UNIT 1

COMBINATIONAL LOGIC

INTRODUCTION

The digital system consists of two types of circuits, namely

(i) Combinational circuits

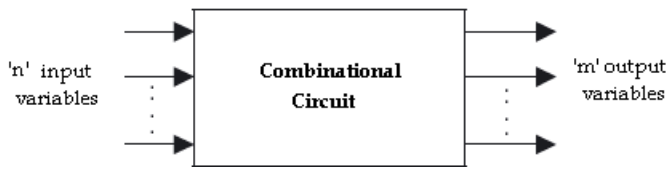
(ii) Sequential circuits

Combinational circuit consists of logic gates whose output at any time is determined from the present combination of inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions.

Sequential logic circuit comprises both logic gates and the state of storage elements such as flip-flops. As a consequence, the output of a sequential circuit depends not only on present value of inputs but also on the past state of inputs.

In the previous chapter, we have discussed binary numbers, codes, Boolean algebra and simplification of Boolean function and logic gates. In this chapter, formulation and analysis of various systematic designs of combinational circuits will be discussed.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from inputs and output signals are generated according to the logic circuits employed in it. Binary information from the given data transforms to desired output data in this process. Both input and output are obviously the binary signals, *i.e.*, both the input and output signals are of two possible states, logic 1 and logic 0.



APEC

For *n* number of input variables to a combinational circuit, 2*n* possible combinations of binary input states are possible. For each possible combination, there is one and only one possible output combination.

A combinational logic circuit can be described by *m* Boolean functions and each output can be expressed in terms of *n* input variables.

DESIGN PROCEDURES

Any combinational circuit can be designed by the following steps of design procedure.

1. The problem is stated.

2. Identify the input and output variables.

3. The input and output variables are assigned letter symbols.

4. Construction of a truth table to meet input -output requirements.

5. Writing Boolean expressions for various output variables in terms of input variables. The simplified Boolean expression is obtained by any method of minimization—algebraic method, Karnaugh map method, or tabulation method.

7. A logic diagram is realized from the simplified Boolean expression using logic gates. The following guidelines should be followed while choosing the preferred form for hardware implementation:

1. The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.

2. There should be a minimum number of interconnections.

3. Limitation on the driving capability of the gates should not be ignored.

Problems

1. Design a combinational circuit with three inputs and one output. The output is 1 when the binary value of the inputs is less than 3. The output is 0 otherwise.

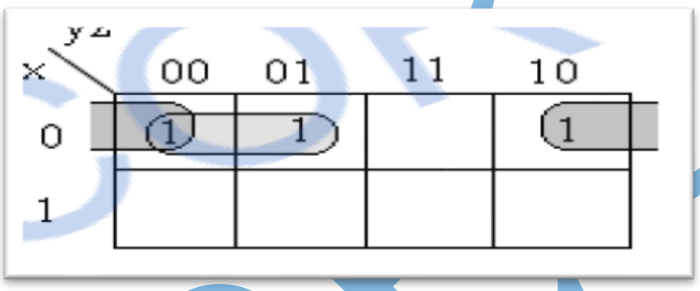
APEC

Solution:

Truth Table**:**

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

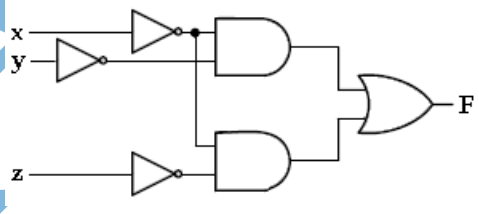
K-map Simplification**:**



$$= x'y' + x'z'$$

Logic Diagram:

The combinational circuit can be drawn as,



ARITHMETIC CIRCUITS

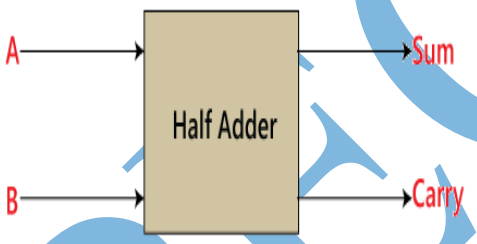
In this section, we will discuss those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the

processes of repeated addition and repeated subtraction. The basic building blocks that form the basis of all hardware used to perform the arithmetic operations on binary numbers are half-adder, full adder, half-subtractor, full-subtractor.

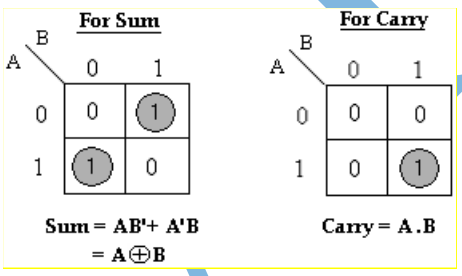
HALF-ADDER

A half-adder is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.

Truth Table:



K-map simplification:



Inputs		Outputs	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

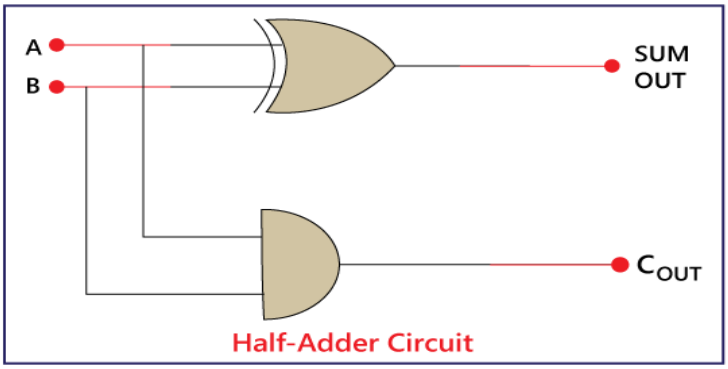
The Boolean expressions for the SUM and CARRY outputs are given by the equations,

$$\text{Sum, } S = A'B + AB' = A \oplus B$$

$$\text{Carry, } C = A \cdot B$$

The first one representing the SUM output is that of an EX-OR gate, the second one representing the CARRY output is that of an AND gate.

The logic diagram of the half adder is,



FULL-ADDER

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs.

Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position. The block diagram of full adder is given by,

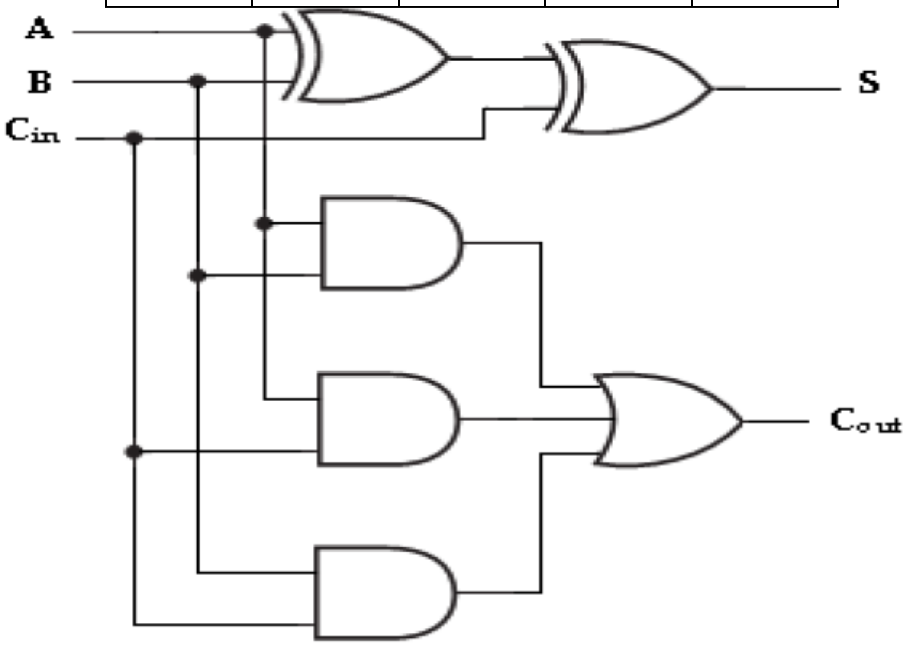


The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. As there are three input variables, eight different input combinations are possible.

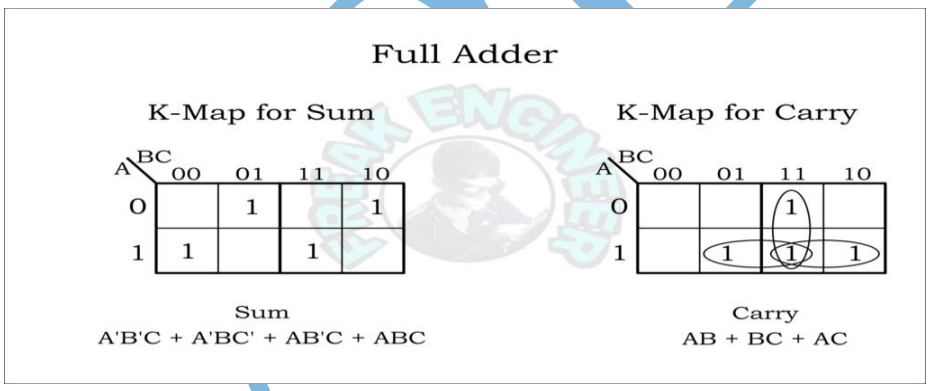
Truth Table:

Inputs			**Outputs**	
A	**B**	**Cin**	**Sum (S)**	**Carry (Cout)**
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



K map



$$\text{Sum, } S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$$
$$= A'(B'C_{in} + BC'_{in}) + A(B'C'_{in} + BC_{in})$$
$$= A'(B \oplus C_{in}) + A(B \oplus C_{in})'$$
$$= A \oplus (B \oplus C_{in})$$

$$\text{Carry, } C_{out} = AB + AC_{in} + BC_{in}.$$

The logic diagram of the full adder can also be implemented with two half-adders and one OR gate. The S output from the second half adder is the exclusive-OR of Cin and the output of the first half-adder, giving

Sum = Cin □ (A □ B) [x □ y = x'y+ xy']

= Cin □ (A'B+AB')

= C'in (A'B+AB') + Cin (A'B+AB')' **[(x'y+xy')'= (xy+x'y')]**

= C'in (A'B+AB') + Cin (AB+A'B')

= A'BC'in + AB'C'in + ABCin + A'B'Cin .

and the carry output is,

Carry, Cout = AB+ Cin (A'B+AB')

= AB+ A'BCin+ AB'Cin

= B (A+A'Cin) + AB'Cin

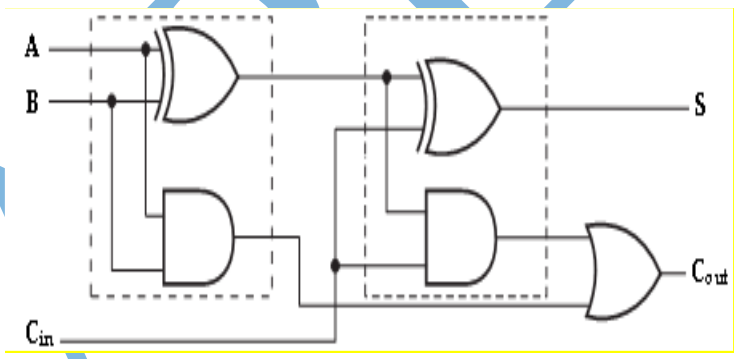
= B (A+Cin) + AB'Cin

= AB + BCin + AB'Cin

= AB + Cin (B + AB')

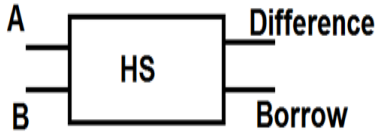
= AB + Cin (A + B)

= AB+ ACin+ BCin.

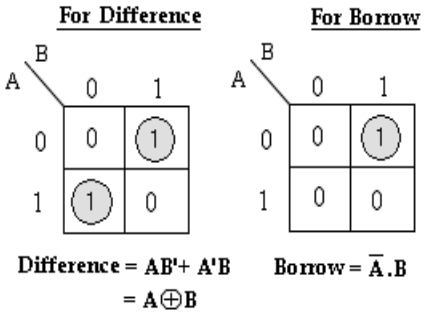


HALF -SUBTRACTOR

A *half-subtractor* is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction.

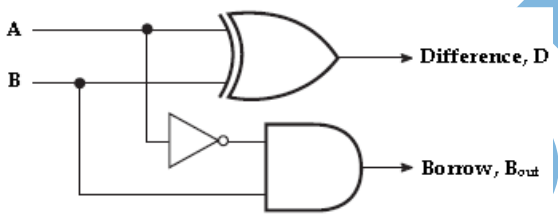


The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown above



The first one representing the DIFFERENCE (**D**)output is that of an exclusive-OR gate, the expression for the BORROW output (**Bout**) is that of an AND gate with input A complemented before it is fed to the gate.

The logic diagram of the half adder is,



Inputs		Outputs	
A	**B**	**D**	**B**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

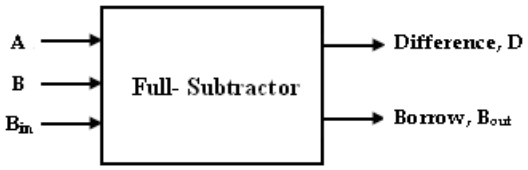
Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, ie., the minuend is complemented, an AND gate can be used to implement the BORROW output.

FULL SUBTRACTOR

A ***full subtractor*** performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.

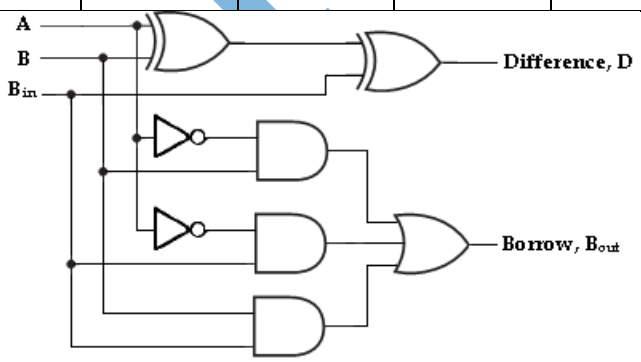
As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin. There are two outputs, namely the DIFFERENCE output D and the BORROW output

Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.



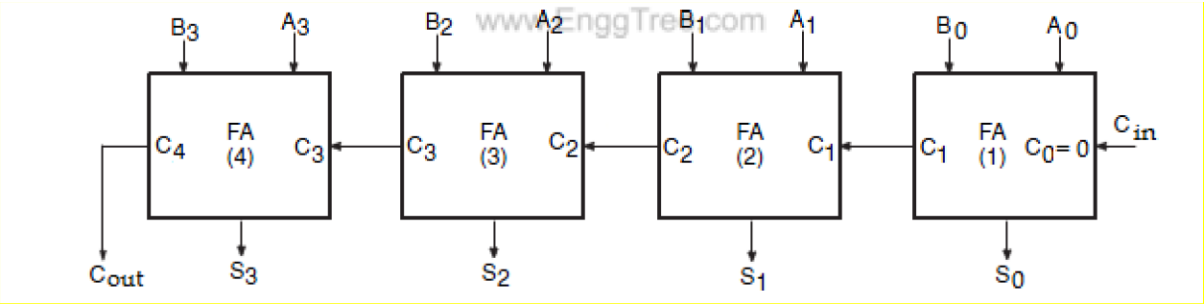
The truth table for full-subtractor is,

A	B	Bin	Difference(D)	Borrow (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



FOUR –BIT BINARY ADDER (PARALLEL ADDER)

The 4-bit binary adder using full adder circuits is capable of adding two4-bit numbers resulting in a 4-bit sum and a carry output as shown in figure below.



4-BITBINARYPARALLELADDER

Since all the bits of augend and add end are fed into the adder circuits simultaneously and the additions in each position are taking place at the same time, this circuit is known as parallel adder.

Letthe4-bit words to be added be represented by,

A3A2A1A0=1111 and B3B2B1B0= 0 0 1 1.

Significant place	4	3	2	1
Input carry	1	1	1	0
Augend word A :	1	1	1	1
Addend word B :	0	0	1	1
	1	0	0	1	0 ← Sum

↑ Output Carry

The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry C0 in the least significant position must be 0. The carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple-carry adder.

In the least significant stage, A0, B0 and C0 (which is 0) are added resulting in sum S0 and carry C1. This carry C1 becomes the carry input to the second stage. Similarly in the second stage, A1, B1 and C1 are added resulting in sum S1 and carry C2, in the third stage, A2, B2 and C2 are added resulting in sum S2 and carry C3, in the third stage, A3, B3 and C3 are added resulting in sum S3 and C4, which is the output carry. Thus the circuit results in a sum (S3 S2 S1 S0) and a carry output (Cout).

Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages. However, there are several methods to reduce this delay. One of the methods of speeding up this process is **look-ahead carry addition** which eliminates the ripple-carry delay.

In the least significant stage, A0, B0 and C0 (which is 0) are added resulting in sum S0 and carry C1. This carry C1 becomes the carry input to the second stage. Similarly in the second stage, A1, B1 and C1 are added resulting in sum S1 and carry C2, in the third stage, A2, B2 and C2 are added resulting in sum S2 and carry C3, in the third stage, A3, B3 and C3 are added resulting in sum S3 and C4, which is the output carry. Thus the circuit results in a sum (S3 S2 S1 S0) and a carry output (Cout).

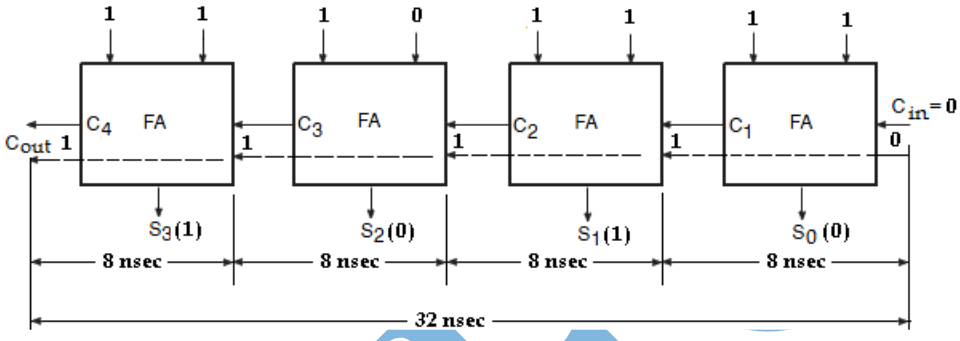
Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages. However, there are several methods to reduce this delay. One of the methods of speeding up this process is **look-ahead carry addition** which eliminates the ripple-carry delay.

CARRY LOOK AHEAD ADDER

In Parallel adder, all the bits of the augend and the addend are available for computation at the same time. The carry output of each full-adder stage is connected to the carry input of the next high-order stage. Since each bit of the sum output depends on the value of the input carry, time delay occurs in the addition process. This time delay is called as **carry propagation delay.**

For example, addition of two numbers (1111+ 1011) gives the result as 1010. Addition of the LSB position produces a carry into the second position. This carry when added to the bits of the second position, produces a carry into the third position. This carry when added to bits of the third position, produces a carry into the last position. The sum bit generated in the last position (MSB) depends on the

carry that was generated by the addition in the previous position. i.e., the adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in an each full-adder. For example, if each full adder is considered to have a propagation delay of 8nsec, then S3 will not react its correct value until 24 nsec after LSB is generated. Therefore total time required to perform addition is 24+ 8 = 32nsec.



The method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition**. This method utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions: carry generate and carry propagate.

Consider the circuit of the full-adder shown above. Here we define two functions: carry generate (Gi) and carry propagate (Pi) as,

$$\text{Carry propagate, } P_i = A_i \oplus B_i$$

$$\text{Carry generate, } G_i = A_i \cdot B_i$$

the output sum and carry can be expressed as,

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i.C_i$$

Gi (carry generate), it produces a carry 1 when both Ai and Bi are 1, regardless of the input carry Ci. Pi (carry propagate), is the term associated with the propagation of the carry from Ci to Ci+1.

The Boolean functions for the carry outputs of each stage and substitute for each Ci its value from the previous equation:

C0= input carry

C1= G0 + P0C0

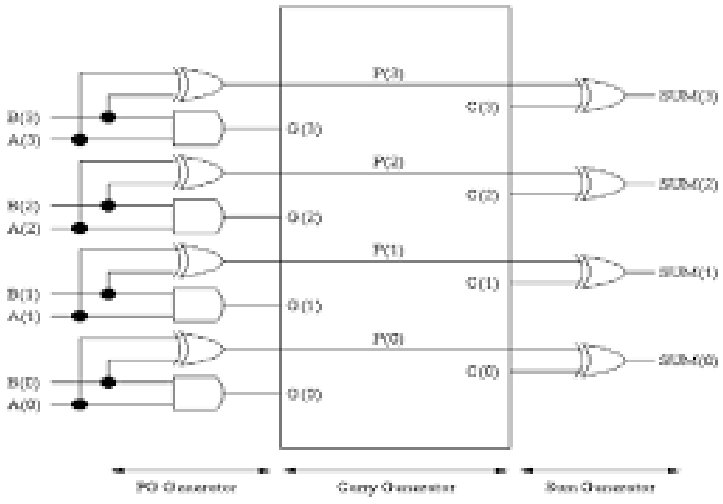
C2= G1 + P1C1 = G1 + P1 (G0 + P0C0)

= G1 + P1G0 + P1P0C0

C3= G2 + P2C2 = G2 + P2 (G1 + P1G0 + P1P0C0)

= G2 + P2G1 + P2P1G0 + P2P1P0C0

Since the Boolean function for each output carry is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate. The three Boolean functions for C1, C2 and C3 are implemented in the carry look-ahead generator as shown below.



Note that C3 does not have to wait for C2 and C1 to propagate; in fact C3 is propagated at the same time as C1 and C2.

Using a Look-ahead Generator we can easily construct a 4-bit parallel adder with a Look-ahead carry scheme. Each sum output requires two exclusive-OR gates. The output of the first exclusive-OR gate generates the Pi variable, and the AND gate generates the Gi variable. The carries are propagated through the carry look-ahead generator and applied as inputs to the second exclusive-OR gate. All output carries are generated after a delay through two levels of gates. Thus, outputs S1 through S3 have equal propagation delay times.

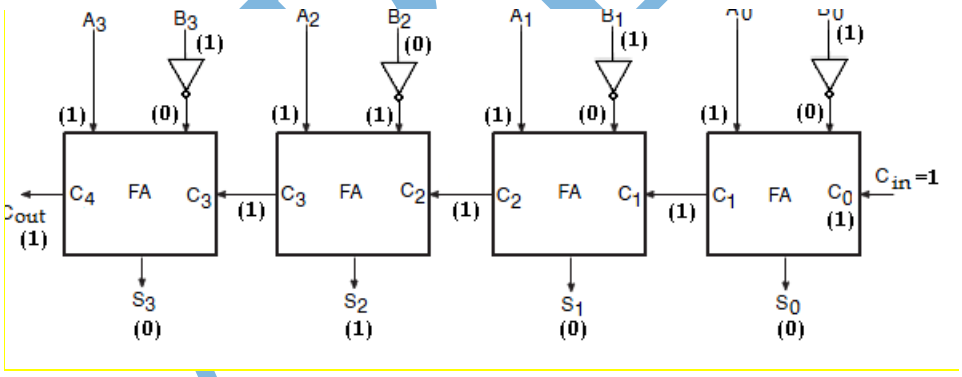
BINARY SUBTRACTOR (PARALLEL SUBTRACTOR)

The subtraction of unsigned binary numbers can be done most conveniently by means of complements. The subtraction (A – B) can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters and a 1 can be added to the sum through the input carry.

The circuit for subtracting (A – B) consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C0 must be equal to 1 when performing subtraction. The operation thus performed becomes A, plus the 1's complement of B, plus1. This is equal to A plus the 2's complement of B.

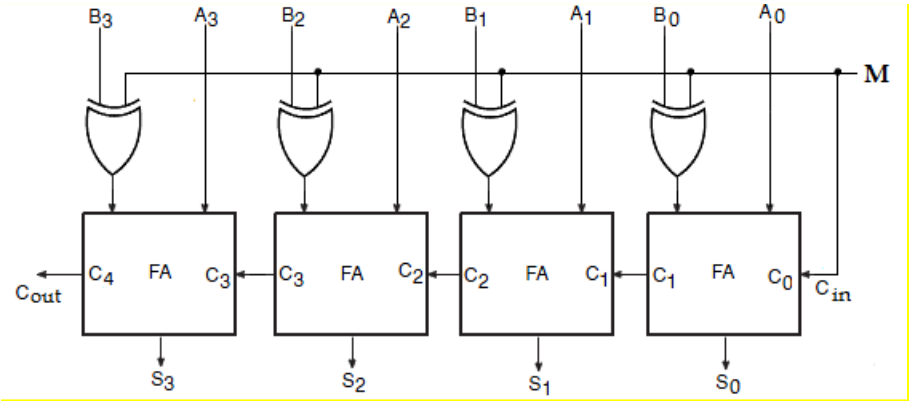
Let the 4-bit words to be subtracted be represented by,

A3 A2 A1 A0= 1 1 1 1 and B3 B2 B1 B0= 1 0 1 1.



PARALLEL ADDER/ SUBTRACTOR

The addition and subtraction operation can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full adder. A 4-bit adder Subtractor circuit is shown below.



The mode input M controls the operation. When M= 0, the circuit is an adder and when M=1, the circuit becomes a Subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. When M=0, we have B□ 0= B. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When M=1, we have B□ 1= B' and C0=1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B. The exclusive-OR with output V is for detecting an overflow.

DECIMAL ADDER (BCD ADDER)

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD bits and produces a sum digit also in BCD.

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9+ 9+1 = 19; the 1 is the sum being an input carry. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

These binary numbers are labeled by symbols C, S3, S2, S1, S0, C is the carry. The columns under the binary sum list the binary values that appear in the outputs of the 4-bit binary adder. The output sum of the two decimal digits must be represented in BCD.

To implement BCD adder:

• For initial addition , a 4-bit binary adder is required,

Combinational circuit to detect whether the sum is greater than 9 and

• One more 4-bit adder to add 6 (0110)2 with the sum of the first 4-bit adder, if the sum is greater than 9 or carry is 1.

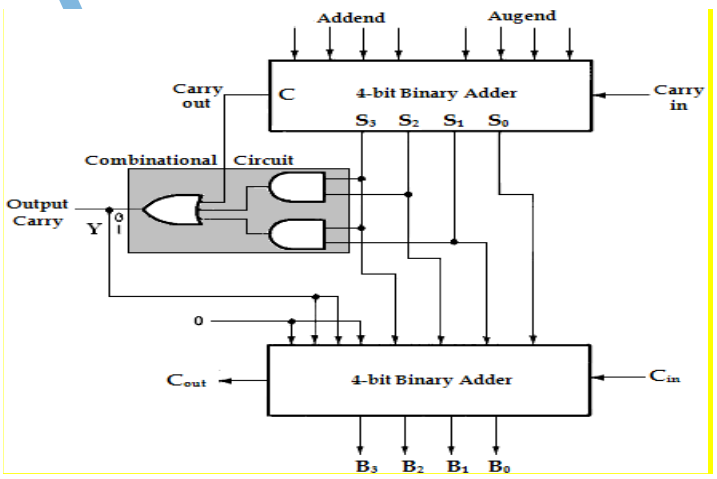
The logic circuit to detect sum greater than 9 can be determined by simplifying the Boolean expression of the given truth table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$Y = S_3S_2 + S_3S_1$

The two decimal digits, together with the input carry, are first added in the top4-bit binary adder to provide the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary (0110)2 is added to the binary sum through the bottom 4-bit adder.

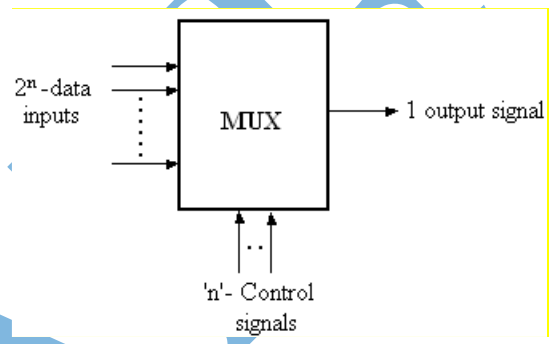


The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. The output carry from one stage must be connected to the input carry of the next higher-order stage

MULTIPLEXER: (DATA SELECTOR)

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

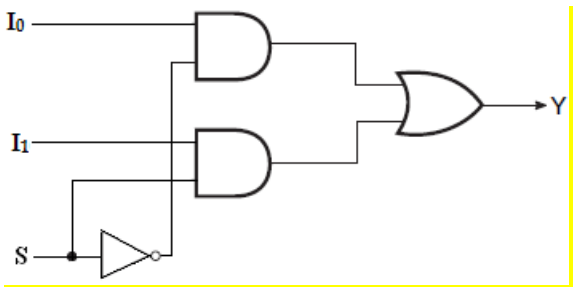
A multiplexer is also called a **data selector**, since it selects one of many inputs and steers the binary information to the output line.



2-to-1- line Multiplexer:

The circuit has two data input lines, one output line and one selection line, S. When S= 0, the upper AND gate is enabled and I0 has a path to the output.

When S=1, the lower AND gate is enabled and I1 has a path to the output.



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

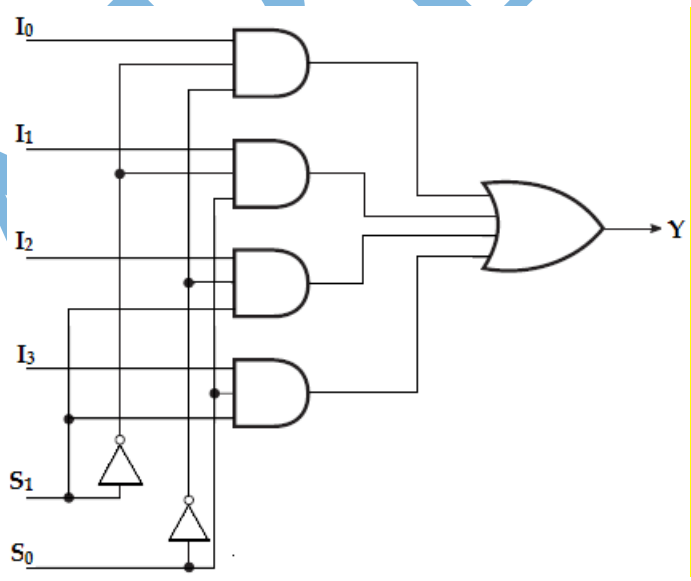
Truth table:

S	Y
0	I0
1	I1

4-to-1-line Multiplexer

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

Each of the four inputs I0 through I3, is applied to one input of AND gate. Selection lines S1 and S0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output



Function table:

S1	S0	Y
0	0	I0
0	1	I1

1	0	I2
1	1	I3

To demonstrate the circuit operation, consider the case when S1S0= 10. The AND gate associated with input I2 has two of its inputs equal to 1 and the third input connected to I2.

The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I2, providing a path from the selected input to the output.

The data output is equal to I0 only if S1= 0 and S0= 0; Y= I0S1'S0'.

The data output is equal to I1 only if S1= 0 and S0= 1; Y= I1S1'S0.

The data output is equal to I2 only if S1= 1 and S0= 0; Y= I2S1S0'.

The data output is equal to I3 only if S1= 1 and S0= 1; Y= I3S1S0.

When these terms are ORed, the total expression for the data output is,

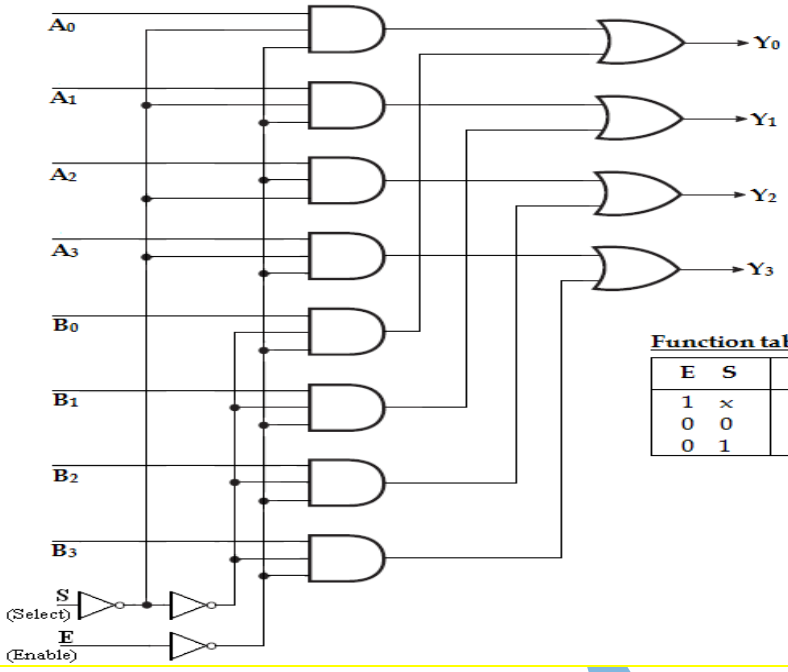
Y= I0S1'S0'+ I1S1'S0 +I2S1S0'+ I3S1S0.

As in decoder, multiplexers may have an enable input to control the operation of the unit.

When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer

Quadruple 2-to-1 Line Multiplexer

APEC



Function table:

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

This circuit has four multiplexers, each capable of selecting one of two input lines. Output Y0 can be selected to come from either A0 or B0. Similarly, output Y1 may have the value of A1 or B1, and so on. Input selection line, S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal operation.

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

Application:

1. They are used as a data selector to select out of many data inputs.
2. They can be used to implement combinational logic circuit.
3. They are used in time multiplexing systems.
4. They are used in frequency multiplexing systems.
5. They are used in A/D and D/A converter.

6. They are used in data acquisition systems.

Implementation of Boolean Function using MUX:

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

F (A, B, C) = Σm (1, 3, 5, 6).

Solution:

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

Select lines= n-1 = 2 (**S1, S0**)

2n-1 to MUX i.e., 22 to 1 = 4 to 1 MUX

Input lines= 2n-1 = 22 = 4 (**D0, D1, D2, D3**)

Implementation table:

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

i i. List the input of the multiplexer

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

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

1. If both the minterms in the column are not circled, apply 0 to the corresponding input.

2. If both the minterms in the column are circled, apply 1 to the corresponding input.

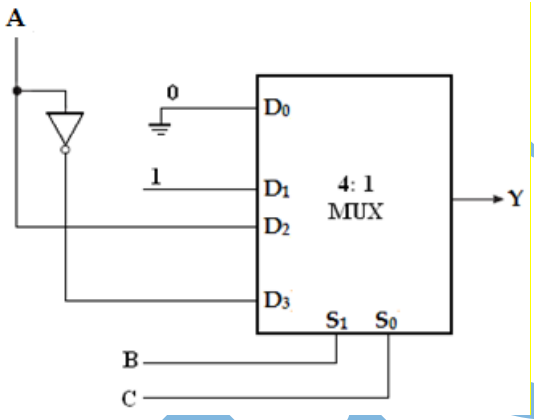
3. If the bottom minterm is circled and the top is not circled, apply C to the input.

4. If the top minterm is circled and the bottom is not circled, apply C' to the input.

APEC

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

Multiplexer Implementation:



2. Implement the Boolean function using 8: 1 and also using 4:1 multiplexer

F (A, B, C, D) = Σm (0, 1, 2, 4, 6, 9, 12, 14)

Solution:

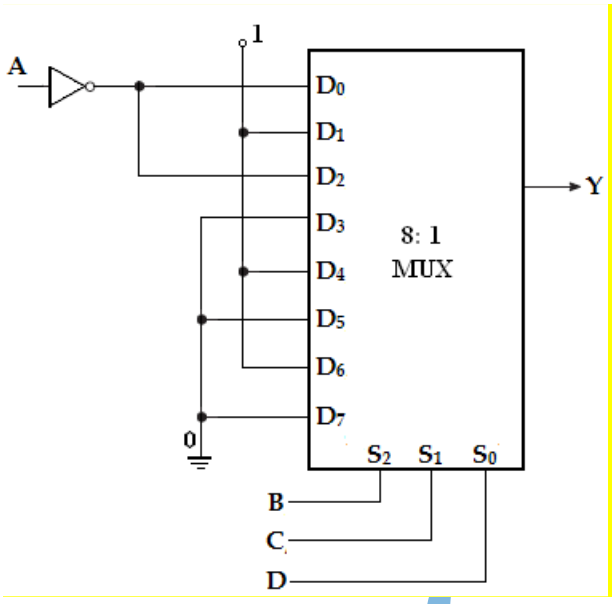
Variables, n= 4 (A, B, C, D)

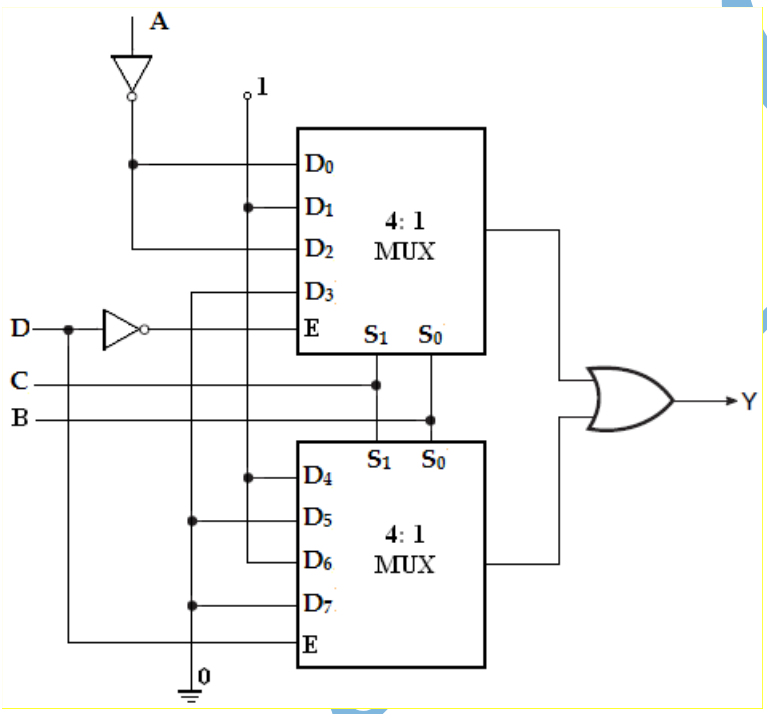
Select lines= n-1 = 3 (**S2, S1, S0**)

2n-1 to MUX i.e., 23 to 1 = 8 to 1 MUX

Input lines= 2n-1 = 23 = 8 (**D0, D1, D2, D3, D4, D5, D6, D7**)

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





3. Implement the Boolean function using 8: 1 multiplexer

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

Solution:

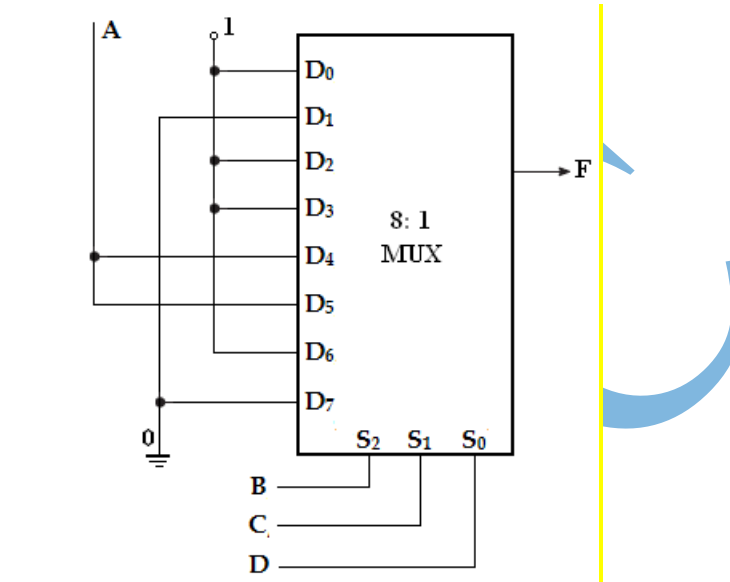
Variables, n= 4 (A, B, C, D)

Select lines= n-1 = 3 (**S2, S1, S0**)

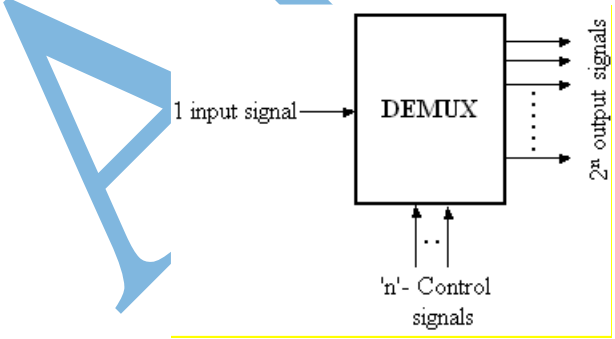
2n-1 to MUX i.e., 23 to 1 = 8 to 1 MUX

Input lines= 2n-1 = 23 = 8 (**D0, D1, D2, D3, D4, D5, D6, D7**)

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



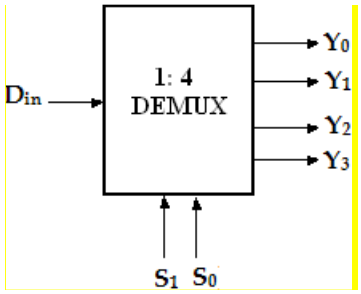
DEMULTIPLEXER



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

1-to-4 line Demultiplexer

A 1-to-4 demultiplexer has a single input, **Din**, four outputs (**Y0 to Y3**) and two select inputs (**S1 and S0**).



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

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

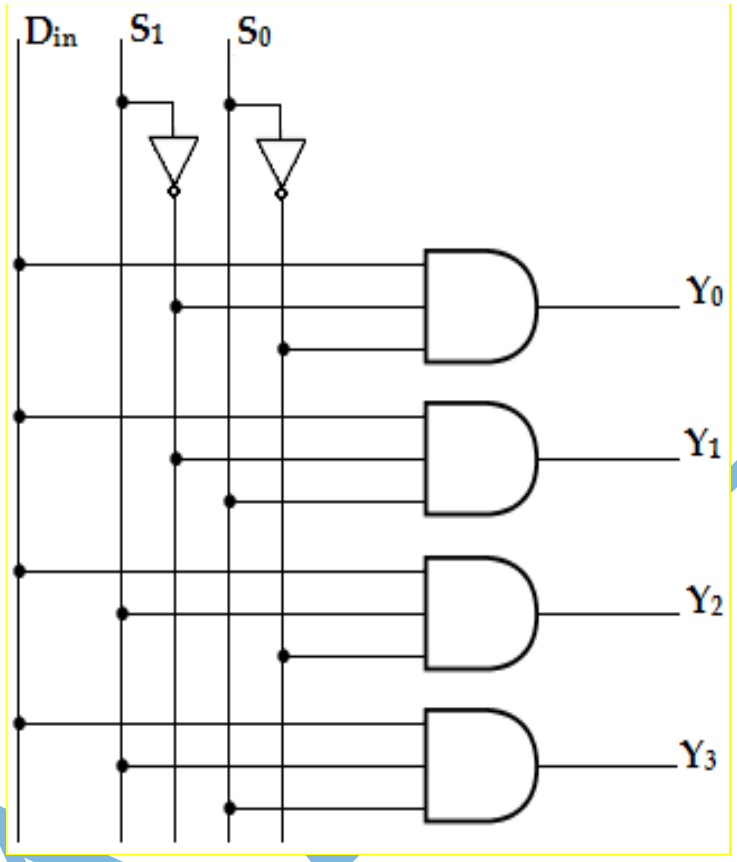
From the truth table, it is clear that the data input, Din is connected to the output Y0, when S1= 0 and S0= 0 and the data input is connected to output Y1 when S1= 0 and S0= 1. Similarly, the data input is connected to output Y2 and Y3 when S1= 1 and S0= 0 and when S1= 1 and S0= 1, respectively. Also, from the truth table, the expression for outputs can be written as follows,

Y0= S1'S0'Din

Y1= S1'S0Din

Y2= S1S0'Din

Y3= S1S0Din



Now, using the above expressions, a 1-to-4 demultiplexer can be implemented using four 3-input AND gates and two NOT gates. Here, the input data line Din, is connected to all the AND gates. The two select lines S1, S0 enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.

1-to-8 Demultiplexer

S3	**S2**	**S1**	**S0**	**Y7**	**Y6**	**Y5**	**Y4**	**Y3**	**Y2**	**Y1**	**Y0**
0	x	x	x	0	0	0	0	0	0	0	0

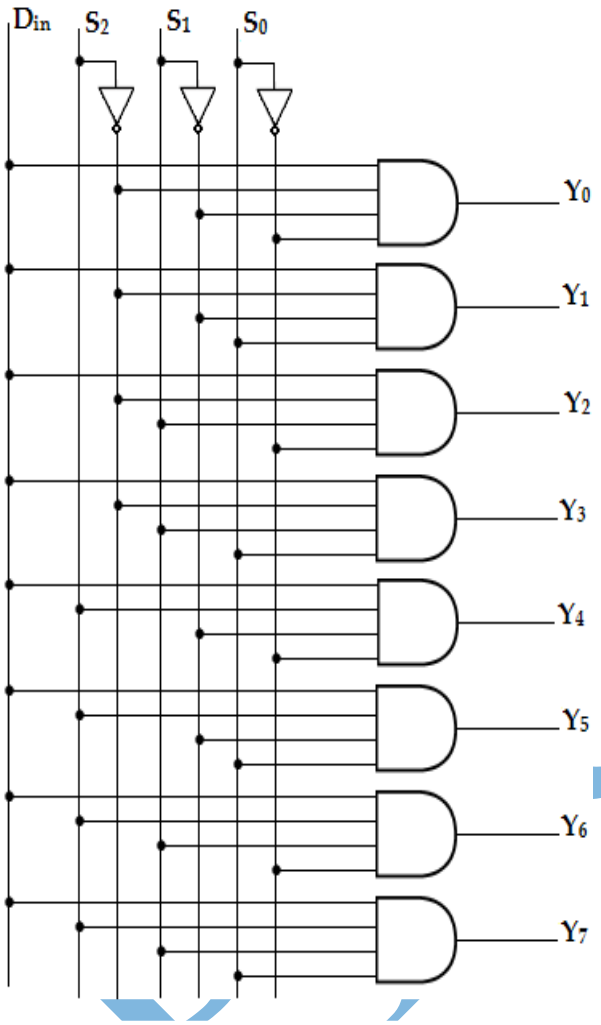
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

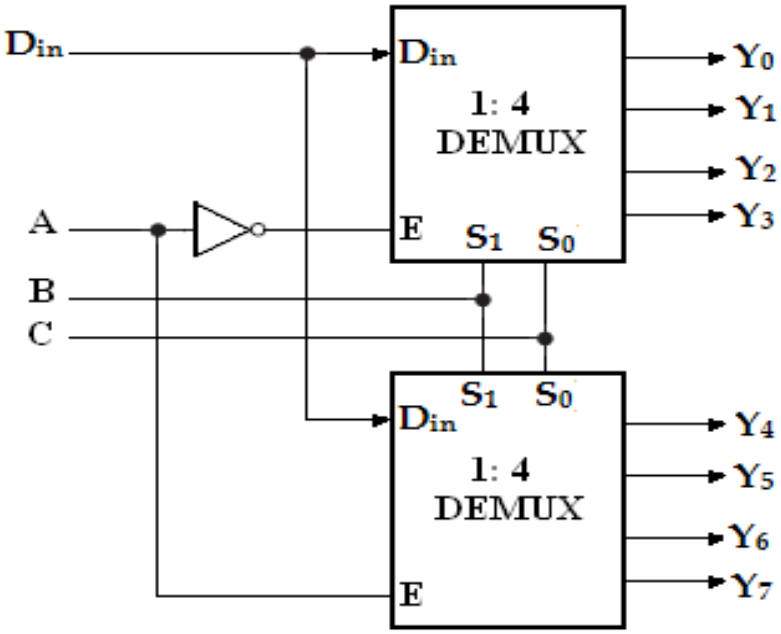
A 1-to-8 demultiplexer has a single input, **Din**, eight outputs (**Y0 to Y7**) and three select inputs (**S2, S1 and S0**). It distributes one input line to eight output lines based on the select inputs. The truth table of 1-to-8 demultiplexer is shown below.

Din

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can be drawn as shown below. Here, the single data line, Din is connected to all the eight AND gates, but only one of the eight AND gates will be enabled by the select input lines. For example, if S2S1S0= 000, then only AND gate-0 will be enabled and thereby the data input, Din will appear at Y0. Similarly, the different combinations of the select inputs, the input Din will appear at the respective output.

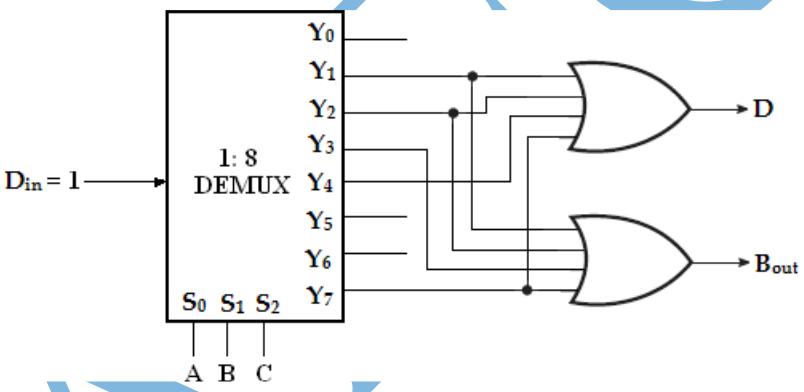
APEC


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


Din
A
B
C
Din
1: 4
DEMUX
E
S1
S0
Y0
Y1
Y2
Y3
S1
S0
Din
1: 4
DEMUX
E
Y4
Y5
Y6
Y7

1.Implement full subtractor using demultiplexer.

Inputs			Outputs	
A	**B**	**Bin**	**Difference(D)**	**Borrow (Bout)**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



Applications:

1. It can be used as a decoder

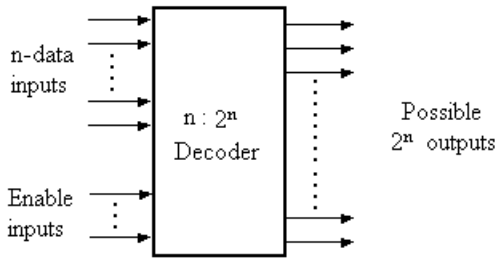
2. It can be used as a data distributer

3. It is used in time division multiplexing at the receiving end as a data separator.

4. It can be used to implement Boolean expressions.

DECODERS

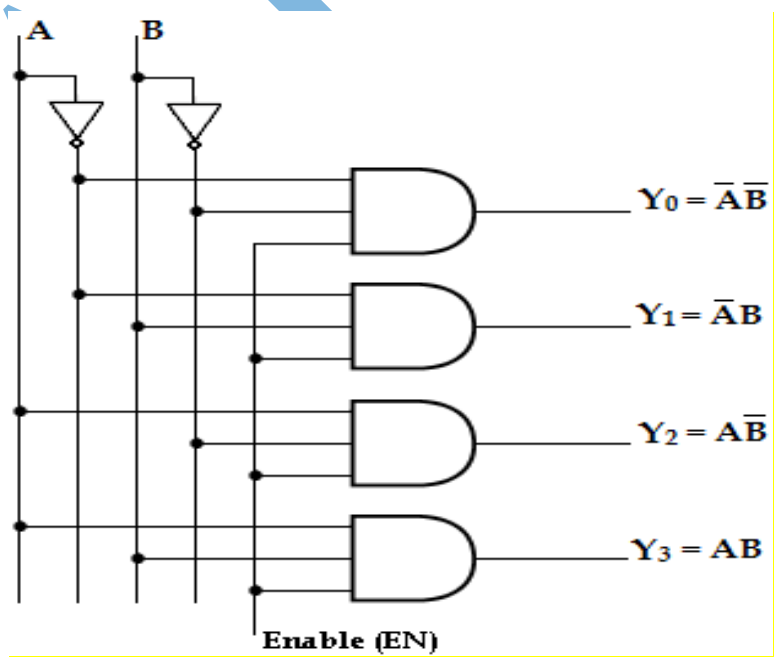
A decoder is a combinational circuit that converts binary information from 'n' input lines to a maximum of '2n' unique output lines. The general structure of decoder circuit is



The encoded information is presented as 'n' inputs producing '2n' possible outputs. The 2n output values are from 0 through 2n-1. A decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

Binary Decoder (2 to 4 decoder)

A binary decoder has 'n' bit binary input and a one activated output out of 2n outputs. A binary decoder is used when it is necessary to activate exactly one of 2n outputs based on an n-bit input value.



Inputs **Outputs**

Enable	**A**	**B**	**Y3**	**Y2**	**Y1**	**Y0**
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

As shown in the truth table, if enable input is 1 (EN= 1) only one of the outputs (Y0 – Y3), is active for a given input.

The output Y0 is active, ie., Y0= 1 when inputs A= B= 0,

Y1 is active when inputs, A= 0 and B= 1,

Y2 is active, when input A= 1 and B= 0,

Y3 is active, when inputs A= B= 1.

3-to-8 Line Decoder

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y0- Y7). Based on the 3 inputs one of the eight outputs is selected.

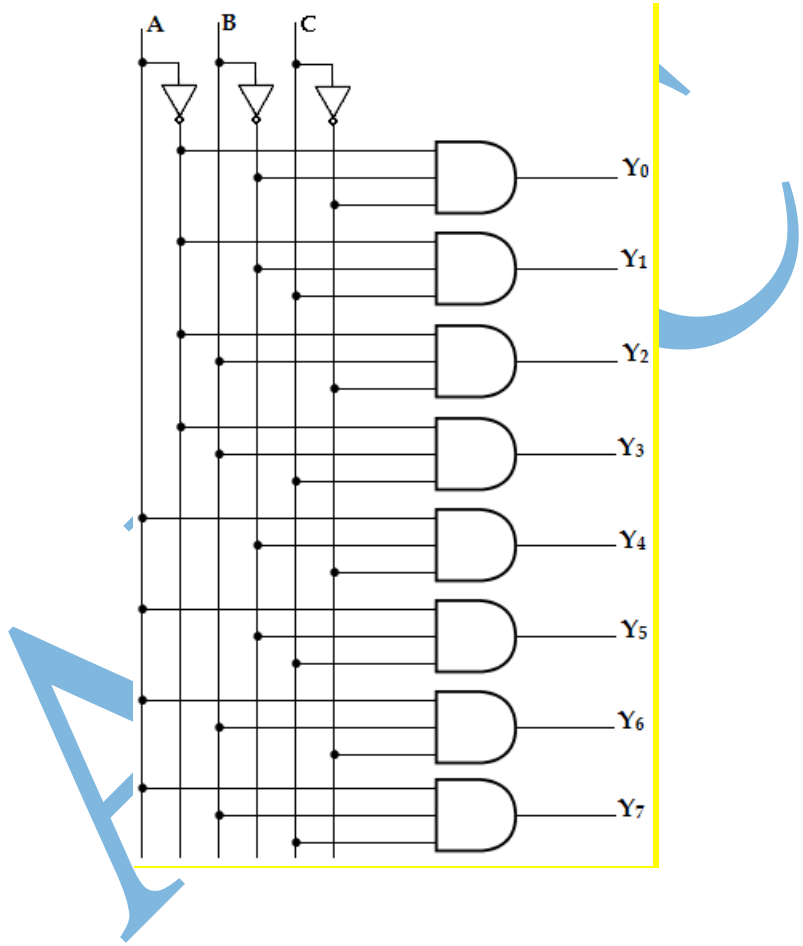
The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion. The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system.

The output variables are mutually exclusive because only one output can be equal to 1 at any one time.

The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

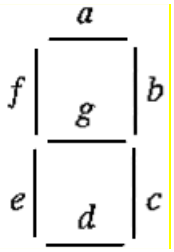
Inputs			**Outputs**							
A	**B**	**C**	**Y0**	**Y1**	**Y2**	**Y3**	**Y4**	**Y5**	**Y6**	**Y7**
0	0	0	1	0	0	0	0	0	0	0

0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



BCD to 7-Segment Display Decoder

A seven-segment display is normally used for displaying any one of the decimal digits, 0 through 9. A BCD-to-seven segment decoder accepts a decimal digit in BCD and generates the corresponding seven-segment code.



Each segment is made up of a material that emits light when current is passed through it. The segments activated during each digit display are tabulated as

Digit	Segments Activated
0	a, b, c, d, e, f
1	b, c
2	a, b, d, e, g
3	a, b, c, d, g
4	b, c, f, g
5	a, c, d, f, g
6	a, c, d, e, f, g
7	a, b, c
8	a, b, c, d, e, f, g
9	a, b, c, d, f, g

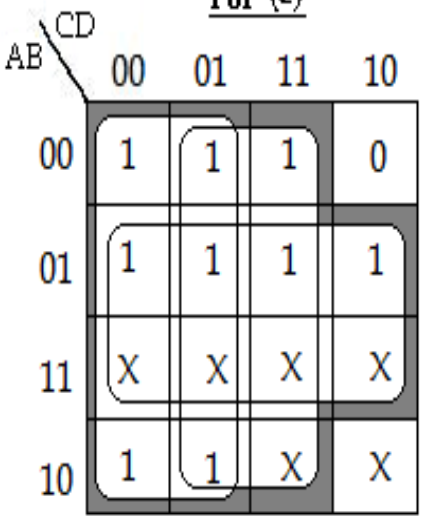
APEC

Truth table:

Digit				BCD code				7-Segment code			
A	**B**	**C**	**D**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0

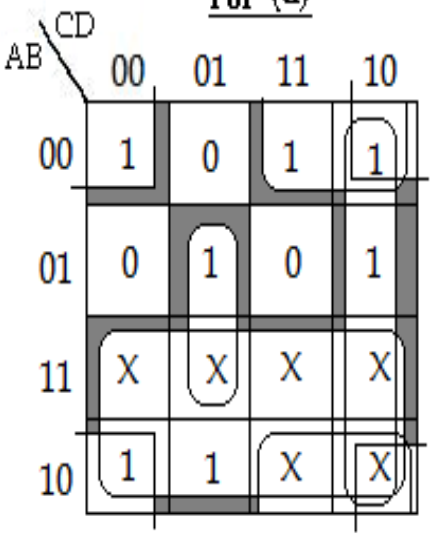
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

For (c)



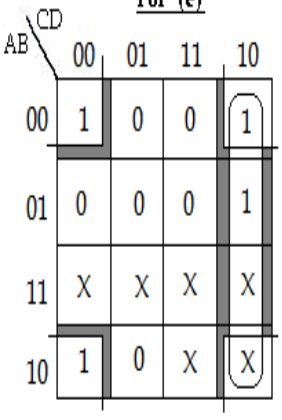
c= B+ C'+ D

For (d)



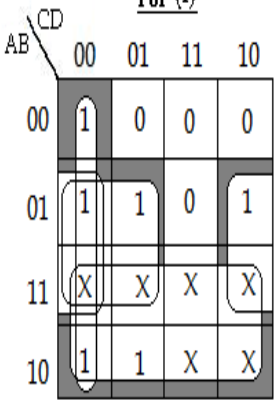
d= B'D'+ CD'+ BC'D+ B'C+ A

For (e)



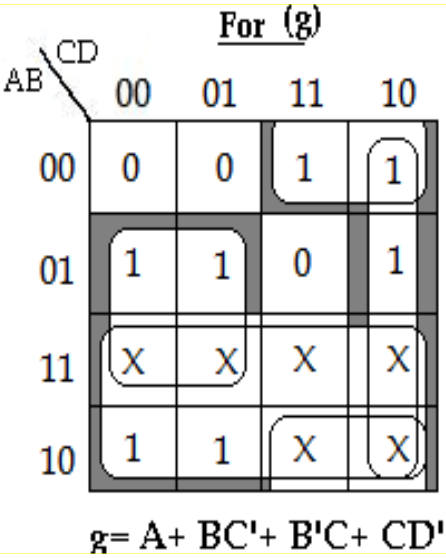
e= B'D'+ CD'

For (f)

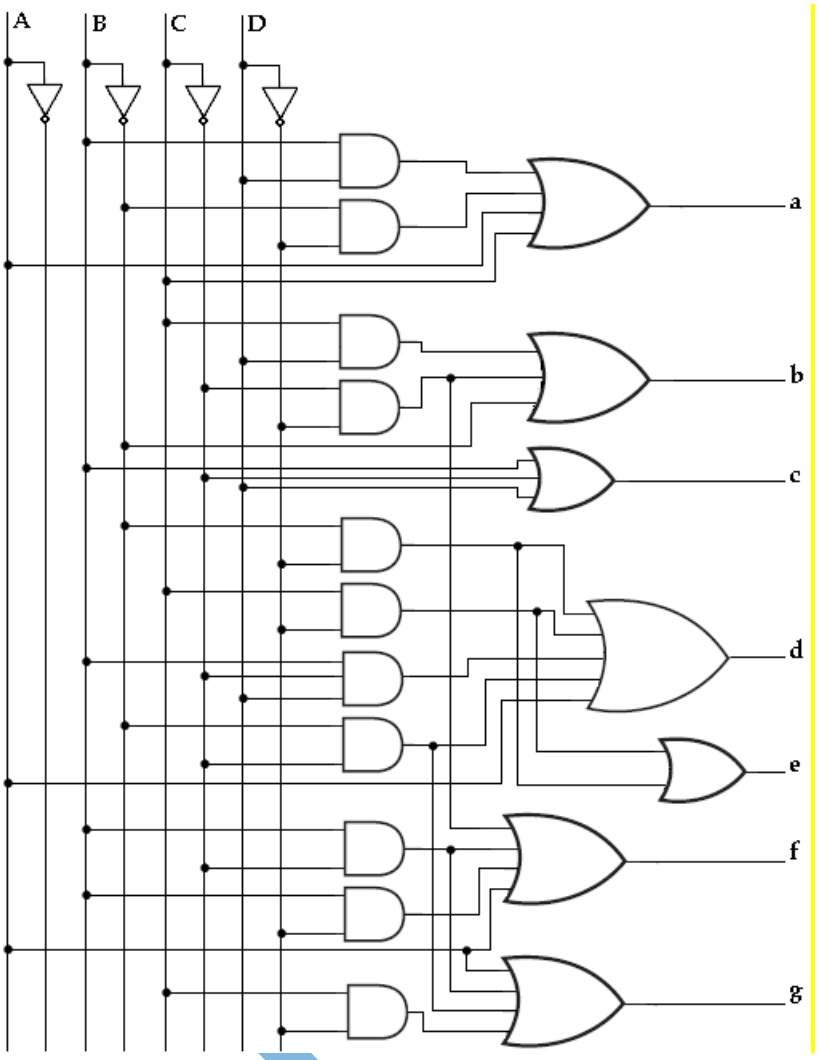


f= A+ C'D'+ BC'+ BD'

For (g)



g= A+ BC'+ B'C+ CD'



Applications of decoders:

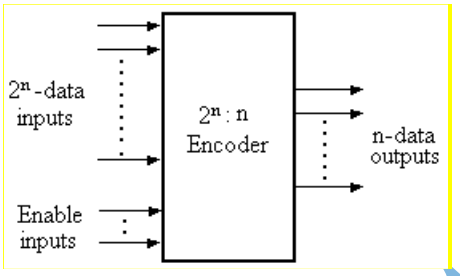
1. Decoders are used in counter system.
2. They are used in analog to digital converter.
3. Decoder outputs can be used to drive a display system.
4. Address decoding
5. Implementation of combinational circuits.
6. Code converters

ENCODERS

An encoder is a digital circuit that performs the inverse operation of a decoder. Hence, the opposite of the decoding process is called encoding. An encoder is a

combinational circuit that converts binary information from 2n input lines to a maximum of ‘n’ unique output lines.

The general structure of encoder circuit is



It has 2n input lines, only one which 1 is active at any time and ‘n’ output lines. It encodes one of the active inputs to a coded binary output with ‘n’ bits. In an encoder, the number of outputs is less than the number of inputs.

Octal-to-Binary Encoder

It has eight inputs (one for each of the octal digits) and the three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time.

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1, when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7 and the output is 1 for digits 4, 5, 6 or 7.

Inputs								**Outputs**		
D0	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**A**	**B**	**C**
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0

0	0	0	0	0	0	0	1	1	1	1

These conditions can be expressed by the following output Boolean functions:

z= D1+ D3+ D5+ D7

y= D2+ D3+ D6+ D7

x= D4+ D5+ D6+ D7

The encoder can be implemented with three OR gates. The encoder defined in the below table, has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, output produces an undefined combination.

For eg., if D3 and D6 are 1 simultaneously, the output of the encoder may be 111. This does not represent either D6 or D3. To resolve this problem, encoder circuits must establish an input priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers and if D3 and D6 are 1 at the same time, the output will be 110 because D6 has higher priority than D3.

Octal-to-Binary Encoder

Another problem in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; this output is same as when D0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate that atleast one input is equal to 1.

Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

In addition to the two outputs x and y, the circuit has a third output, V (valid bit indicator). It is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.

Another problem in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; this output is same as when D0 is equal to 1.

The discrepancy can be resolved by providing one more output to indicate that atleast one input is equal to 1.

Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

In addition to the two outputs x and y, the circuit has a third output, V (valid bit indicator). It is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The higher the subscript number, higher the priority of the input. Input D3, has the highest priority. So, regardless of the values of the other inputs, when D3 is 1, the output for xy is 11.

D2 has the next priority level. The output is 10, if D2= 1 provided D3= 0. The output for D1 is generated only if higher priority inputs are 0, and so on down the priority levels.

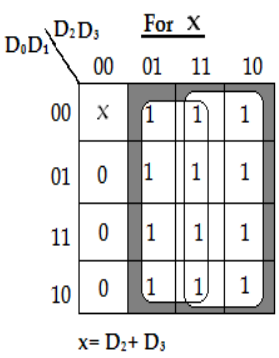
Inputs				**Outputs**		
D0	**D1**	**D2**	**D3**	**x**	**y**	**V**
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

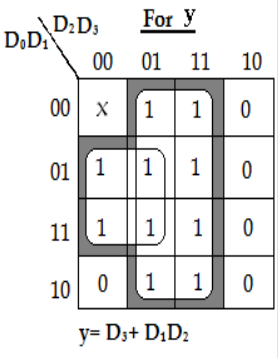
Although the above table has only five rows, when each don't care condition is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example, the third row in the table with X100 represents minterms 0100 and 1100. The don't care condition is replaced by 0 and 1 as shown in the table below.

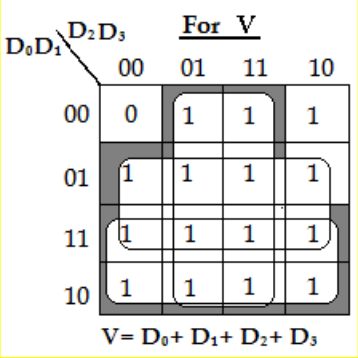
Modified Truth table:

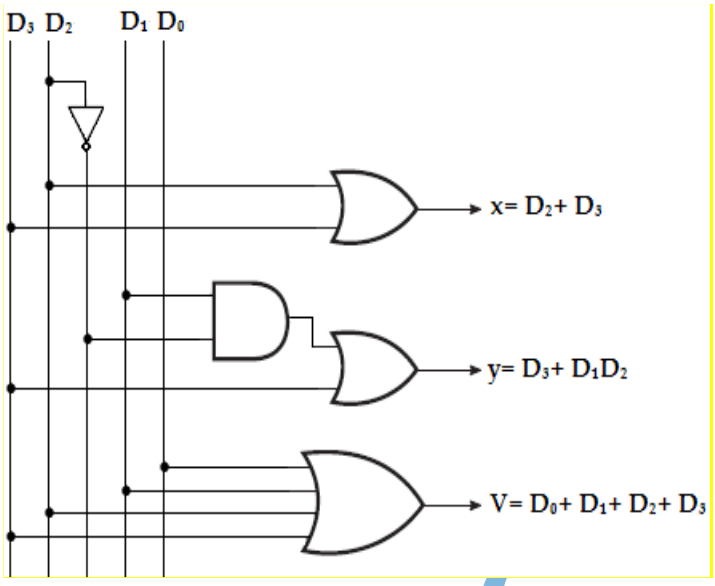
Inputs				Outputs		
D0	**D1**	**D2**	**D3**	**x**	**y**	**V**
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0	1	0	1
0	1	1	0			
1	0	1	0			
1	1	1	0			
0	0	0	1	1	1	1
0	0	1	1			
0	1	0	1			
0	1	1	1			
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

APEC



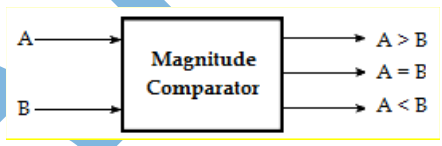






MAGNITUDE COMPARATOR

A ***magnitude comparator*** is a combinational circuit that compares two given numbers (A and B) and determines whether one is equal to, less than or greater than the other. The output is in the form of three binary variables representing the conditions A = B, A>B and A<B, if A and B are the two numbers being compared.



For comparison of two *n*-bit numbers, the classical method to achieve the Boolean expressions requires a truth table of 22*n* entries and becomes too lengthy and cumbersome.

2-bit Magnitude Comparator

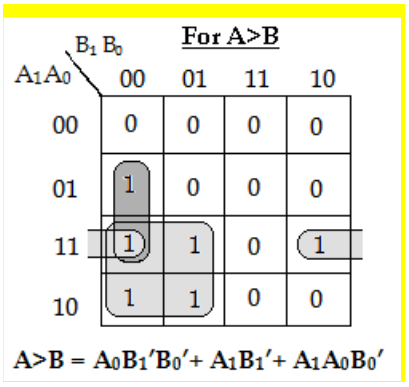
The truth table of 2-bit comparator is given in table below

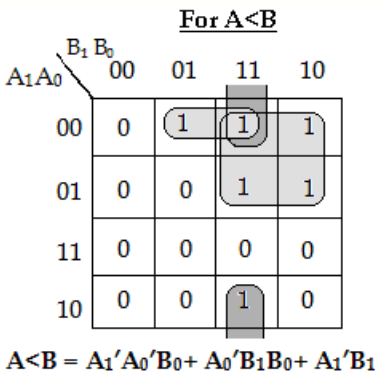
Truth table:

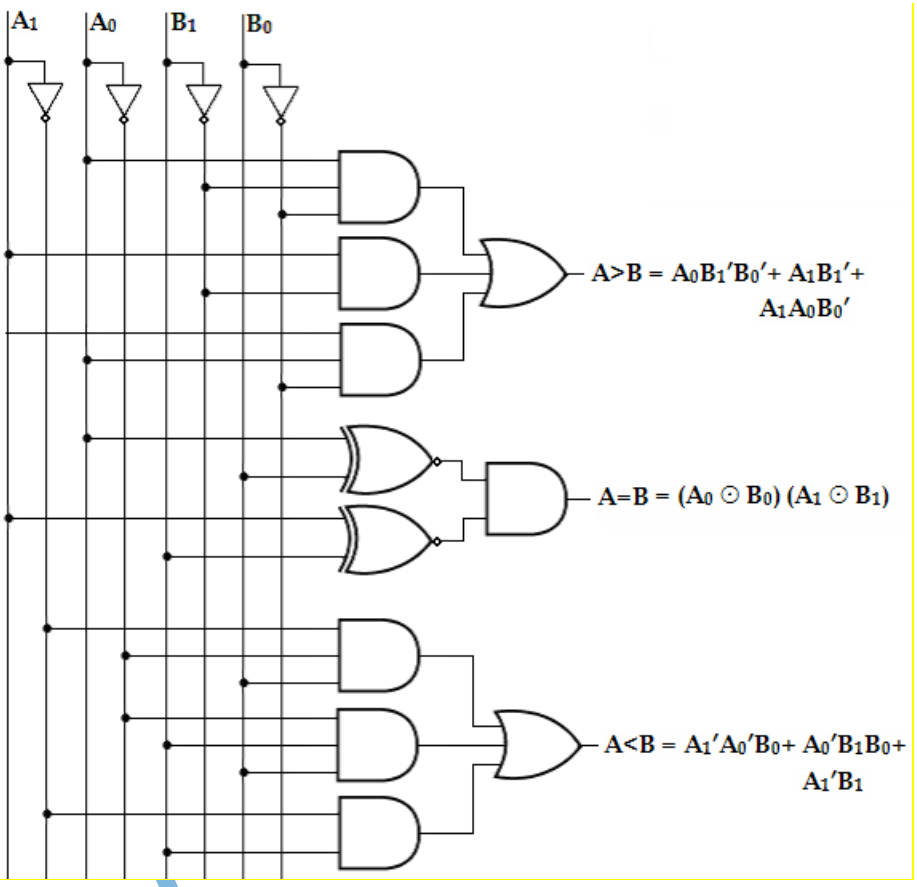
Inputs				**Outputs**		
A1	**A0**	**B1**	**B0**	**A>B**	**A=B**	**A<B**

0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

APEC







4-bit Magnitude Comparator:

Let us consider the two binary numbers A and B with four digits each. Write the coefficient of the numbers in descending order as,

A = A3A2A1A0

B = B3 B2 B1 B0,

Each subscripted letter represents one of the digits in the number. It is observed from the bit contents of two numbers that A = B when A3 = B3, A2 = B2, A1 = B1 and A0 = B0. When the numbers are binary they possess the value of either 1

or 0, the equality relation of each pair can be expressed logically by the equivalence function as,

$$Xi = AiBi + Ai'Bi' \quad \text{for } i = 1, 2, 3, 4.$$

$$Xi = (A \oplus B)'. \quad \text{or, } Xi' = A \oplus B$$

$$Xi = (AiBi' + Ai'Bi)'.$$

where, X*i* =*1* only if the pair of bits in position i are equal (ie., if both are 1 or both are 0).

To satisfy the equality condition of two numbers A and B, it is necessary that all X*i* must be equal to logic 1. This indicates the AND operation of all X*i* variables. In other words, we can write the Boolean expression for two equal 4-bit numbers.

(A = B) = X3X2X1 X0.

The binary variable (A=B) is equal to 1 only if all pairs of digits of the two numbers are equal.

To determine if A is greater than or less than B, we inspect the relative magnitudes of pairs of significant bits starting from the most significant bit. If the two digits of the most significant position are equal, the next significant pair of digits is compared. The comparison process is continued until a pair of unequal digits is found. It may be concluded that A>B, if the corresponding digit of A is 1 and B is 0. If the corresponding digit of A is 0 and B is 1, we conclude that A<B. Therefore, we can derive the logical expression of such sequential comparison by the following two Boolean functions,

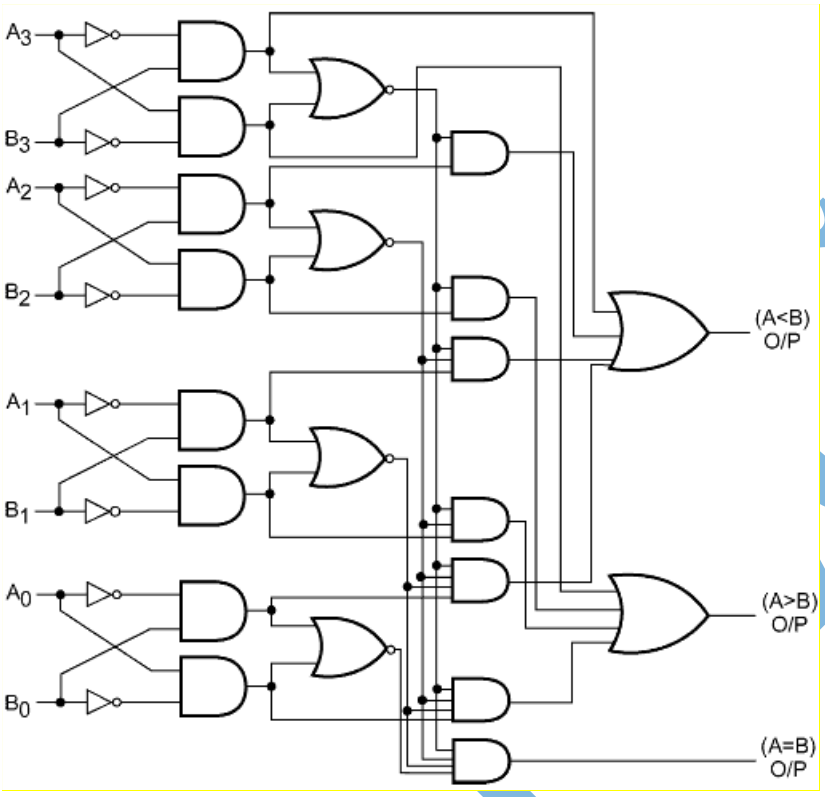
(A>B) = A3B3′ +X3A2B2′ +X3X2A1B1′ +X3X2X1A0B0′

(A<B) = A3′B3 +X3A2′B2 +X3X2A1′B1 +X3X2X1A0′B0

The symbols (A>B) and (A<B) are binary output variables that are equal to 1 when A>B or A<B, respectively.

APEC

The gate implementation of the three output variables just derived is simpler than it seems because it involves a certain amount of repetition. The unequal outputs can use the same gates that are needed to generate the equal output. The logic diagram of the 4-bit magnitude comparator is shown below,



The four x outputs are generated with exclusive-NOR circuits and applied to an AND gate to give the binary output variable (A=B). The other two outputs use the x variables to generate the Boolean functions listed above. This is a multilevel implementation and has a regular pattern