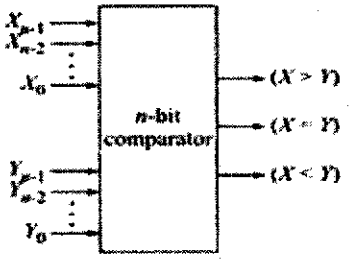
On the other hand, suppose the input data $X_7 \ldots X_0$ has odd-parity. Then, the 4th entry of the above Table, tells us, the ∑ ODD OUTPUT (pin 6) is low. Again, the 9-bit number $X_8 \ldots X_0$ coming out of the circuit has odd parity.

Hence, the circuit shown in the above Fig, always generates a 9-bit output with odd parity.

MAGNITUDE COMPARATOR:

Magnitude comparator compares magnitude of two n-bit numbers, say X and Y, and activates one of these three outputs X < Y, X = Y, and X > Y.

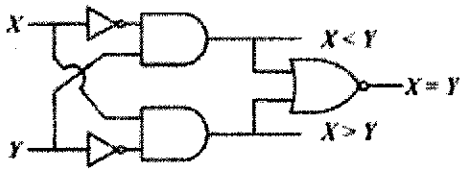
The Fig presents block diagram of such a comparator.



Block Diagram of Magnitude Comparator

Bit Comparator:

Inputs			Outputs	
X	Y	(X > Y) G	(X < Y): L	(X = Y): E
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1
Logic Equations:		G = XY'	L = X'Y	E = X'Y' + XY = (XY' + X'Y)' = (G L)'



Truth Table, Logic Equations & Circuit Diagram for a 1-Bit Comparator

2-Bit Comparator: We can form a 4-variable (X: X_1X_0 and Y: Y_1Y_0) truth table and get logic equations through and simplification technique. But this procedure will become very complex. Hence, we shall use the truth table of 1-bit comparator that generates less than, equal to, and greater than terms. Hence;

Bit-wise greater than (G): $G_1 = X_1\bar{Y}_1$ $G_0 = X_0\bar{Y}_0$

Bit-wise less than (L): $L_1 = \bar{X}_1Y_1$ $L_0 = \bar{X}_0Y_0$

Bit-wise equality term (E): $E_1 = \overline{(G_1 \quad L_1)}$ $E_0 = \overline{(G_0 \quad L_0)}$

From these definitions; we can easily write, 2-bit comparator outputs as follows:

(X = Y) if $X_1 = Y_1$ & $X_0 = Y_0$ $(X = Y) = E_1.E_0$

(X > Y) if $X_1 > Y_1$ or $X_1 = Y_1$ & $X_0 > Y_0$ $(X > Y) = G_1 + E_1.G_0$

(X < Y) if $X_1 < Y_1$ or $X_1 = Y_1$ & $X_0 < Y_0$ $(X < Y) = L_1 + E_1.L_0$

The logic followed is this: X = Y, when both the bits are equal. X > Y if MSB of X is higher ($G_1 = 1$) than that of Y. If MSB is equal (given by $E_1 = 1$), then LSB of X and Y is checked and if found higher ($G_0 = 1$), the condition X > Y is fulfilled. Similar logic gives us the X < Y term.

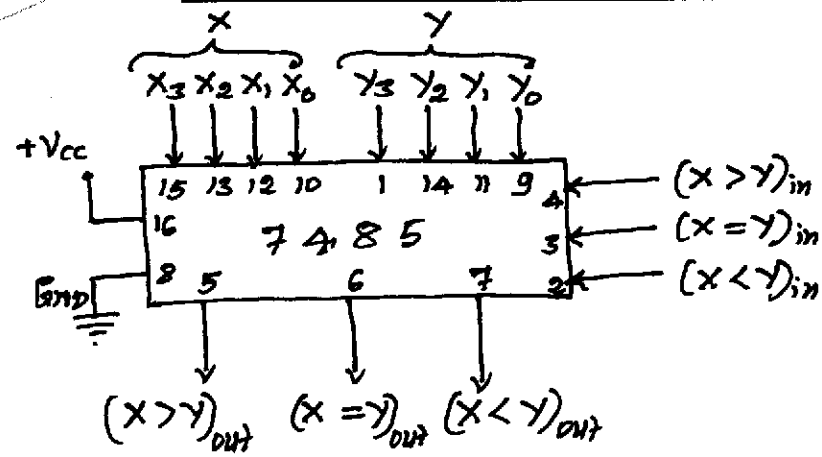
Thus for nay two *n*-bit numbers X: $X_{n-1}\ X_{n-1} \ldots X_0$ and Y: $Y_{n-1}\ Y_{n-2} \ldots Y_0$ We can write –

$(X = Y) = E_{n-1}.E_{n-2} \ldots E_0$

$(X > Y) = G_{n-1} + E_{n-1}.G_{n-2} + \ldots + E_{n-1}.E_{n-2} \ldots E_1.G_0$

$(X < Y) = L_{n-1} + E_{n-1}.L_{n-2} + \ldots + E_{n-1}.E_{n-2} \ldots E_1.L_0$

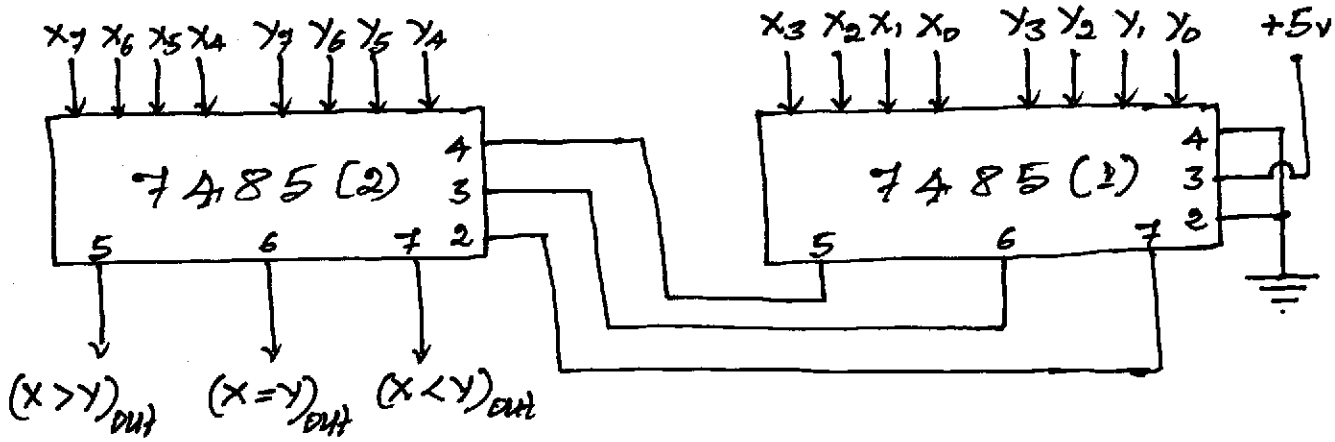
The 7485: The block diagram of IC 7485, which compares two 4-bit numbers, is shown in the following Fig. This is a 16-pin IC. Note that the IC has three additional inputs in the form of $(X = Y)_{in}$, $X > Y)_{in}$, and $(X < Y)_{in}$. These inputs are used when we need to cascade more than one IC 7485 to compare numbers having more than 4-bits. When IC 7485 is not used in cascade, we keep $(X = Y)_{in} = 1$, $X > Y)_{in} = 0$, and $(X < Y)_{in} = 0$.



Functional Diagram of IC 7485

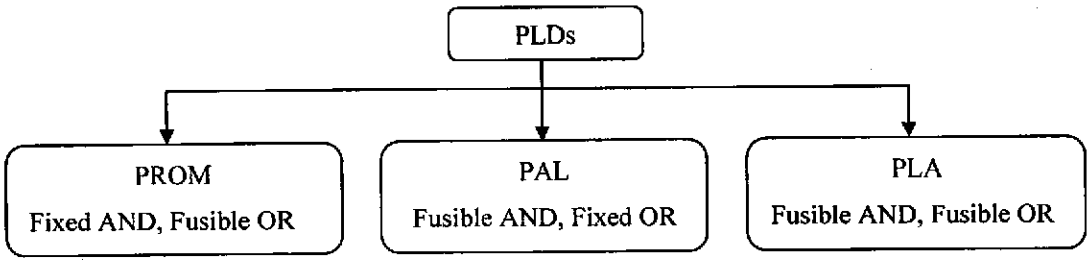
Problem: *Show how two IC 7485 can be used to compare magnitude of two 8-bit numbers.*

Solution:

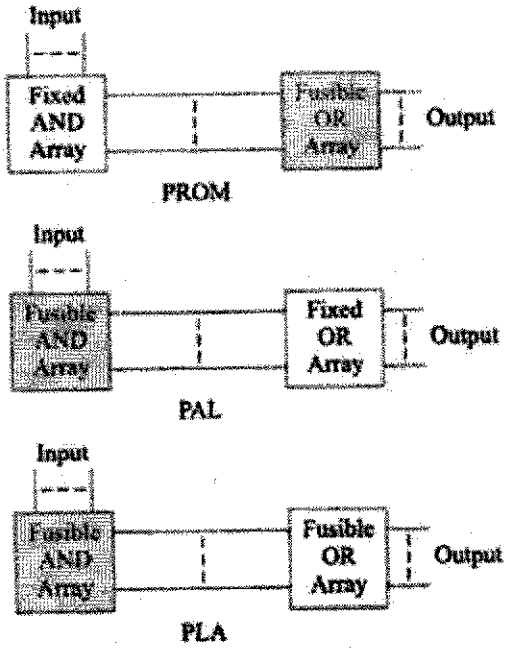


8-Bit Comparator from Two 4-Bit Comparators

NOTE: *Programmable Logic Device (PLD)* is an electronic component, used to build reconfigurable digital circuits. Programmable Read Only Memory (PROM), Programmable Array Logic (PAL), and Programmable Logic Array (PLA) are included in the general classification.



General Classification of PLDs



Basic Operation of Three PLDs

In PLDs, the input signals are presented to an array of AND gates, while the outputs are taken from an array of OR gates.

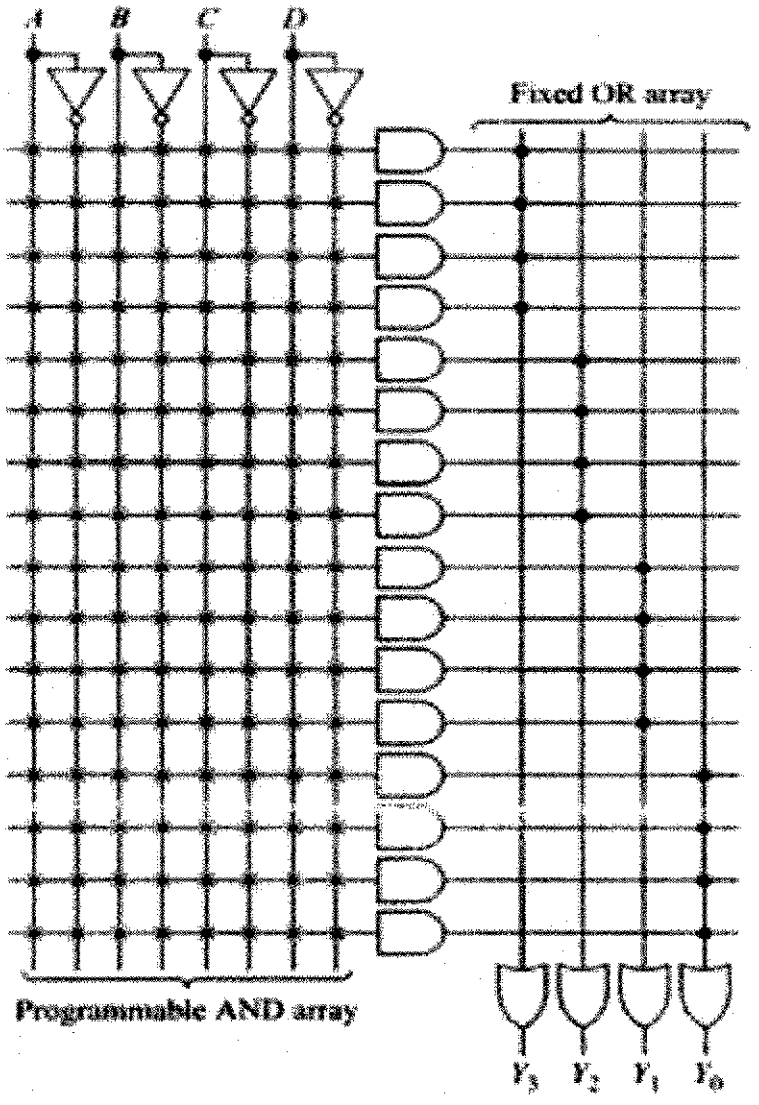
- The input AND gate array used in a PROM is fixed and cannot be altered, while the output OR gate array is fusible-linked, and can be programmed.
- The output OR gate array used in a PAL is fixed and cannot be altered, while the input AND gate array is fusible-linked, and can be programmed.
- In PLA, both its input AND gate array and output OR gate array are fusible-linked, and can be programmed.

PROGRAMMABLE ARRAY LOGIC (PAL):

Programmable Array Logic (PAL) is a programmable array of logic gates on a single chip. PALs are another design solution, similar to SOPs, POSs, and multiplexer logic.

A PAL has a programmable AND array and a fixed OR array. The following Fig shows a PAL with 4 inputs and 4 outputs. The x's on the input side are fusible links, while the solid bullets on the output side are fixed connections.

Commercially available PALs: 10H8 – 10 input and 8 output AND-OR
16H2 – 16 input and 2 output AND-OR
14L4 – 14 input and 4 output AND-OR-INVERT



NOTE:

1. PALs are not universal logic solution; because, only some of the fundamental products can be generated and ORed at the final outputs.
2. PALs have enough flexibility to produce all kinds of complicated logic functions.
3. PALs have the advantage of 16 inputs compared to typical limit of 8 inputs for PROMs.

Problem: *Generate the following Boolean function using PAL.*

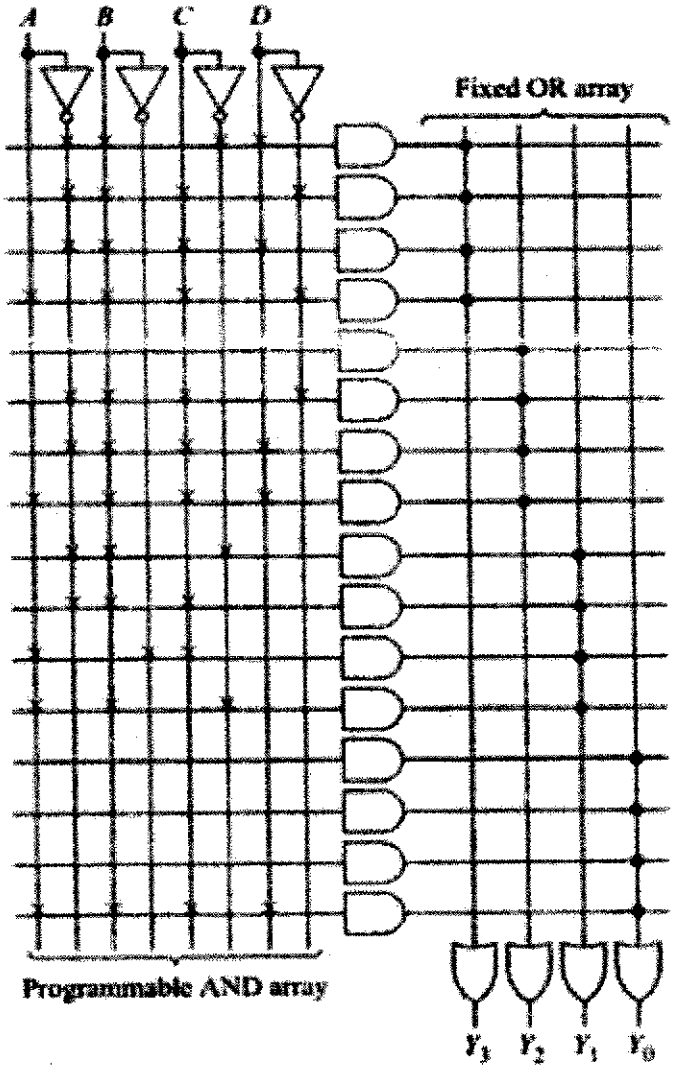
$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D}$$

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

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

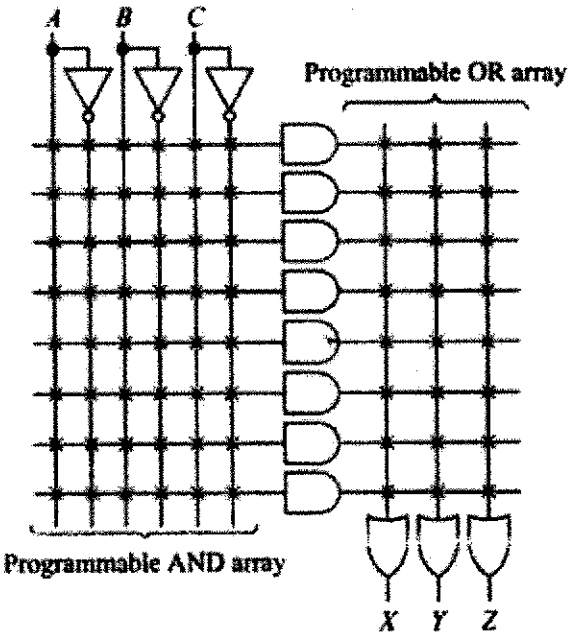
$$Y_0 = ABCD$$

Solution: Start with first equation. The first desired product is A'BC'D, which is marked as shown in the following Fig. The fixed OR connections on the output side imply that the first OR gate produces an output of first equation.

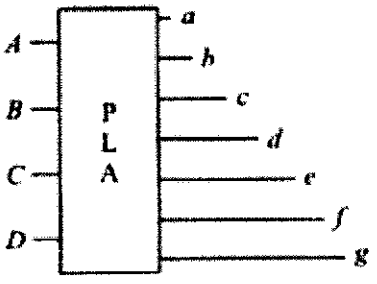


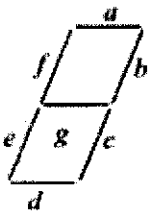
PROGRAMMABLE LOGIC ARRAY (PLA):

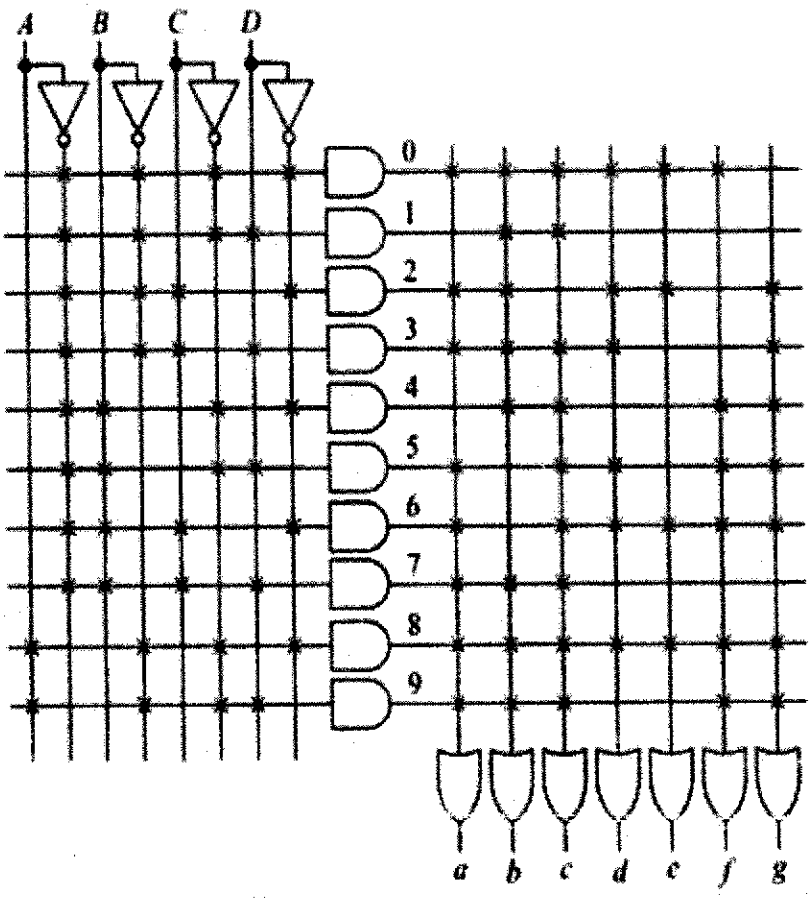
A PLA having 3 input variables (ABC) and 3 output variables (XYZ) is illustrated in the following Fig. Eight AND gates are required to decode the 8 possible input states. There are three OR gates that can be used to generate logic functions at the output. Note that, there could be additional OR gates at the output, if desired.



Example: Suppose it is desired to use a PLA to recognize each of the 10 decimal digits represented in binary form and to correctly drive a 7-segment display. Then, the PLA must have 4 inputs, as shown in the following Fig. Four bits are required to represent the 10 decimal numbers. There must be 7 outputs (*abcdefg*).







7-Segment Decoder using PLA

The circuit given in the above Fig shows the links after programming. The input AND-gate array is programmed such that, each AND gate decodes one of the decimal numbers. Then, links are removed from the output OR-gate array, such that proper segments of the indicator are illuminated. For example, when ABCD = LHLH, segments *afgcd* are illuminated to display the decimal number 5.

NOTE: Many PLDs are programmable only at the factory. However, PLDs can be programmed by the user. These are said to be *field-programmable*, and the letter F is often used to indicate this fact. For example, the Texas Instruments TIFPLA840 is field-programmable PLA with 14 inputs, 32 AND gates, and 6 OR gates; describes as 14 x 32 x 6 FPLA.

HDL IMPLEMENTATION OF DATA PROCESSING CIRCUITS:

The data flow model provides a different use of keyword ***assign*** in the form of –

Recall:

Structural ↔ Gates / Circuits

Dataflow ↔ ***assign*** / Expression

Behavioral ↔ ***always*** / Truth table

assign X = S ? A : B;

If S = 1, X = A, and if S = 0, X = B.

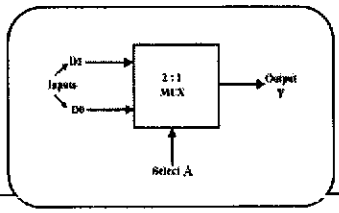
2-to-1 Multiplexer: The data flow model and behavioral model for a 2-to-1 multiplexer is given below:

```
module mux2to1(A,D0,D1,Y);
  input A,D0,D1; /* Circuit shown */
  output Y;
  assign Y=(~A&D0)|(A&D1);
endmodule
```

```
module mux2to1(A,D0,D1,Y);
  input A,D0,D1; /* Circuit shown */

  output Y;
  assign Y= A ? D1 : D0; /*Conditional
   assignment*/
endmodule
```

```
module mux2to1(A,D0,D1,Y);
  input A,D0,D1; /* Circuit shown*/

  output Y;
  reg Y;
  always @ (A or D0 or D1)
     if (A==1) Y=D1;
     else Y=D0;
  endmodule
```

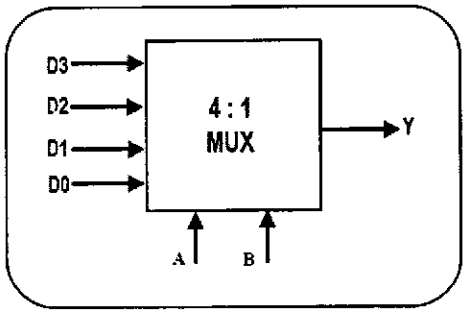
```
module mux2to1(A,D0,D1,Y);
  input A,D0,D1; /* Circuit shown */

  output Y;
  reg Y;
  always @ (A or D0 or D1)
     case (A)
        0 : Y=D0;
        1 : Y=D1;
     endcase
endmodule
```



Problem: *Design a 4 –to-1 multiplexer (for the Fig given below), using conditional* ***assign*** *and case statements.*

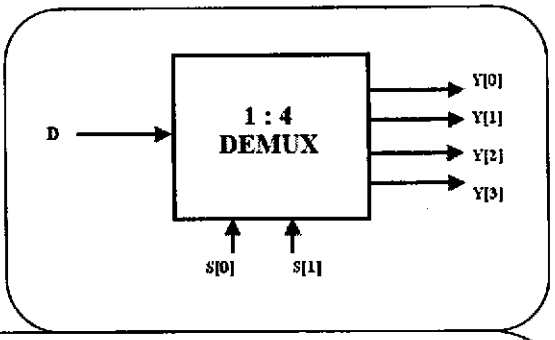
Solution:

```
module mux4to1(A,B,D0,D1,D2,D3,Y);
  input A,B,D0,D1,D2,D3;
  output Y; /* Circuit shown */

  assign  Y = A ?( B ? D3 : D2):(B ?
  D1 : D0);
endmodule
```




```
module mux4to1(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)) /*Concatenation of A
     and B, A is MSB*/
     0:Y=D0; /*Two binary digit can
        generate*/
     1:Y=D1; /*four different values
        0,1,2,3 for*/
     2:Y=D2; /*binary combination
        00, 01,10*/
     3: Y=D3; //and 11 respectively
   endcase
endmodule
```

Representation in HDL: A 1-to-4 demultiplexer can also be served as 2-to-4 decoder. The Verilog code for this demultiplexer / decoder is given below:




```
module demux1to4(S,D,Y);
  input [1:0] S;
  input  D;
  output [3:0] Y;
  reg [3:0] Y;
  always @ (S or D)
     case ({D,S}) //Concatenation of D  and S to give 3 bits, D is MSB
        3'b100 : Y= 4'b0001; /*Binary representation, refer to Section 2-5.
       If D=1, S=00, Y=0001*/
         3'b101 : Y= 4'b0010; // if D=1, S=01, Y=0010
         3'b110 : Y= 4'b0100; // if D=1, S=10, Y=0100
         3'b111 : Y= 4'b1000; // if D=1, S=11, Y=1000
          default : Y= 4'b0000; //For other combinations D=0, then Y=0000
    endcase
  endmodule
```

Problem: *A Verilog HDL code for a digital circuit is given as follows. Can you describe the function it performs? Can it be related to any logic circuit?*

```
module unknown (A, B, C, Y);
        input [3:0] A, B;
        input [2:0] C
        output [2:0] Y
        reg [2:0] Y;
        always @ (A or B or C)
                if (A<B) Y = 3'b001;
                else if (A>B) Y = 3'010;
                else Y = C;
endmodule
```

Solution: The circuit described by the HDL compares two numbers A and B and generates a 3-bit output Y. It also has a 3-bit input C. If A is less than B, output Y = 001 and does not depend on C. Similarly, if A is greater than B, output Y = 010, irrespective of C. But, if these two conditions are not met, i.e. if A = B, then Y = C.

If we consider three bits of Y represent (starting from MSB) A = B, A > B, and A < B respectively then, this circuit represents a 4-bit magnitude comparator, where C represents comparator output of previous stage. Hence, this is similar to IC ~~7458~~ 7485.

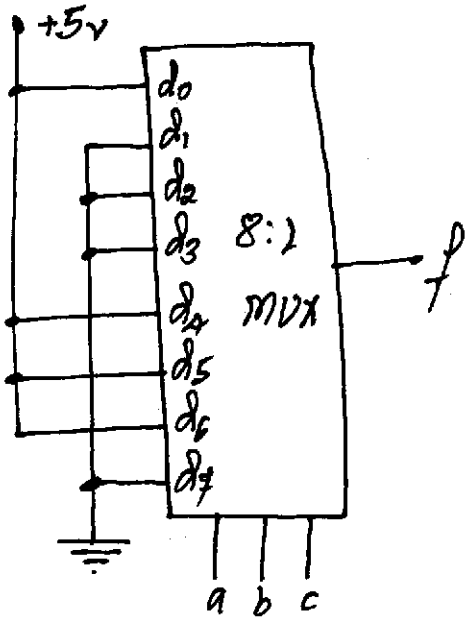
Problem: *Implement the following functions using 8:1 Mux*

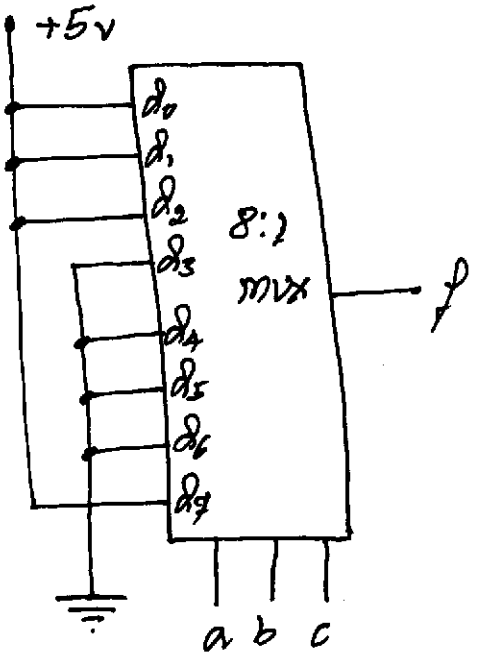
a) f (a, b, c) = ∑ (0, 4, 5, 6)

b) f (a, b, c) = ∑ (0, 1, 2, 7)

HW *c) f (a, b, c) = ∑ (0, 4, 5, 7)*

Solution:





Problem: *Implement the following functions using 4:1 Mux*

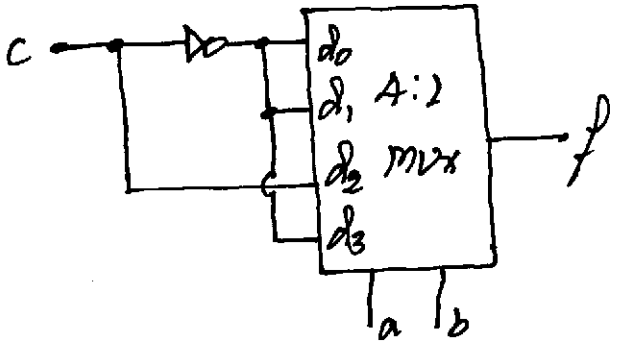
a) f (a, b, c) = ∑ (0, 4, 5, 6)

b) f (a, b, c) = ∑ (0, 1, 2, 7)

HW *c) f (a, b, c) = ∑ (0, 4, 5, 7)*

Solution:

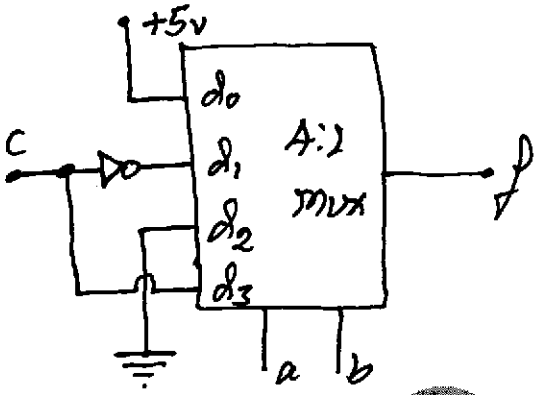
(a) $f = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c}$



(b) $f = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc$

$= \bar{a}\bar{b}(\bar{c}+c) + \bar{a}b(\bar{c}) + ab(c)$

$= \bar{a}\bar{b}\cdot 1 + \bar{a}b(\bar{c}) + a\bar{b}(0) + ab(c)$



Problem: *Implement f (a, b, c) = ∑ (1, 4, 5, 7) using 4:1 Mux, taking –*

a) *a and b as address lines*

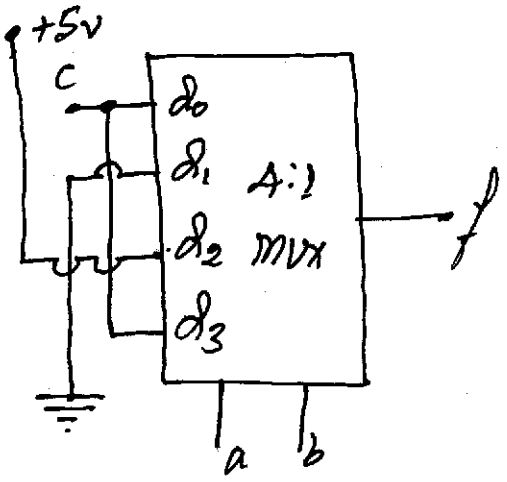
b) *b and c as address lines.*

Solution:

ⓐ $f = \bar{a}\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c + abc$

$= \bar{a}\bar{b}(c) + a\bar{b}(\bar{c}+c) + ab(c)$

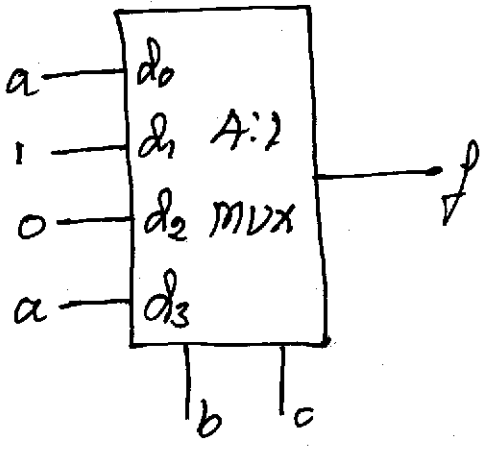
$= \bar{a}\bar{b}(c) + \bar{a}b(0) + a\bar{b}(1) + ab(c)$



ⓑ $f = \bar{a}\bar{b}c + a\bar{b}\bar{c} + a\bar{b}c + abc$

$= \bar{b}\bar{c}(a) + \bar{b}c(\bar{a}+a) + bc(a)$

$= \bar{b}\bar{c}(a) + \bar{b}c(1) + b\bar{c}(0) + bc(a)$



Home Work: *Implement f (a, b, c) = ∑ (0, 1, 2, 4) using b and c as address lines.*

Solution:

Problem: *Implement the Boolean function f (a, b, c, d) = ∑ (4, 5, 7, 8, 10, 12, 15) using 4:1 Mux and external gates if –*

a) a and b are connected to select lines

b) c and d are connected to select lines.

Solution:

(a) $f = \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}cd + ab\bar{c}\bar{d} + abcd.$

$= \bar{a}\bar{b}[0] + \bar{a}b[\bar{c}\bar{d} + \bar{c}d + cd] + a\bar{b}[\bar{c}\bar{d} + c\bar{d}] + ab[\bar{c}\bar{d} + cd]$

$= \bar{a}\bar{b}[0] + \bar{a}b[\bar{c} + d] + a\bar{b}[\bar{d}] + ab[\overline{c \oplus d}] \quad [\because \bar{c} + cd = \bar{c} + d].$

(b) $f = \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}cd + ab\bar{c}\bar{d} + abcd.$

$= \bar{c}\bar{d}[\bar{a}b + a\bar{b} + ab] + \bar{c}d[\bar{a}b] + c\bar{d}[a\bar{b}] + cd[\bar{a}b + ab]$

$= \bar{c}\bar{d}[a + b] + \bar{c}d[\bar{a}b] + c\bar{d}[a\bar{b}] + cd[b] \quad [\because a + \bar{a}b = a + b]$

Home Work: *Implement* $u = ad + b\bar{c} + bd$ *using –*

(i) 16:1 Mux *(ii) 8:1 Mux* *(iii) 4:1 Mux.*

Solution:

Try these: *(i) Design a full adder using a 4-to-1 multiplexer, a 2-to-1 multiplexer, and a decoder*

(ii) Design a full subtractor using a 4-to-1 multiplexer, a 2-to-1 multiplexer, and a decoder.

ANALOG AND DIGITAL ELECTRONICS

ARITHMETIC CIRCUITS

ARITHMETIC BUILDING BLOCKS:

Half-adder, Full-adder and Controller Inverter are three basic circuits that will be used as building blocks.

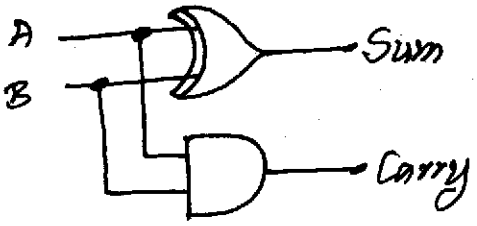
Half Adder: When we add two binary numbers, we start with the least significant column. This means that, we have to add two bits with the possibility of a carry. The circuit used for this is called a *half-adder*. The half-adder performs binary addition.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

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

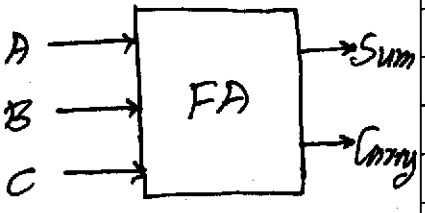
$= A \oplus B$

$Carry = AB$



Half-Adder Truth Table & Circuit Diagram

Full Adder: A logic circuit that can add three bits at a time is called a *full-adder*. The third bit is the carry from a lower column. Full-adder performs binary addition on three bits.



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

$Sum = \sum m(1, 2, 4, 7)$

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

$= C[\bar{A}\bar{B} + AB] + \bar{C}[\bar{A}B + A\bar{B}]$

$= C\overline{[\bar{A}B + A\bar{B}]} + \bar{C}[\bar{A}B + A\bar{B}]$

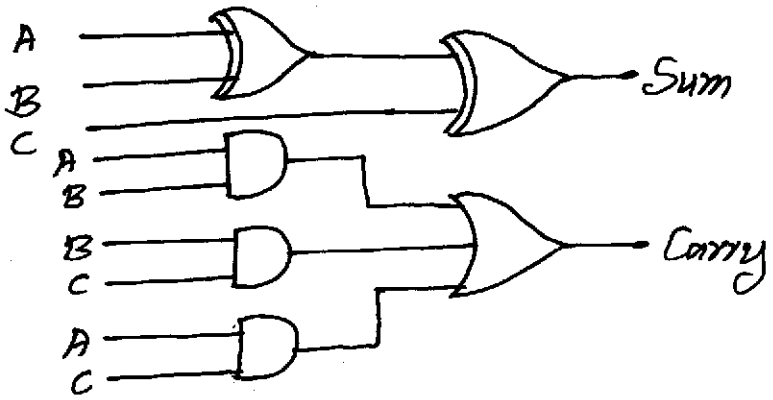
$= A \oplus B \oplus C.$

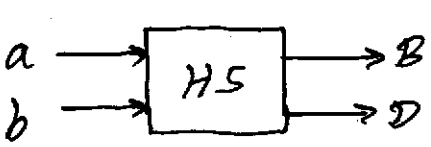
$Carry = \sum m(3, 5, 6, 7)$

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

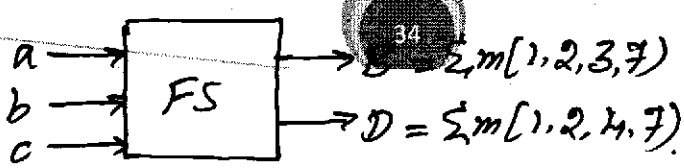
$= AB + BC + AC.$

	$\bar{C}$	C
$\bar{A}\bar{B}$	0	0
$\bar{A}B$	0	1
AB	1	1
$A\bar{B}$	0	1





a	b	Borrow	Diff.
0	0	0	0
0	1	1	1
1	0	0	1



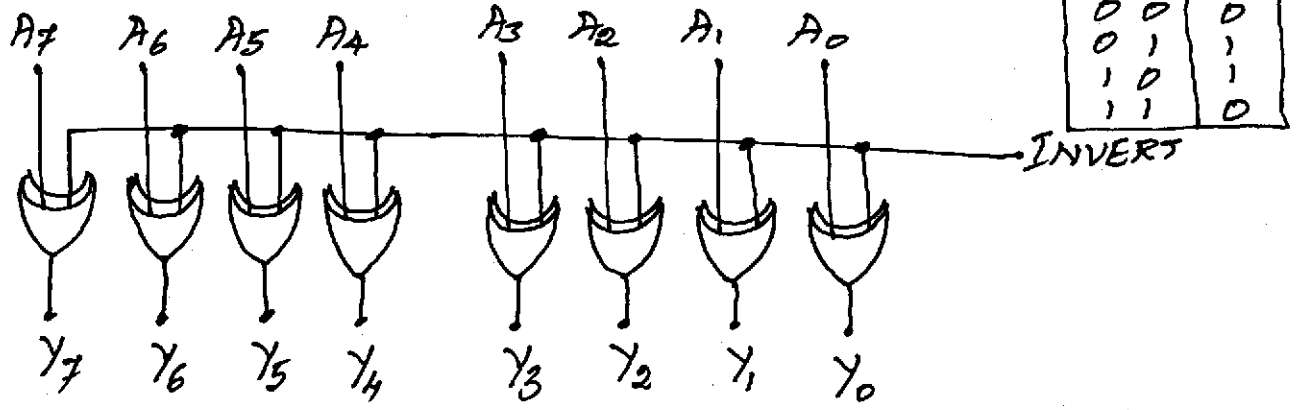
General representation of full-adder which adds *i*-th bit A_i and B_i of two numbers A and B along with the carry *(i-1)*th bit could be

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_iB_i + B_iC_{i-1} + A_iC_{i-1} \quad or \quad C_i = A_iB_i + (A_i + B_i)C_{i-1}$$

where, S_i and C_i are the sum and carry bits generated from the full-adder.

Controlled Inverter: The following Fig shows a *controlled inverter*.



Controlled Inverter

When INVERT is low, it transmits the 8-bit input to the output; when INVERT is high, it transmits the 1's complement. For example –

If the input number is $A_7 \ldots A_0 = 0110\ 1110$, a low INVERT produces $Y_7 \ldots Y_0 = 0110\ 1110$.

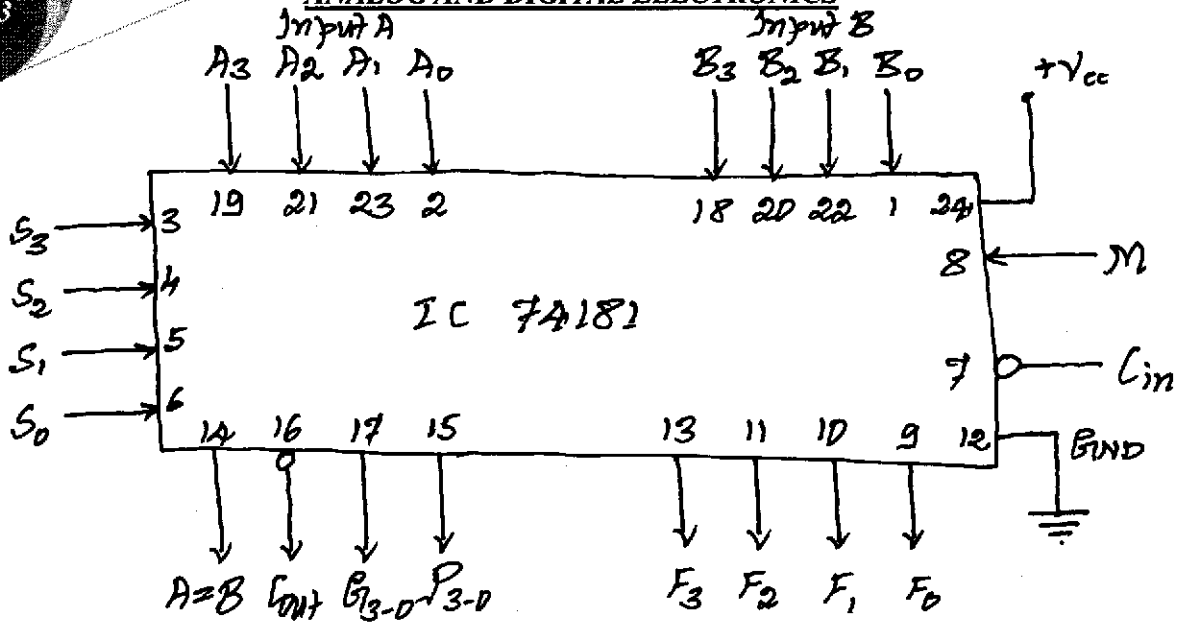
But a high INVERT results in $Y_7 \ldots Y_0 = 1001\ 0001$.

The controlled inverter finds application in subtraction. During subtraction, we first need to take 2's complement of the subtrahend. Then, we can add the complemented subtrahend to obtain the answer. With a controlled inverter, we can produce 1's complement and proceed further accordingly.

ARITHMETIC LOGIC UNIT (ALU):

ALU is multifunctional device that can perform both arithmetic and logic function. A mode selector input (M) decides whether ALU performs a logic operation or an arithmetic operation. As an arithmetic unit, along with normal addition, subtraction, etc., it can also perform increment, decrement operations. As logic unit, it can perform usual OR, AND, NOT, Ex-OR, and many more complex logic functions. It also comes with PRESET and CLEAR options, invoking which all the function outputs are made 1 and 0 respectively. ALU is an integral part of Central Processing Unit (CPU) of a computer.

IC 74181 is a 4-bit ALU that can generate 16 different kinds of outputs, in each mode, selected by four selection inputs S_3, S_2, S_1, and S_0.



Functional Diagram of ALU IC 74181

S3	S2	S1	S0	M = 1 Logic Function	M = 0 Arithmetic Function C_{in} = 1 (For C_{in} = 0, add 1 to F)
0	0	0	0	F = A'	F = A
0	0	0	1	F = (A + B)'	F = A + B
0	0	1	0	F = A'B	F = A + B'
0	0	1	1	F = 0	F = minus 1
0	1	0	0	F = (AB)'	F = A plus (AB')
0	1	0	1	F = B'	F = (A + B) plus (AB')
0	1	1	0	F = A ⊕ B	F = A minus B minus 1
0	1	1	1	F = AB'	F = AB' minus 1
1	0	0	0	F = A' + B	F = A plus (AB)
1	0	0	1	F = (A ⊕ B)'	F = A plus B
1	0	1	0	F = B	F = (A + B') plus (AB)
1	0	1	1	F = AB	F = AB minus 1
1	1	0	0	F = 1	F = A plus A
1	1	0	1	F = A + B'	F = (A + B) plus A
1	1	1	0	F = A + B	F = (A + B') plus A
1	1	1	1	F = A	F = A minus 1

ALU IC 74181 Truth Table

Problem: Show how A>B output can be generated in IC 74181 ALU. Also show how A≥B condition can be checked. Show how bits of input A shifted to left by one unit appear at output F in IC 74181.

note that, the data inputs A and B are active high signals and C_{in} and C_{out} are active low signals. The outputs C_{out}, G_{3-0}, and P_{3-0} are useful in addition and subtraction of more than 4-bits.

Logic operations are done bit-wise, by making M = 1 and choosing appropriate select inputs. Carry is inhibited for M = 1. For example, to perform AND operation between two 4-bit numbers (1011) A and (0111) B, make S = 1011 (refer truth table) and M = 1 to choose the logic function. The output will be F = 0011.

For arithmetic operations, M = 0 to be chosen. For example, to add decimal numbers 6 with 4, we have to place 0110 for 6 (at A) and 0100 for 4 (at B). Then with S = 1001 (refer truth table), C_{in} = 1 (active low), the output generated is, F = 1010 – decimal equivalent of 10.

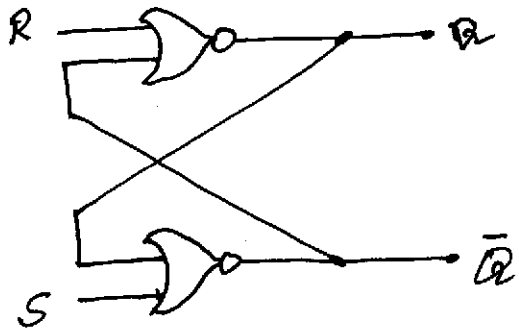
FLIP-FLOPS

- Any device or circuit that has two stable states is said to be *bistable*. For example, a toggle switch – a switch is also said to have *memory*, since it will remain as set until someone changes its position.
- A *flip-flop* is a bistable electronic circuit that has two stable states – i.e. its output is either 0 or +5 V.
- The flip-flop also has memory, since its output will remain as set until something is done to change it.
- The flip-flop is often called a *latch*, since it will hold (or latch) in either stable state.

SR FLIP-FLOPS:

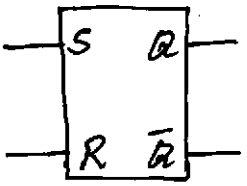
NOR-Gate Latch:

Two 2-input NOR gates are connected as shown in the following Fig to form a flip-flop. The flip-flop has two outputs, Q and Q'. There are two inputs to the flip-flop defined as S and R. The input/output possibility for this SR flip-flop is also given below.



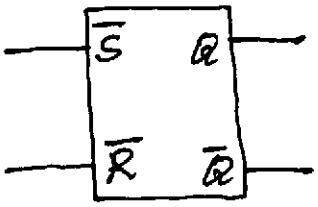
S	R	Q	Action
0	0	Last Sate	No Change
0	1	0	RESET
1	0	1	SET
1	1	?	Forbidden

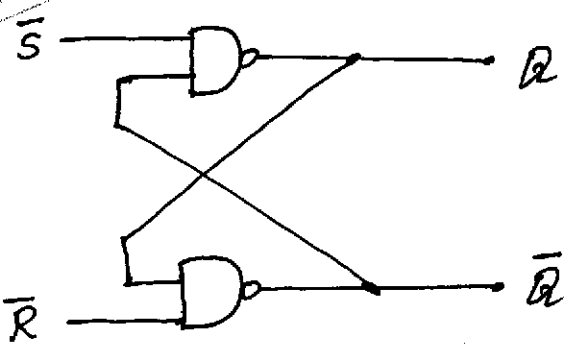
NOR-Gate SR Flip-Flop, Symbol & Its Truth Table



NAND-Gate Latch:

A slightly different latch can be constructed by using NAND gates, as shown in the following Fig.



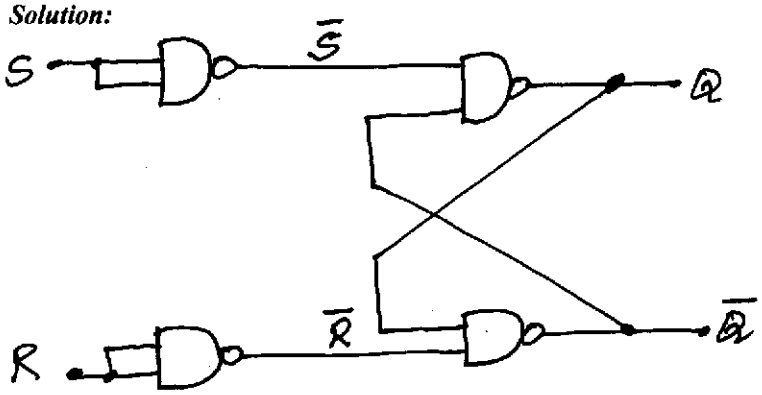


$\bar{S}$	$\bar{R}$	Q	Action
1	1	Last Sate	No Change
1	0	0	RESET
0	1	1	SET
0	0	?	Forbidden

NAND-Gate $\bar{S}\bar{R}$ Flip-Flop, Symbol & Its Truth Table

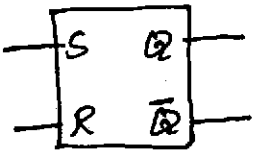
Problem: *Show how to convert the $\bar{S}\bar{R}$ flip-flop into RS flip-flop.*

Solution:



S	R	Q	Action
0	0	Last Sate	No Change
0	1	0	RESET
1	0	1	SET
1	1	?	Forbidden

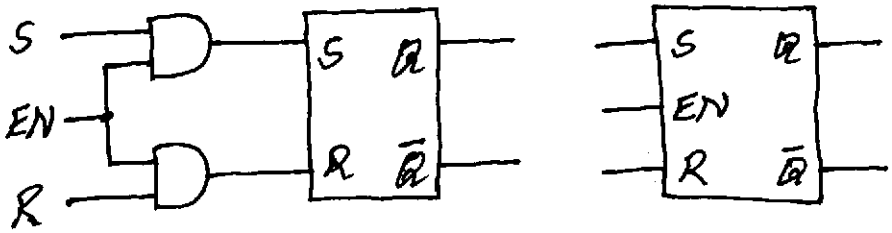
NAND-Gate SR Flip-Flop, Symbol & Its Truth Table



GATED FLIP-FLOPS:

Clocked RS Flip-Flop: The addition of two AND gates at the S and R inputs (as shown in the following Fig) will result in a flip-flop that can be enabled or disabled.

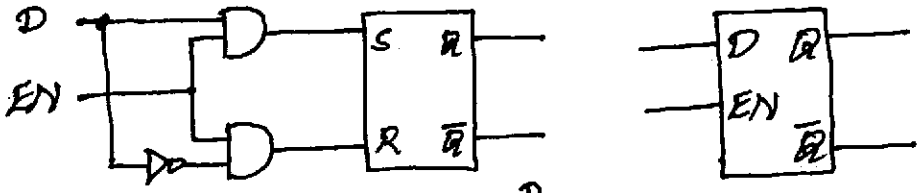
- ✓ When the ENABLE input is low, the AND gate (both) outputs must be low. Hence, neither R nor S will have any effect on the flip-flop output Q.
- ✓ When the ENABLE (EN) input is high, information at the S and R inputs will be transmitted directly to the outputs. Hence, the output will change in response to the inputs as long as the ENABLE is high.



EN	S	R	Q_{n+1}	Action
0	X	X	Q_n	No Change
1	0	0	Q_n	No Change
1	0	1	0	RESET
1	1	0	1	SET
1	1	1	?	Forbidden

Clocked SR Flip-Flop, Symbol & Its Truth Table

Clocked D Flip-Flop: A D flip-flop is a bistable circuit whose D input is transferred to the output when EN is high.

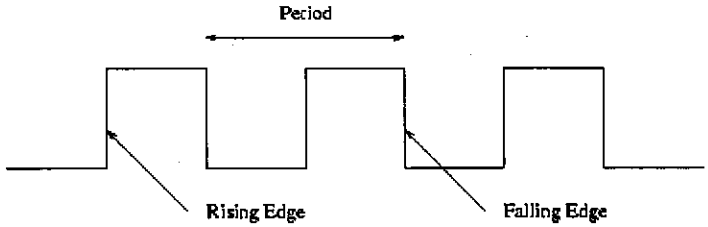


EN	D	Q_{n+1}
0	X	Q_n (No Change)
1	0	0
1	1	1

Clocked D Flip-Flop, Symbol & Its Truth Table

NOTE:

1. Clock cycle:

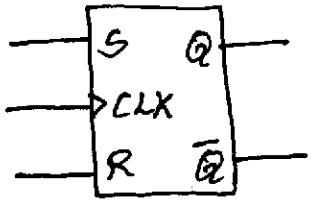


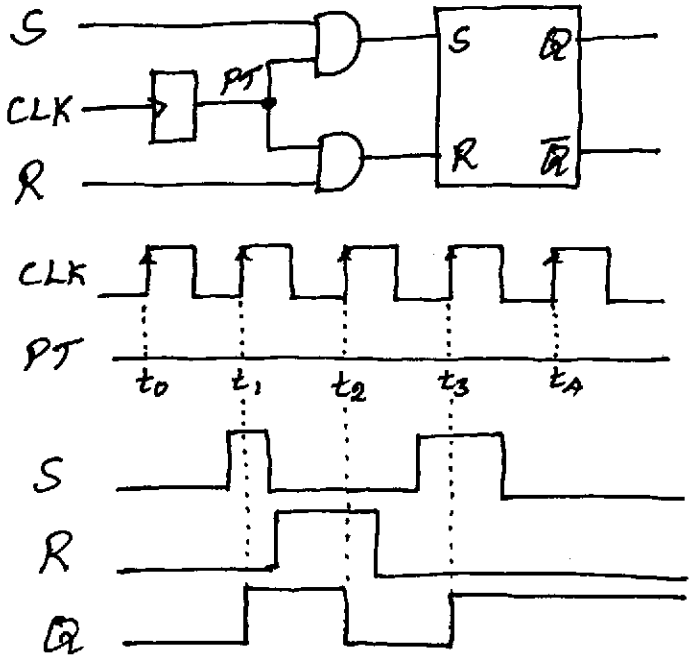
2. A low-to-high transition is *positive transition (PT)*. A circuit that changes state at this time is known as *positive-edge triggered.*
3. A high-to-low transition is *negative transition (NT)*. A circuit that changes state at this time is known as *negative-edge triggered.*

EDGE-TRIGGERED SR FLIP-FLOP:

Positive-Edge-Triggered SR Flip-Flops:

C	S	R	Q_{n+1}	Action
↑	0	0	Q_n	No Change
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	?	Illegal

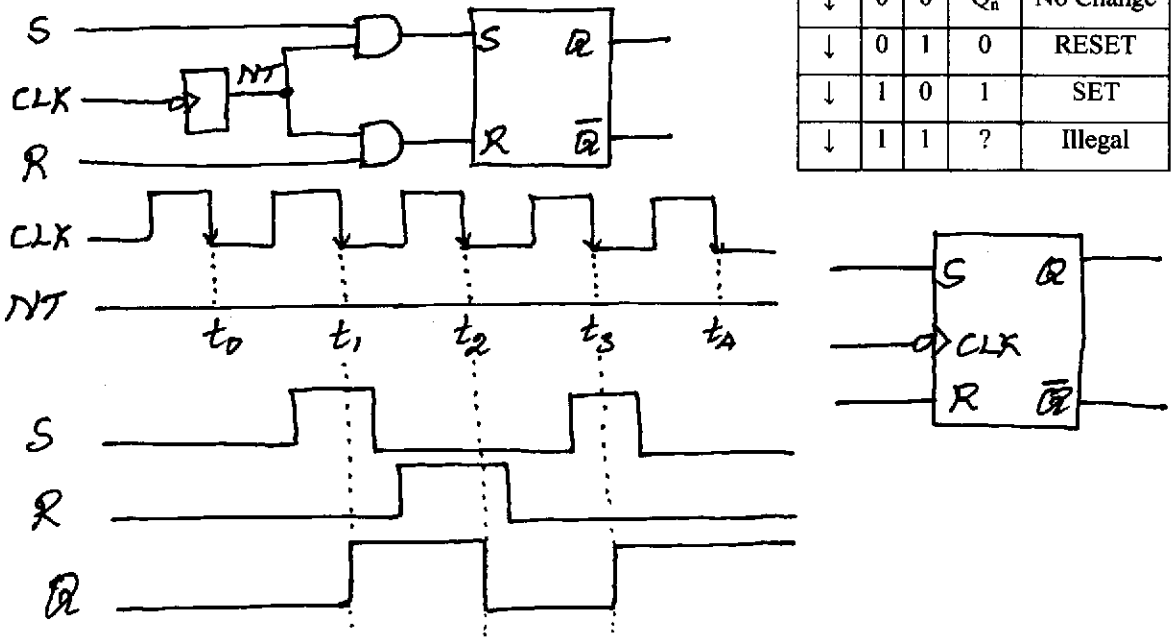




Positive-Edge-Triggered SR Flip-Flop, Symbol, Truth Table, & Waveform

Negative-Edge-Triggered SR Flip-Flops:

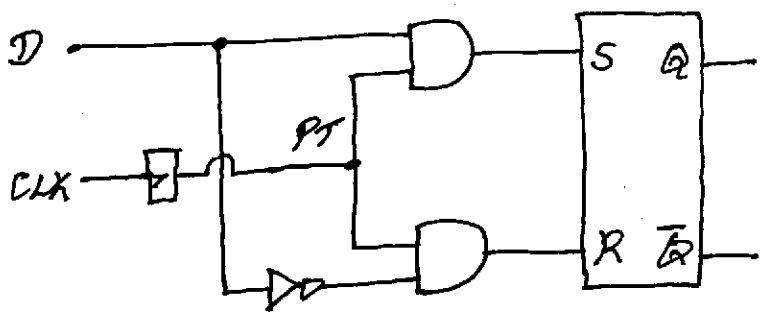
EN	S	R	Q_{n+1}	Action
↓	0	0	Q_n	No Change
↓	0	1	0	RESET
↓	1	0	1	SET
↓	1	1	?	Illegal



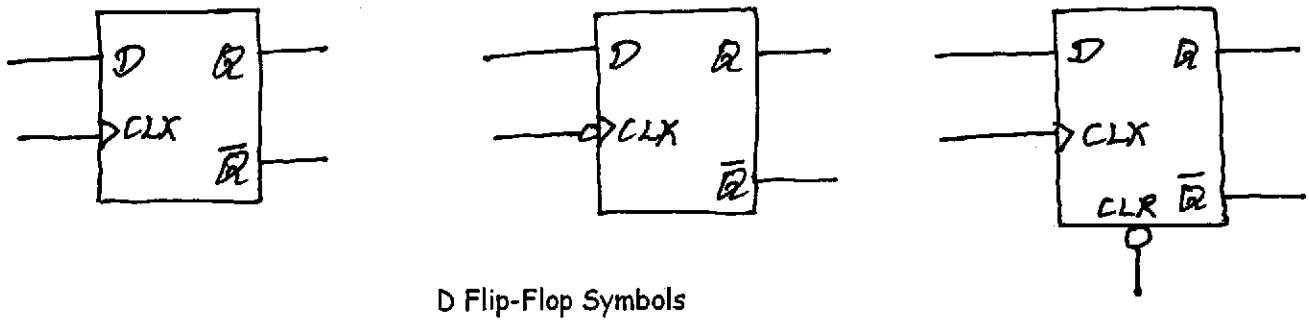
Negative-Edge-Triggered SR Flip-Flop, Symbol, Truth Table, & Waveform

EDGE-TRIGGERD D FLIP-FLOPS:

C	D	Q_{n+1}
0	X	Q_n (No Change)
↑	0	0
↑	1	1



Positive-Edge-Triggered D Flip-Flop & Its Truth Table

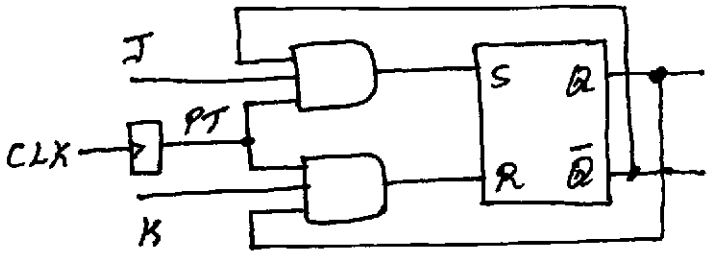


D Flip-Flop Symbols

EDGE-TRIGGERED JK FLIP-FLOPS:

Setting S = R = 1 with an edge-triggered SR flip-flop forces both Q and Q' to the same logic level. This is an illegal condition; hence, it is not possible to find the final state of Q. The JK flip-flop accounts for this illegal input.

Positive-Edge-Triggered JK Flip-Flops:



Positive-Edge-Triggered JK Flip-Flop

C	J	K	Q_{n+1}	Action
↑	0	0	Q_n	No Change
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	$\overline{Q_n}$	Toggle

Working:

1. When J and K both are low, both AND gates are disabled. Therefore, clock pulses have no effect. Hence, Q retains its previous value.
2. When J is low and K is high, the upper AND gate is disabled. So, there is no way to set the flip-flop, and the only possibility is reset.

   If Q is low in the previous state, same low value will be maintained as the next positive clock edge arrives. If the Q is high in the previous state, the lower AND gate passes a RESET pulse as soon as the next positive clock edge arrives. This forces Q to become low.
3. When J is high and K is low, the lower gate is disabled. So, it's impossible to reset the flip-flop.

   If Q is low in the previous state, $\bar{Q}$ is high; therefore the upper AND gate passes SET pulse on the next positive clock edge. This drives Q into high state. If Q is high in the previous state, $\bar{Q}$ is low; therefore, the SET signal through the upper AND gate will be 0. This maintains Q in the high state.
4. When J and K are both high, it's possible to set or reset the flip-flop. If Q is high, the lower AND gate passes a RESET pulse on the next PT. If Q is low, the upper AND gate passes a SET pulse on the next PT. Either the way, the Q changes to the complement of the previous state. Hence, flip-flop will toggle.

JK Flip-Flop Symbols

By: **MAHESH PRASANNA K.,**

DEPT. OF CSE, VCET.

****

****