UNIT -3
COMBINATIONAL LOGIC
TOPIC – 1
COMBINATIONAL CIRCUITS ANALYSIS & DESIGN PROCEDURE

Digital logic circuits

- There are two types of Digital circuits depending on their output and memory used:
  (i) Combinational circuit, and (ii) Sequential circuit
- A combinational logic circuit is one whose output solely depends on its current inputs.
- Sequential circuits are built using combinational circuits and memory elements called "flip-flops".
- These circuits generate output that depends on the current and previous states.

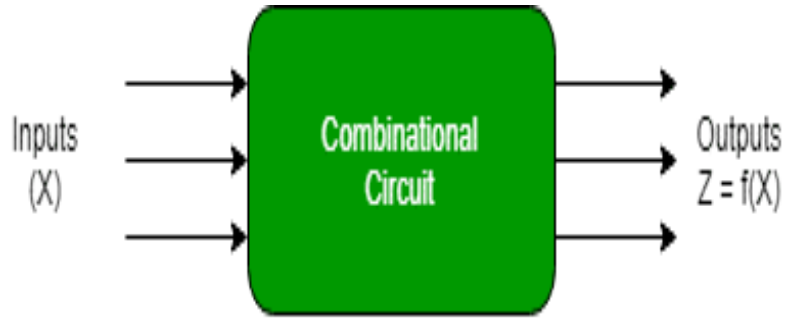


Figure: Combinational Circuits

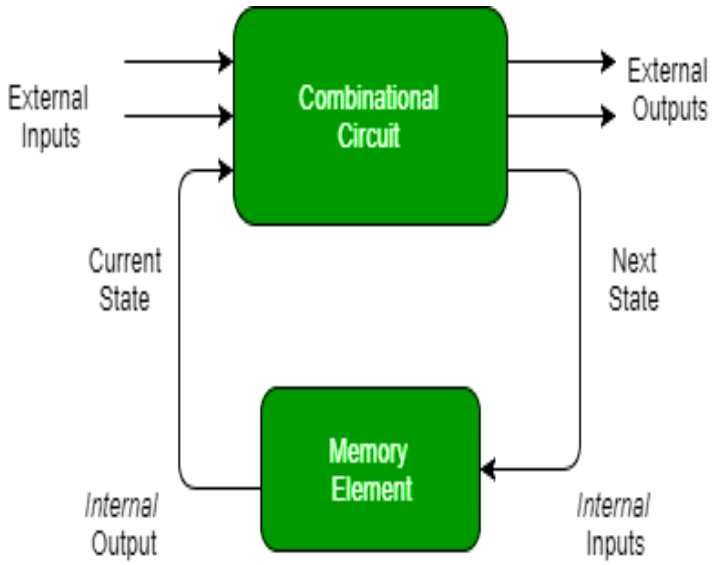
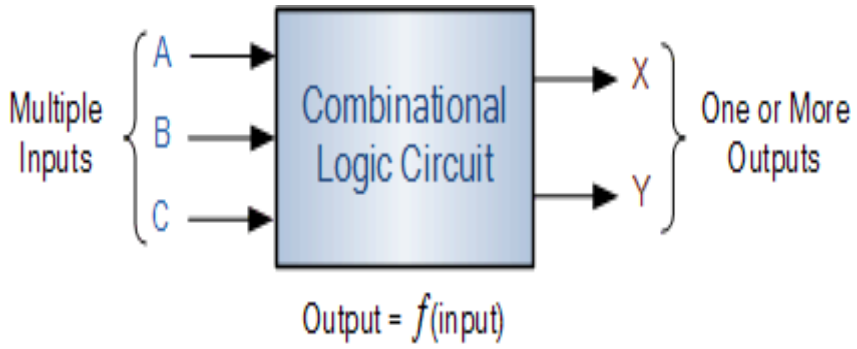


Figure: Sequential Circuit

Combinational Logic Circuits

- Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs.



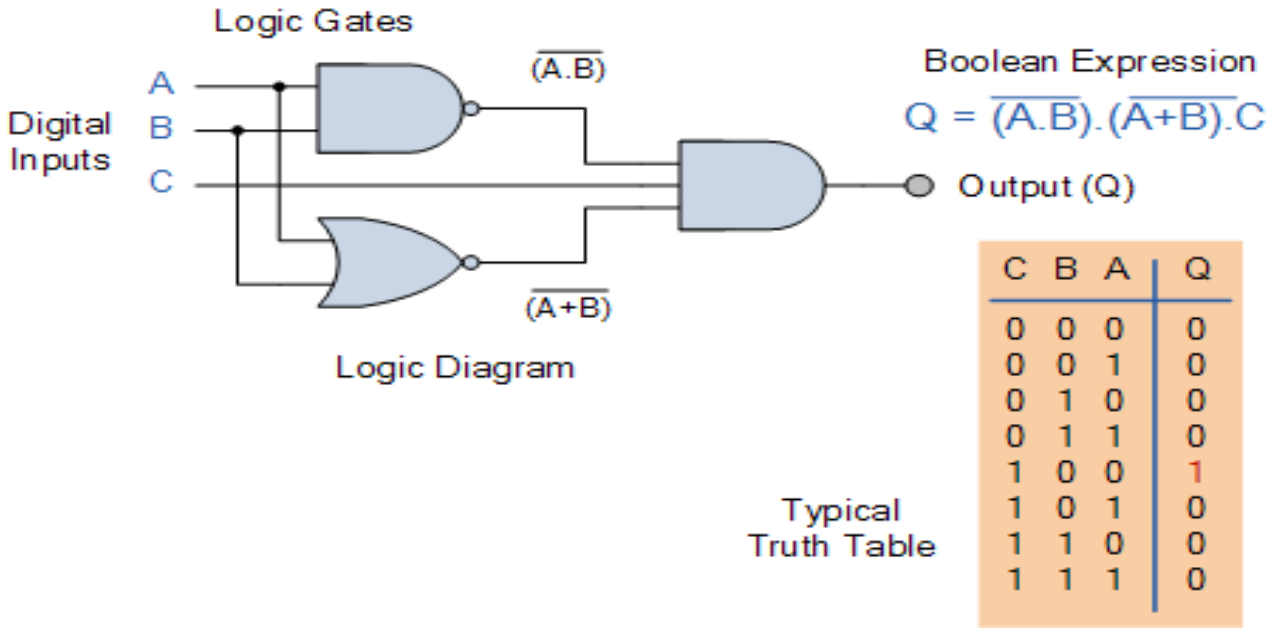
- The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

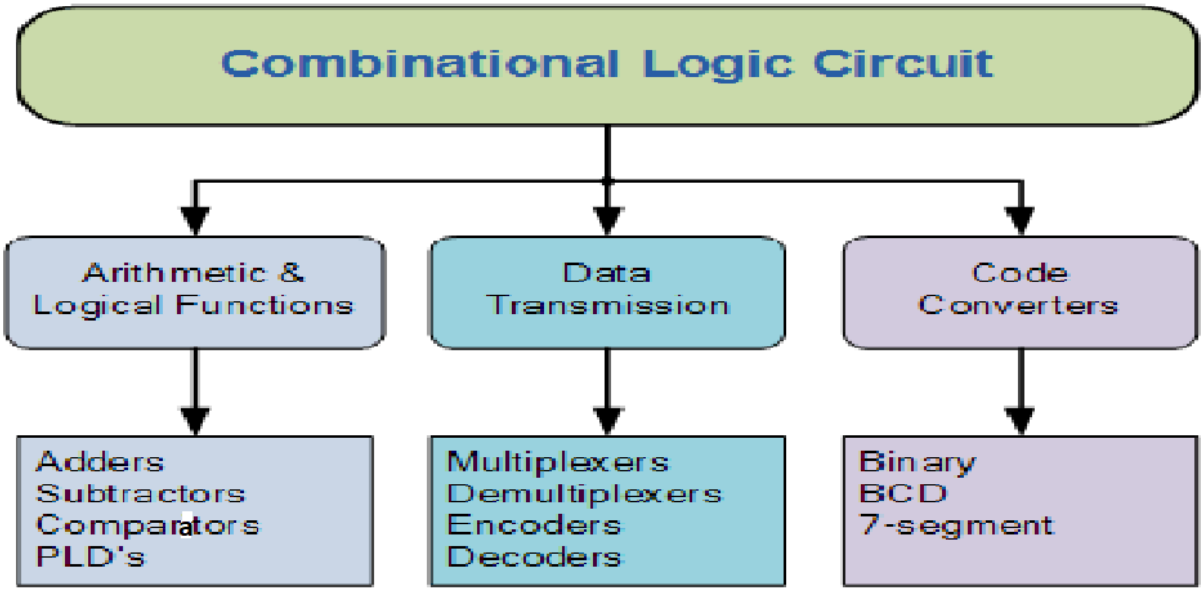
- The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependent at all times on the combination of its inputs. Thus a combinational circuit is memoryless.

- **Combinational Logic Circuits** are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits.

- The three main ways of specifying the function of a combinational logic circuit are:

1. **Boolean Algebra** – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.
2. **Truth Table** – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
3. **Logic Diagram** – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.





Design Procedure:

The procedure involves the following steps:

- Determine the required inputs and outputs and assign a symbol to each.
- Derive the truth table that defines the required relationship between inputs and outputs.
- Obtain the simplified Boolean functions for each output as a function of the input variables.
- Draw the logic diagram and verify the correctness of the design.

UNIT -3

COMBINATIONAL LOGIC

TOPIC – 2

BINARY ADDERS(HALF-ADDER,FULL-ADDER&BINARY PARALLEL ADDER)

Half Adder:

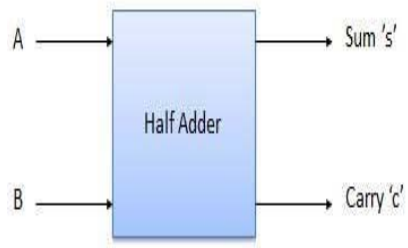
Half adder is a combinational logic circuit with two inputs and two outputs.

The half adder circuit is designed to add two single bit binary number A and B.

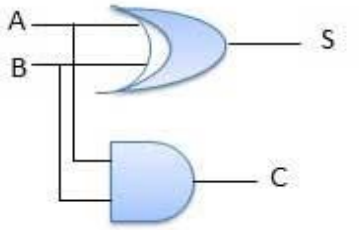
It is the basic building block for addition of two single bit numbers.

This circuit has two outputs carry and sum.

Block diagram **Truth Table** **Circuit Diagram**



Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



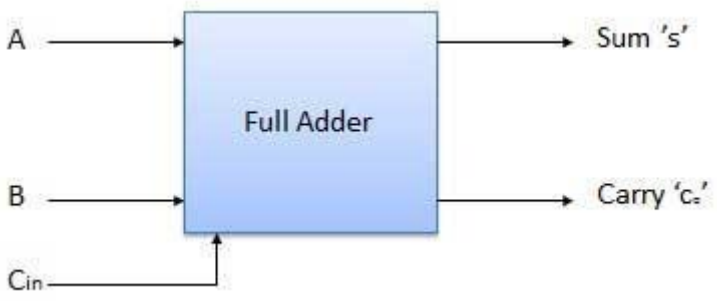
Half adder sum and carry equation is

Sum(S)= A ⊕ B =A'B+AB' **Carry(C) = A.B**

Full Adder:

- Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry C_{IN}.
- The full adder is a three input and two output combinational circuit.

Block diagram



TRUTH TABLE:

Inputs			Outputs	
A	B	C – IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the above table,

1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added.
2. 'C_{in}' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

Logical Expression for SUM: $=\Sigma\, m\,(1,2,4,7) = A'\ B'\ C_{IN} + A'\ B\ C'_{IN} + A\ B'\ C'_{IN} + A\ B\ C_{IN}$

$= C_{IN}\ (A'\ B' + A\ B) + C'_{IN}\ (A'\ B + A\ B')$

$= C_{IN}(\overline{A \oplus B}) + C'_{IN}\ (A \oplus B)$

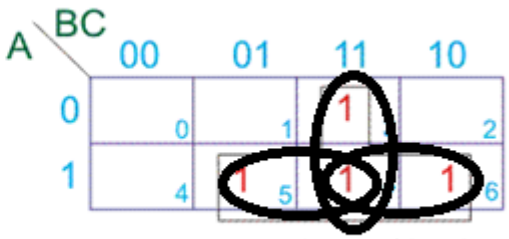
$= (A \oplus B) \oplus C_{IN}$

$= A \oplus B \oplus C_{IN}$

Logical Expression for C-OUT: $= \Sigma\, m\,(3,5,6,7) = A'\ B\ C_{IN} + A\ B'\ C_{IN} + A\ B\ C'_{IN} + A\ B\ C_{IN}$

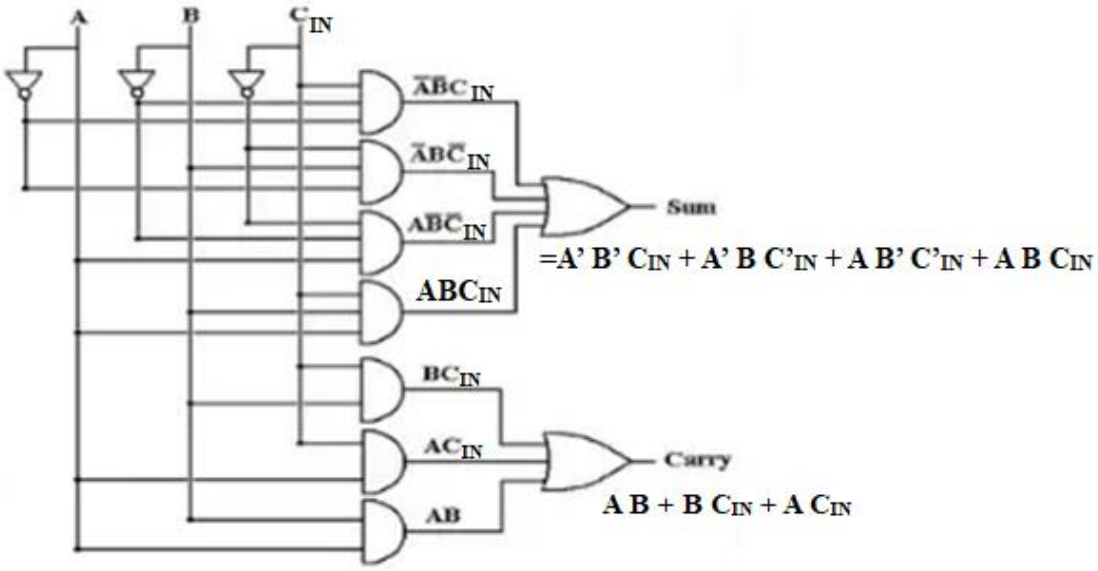
$= A\ B + B\ C_{IN} + A\ C_{IN}$

K-map for Sum (S)



K-map for Carry (C_{out})

AOI Logic diagram:



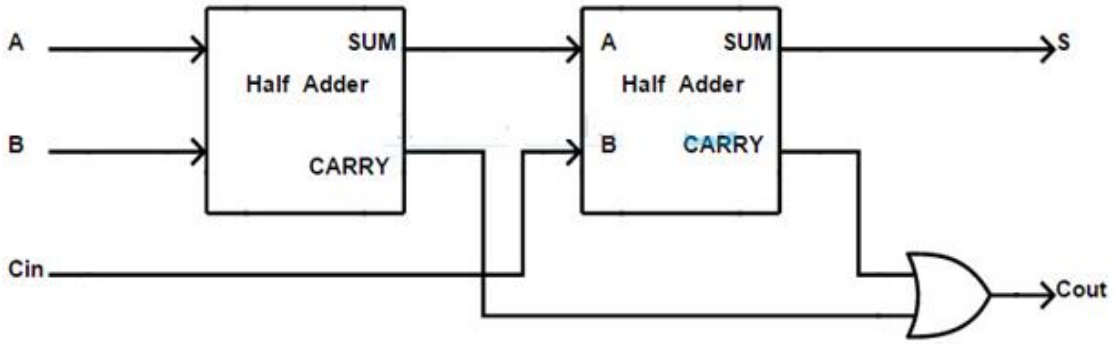
Implementation of Full Adder using Half Adders:

A full adder can be formed by logically connecting two half adders.

$$
\begin{aligned}
S &= A\overline{B}\overline{C} + \overline{A}\,\overline{B}C + ABC + \overline{A}B\overline{C} \\
&= C\,(AB + \overline{A}\,\overline{B}) + \overline{C}\,(\overline{A}B + A\,\overline{B}) \\
&= C\,\overline{(\overline{A}B + A\,\overline{B})} + \overline{C}\,(\overline{A}B + A\,\overline{B}) \\
&= C\,(\overline{A \oplus B}) + \overline{C}\,(A \oplus B) \;= A \oplus B \oplus C.
\end{aligned}
$$

$$
\begin{aligned}
C_{out} &= \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \\
&= (\overline{A}B + A\overline{B})\,C + AB\,(\overline{C} + C) \\
&= (A \oplus B).C + AB.
\end{aligned}
$$

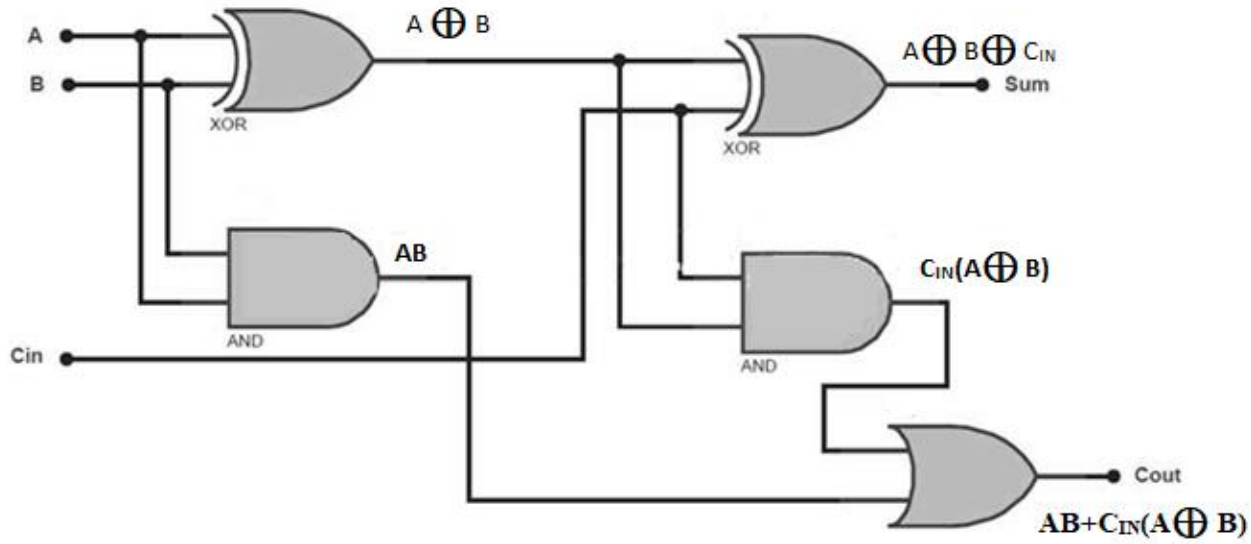
The following is a block diagram that shows the implementation of a full adder using two half adders.



- The first half adder has two single-bit binary inputs A and B. As we know that, the half adder produces two outputs, i.e., Sum and Carry.
- The 'Sum' output of the first adder will be the first input of the second half adder, and the 'Carry' output of the first adder will be the second input of the second half adder.
- The second half adder will again provide 'Sum' and 'Carry'. The final outcome of the Full adder circuit is the 'Sum' bit.
- In order to find the final output of the 'Carry', we provide the 'Carry' output of the first and the second adder into the OR gate.
- The outcome of the OR gate will be the final carry out of the full adder circuit.

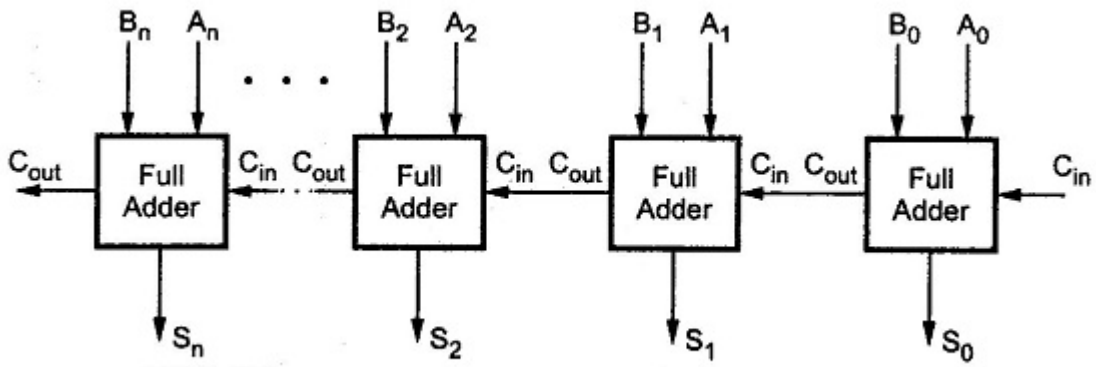
The MSB is represented by the final 'Carry' bit.

The full adder logic circuit can be constructed using the **'AND'** and **the 'XOR' gate** with an **OR gate**.



N-Bit Parallel Adder:

- The Full Adder is capable of adding only two single digit binary numbers along with a carry input.
- To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade.
- The carry output of the previous full adder is connected to carry input of the next full adder.

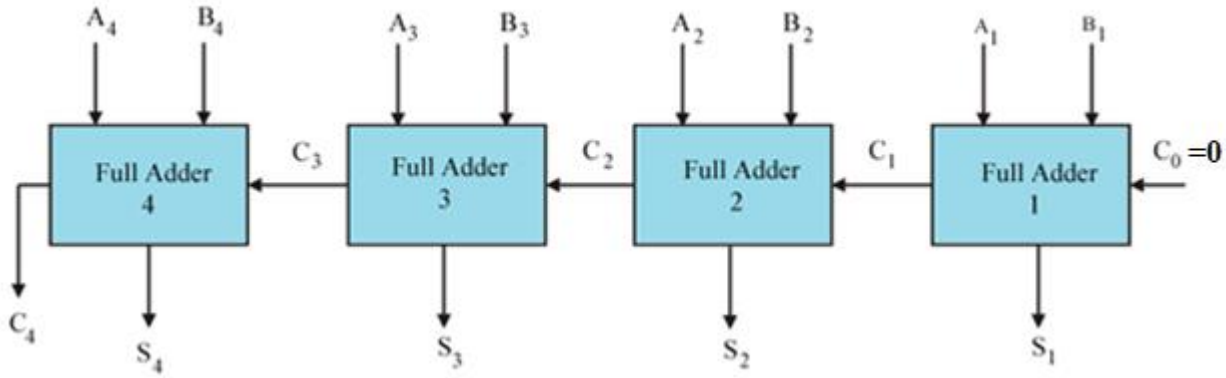


Block diagram of n-bit parallel adder

4-Bit Parallel Adder:

- In the block diagram, A_0 and B_0 represent the LSB of the four-bit words A and B. Hence Full Adder-0 is the lowest stage.
- Hence its C_0 has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig.
- The four-bit parallel adder is a very common logic circuit.
- Parallel Adder is a digital circuit that efficiently adds more than 1-bit binary numbers. Parallel Adders are implemented using Full Adders.

Block Diagram



How does Parallel Adder Work:

- To understand the working principle of Parallel Adder, Let us understand the construction of Parallel Adder as shown in the Figure.
- 4- bit Parallel Adder is designed using 4 Full Adders FA_1, FA_2, FA_3, FA_4 . Full Adder FA_1 adds A_1, B_1 along with carry C_0 to generate Sum S_1 and Carry bit C_1 and this Carry bit is connected to FA_2.
- FA_2 accepts this Carry C_1 and adds with its inputs A_2 and B_2 to generate Sum S_2 and Carry C_2. This bit C_2 is connected to FA_3.

- This process continues till the last Full Adder. FA_4 accepts the carry bit C_3 and adds with its input A_4 and B_4 to generate the final output S_4 along with the last carry bit C_4.
- Let us examine the justification of the above circuit by taking an example of addition of two 4-bit binary numbers.
- Let us add 1011with 1101.

$$\begin{array}{r} 1\,0\,1\,1 \\ 1\,1\,0\,1 \\ \hline 1\,1\,0\,0\,0 \end{array}$$

$$Here,\ A_1 = 1,\ A_2 = 1,\ A_3 = 0\ A_4 = 1.$$
$$B_1 = 1,\ B_2 = 0,\ B_3 = 1\ B_4 = 1.$$

- As there is no previous carry $C_0 = 0$.

Therefore, final result of the addition would be

$$C_4\ S_4\ S_3\ S_2\ S_1 = 11000$$

$$Now,\ C_0 + A_1 + B_1 = 0 + 1 + 1 = 10 \rightarrow S_1 = 0, C_1 = 1$$
$$C_1 + A_2 + B_2 = 1 + 1 + 0 = 10 \rightarrow S_2 = 0, C_2 = 1$$
$$C_2 + A_3 + B_3 = 1 + 0 + 1 = 10 \rightarrow S_3 = 0, C_3 = 1$$
$$C_3 + A_4 + B_4 = 1 + 1 + 1 = 10 \rightarrow S_4 = 1, C_4 = 1$$

UNIT -3

COMBINATIONAL LOGIC

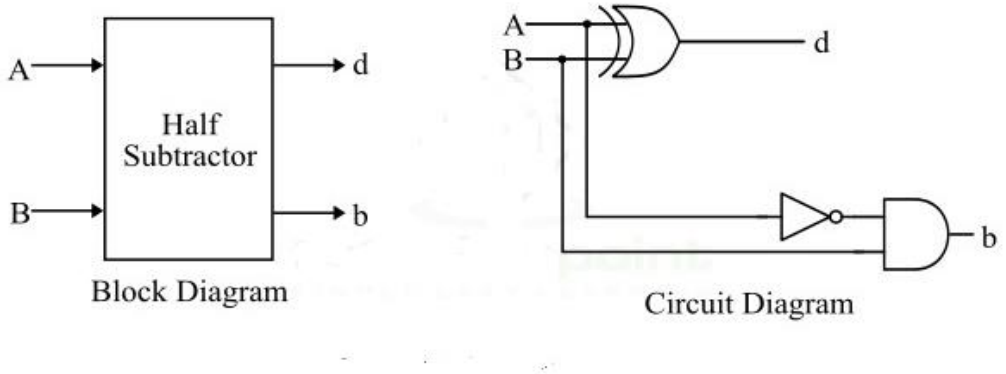
TOPIC – 3

BINARY SUBTRACTOR(HALF-SUBTRACTOR,FULL-SUTRACTOR&BINARY PARALLEL SUBTRACTOR)

HALF SUBTRACTORS:

- Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow).
- It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed.
- In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

The block diagram and logic circuit diagram of the half subtractor is shown in Figure below



- Hence, from the logic circuit diagram, it is clear that a half subtractor can be realized using an XOR gate together with a NOT gate and an AND gate.
- In half subtractor , A and B are the inputs, d and b are the outputs. Where d indicates the difference and b indicates the borrow output.
- The borrow output (b) is the signal that tells the next stage that a 1 has been borrowed.
- From the logic circuit diagram of the half subtractor, it is clear that the difference bit (d) is obtained by the XOR operation of the two inputs A and B, and the borrow bit is obtained by AND operation of the compliment of the minuend (A') with the subtrahend (B).

Operation of Half Subtractor:

Half subtractor performs its operation to find the difference of two binary digits according to the rules of binary subtraction, which are as follows −

- The output borrow of b is zero (0) as long as the minuend bit (A) is greater than or equal to the subtrahend bit (B), i.e. $A \geq B$. The output borrow is a 1 when A = 0 and B = 1.

Truth Table of Half Subtractor

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-Map for Half Subtractor

We can use the K-Map (or Karnaugh Map), a method for simplifying Boolean algebra, to determine equations of the difference bit (d) and the output borrow (b).

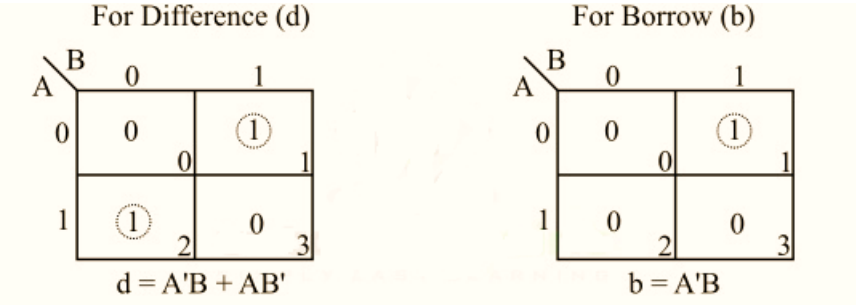


Figure - K Map for Half Subtractor

The difference bit (d) of the half subtractor is given by XORing the two inputs A and B.

Therefore, **Difference(d)=** A′B+AB′ = **A⊕B**

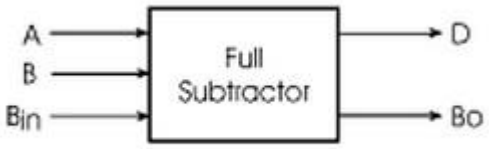
The borrow (b) of the half subtractor is the AND of A' (compliment of A) and B.

Therefore, **Borrow (b)=A′B**

FULL SUBTRACTOR:

- A **full-subtractor** is a combinational circuit that has three inputs A, B, B_{in} and two outputs D and B_O.
- Where, A is the minuend, B is subtrahend, b_{in} is borrow produced by the previous stage, D is the difference output and B_O is the borrow output.
- As we know that the half-subtractor can only be used for subtraction of LSB (least significant bit) of binary numbers.
- If there is any borrow during the subtraction of the LSBs of two binary numbers, then it will affect the subtraction of next stages.
- Therefore, the subtraction with borrow are performed by a full subtractor.

Block Diagram of Full-Adder:



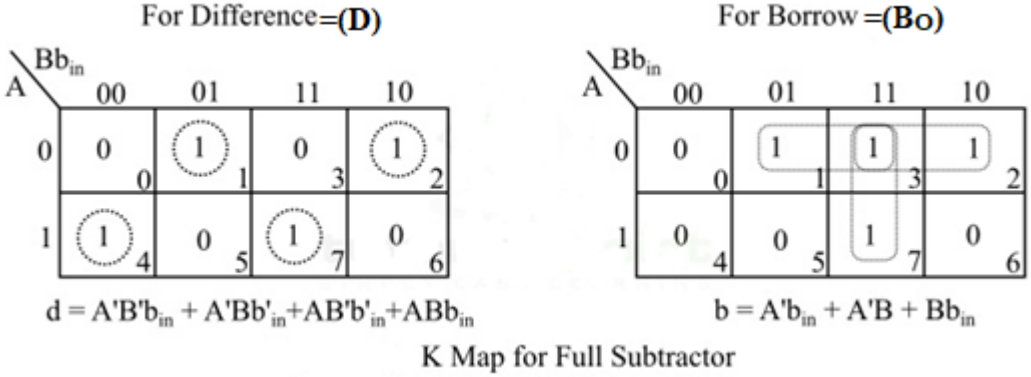
Truth Table of Full-Subtractor

The truth table is one that gives relationship between input and output of a logic circuit.

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
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

K-Map for Full Subtractor

We can use the K-Map (or Karnaugh Map), a method for simplifying Boolean algebra, to determine equations of the difference bit (D) and the output borrow bit (B_O).



K Map for Full Subtractor

Logical Expression for Difference(D): $=\Sigma\, m\,(1,2,4,7) = A'\ B'\ b_{in} + A'\ B\ b'_{in} + A\ B'\ b'_{in} + A\ B\ b_{in}$

$= b_{in}\ (A'\ B' + A\ B) + b'_{in}(A'\ B + A\ B')$

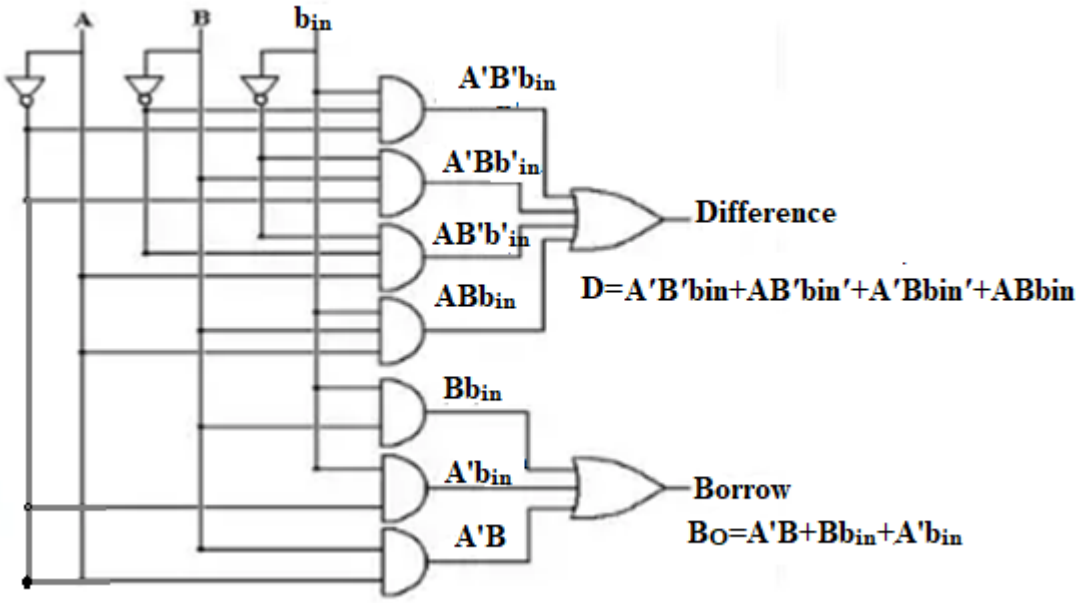
$= ((A \oplus B) \oplus b_{in})$

$= A \oplus B \oplus b_{in}$

Logical Expression for Borrow(Bo): = Σ m (3,5,6,7) = A’ B b_{in} + A B’ b_{in} + A B b'_{in} + A B b_{in}

=A’B + B b_{in} + A’b_{in}

AOI Logic diagram:



Implementation of Full Subtractor using Half Subtractor:

The difference (D) of the full subtractor is the XOR of A, B, and b_{in}. Therefore,

Difference, D = A’B’b_{in} + A’Bb'_{in} + AB’b'_{in} + ABb_{in}

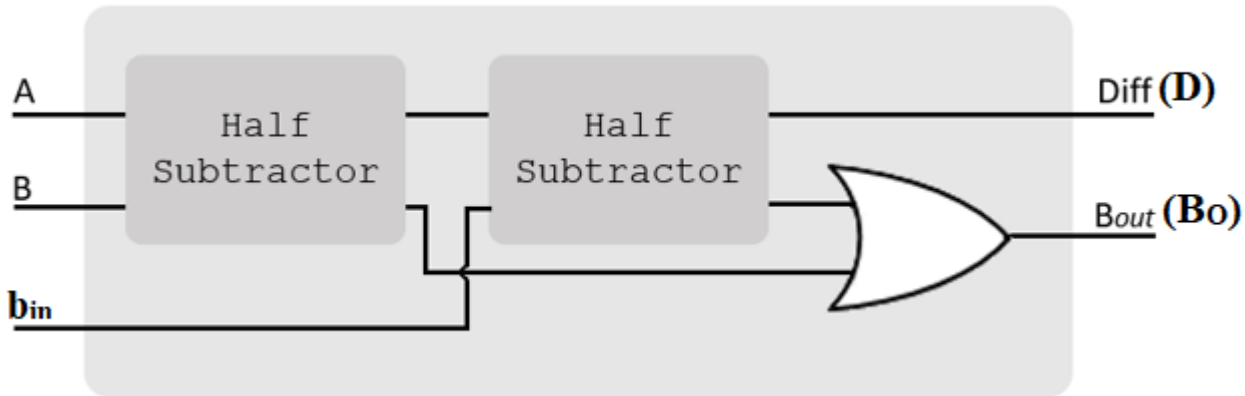
= b_{in} (A’ B’ + A B) +b'_{in}(A’ B + A B’)

= (A ⊕ B) ⊕ b_{in}

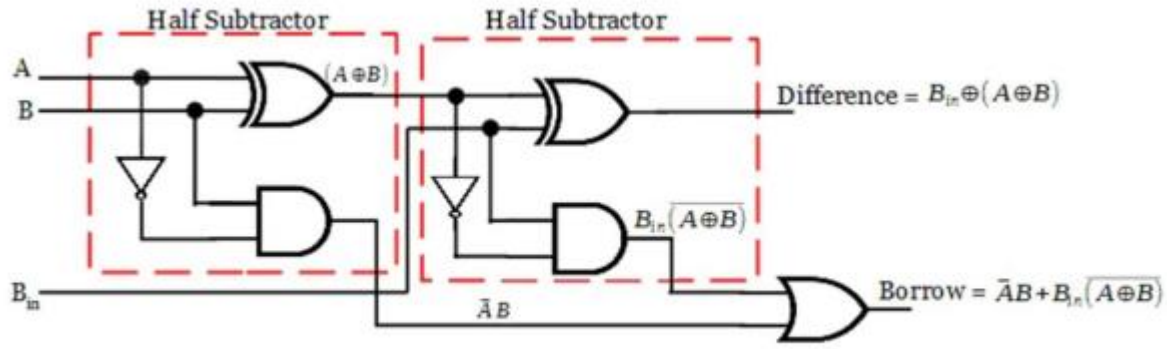
D = A ⊕ B ⊕ b_{in}

The borrow (B_O)= A’ B b_{in} + A B’ b_{in} + A B b'_{in} + A B b_{in}

B_O = A′B+(A⊕B)′ bin



Circuit Diagram:



Realization of full subtractor with two half subtractors

Therefore, we can realize the full-subtractor using two XOR gates, two NOT gates, two AND gates, and one OR gate.

Operation of Full Subtractor

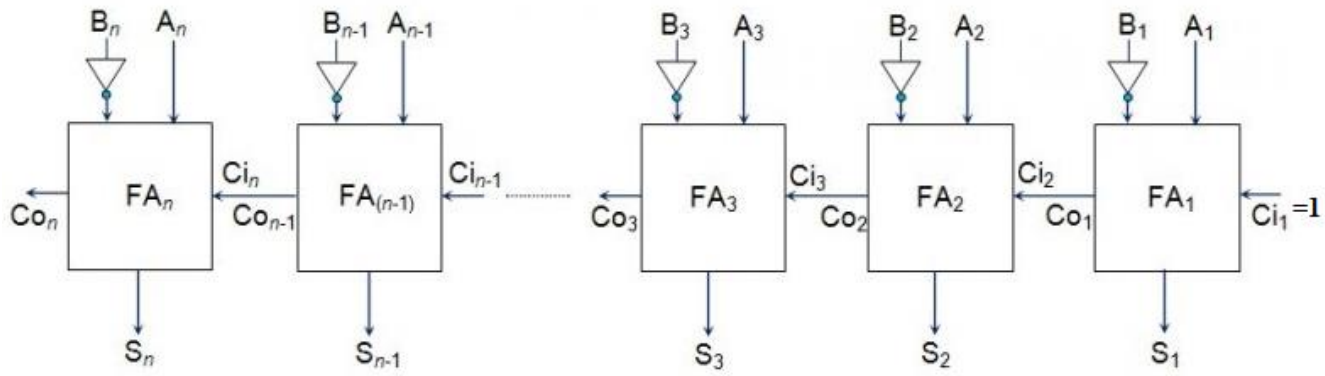
Full subtractor performs its operation to find the difference of two binary numbers according to the rules of binary subtraction, which are as follows −

In the case of full subtractor, the 1s and 0s for the output variables (difference and borrow) are determined from the subtraction of A – B – b_{in}.

From the logic circuit diagram of the full subtractor, it is clear that the difference bit (D) is obtained by the XOR operation of the two inputs A, B, and b_{in}, and the output borrow bit (B_O) is obtained by NOT, AND, and OR operations of variable A, B, and b_{in}.

N-BIT PARALLEL SUBTRACTOR

- The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted.
- For example we can perform the subtraction (A-B) by adding either 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.



- The subtractor can be arranged with a combination of all Full-Adders with the subtrahend complement i/p.

- The procedure of subtraction can be done by considering the addition of minuend with subtrahend's 2's complement. So that parallel subtraction can be done.

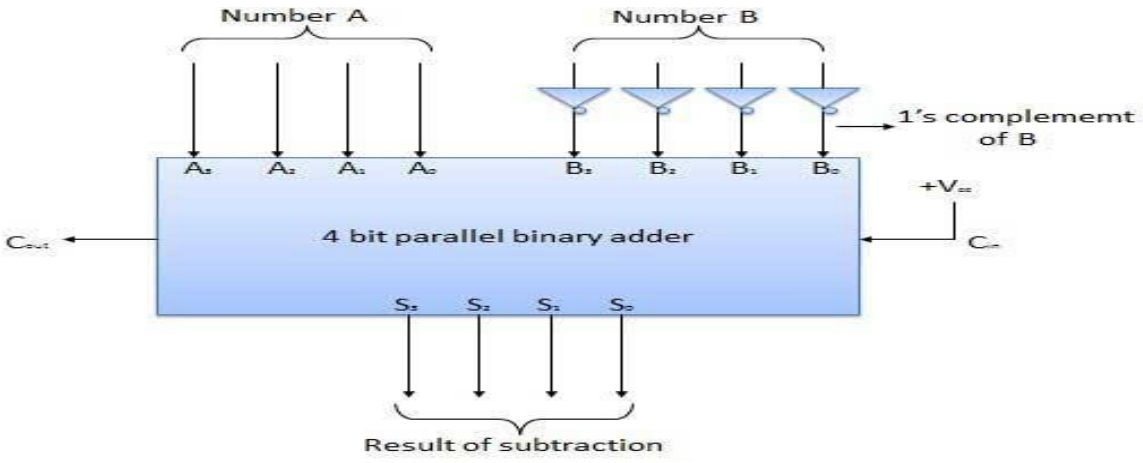
- The two's complement of a number can be done by converting the binary number into 1's complement. Here 1's complement is to negate the binary number. Here, by adding 1 to LSB bit of 1's complement, 2's complement can be obtained.

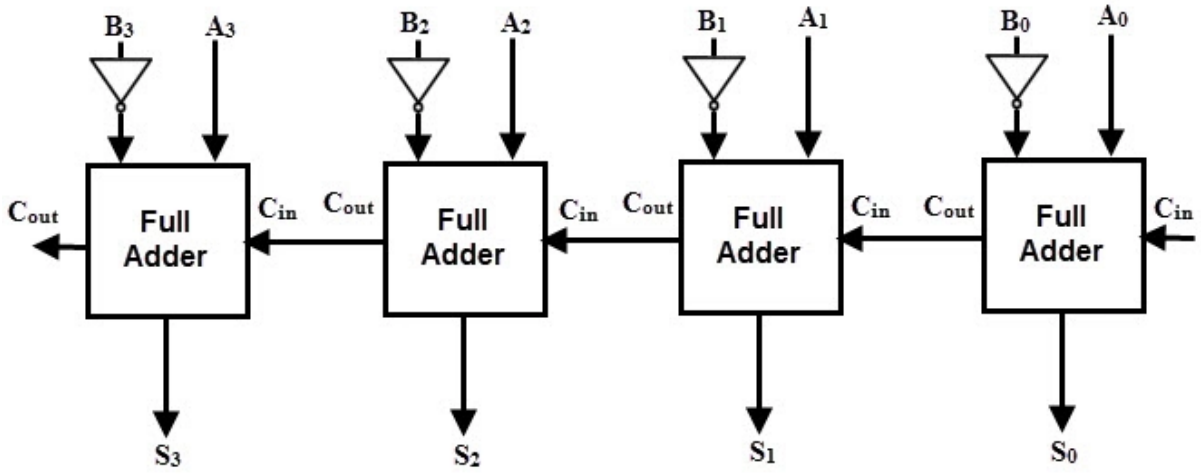
- The 1's complement of 'B' can be attained through the NOT logic gate & '1' is added throughout the carry to get the 2's complement of 'B'. Further, this is added to 'A' to perform the arithmetic subtraction.
- This procedure will continue till the final full adder like 'FAn' and it utilizes the carry bit 'Cn' to include with its i/p 'An' as well as 2's complement of 'Bn' to produce the final output bit with final carry bit 'Cout'.

4 BIT PARALLEL SUBTRACTOR

- The number to be subtracted (B) is first passed through inverters to obtain its 1's complement.
- The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S_3 S_2 S_1 S_0 represents the result of binary subtraction (A-B) and carry output C_{out} represents the polarity of the result.
- If A > B then Cout = 1 and the result is positive.
- If A< B then C_{out} = 0 and the result is in the 2's complement form.

Block diagram





- If we want to subtract using adder we can perform the operation, **D=A+(-B)**
- If we use negative numbers, we can perform subtraction operation.
- In binary number, we can represent positive number to negative by doing it's 2's complement operation.
- In 2's complement first we make number complement of original number and we add binary 1 in it. So,

$$-B = \overline{B} + 1$$

Hence subtraction operation becomes.

$$A - B = A + \overline{B} + 1$$

- The minuend bits are given directly to the full adder.
- The subtrahend bits are inverted or complemented by using a NOT gate and given to the full adder.
- The carry input for the first full adder is set to '1'.
- The sum output of each full adder $S_3\,S_2\,S_1\,S_0$ corresponds to the difference output $D_3\,D_2\,D_1\,D_0$ of each bit in a multi-bit binary number.
- If there is any carry produced at the nth block, it will be seen at C_4.

EXAMPLE: **Minuend: 1101**

Subtrahend: 1010

- Convert the subtrahend into its 2's complement form by inverting all the bits and adding

1010 -> 0101 + 1 = 0110

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 0\ 1\ 1 \end{array}$$

- Ignore any carry from the previous addition
- The resulting binary number is the difference between the minuend and subtrahend in binary form:

0011 = 3 in decimal

UNIT -3

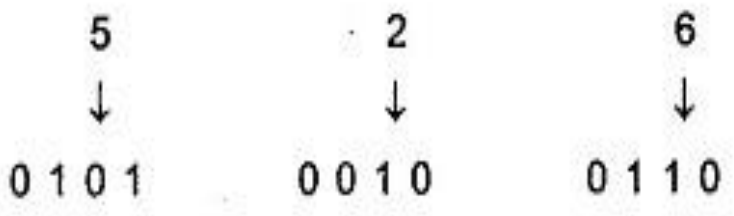
COMBINATIONAL LOGIC

TOPIC – 4

DECIMAL ADDER

DECIMAL ADDER

- The digital systems handle the decimal number in the form of binary coded decimal numbers (BCD).
- A BCD Adder Circuit that adds two BCD digits and produces a sum digit also in BCD.
- BCD numbers use 10 digits, 0 to 9 are represented in the binary form 0 0 0 0 to 1 0 0 1, i.e. each BCD digit is represented as a 4-bit binary number.
- When we write BCD number say 526, it can be represented as



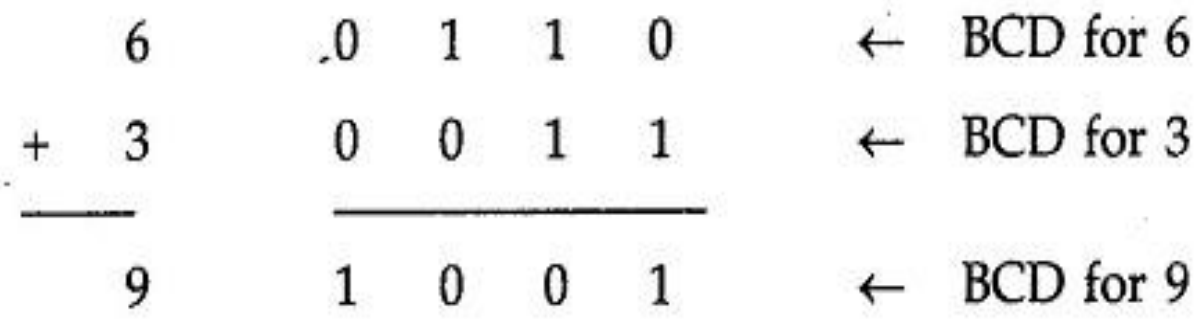
Here, we should note that BCD cannot be greater than 9.

OPERATION OF BCD ADDER

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Case 1: Sum Equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.



The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

Case2: Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

```
    6        0 1 1 0      ← BCD for 6
+   8        1 0 0 0      ← BCD for 8
------   -----------
   14        1 1 1 0      ← Invalid BCD number
```

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

```
    6        0 1 1 0      ← BCD for 6
+   8        1 0 0 0      ← BCD for 8
------   -----------
   14        1 1 1 0      ← Invalid BCD number
          +  0 1 1 0      ← Add 6 for correction
          -----------
0 0 0 1     0 1 0 0
                         ← BCD for 14
    1           4
```

After addition of 6 carry is produced into the second decimal position.

Case3: Sum equals 9 or less with carry 1

Let us consider addition of 8 and 9 in BCD

```
    8                    1 0 0 0   ← BCD for 8
+   9                    1 0 0 1   ← BCD for 9
------   ---------------------
   17    0 0 0 1        0 0 0 1   ← Incorrect BCD result
```

In this, case, result (0001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown below

```
    8                    1 0 0 0   ← BCD for 8
+   9                    1 0 0 1   ← BCD for 9
------   ---------------------
   17    0 0 0 1        0 0 0 1   ← Incorrect BCD result
      +  0 0 0 0        0 1 1 0   ← Add 6 for correction
         ---------------------
         0 0 0 1        0 1 1 1   ← BCD for 17
```

- Going through these three cases of BCD addition we can summarize the BCD addition procedure as follows:

1. Add two BCD numbers using ordinary binary addition.

2. If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.
3. If the four-bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

Thus, to implement BCD Adder Circuit we require:

- 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add 0110_2 in the sum if sum is greater than 9 or carry is 1.

TRUTH TABLE

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given BCD Adder 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

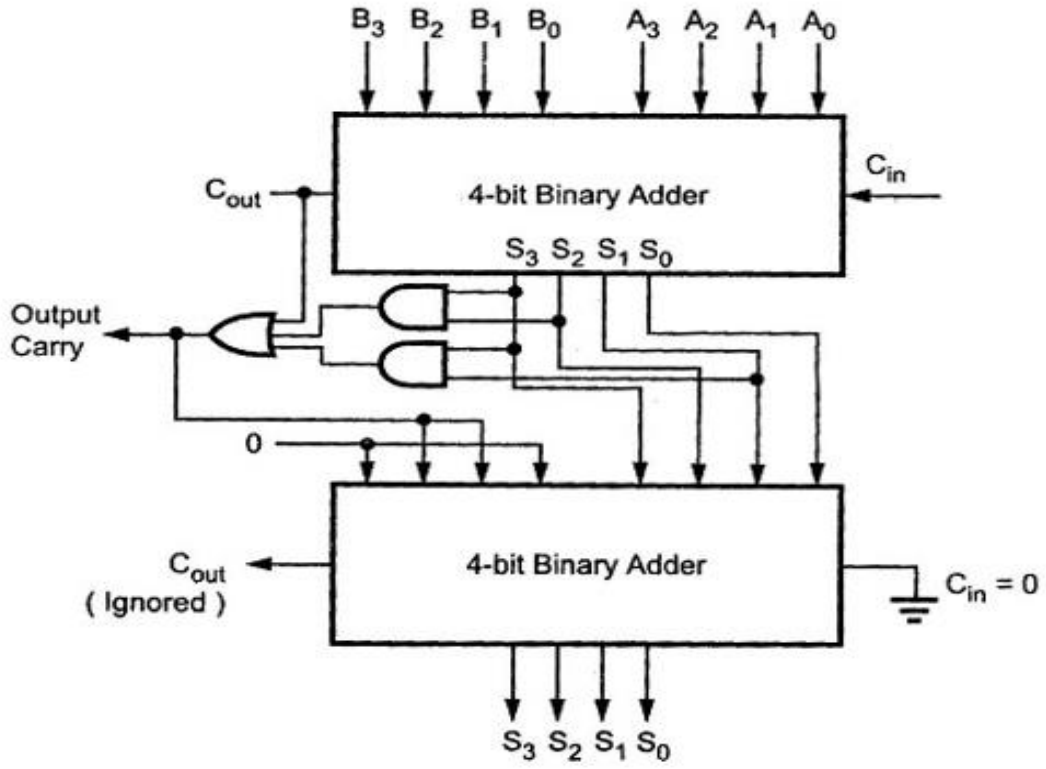
- **K - MAP**

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

$$Y = S_3S_2 + S_3S_1$$

- **BLOCK DIAGRAM**

With this design information we can draw the BCD Adder Block Diagram, as shown in the Fig below.



The two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum. When the output carry is equal to zero (i.e. when sum ≤ 9 and $C_{out} = 0$) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum > 9 or $C_{out} = 1$), binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

Example: 1001+1000

First, add both the numbers using a 4-bit binary adder and pass the input carry to 0.
The binary adder produced the result 0001 and carried output 'C_{out}' 1.
Then, find the 'Output Carry' value to identify that the produced BCD is invalid or valid using the expression Output Carry=$C_{out}+S_3S_2+S_3S_1$.
C_{out}= 1
S_3= 0
S_2= 0
S_1= 0
Output Carry = 1+0*0+0*0
Output Carry = 1+0+0
Output Carry = 1
The value of Output Carry is 1, which expresses that the produced BCD code is invalid.
Then, add the output of the 1st 4-bit binary adder with 0110.
= 0001+0110
= 0111
The BCD is represented by the carry output as:
BCD=C_{out} $S_3S_2S_1$ =1 0 1 1 1

UNIT -3

COMBINATIONAL LOGIC

TOPIC – 5

BINARY MULTIPLIER

DIGITAL BINARY MULTIPLIER

- A **Binary multiplier** is a combinational logic circuit used for **multiplying two binary numbers.**
- The two numbers are more specifically known as **multiplicand** and **multiplier** and the result is known as a **product.**
- The multiplicand & multiplier can be of various bit size
- The bit size of the product is equal to the sum of the bit size of multiplier & multiplicand.
- Binary multiplication of more than 1-bit numbers contains 2 steps.

Step1- Bit-wise multiplication known as partial product.

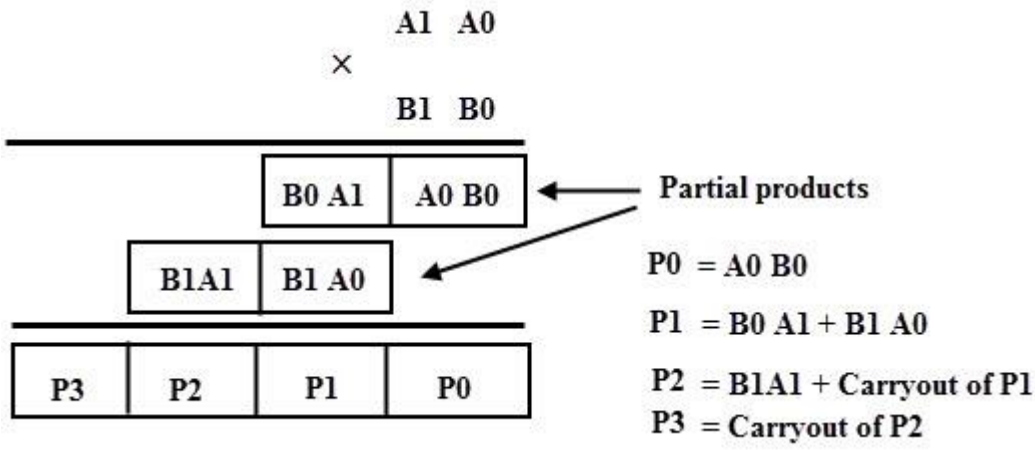
Step2-Adding all partial products to obtain product of given numbers.

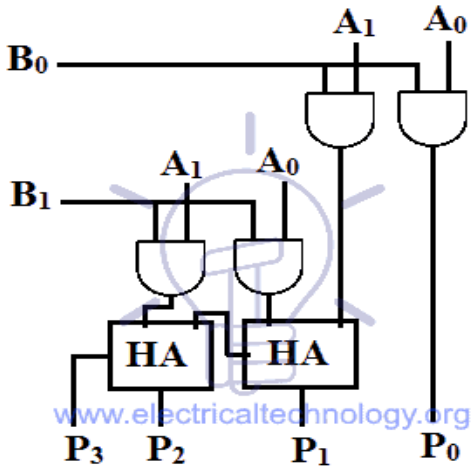
➢ 2×2 BIT MULTIPLIER:

OPERATION

This multiplier can multiply two numbers having bit size = 2 i.e. the multiplier and multiplicand can be of 2 bits.

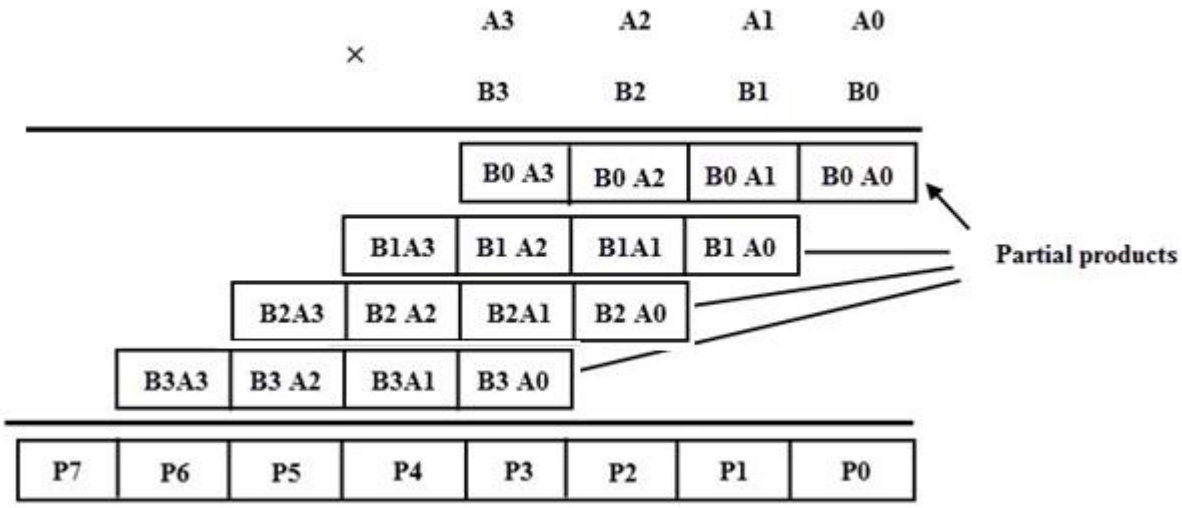
- The product bit size will be the sum of the bit size of the input i.e. **2+2=4**.
- The maximum range of its output is **3 x 3 = 9**. So we can accommodate decimal 9 in 4 bits.
- Multiplicand A_1 A_0 & multiplier B_1 B_0 & P_3 P_2 P_1 P_0 as a product of the 2×2 multiplier.
- The multiplicand is multiplied with each bit of the multiplier (from LSB to MSB) to obtain partial products.
- The number of partial products is equal to the number of bits in the multiplier





- This multiplication is implemented by a combinational circuit such that the multiplication is performed with AND gates whereas the addition is carried out by using half adders.
- The first partial product is obtained by the AND gate which is nothing but a least significant bit of the multiplication result.
- Since the second partial product is shifted to the left position, the first partial product second term and second partial product first term is added by half adder and produce the sum output along with the carry out.
- This carry out is added at the next half adder as an input. Likewise, it produces the multiplication result of two binary numbers by using the simple circuit configuration.
- The multiplication of the two 2-bit number results a 4-bit binary number.

➢ **4×4 BIT MULTIPLIER USING 4-BIT FULL ADDERS:**



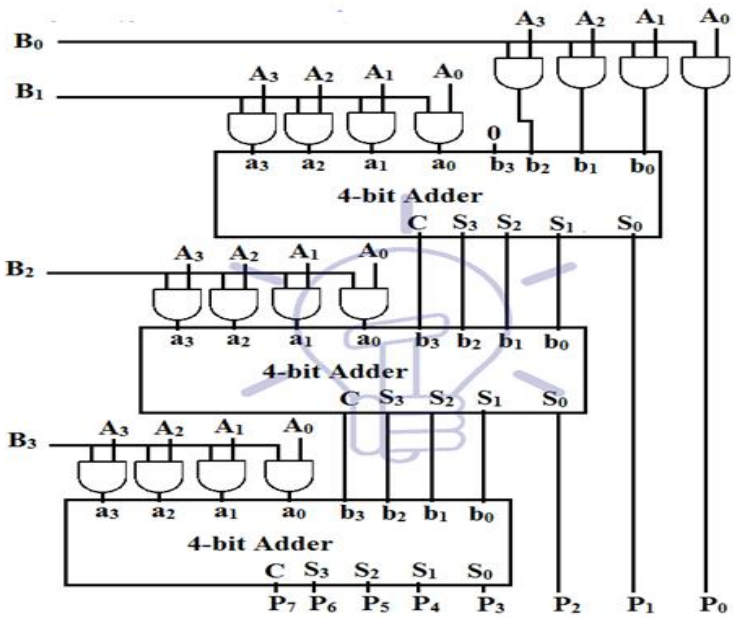
- This multiplier can multiply a binary number of 4-bit size & gives a product of 8-bit size because the bit size of the product is equal to the sum of bit size of multiplier and multiplicand.
- The maximum number it can calculate us **15 x 15 = 225**. You can also evaluate the number of bits from the maximum output range.

- Suppose multiplicand $A_3A_2A_1A_0$ & multiplier $B_3B_2B_1B_0$ & product as $P_7\ P_6\ P_5\ P_4\ P_3\ P_2\ P_1\ P_0$ for 4×4 multiplier.
- In 4×4 multiplier, there are 4 partial products, and we need to add these partial products to get the product of multiplier.
- They can be added using 4-bit full adders or single bit adders (half-adder & full-adder).
- The design using Single bit adders is very complicated compared to using 4-bit full adders (4-bit Parallel adder)**.**
- In the above operation the first partial product is obtained by multiplying B_0 with $A_3A_2\ A_1A_0$, the second partial product is formed by multiplying B_1 with $A_3A_2\ A_1A_0$, likewise for 3rd and 4th partial products. So these partial products can be implemented with AND gates.
- These partial products are then added by using 4 bit parallel adder. The three most significant bits of first partial product with carry (considered as zero) are added with second partial term in the first full adder.
- Then the result is added to the next partial product with carry out and it goes on till the final partial product, finally it produces 8 bit sum which indicates the multiplication value of the two binary numbers.

EXAMPLES:

```
  1 0 1 0   ------>  Multiplicand
× 1 0 1 1   ------>  Multiplier
-----------
  1 0 1 0   ------>  Partial product 1
1 0 1 0     ------>  Partial product 2
0 0 0 0     ------>  Partial product 3
1 0 1 0     ------>  Partial product 4
-----------
1 1 0 1 1 1 0
-----------
```

```
1101X1101
---------
    1101  ------>
    0000  ------>   PARTIAL
    1101  ------>   PRODUCTS
    1101  ------>
---------
10101001
```



UNIT -3

COMBINATIONAL LOGIC

TOPIC – 6

2-BIT MAGNITUDE COMPARATOR

MAGNITUDE COMPARATOR:

- A magnitude Comparator is a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is equal, less than, or greater than the other binary number.
- We logically design a circuit for which we will have two inputs one for A and the other for B and have three output terminals, one for A > B condition, one for A = B condition, and one for A < B condition.

- The circuit works by comparing the bits of the two numbers starting from the Most Significant Bit (MSB) and moving toward the Least Significant Bit (LSB).
- At each bit position, the two corresponding bits of the numbers are compared. If the bit in the first number is greater than the corresponding bit in the second number, the A>B output is set to 1, and the circuit immediately determines that the first number is greater than the second.
- Similarly, if the bit in the second number is greater than the corresponding bit in the first number, the A<B output is set to 1, and the circuit immediately determines that the first number is less than the second.
- If the two corresponding bits are equal, the circuit moves to the next bit position and compares the next pair of bits.
- This process continues until all the bits have been compared. At any point in the comparison, the circuit determines that the first number is greater or less than the second number, the comparison is terminated, and the appropriate output is generated.
- If all the bits are equal, the circuit generates an **A=B output**, indicating that the two numbers are equal.
- We can implement a magnitude comparator, such as using a combination of XOR, AND, and OR gates.

➢ 1-BIT MAGNITUDE COMPARATOR

- A comparator used to compare two bits is called a single-bit comparator.
- It consists of two inputs each for two single-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.
- The truth table for a 1-bit comparator is given below.
- From the truth table logical expressions for each output can be expressed as follows.

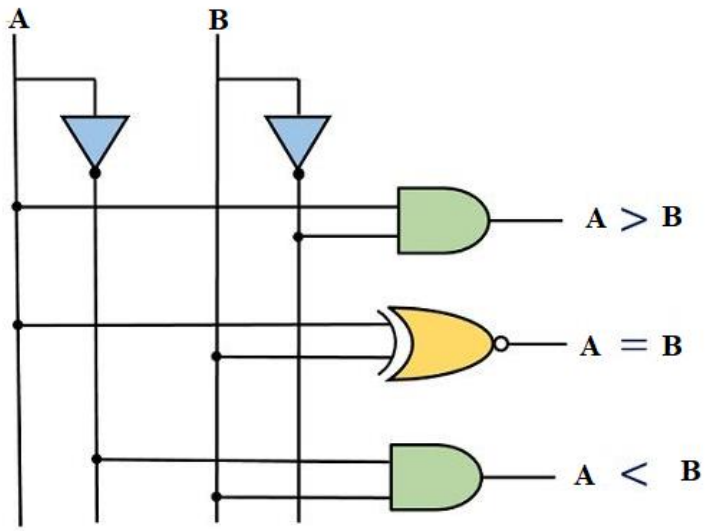
A>B: AB’

A<B: A’B

A=B: A'B' + AB

A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

LOGIC DIAGRAM OF 1-BIT MAGNITUDE COMPARATOR:



Logic Circuit of 1-bit Magnitude Comparator

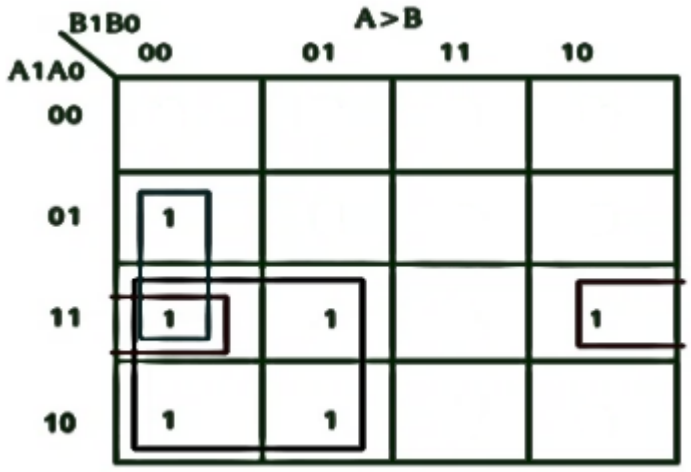
:

➢ 2-BIT MAGNITUDE COMPARATOR

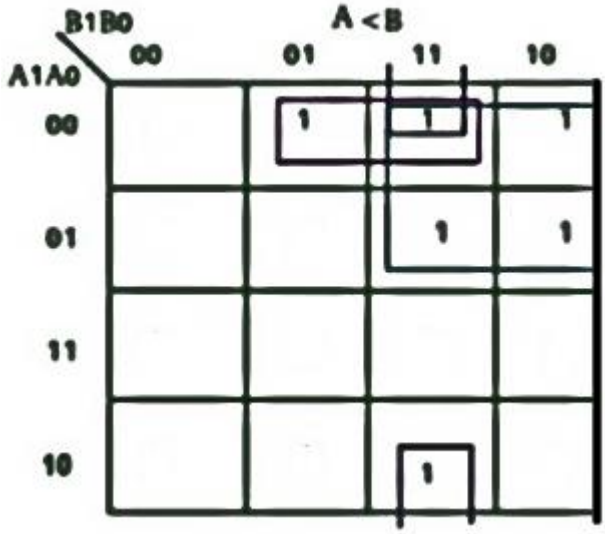
- A comparator used to compare two binary numbers of each of two bits is called a 2-bit Magnitude comparator.
- It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

Truth Table of Output A>B,A<B&A=B

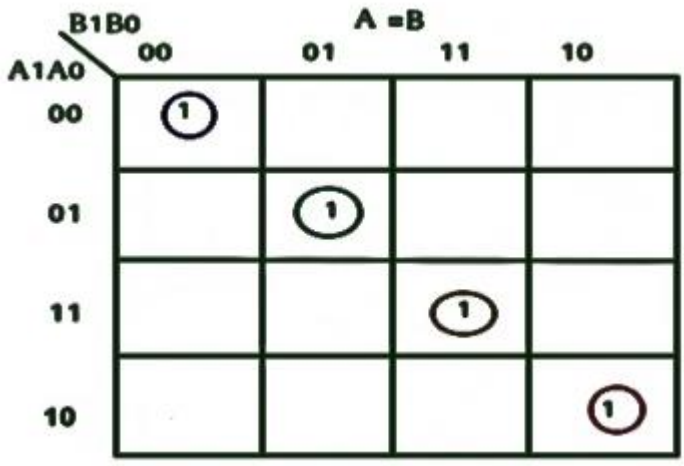
INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0



A>B: A1B1' + A0B1'B0' + A1A0B0'



A<B: A1'B1 + A0'B1B0 +A1'A0'B0



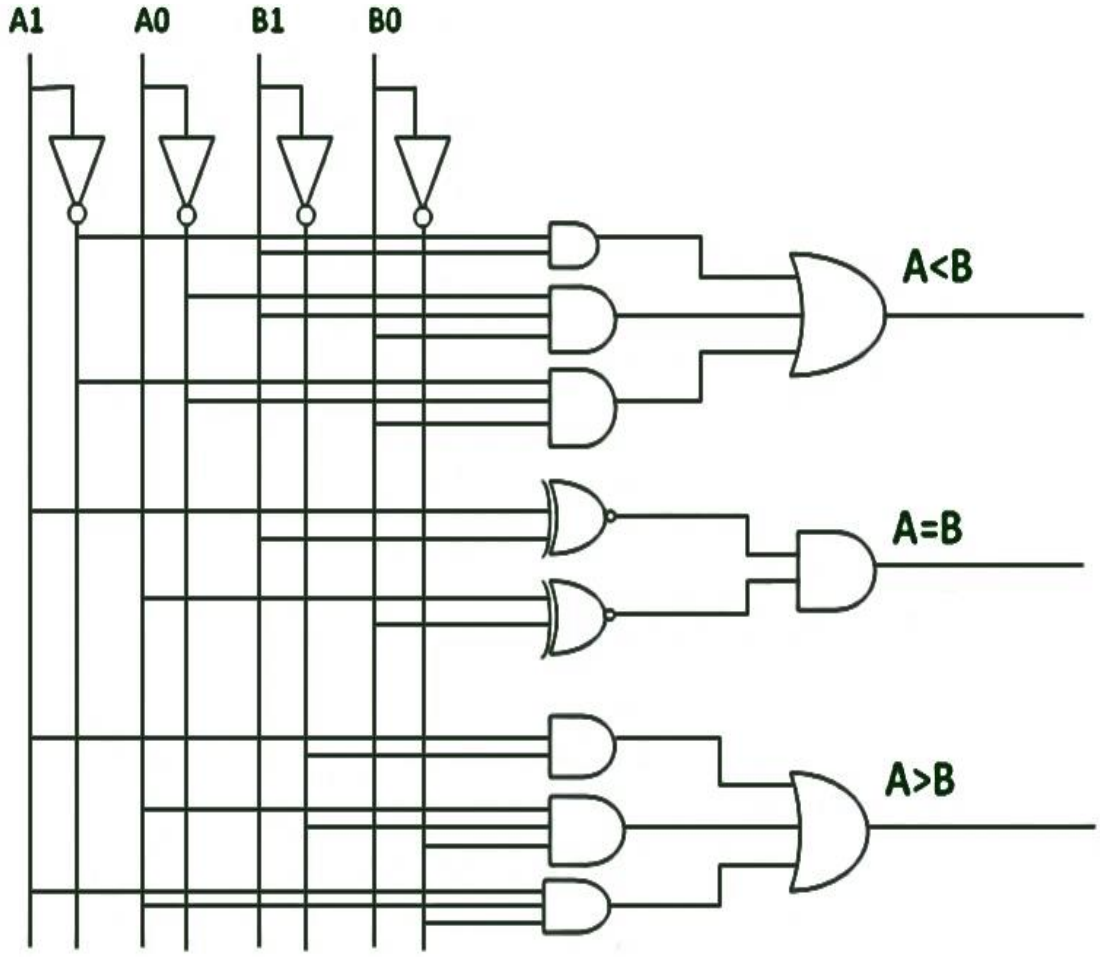
A=B: A1'A0'B1'B0'+A1'A0B1'B0+ A1A0B1B0 + A1A0'B1B0'

: A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0')

: (A0B0 + A0'B0') (A1B1 + A1'B1')

: (A0ΘB0) (A1ΘB1)

LOGIC CIRCUIT DIAGRAM:



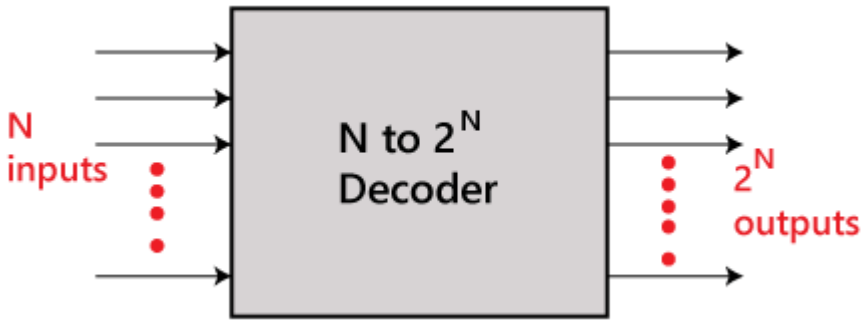
UNIT -3
COMBINATIONAL LOGIC
TOPIC – 7
2 : 4 Decoder & 3 : 8 Decoder

Decoder

The combinational circuit that change the binary information into 2^N output lines is known as **Decoders.**

The binary information is passed in the form of N input lines. The output lines define the 2^N-bit code for the binary information.

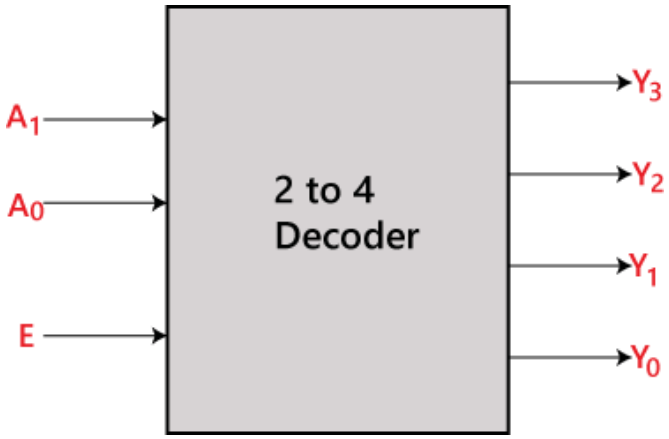
The produced 2^N-bit output code is equivalent to the binary information.



2 to 4 line decoder:

- In the 2 to 4 line decoder, there is a total of three inputs, i.e., A_0,and A_1 and E and four outputs, i.e., Y_0, Y_1, Y_2, and Y_3.
- For each combination of inputs, when the enable 'E' is set to 1,one of these four outputs will be 1.

The block diagram and the truth table of the 2 to 4 line decoderare given below.



Truth Table:

Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

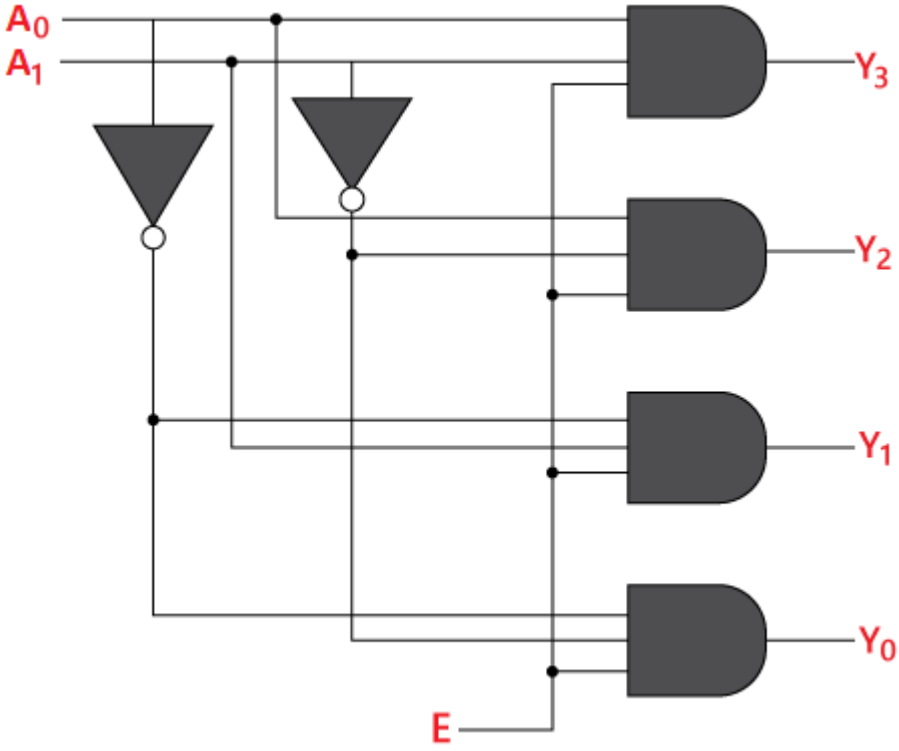
The logical expression of the term Y0, Y0, Y2, and Y3 is as follows:

$Y_3=E.A_1.A_0$

$Y_2=E.A_1.A_0'$

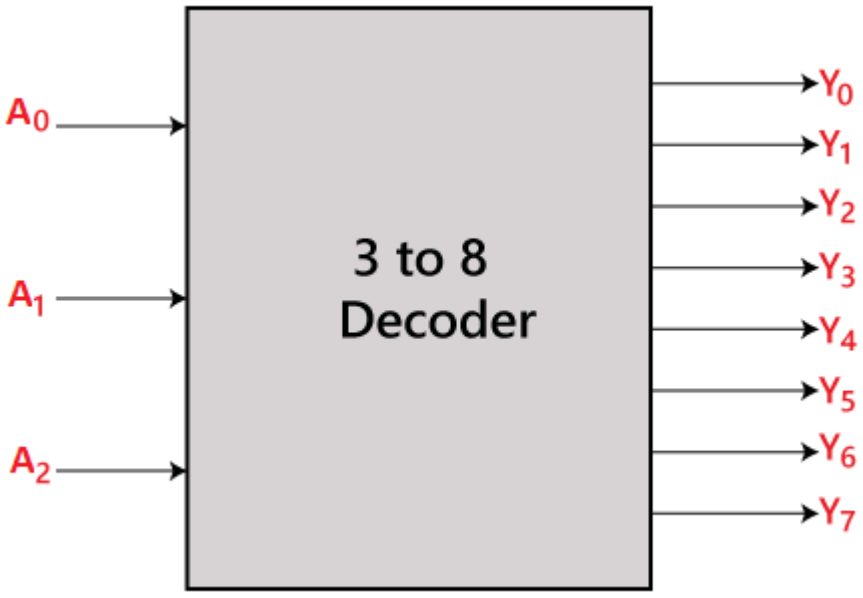
$Y_1=E.A_1'.A_0$

$Y0=E.A_1'.A_0'$



3 to 8 line decoder:

- The 3 to 8 line decoder is also known as Binary to Octal Decoder.
- In a 3 to 8 line decoder, there is a total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2.
- This circuit has an enable input 'E'. Just like 2 to 4 line decoder, when enable 'E' is set to 1, one of these eight outputs will be 1.
- The block diagram and the truth table of the 3 to 8 line encoder are given below.



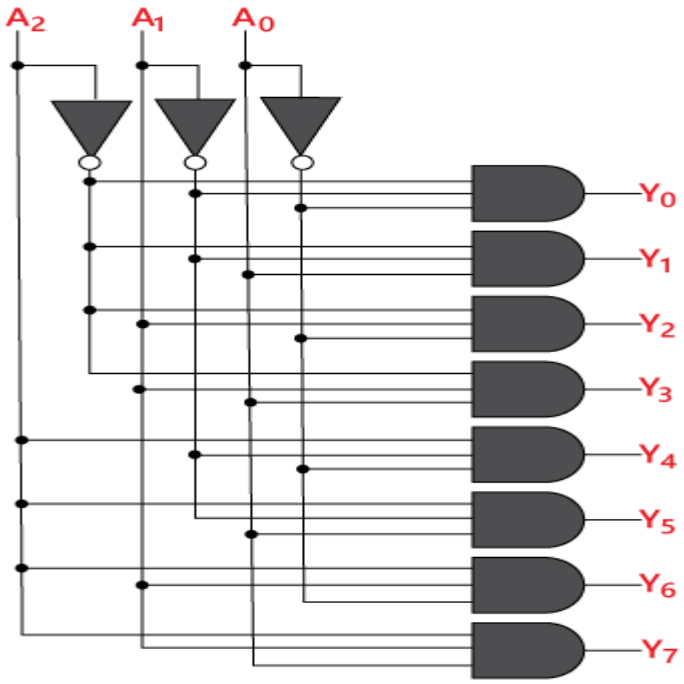
Truth Table:

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
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

The logical expression of the term Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 is as follows:

$Y_0=A_2'.\ A_1'.\ A_0'$
$Y_1=A_2'.\ A_1'.A_0$
$Y_2=\ A_2'.A_1.\ A_0'$
$Y_3=A_2'.A_1\ .\ A_0$
$Y_4=A_2\ .A_1'.\ A_0'$
$Y_5=A_2.\ A_1'.\ A_0$
$Y_6=\ A_2.A_1.A_0'$
$Y_7=A_2.A_1\ .A_0$

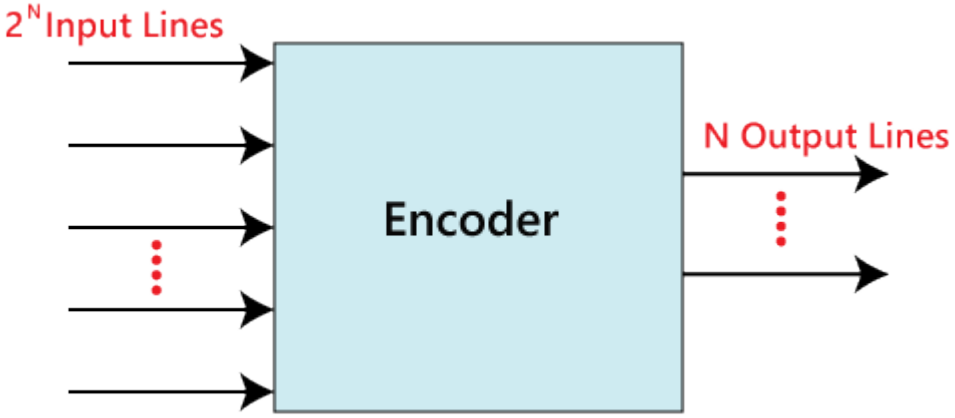
Logical circuit of the above expressions is given below:



UNIT -3
COMBINATIONAL LOGIC
TOPIC – 8
4 to 2 Encoder & 8 to 3 Encoder

Encoders:

• The combinational circuits that change the binary information into N output lines are known as Encoders.
• The binary information is passed in the form of 2^N input lines. The output lines define the N-bit code for the binary information.
• In simple words, the Encoder performs the reverse operation of the Decoder. At a time, only one input line is activated for simplicity.
• The produced N-bit output code is equivalent to the binary information.



4 to 2 line Encoder:

The 4 to 2 Encoder consists of **four inputs Y3, Y2, Y1 & Y0, and two outputs A1 & A0**. At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The figure below shows the logic symbol of the 4 to 2 line Encoder.

The Truth table of 4 to 2 encoders is as follows:

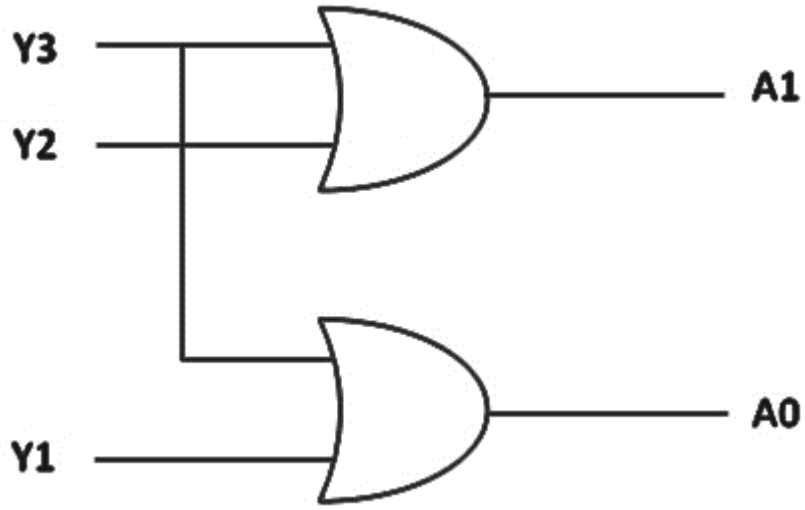
INPUTS				OUTPUTS	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Logical expression for A1 and A0:

A1 = Y3 + Y2

A0 = Y3 + Y1

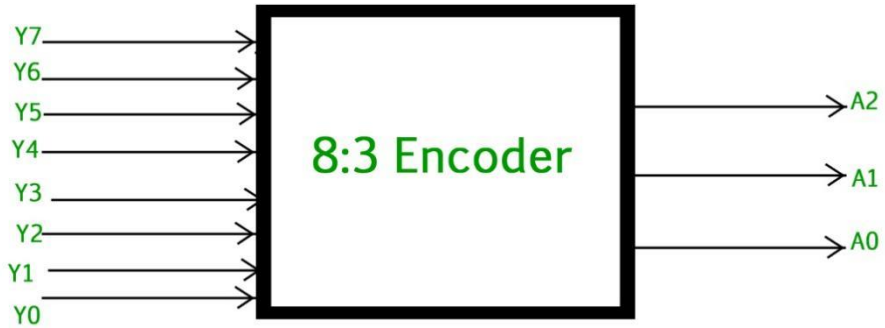
The above two Boolean functions A1 and A0 can be implemented using two input OR gates :



Implementation using OR Gate

Octal to Binary Encoder (8 to 3 Encoder):

The 8 to 3 Encoder or octal to Binary encoder consists of **8 inputs**: Y7 to Y0 and **3 outputs**: A2, A1 & A0. Each input line corresponds to each octal digit and three outputs generate corresponding binary code. The figure below shows the logic symbol of octal to the binary encoder.



The truth table for the 8 to 3 encoder is as follows:

INPUTS								OUTPUTS		
Y7	**Y6**	**Y5**	**Y4**	**Y3**	**Y2**	**Y1**	**Y0**	**A2**	**A1**	**A0**
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

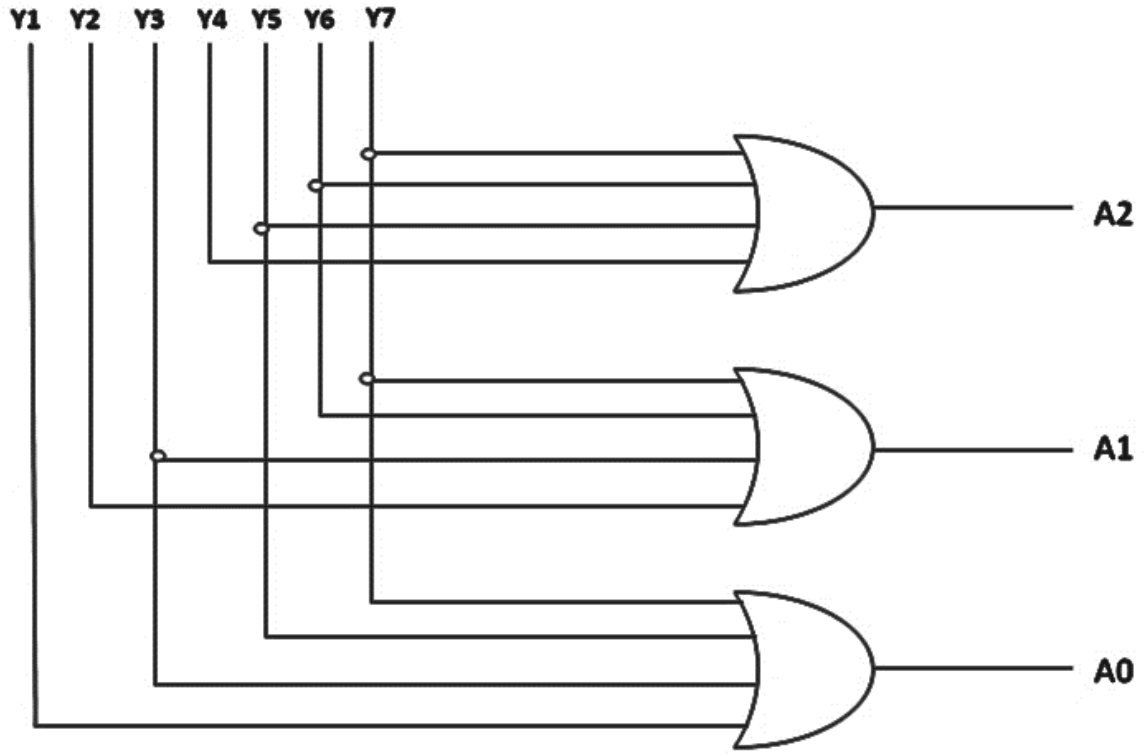
Logical expression for A2, A1, and A0:

A2 = Y7 + Y6 + Y5 + Y4

A1 = Y7 + Y6 + Y3 + Y2

A0 = Y7 + Y5 + Y3 + Y1

The above two Boolean functions A2, A1, and A0 can be implemented using four input OR gates.



Application of Encoders

- Encoders are very common electronic circuits used in all digital systems.
- Encoders are used to translate the decimal values to the binary in order to perform binary functions such as addition, subtraction, multiplication, etc.
- Other applications, especially for Priority Encoders may include detecting interrupts in microprocessor applications.

UNIT -3
COMBINATIONAL LOGIC
TOPIC – 9

4 to 1 line Multiplexer & 8 to 1 line Multiplexer

Multiplexers:

• Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.
• Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as Mux.
• Multiplexers are also known as **data selectors, parallel to serial converter, many to one circuit**.

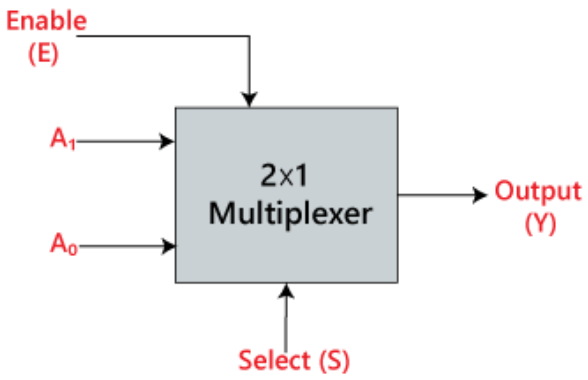
Types of Multiplexers

- 2-1 multiplexer (1select line)
- 4-1 multiplexer (2 select lines)
- 8-1 multiplexer (3 select lines)
- 16-1 multiplexer (4 select lines)

2X1 Multiplexer:

• A 2×1 multiplexer is a combinational logic circuit that has only two inputs, i.e, A0 and A1, 1 selection line, i.e., S0, and single outputs, i.e., Y.
• On the basis of the combination of inputs that are present at the selection line S0, one of these 2 inputs forward to the output line.
• It is the simplest type of multiplexer in digital electronics.
• The 2X1 multiplexer's block diagram and truth table are shown below.

Block Diagram:

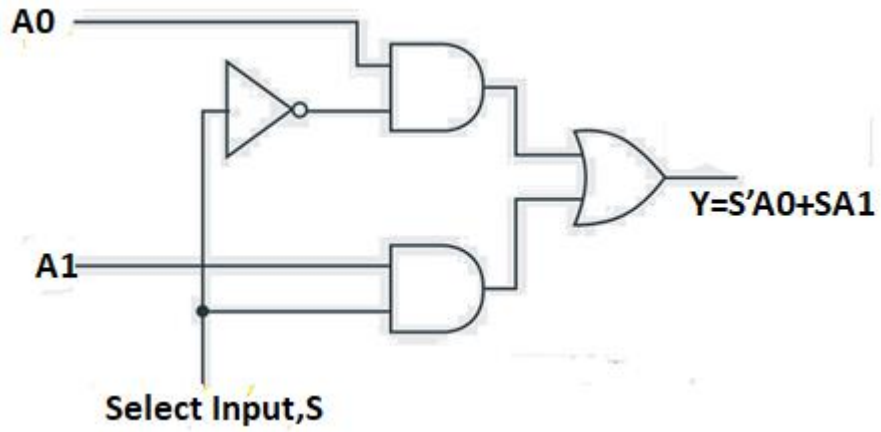


Truth Table:

INPUTS	Outputs
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

Y=S'A0+SA1

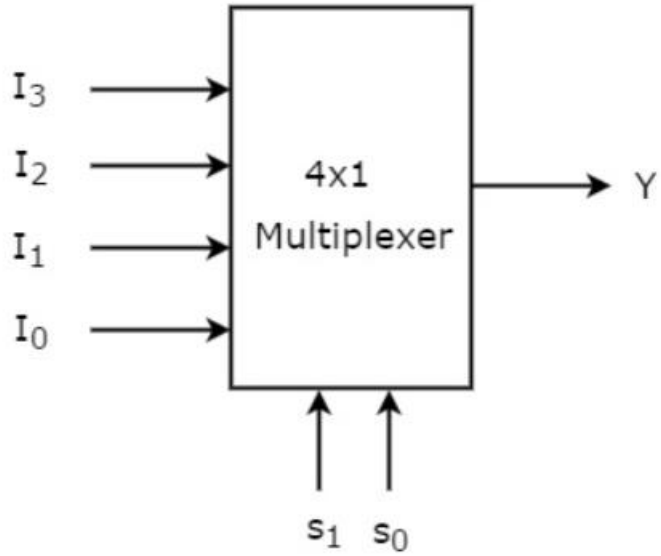


- From the above output expression, the logic circuit of 2-to-1 multiplexer can be implemented using logic gates as shown in figure.
- It consists of two AND gates, one NOT gate and one OR gate.
- When the select line, S=0, the output of the lower AND gate is zero, but the output of upper AND gate is D0. Thus, the output generated by the OR gate is equal to A0.
- Similarly, when S=1, the output of the upper AND gate is zero, but the output of lower AND gate is A1.
- Therefore, the output of the OR gate is A1.
- Thus, the above given Boolean expression is satisfied by this circuit.

4x1 Multiplexer:

• 4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y.
• The **block diagram** of 4x1 Multiplexer is shown in the following figure.

Block Diagram:



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

Truth Table:

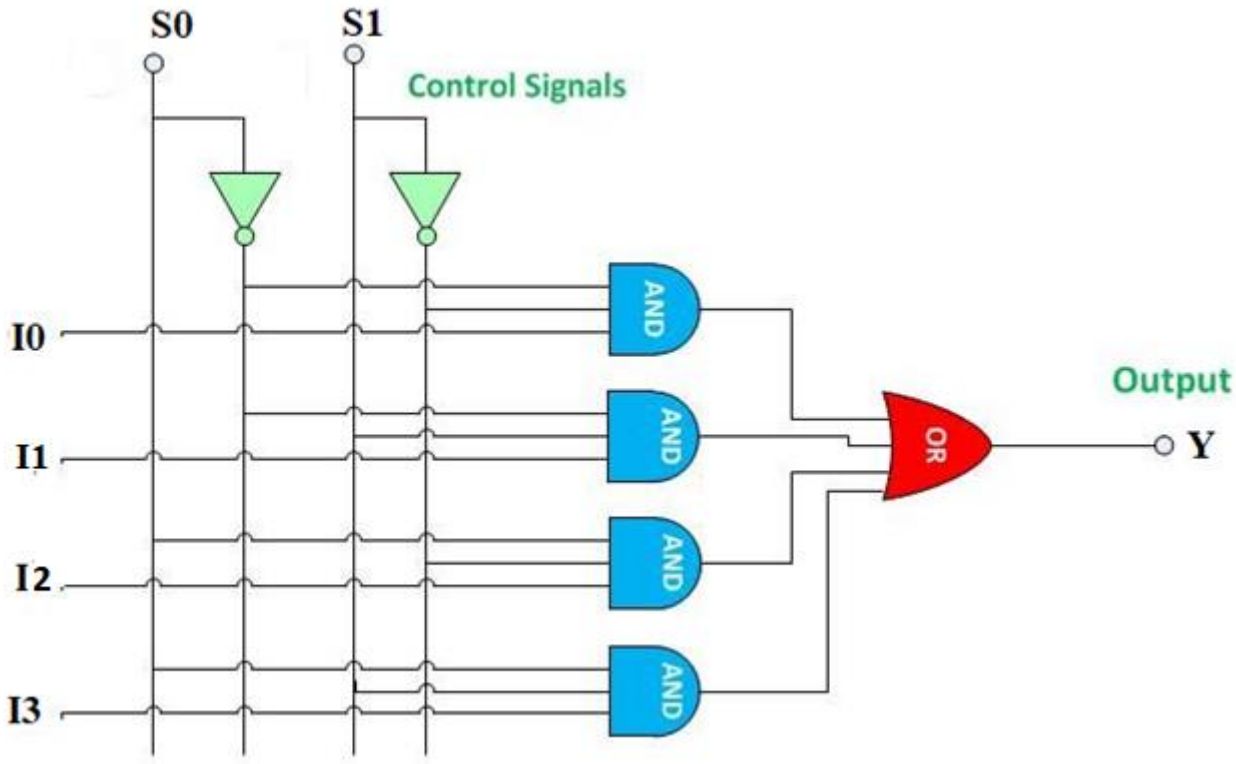
Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

The logical expression of the term Y is as follows:

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

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

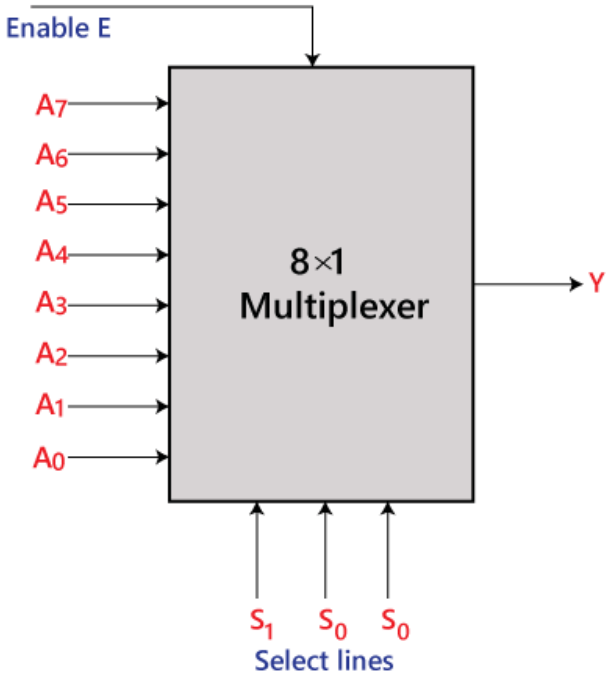
• We can implement this Boolean function using Inverters, AND gates & OR gate. The circuit diagram of 4x1 multiplexer is shown in the following figure.



8x1 Multiplexer:

- In the 8 to 1 multiplexer, there are total eight inputs, i.e., A0, A1, A2, A3, A4, A5, A6, and A7, 3 selection lines, i.e., S0, S1and S2 and single output, i.e., Y.
- On the basis of the combination of inputs that are present at the selection lines S0, S1, and S2, one of these 8 inputs are connected to the output.
- The block diagram and the truth table of the 8×1 multiplexer are given below.

Block diagram of 8X1 Multiplexer:



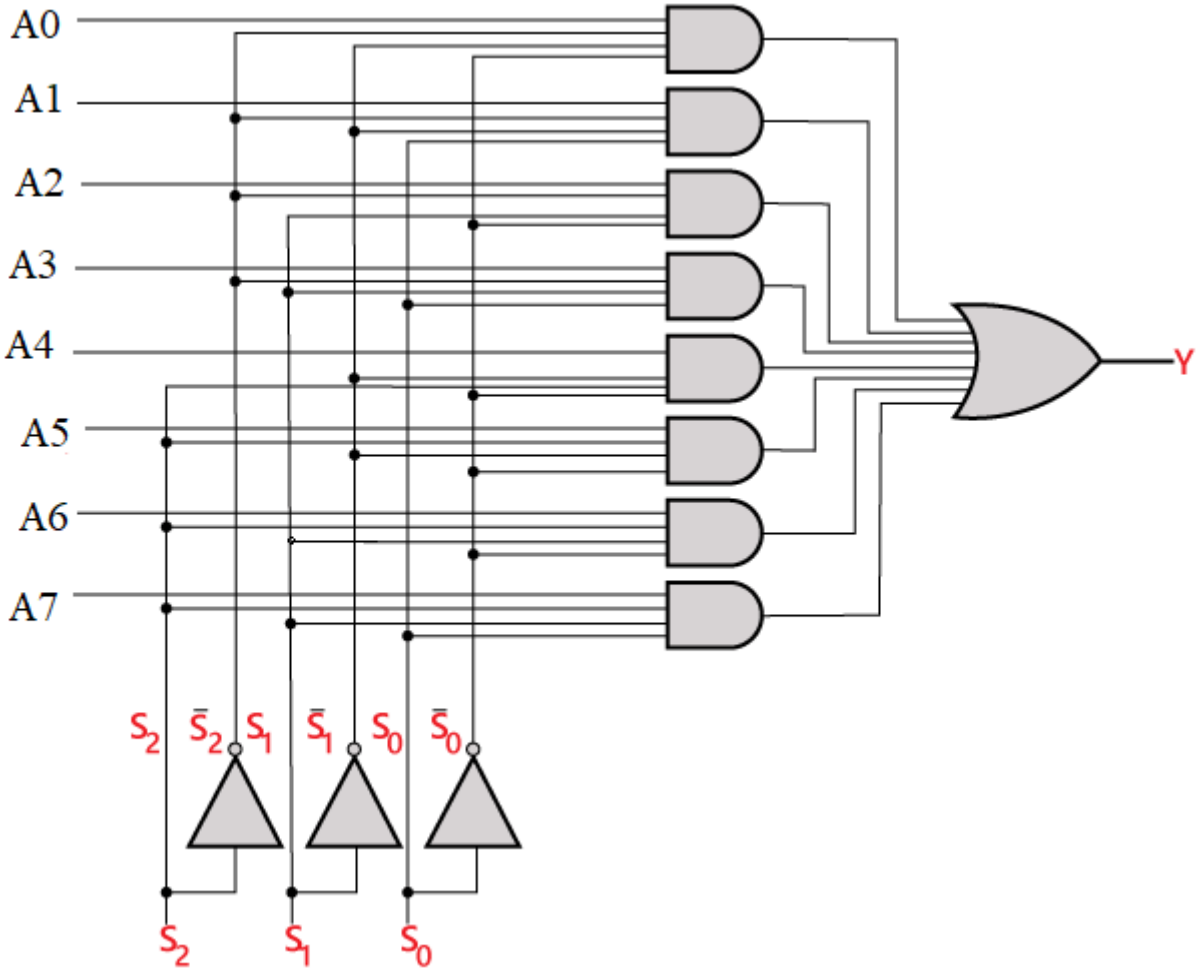
Truth Table:

INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

The logical expression of the term Y is as follows:

Y=S2'.S1'.S0'.A0+ S2'.S1'.S0.A1+ S2'.S1.S0'.A2+ S2'.S1.S0.A3+ S2.S1'.S0'.A4+ S2.S1'.S0.A5+ S2.S1.S0'.A6+ S2.S1.S0.A7

Logical circuit of the above expression is given below:

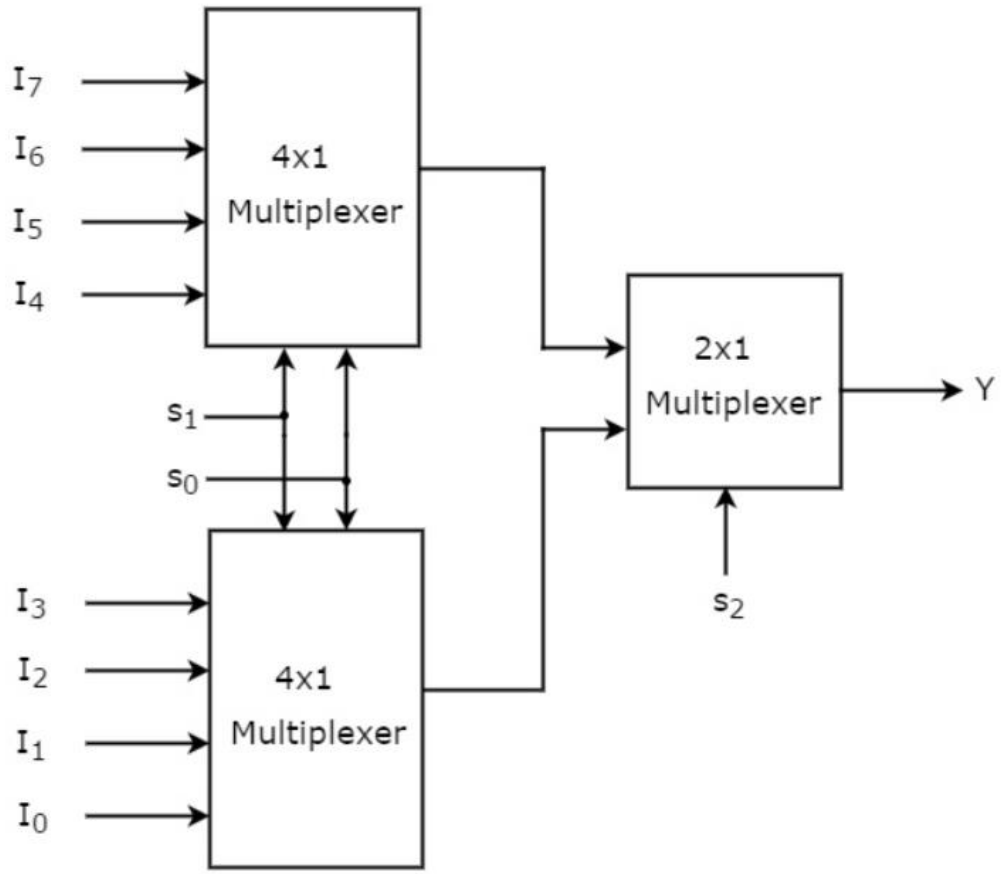


Implementation of Higher-order Multiplexers:

• Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.
• Let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer.
• We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.
• So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.
• Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

• We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



• The same **selection lines, S_1 & S_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, S_1 S_0.
• The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, S_2** is applied to 2x1 Multiplexer.
• If S2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I3 to I0 based on the values of selection lines S1 & S0.
• If S2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I7 to I4 based on the values of selection lines S1 & S0.

UNIT -3
COMBINATIONAL LOGIC
TOPIC – 10
1 to 4 line Demultiplexer &
1 to 8 line Demultiplexer

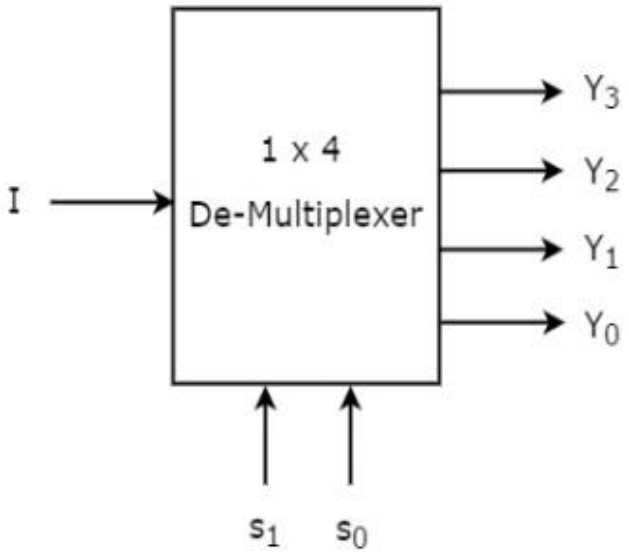
Demultiplexer:

- Demultiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.
- Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. Demultiplexer is also called as De-Mux.

1x4 Demultiplexer

- 1x4 Demultiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The block diagram of 1x4 Demultiplexer is shown in the following figure.

Block diagram of 1 x 4 Demultiplexer:



- The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The Truth table of 1x4 Demultiplexer is shown below.

Truth Table:

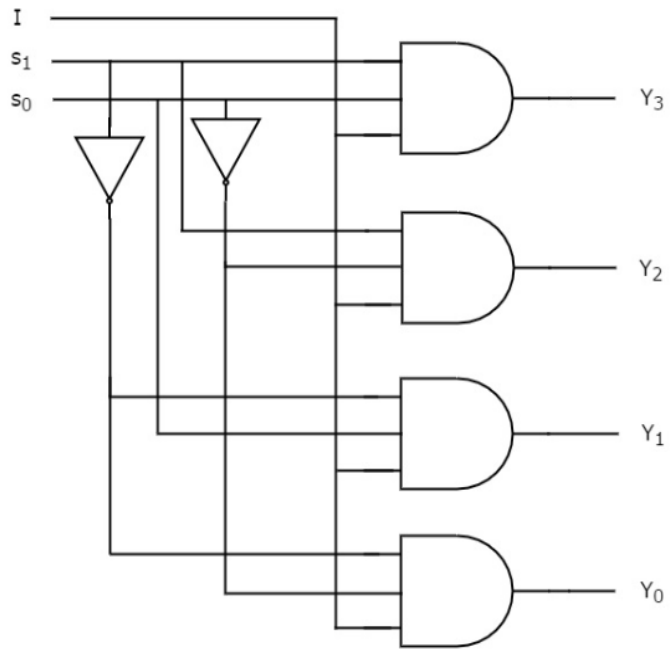
INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

$Y_0=S_1'S_0'A$
$Y_1=S_1'S_0\ A$
$Y_2=S_1\ S_0'A$
$Y_3=S_1\ S_0\ A$

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

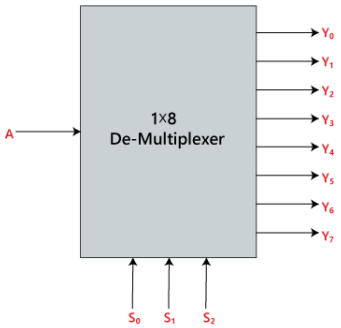
Logical circuit of the above expression is given below:



1x8 Demultiplexer:

- In 1 to 8 Demultiplexer, there are total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7, 3 selection lines, i.e., S_0, $S_1$and S_2 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S0,S1 and S2, the input will be connected to one of these outputs. The block diagram and the truth table of the 1×8 Demultiplexer are given below.

Block diagram



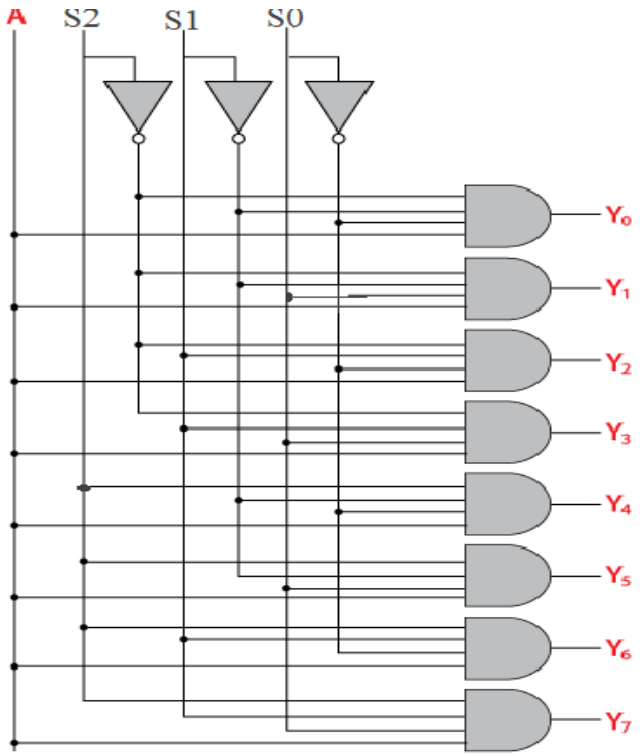
Truth Table:

INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

TThe logical expression of the term Y is as follows:

$Y_0=S_2'.S_0'.S_1'.A$
$Y_1=S_2'.S_1'.S_0.A$
$Y_2=S_2'.S_1.S_0'.A$
$Y_3=S_2'.S_1.S_0.A$
$Y_4=S_2.S_1'.S_0'.A$
$Y_5=S_2.S_1'.S_0.A$
$Y_6=S_2..S_1.S_0'.A$
$Y_7=S_2.S_1.S_0.A$

Logical circuit of the above expressions is given below:



Implementation of Higher-order Demultiplexers:

- Now, let us implement the following two higher-order Demultiplexers using lower-order Demultiplexers.

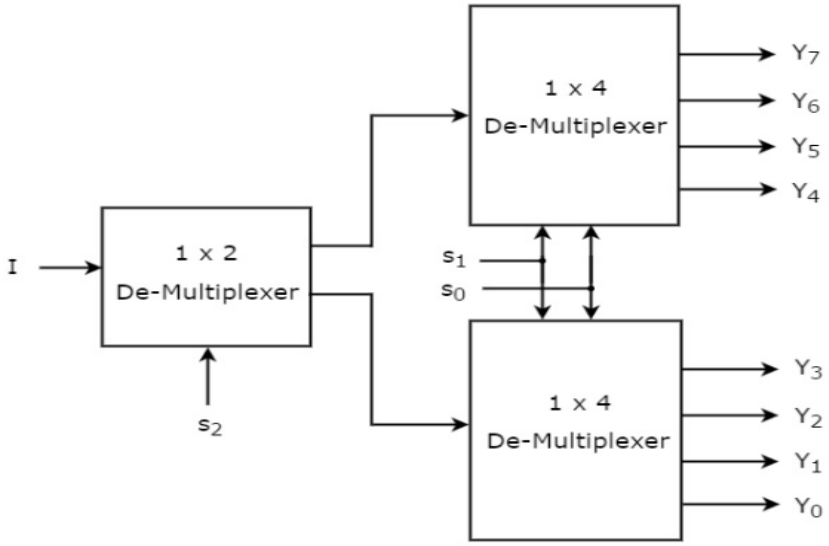
1x8 Demultiplexer:

- Let us implement 1x8 Demultiplexer using 1x4 Demultiplexers and 1x2 Demultiplexer. We know that 1x4 Demultiplexer has single input, two selection lines and four outputs. Whereas, 1x8 Demultiplexer has single input, three selection lines and eight outputs.
- So, we require two **1x4 Demultiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 Demultiplexer will be the overall input of 1x8 Demultiplexer.
- Let the 1x8 Demultiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0.

The **Truth table** of 1x8 Demultiplexer is shown below.

INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

- We can implement 1x8 Demultiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 Demultiplexer is shown in the following figure.



• The common **selection lines, s_1 & s_0** are applied to both 1x4 Demultiplexers. The outputs of upper 1x4 Demultiplexer are Y_7 to Y_4 and the outputs of lower 1x4 Demultiplexer are Y_3 to Y_0.
• The other **selection line, s_2** is applied to 1x2 Demultiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 Demultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 Demultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

UNIT -3
COMBINATIONAL LOGIC
TOPIC – 11
Implementation of Boolean Functions Using Multiplexers and Decoders

Implementation of Boolean Functions Using Multiplexers and Decoders:

1. Implement the given Boolean function F(A,B,C,D)= (1,2,4,7,10,11,13,15) using Decoder?

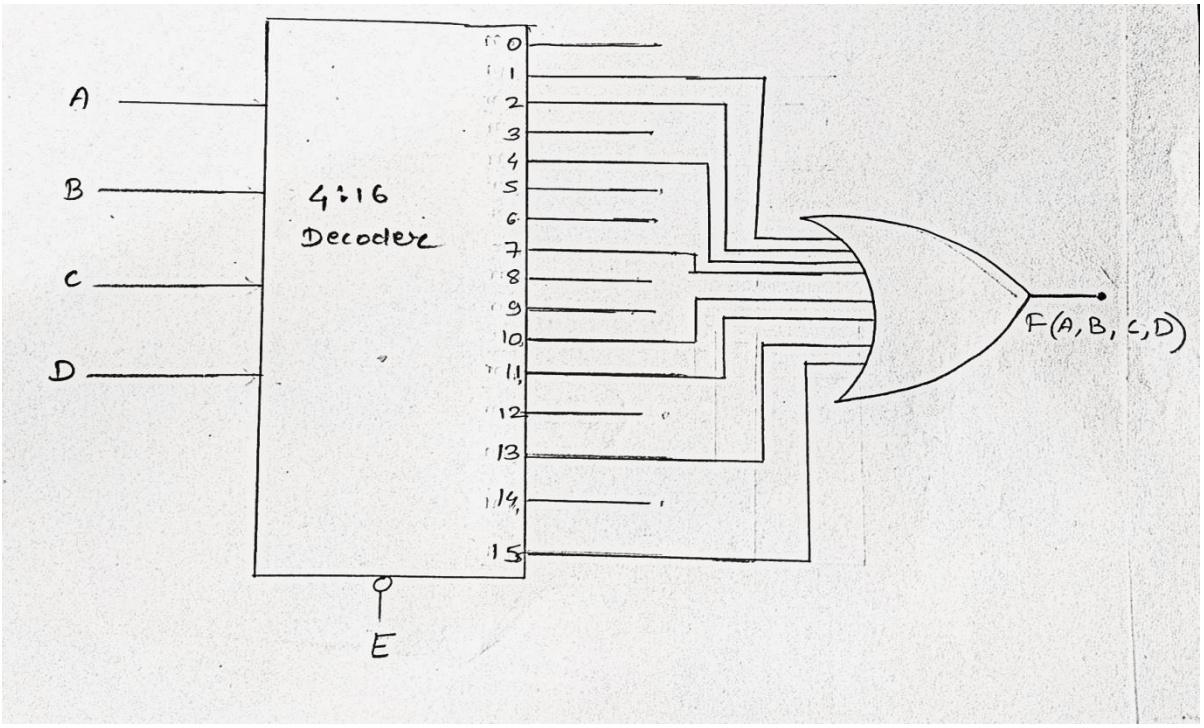
Here the function have 4inputs- A,B,C,D .We will use a 4:16 decoder and an OR gate is used to implement this function

1)First count the number of inputs given in Function. In the above example F(A,B,C,D),there are 4 inputs.

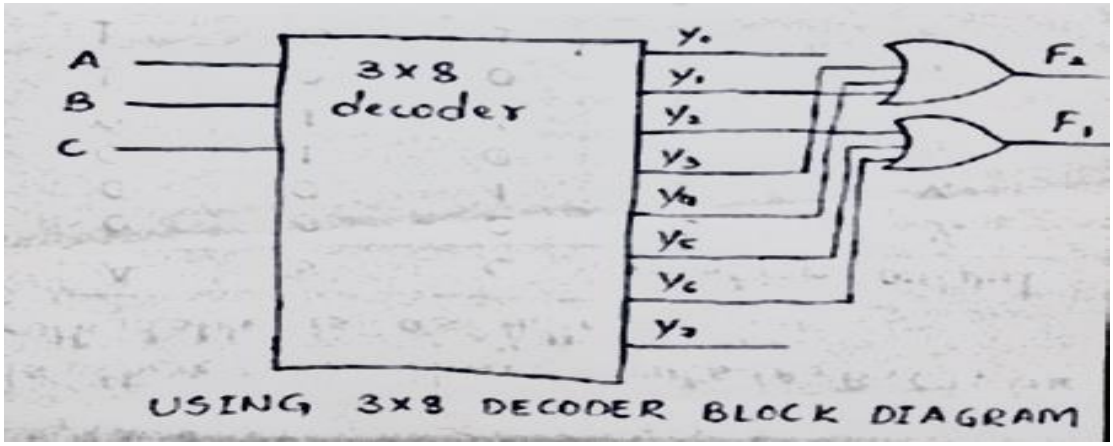
2)Select type of decoder to implement it, i.e., n:2n decoder . Since, we use 4:16 decoder

3)Now, among the decoder outputs, select only the output number included in Function. Then through these outputs as inputs to an OR gate to produce final output as function. Here, output involves minterms of 1,2,4,7,10,11,13,15.Select them and connect with OR gate to implement the given function.

4)Thus, the given Boolean function is implemented with decoder and OR gate.

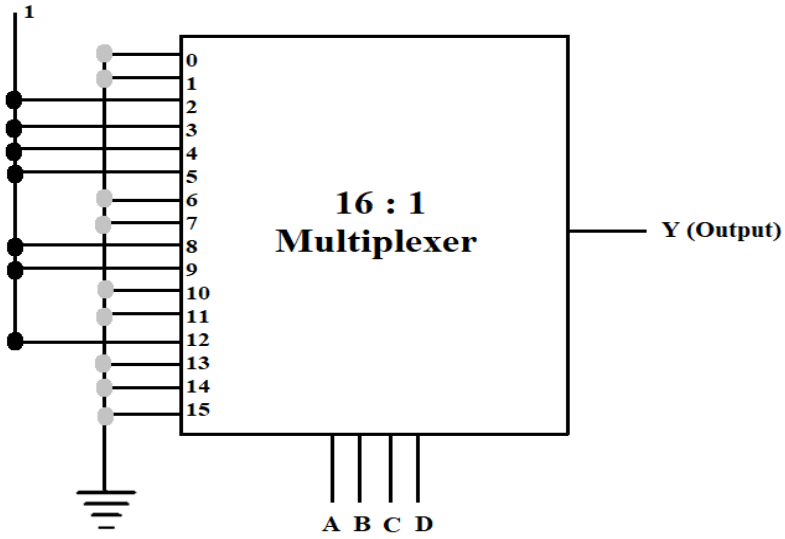


2. A combinational circuit is specified by the following Boolean function: f1(A,B,C)=Σm(2,5,6),f2(A,B,C)=Σm(1,3,4).Implement the circuit with a 3x8 decoder and OR gates. Use block diagram for the decoder?



3. Implement the function F(ABCD) = ∑m(2,3,4,5,8,9,12) using a) 16:1 mux b) Using one 8: 1Mux

- In the given Boolean expression, there are 4 variables.Hence, use $2^n = 2^4 : 1 = 16:1$ multiplexer.
- So, the mux has 16 input lines, 4 selection lines, and 1 output.
- The inputs, corresponding to the min-terms (2,3,4,5,8,9,12) are connected to logic 1 and the remaining terms to logic 0 (Grounded).
- The given input variables are connected as 4 selection lines.
- The 16:1 multiplexer can be drawn as follows:



b) Implementation with 8:1 MUX

- In the given boolean expression, there are 4 variables.
- Hence, we should use 24 : 1 = 16:1 multiplexer.
- But as per the question, it is to be implemented with 8:1 mux.
- For the 8:1 multiplexer, there should be 3 selection lines.
- So from the given 4 variables, the 3 least significant variables(B, C, D) are used as selection line inputs.
- Let us derive the eight inputs of the 8:1 multiplexer using the Implementation Table.
- The eight inputs are listed column-wise and all the minterms are written under the eight inputs in rows as shown below the implementation table:

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

Implementation Table