Lecture of Module 3

Combinational Circuits

Overview

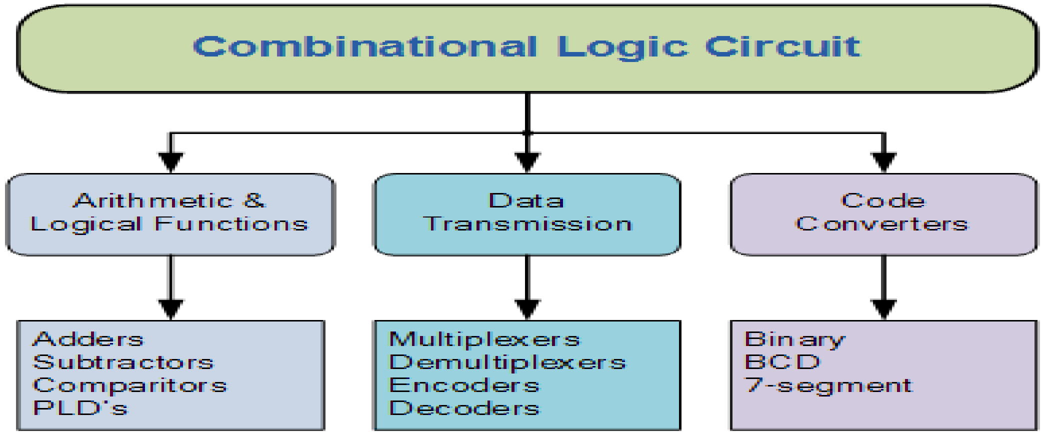
- **Introduction**
- **Half Adder**
- **Full Adder**
- **Half Subtractor**
- **Full Subtractor**
- **Ripple/Parallel Adder**
- **Adder-Subtractor**
- **Look-ahead carry Adder**

Introduction

The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state(s), logic “0” or(and) logic “1”, at any given instant.

Combinational logic circuits give us many useful devices.

One of the simplest is the *half adder*, which finds the sum of two bits.



Combinational Logic Circuit

Arithmetic & Logical Functions

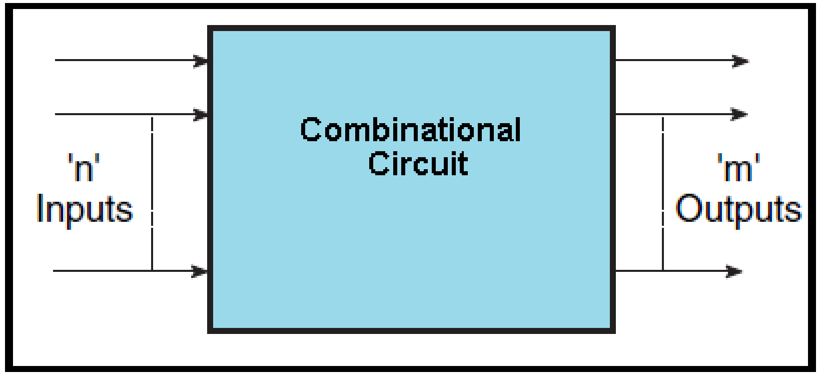
Data Transmission

Code Converters

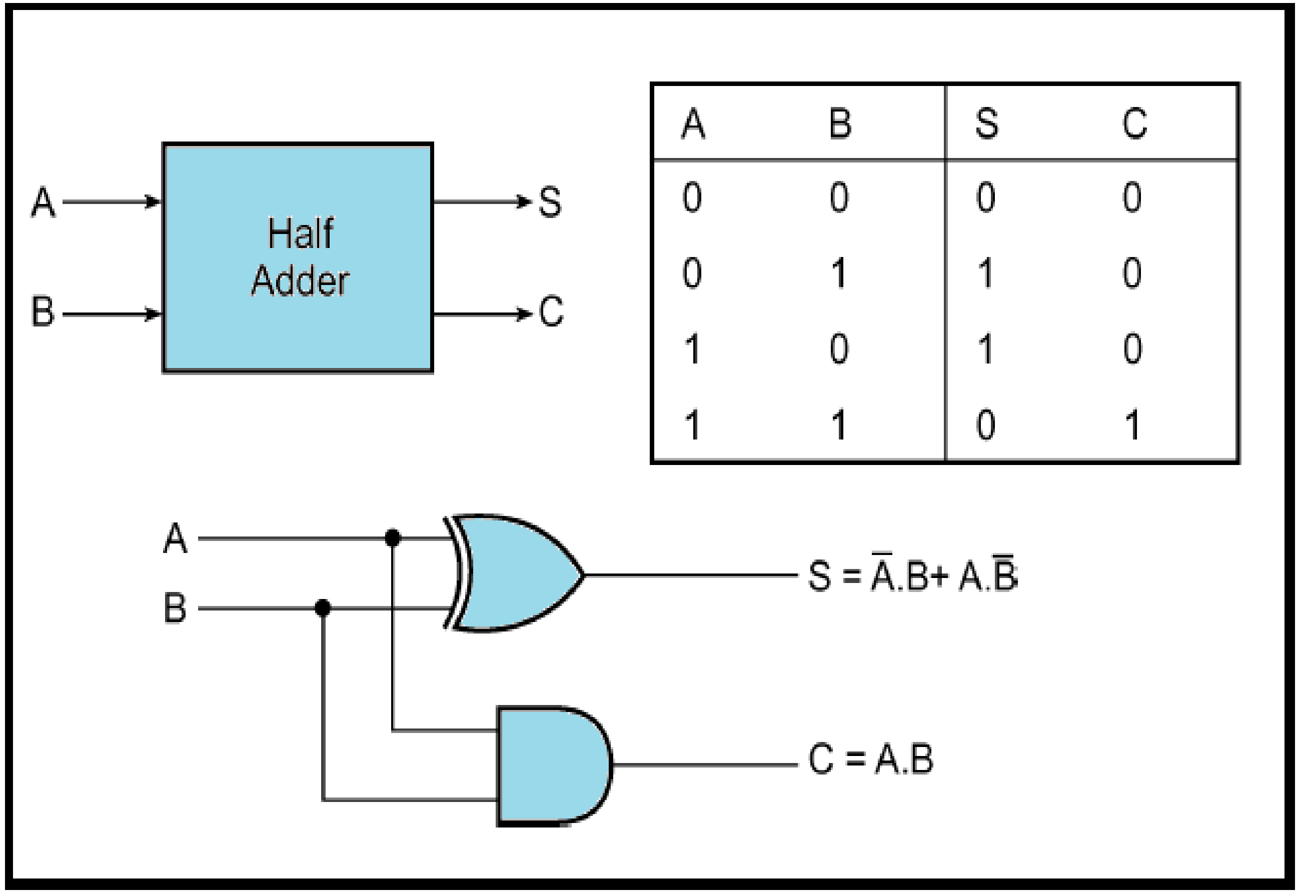
Adders
Subtractors
Comparitors
PLD's

Multiplexers
Demultiplexers
Encoders
Decoders

Binary
BCD
7-segment



Half Adder



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = \overline{A}.B + A.\overline{B}$

$C = A.B$

For Carry

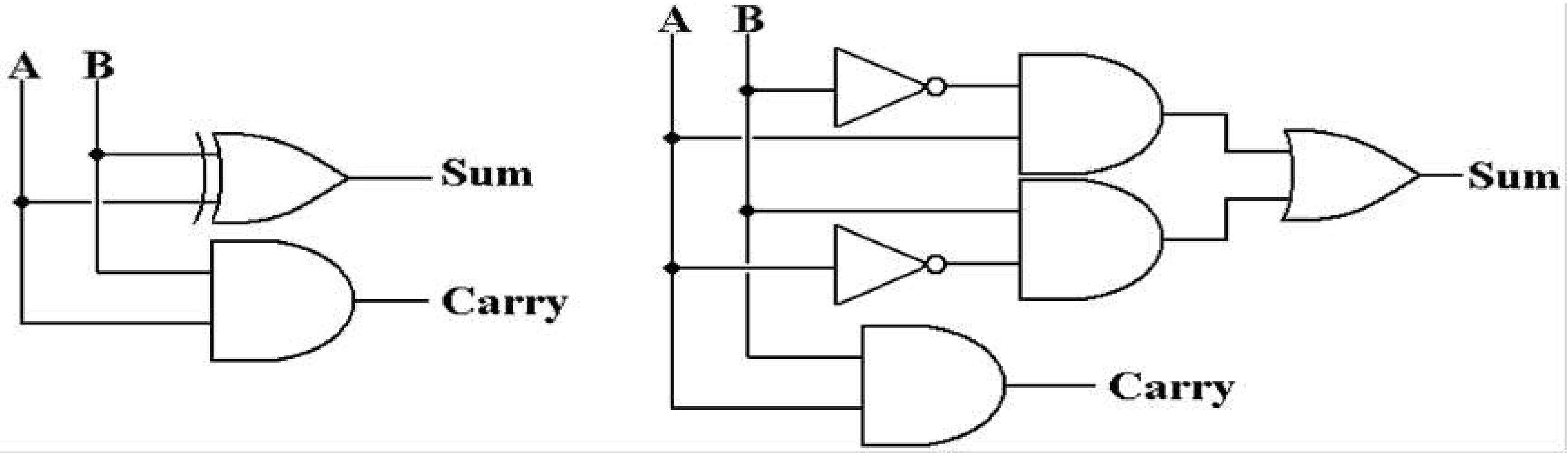
A \ B	0	1
0	0	0
1	0	1

Carry = AB

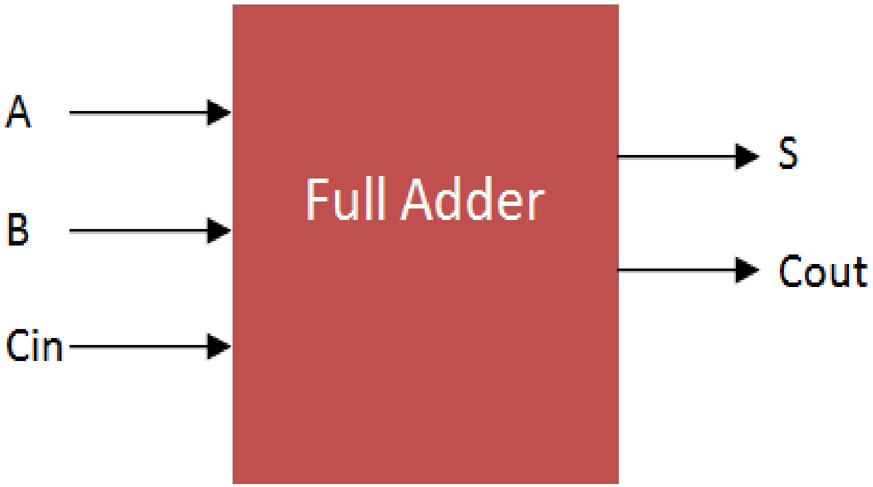
For Sum

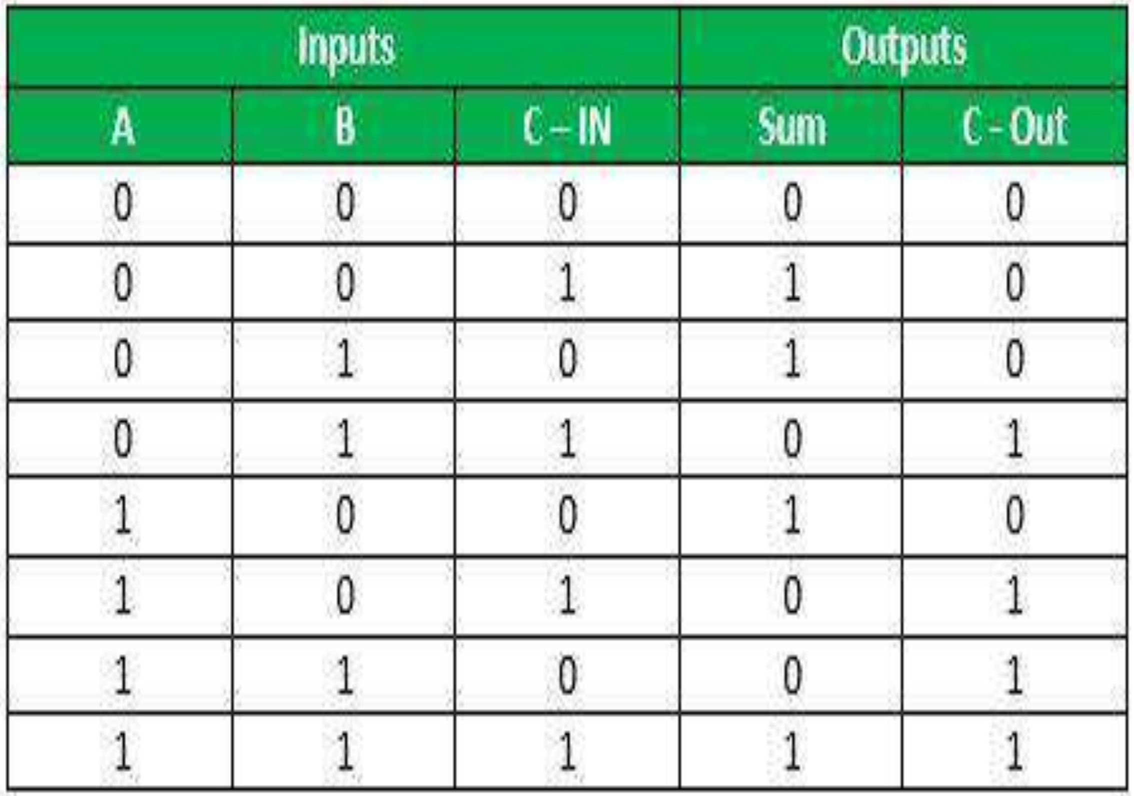
A \ B	0	1
0	0	1
1	1	0

Sum = $\overline{A}B+A\overline{B}$
$=A\oplus B$


A
B
Sum
Carry
A
B
Sum
Carry

Full Adder





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

A \ BC	00	01	11	10
0	$_0$	1 $_1$	$_3$	1 $_2$
1	1 $_4$	$_5$	1 $_7$	$_6$

K-map for Sum (S)

A \ BC	00	01	11	10
0	$_0$	$_1$	1 $_3$	$_2$
1	$_4$	1 $_5$	1 $_7$	1 $_6$

K-map for Carry (C_{out})

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

$$S = A \oplus B \oplus C$$

$$C = AB + BC + CA$$

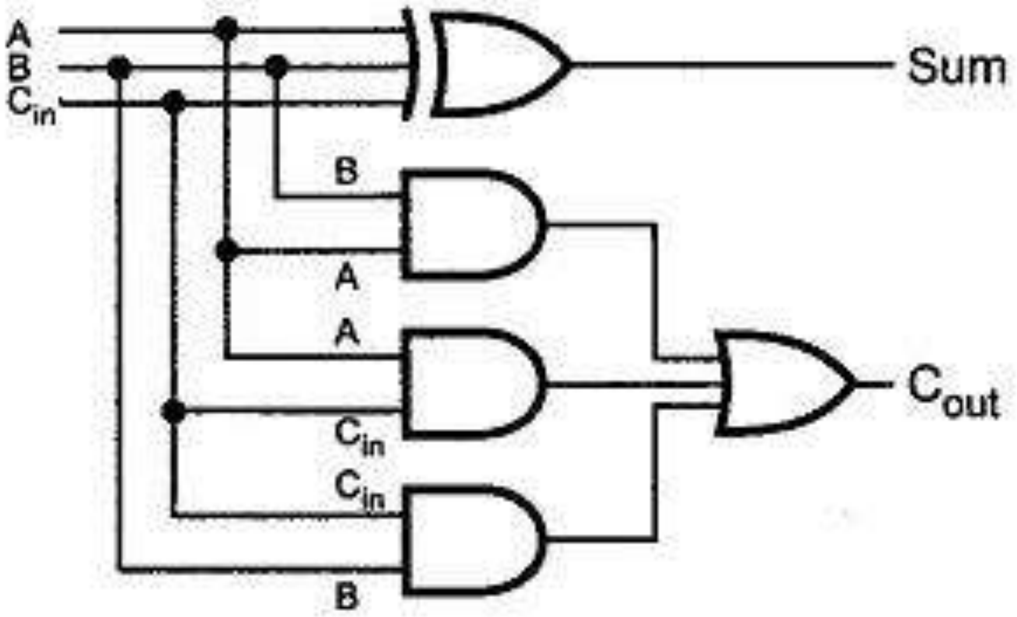
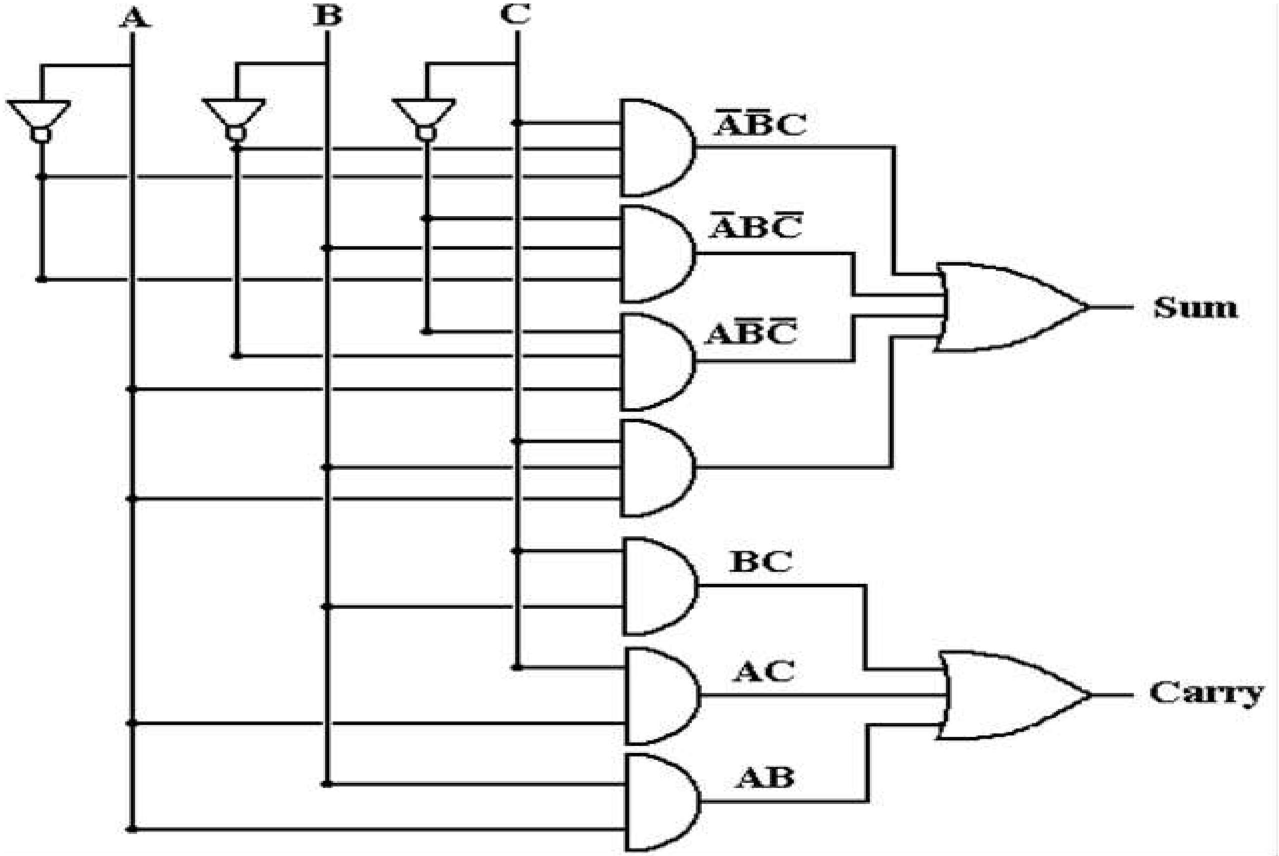


Fig. 3.17 Implementation of full-adder

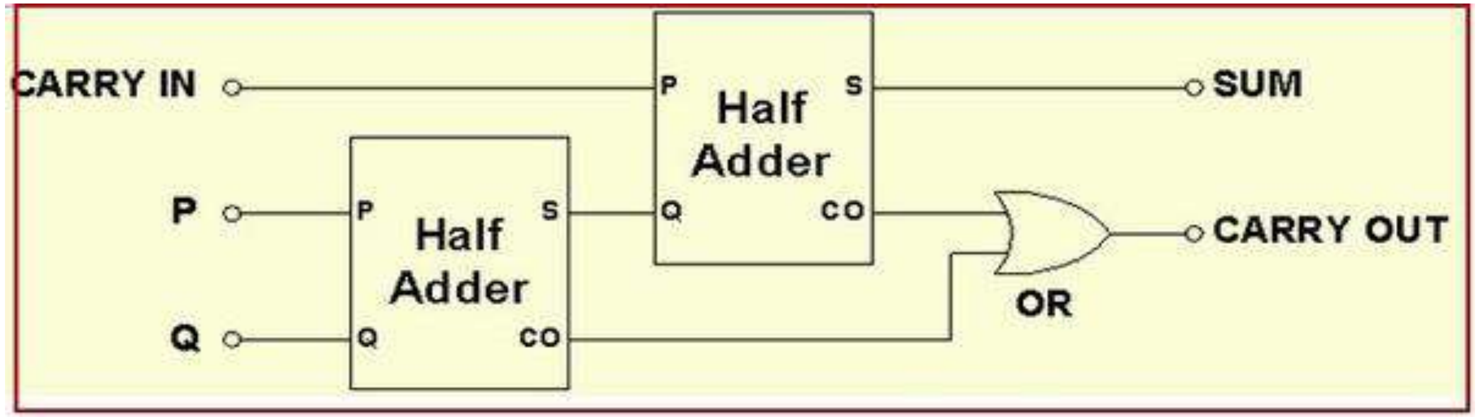
$$S = \overline{A}\,\overline{B}C+\overline{A}B\overline{C}+A\overline{B}\,\overline{C}+ABC$$

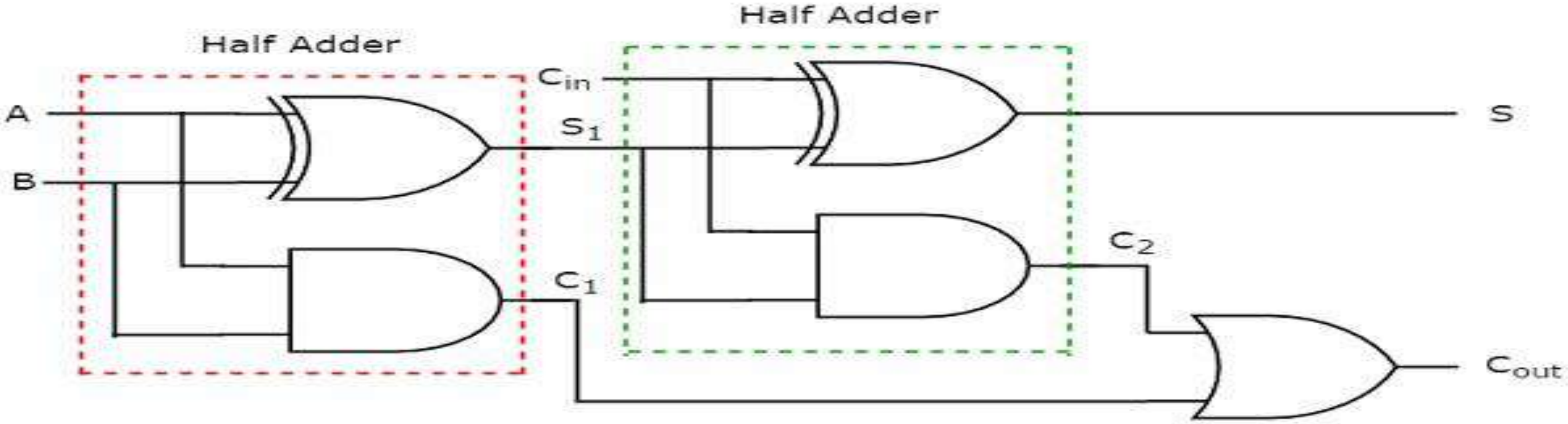
$$S=A \oplus B \oplus C$$

$$C=AB+BC+CA$$

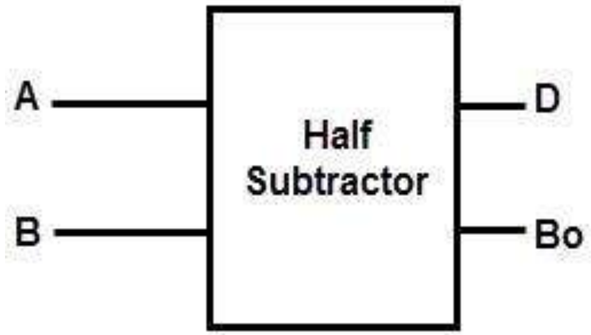

A
B
C
$\overline{A}\overline{B}C$
$\overline{A}B\overline{C}$
$A\overline{B}\overline{C}$
Sum
BC
AC
AB
Carry

Full Adder using Half Adders





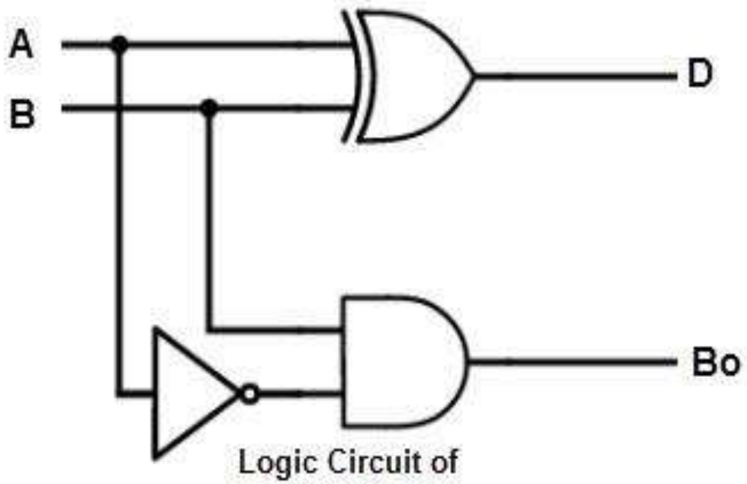
Half Subtractor



Block Diagram

A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table



Logic Circuit of Half Subtractor

For D:

A \ B	$\bar{B}$	B
$\bar{A}$		1
A	1	

$D = A \oplus B$

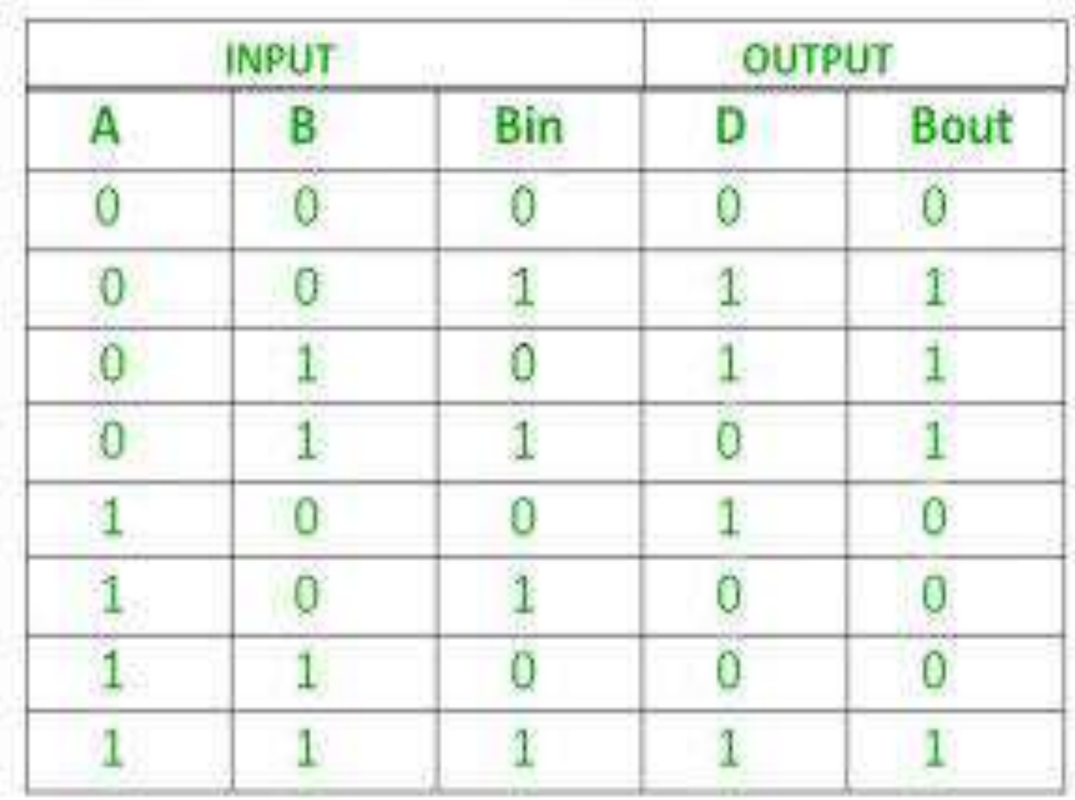
For b:

A \ B	$\bar{B}$	B
$\bar{A}$		1
A		

$b = \bar{A}.B$

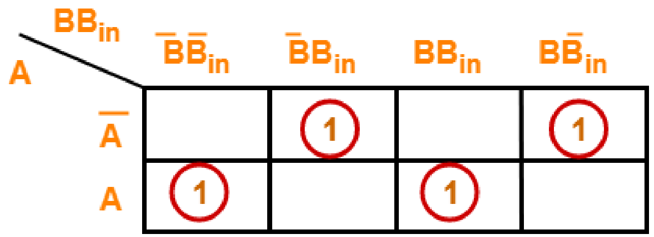
K Maps

Full Subtractor



INPUT			OUTPUT	
A	B	Bin	D	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

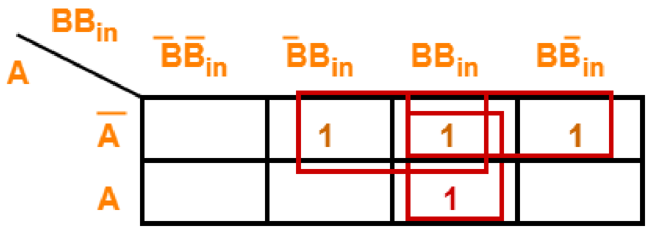
For D:



A \ BB_{in}	$\bar{B}\bar{B}_{in}$	$\bar{B}B_{in}$	BB_{in}	$B\bar{B}_{in}$
$\bar{A}$		1		1
A	1		1	

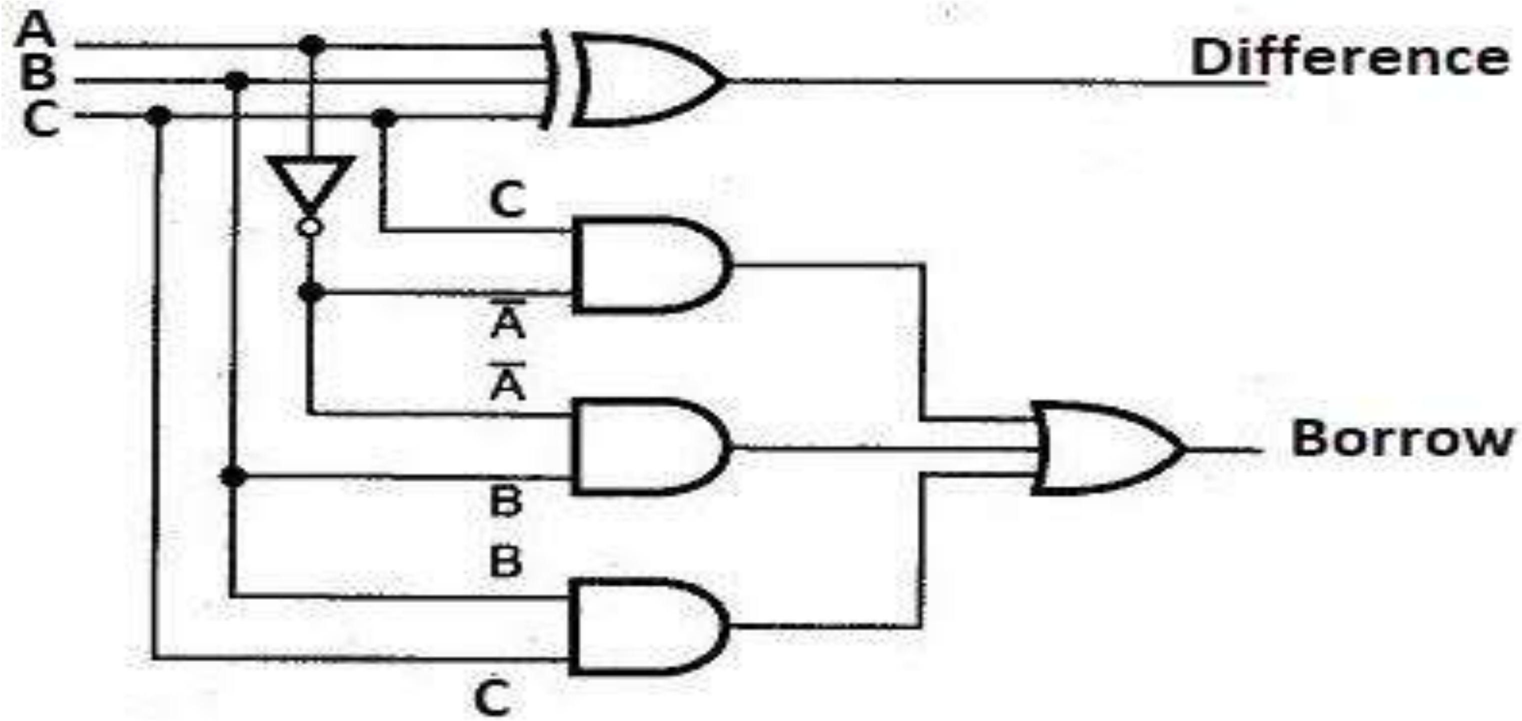
$$\boxed{D = A \oplus B \oplus B_{in}}$$

For B_{in}:

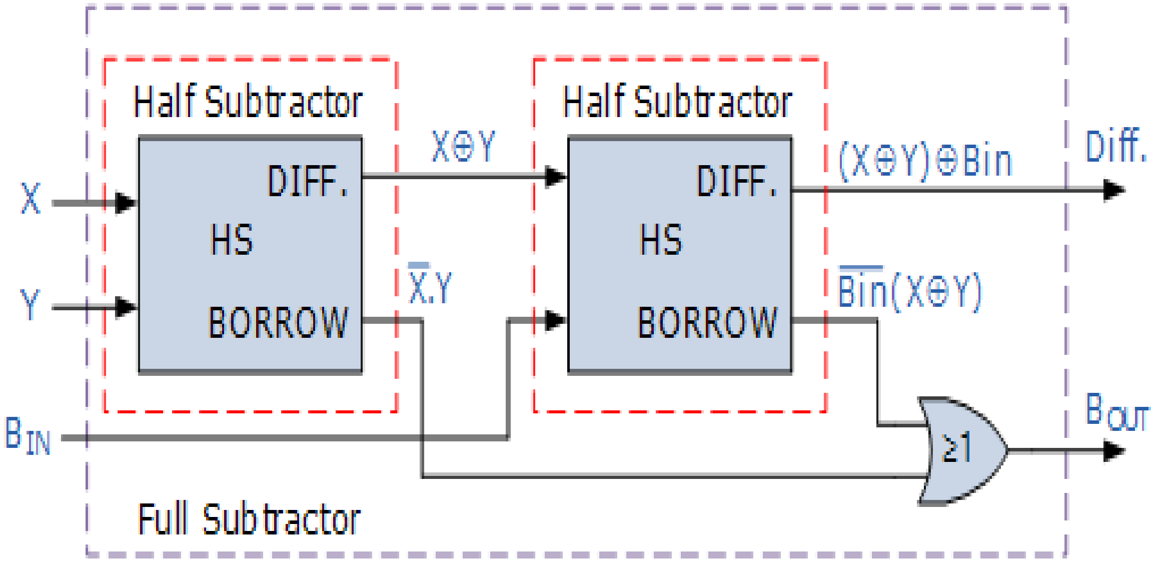


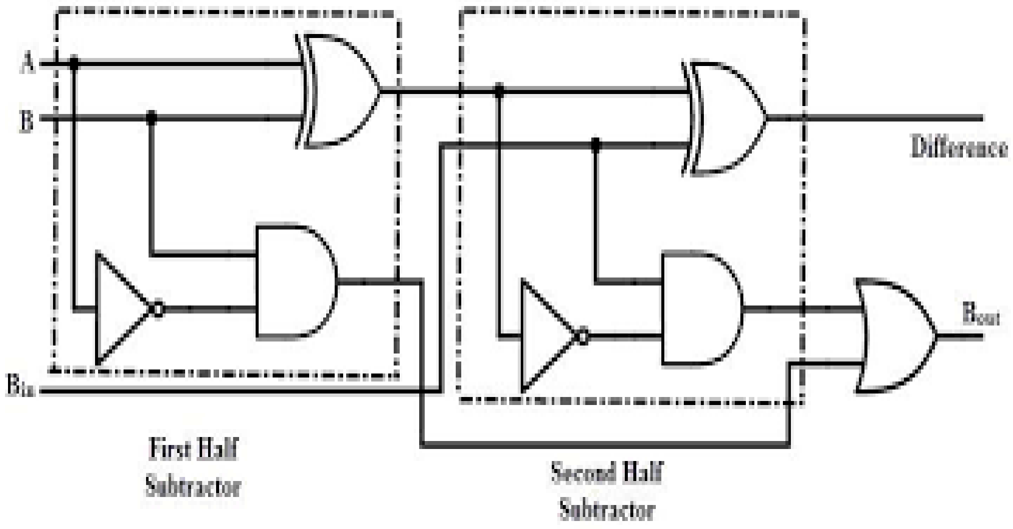
A \ BB_{in}	$\bar{B}\bar{B}_{in}$	$\bar{B}B_{in}$	BB_{in}	$B\bar{B}_{in}$
$\bar{A}$		1	1	1
A			1	

$$\boxed{B_{out} = \bar{A}\,B + (\bar{A} + B)\,B_{in}}$$


A
B
C
Difference
C
$\overline{A}$
$\overline{A}$
B
B
C
Borrow

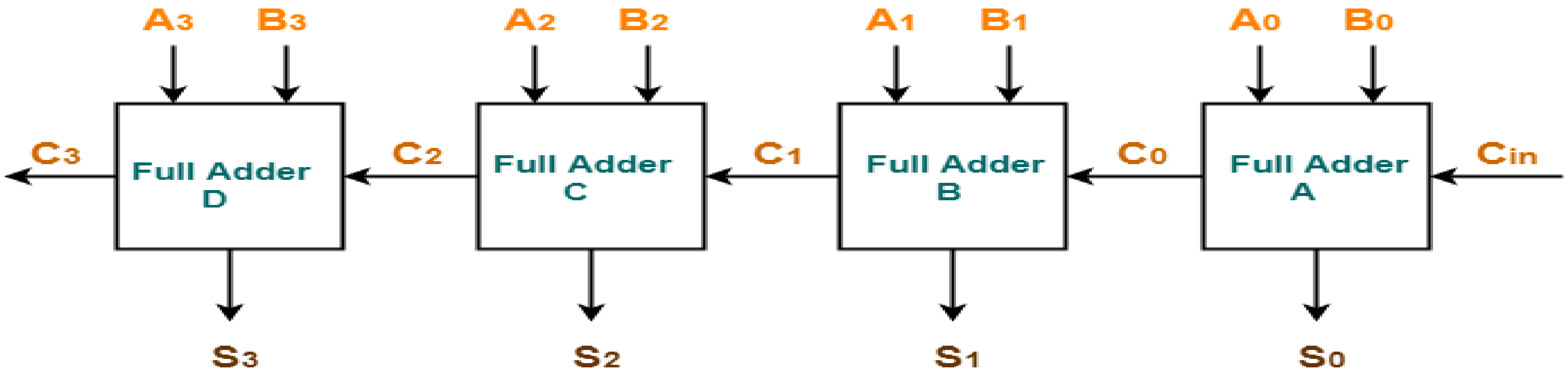
Full Subtractor using Half Subtractor





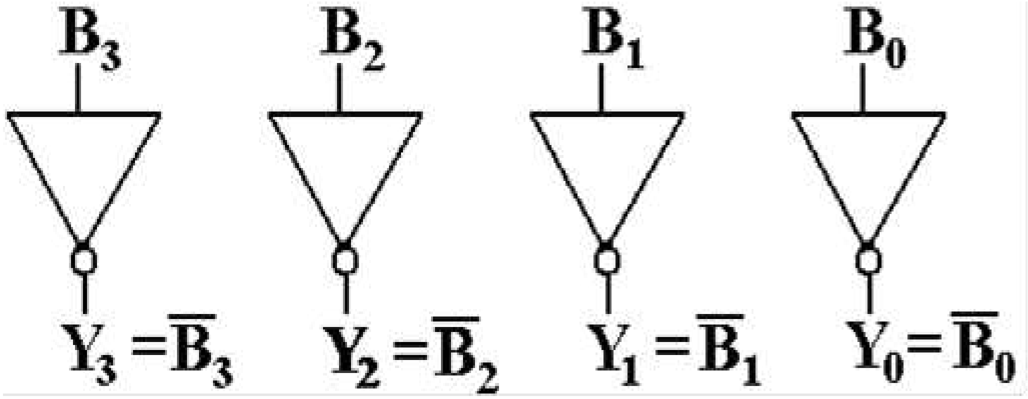
Ripple/ Parallel Adder

- Just as we combined half adders to make a full adder, full adders can connected in series.
- The carry bit "ripples" from one adder to the next; hence, this configuration is called a *ripple-carry adder*.



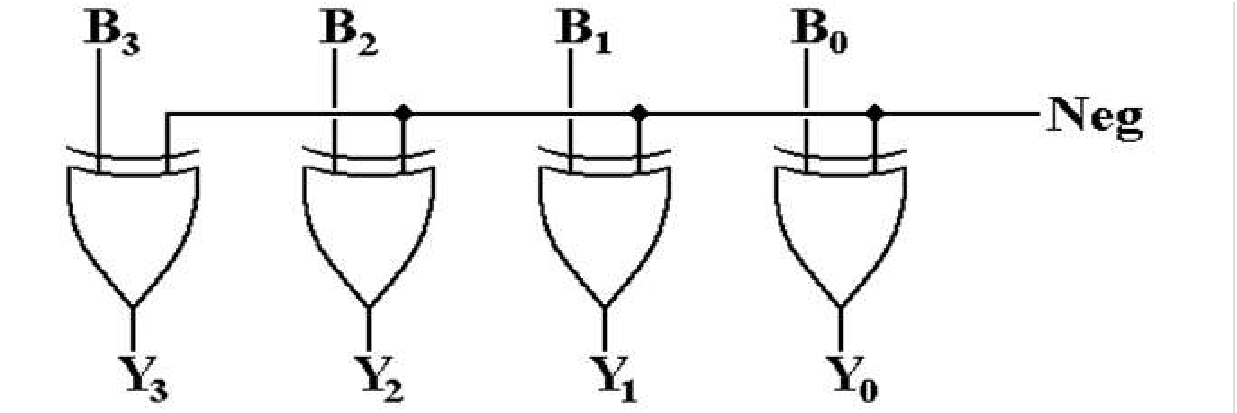
4-bit Ripple Carry Adder

One’s Complement Circuit



In order to make an adder/subtractor, it is necessary to use a gate that can either pass the value through or generate its one’s–complement.

The exclusive OR gate, XOR, is exactly what we need.

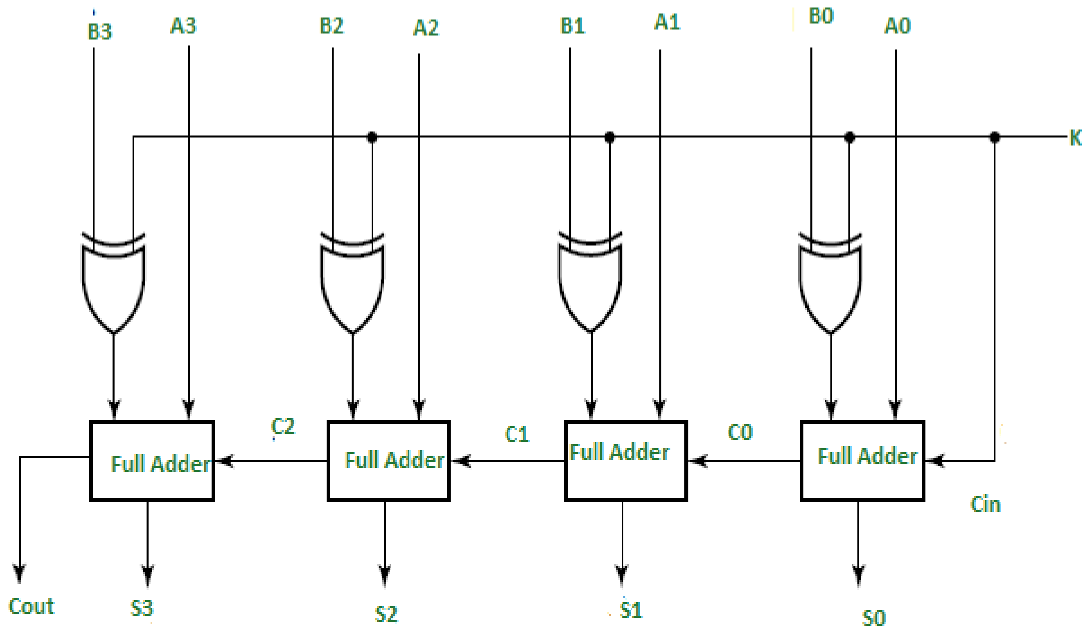


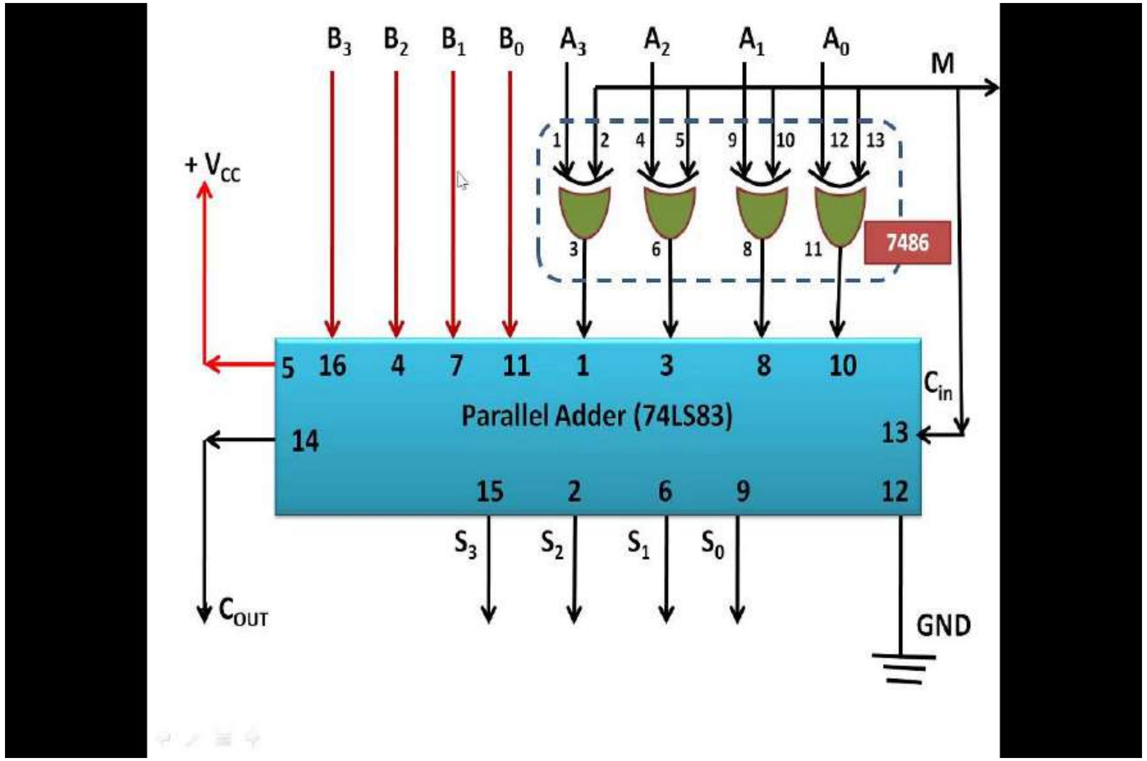
If Neg = 0 Then $Y_3 = B_3, Y_2 = B_2, Y_1 = B_1$, and $Y_0 = B_0$

If Neg = 1 Then $Y_3 = \overline{B_3}, Y_2 = \overline{B_2}, Y_1 = \overline{B_1}$, and $Y_0 = \overline{B_0}$

This is controlled by a binary signal: **Neg**.
Let B = 1011.
If Neg = 0, then Y = 1011.
If Neg = 1, then Y = 0100.

Adder-Subtractor





- In any combinational circuit, the signal must propagate through the gates before the correct output is available in the output terminal.
- The total propagation time equal to the propagation delay of a typical gate times multiplied with the gate levels in the circuit.
- The propagation delay time in a parallel adder is the time it takes the carry to propagate through the full adder.
- In each full adder the carry out from the carry in passes through two gate levels.
- For n-bit parallel adder the total gate delay will be 2n.

- So, the carry propagation time is a limiting factor on the speed with which two numbers are added in parallel.
- To avoid that another adder is widely used which employs the principle of Look-ahead carry.
- The adder designed using the principle of Look-ahead carry is called as Look-ahead carry adder or Carry look-ahead adder.

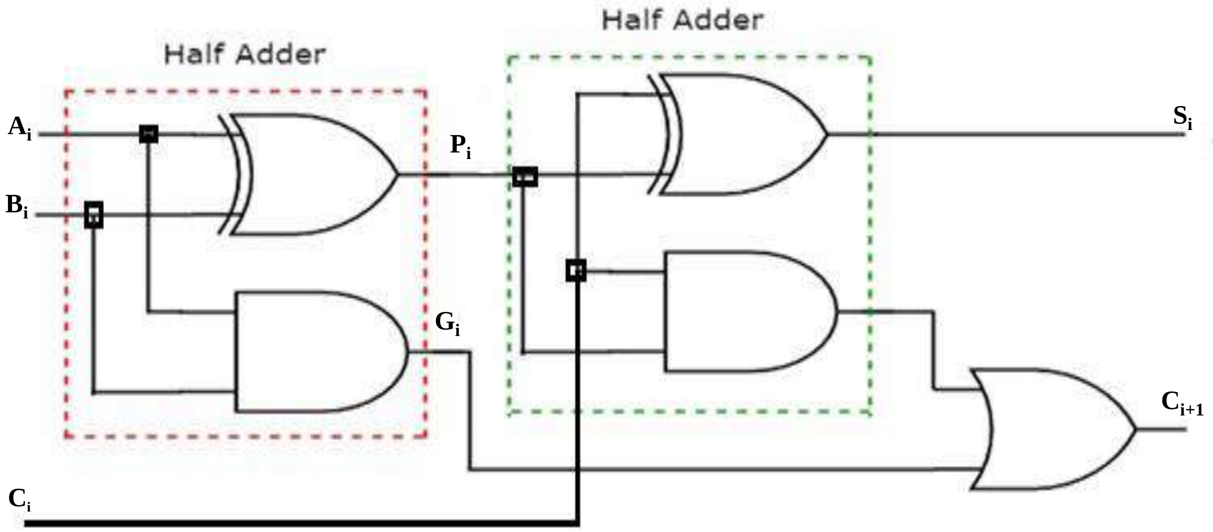
Look-Ahead Carry Adder

$P_i = A_i \oplus B_i$
$G_i = A_i\, B_i$

The output Sum and Carry can be expressed as:

$S_i = P_i \oplus Ci$
$C_{i+1} = G_i + P_i\, C_i$

G_i is called as carry generator and P_i is called as carry propagator.



These equations show that a carry signal will be generated in two cases:

1) if both bits A_i and B_i are **1**
2) if either A_i or B_i is **1** and the carry-in C_i is **1**.

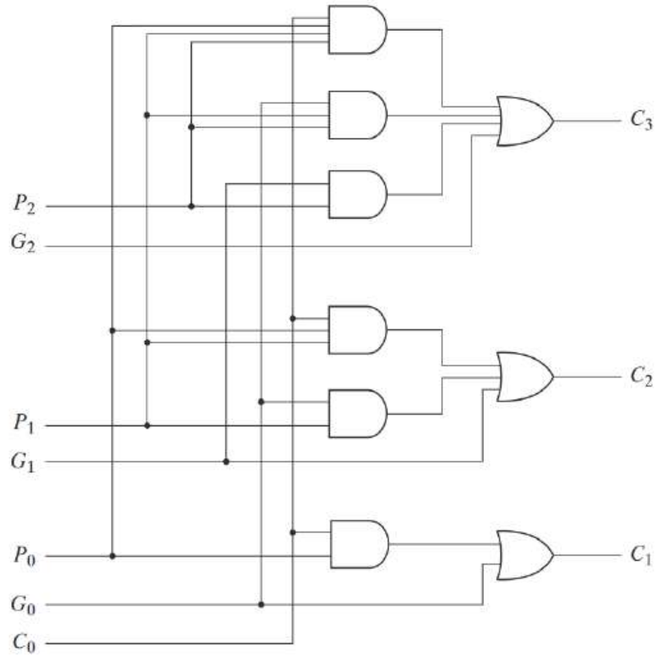
Let's apply these equations for a 4-bit adder:

$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$
$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

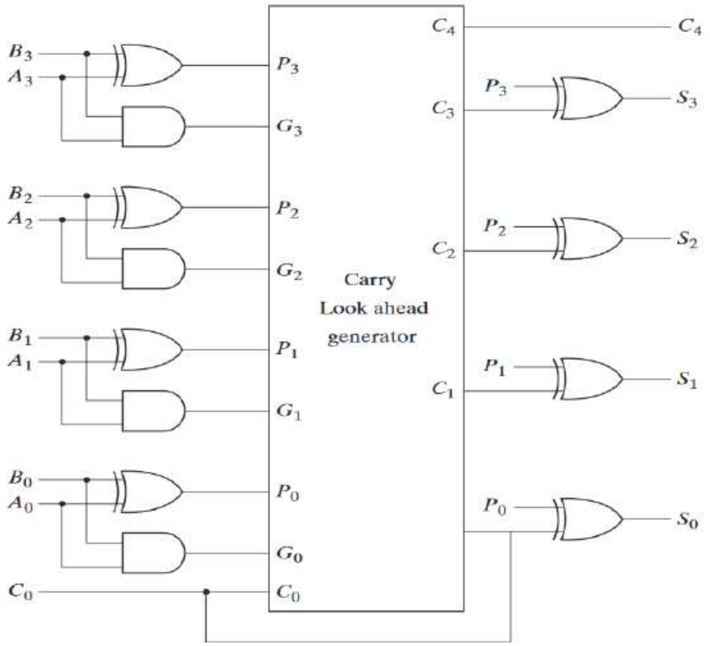
- These expressions show that C_2, C_3 and C_4 do not depend on its previous carry-in.
- Therefore C_4 does not need to wait for C_3 to propagate.
- As soon as C_0 is computed, C_4can reach steady state.
- The same is also true for C_2 and $C_{3.}$
- The general expression is

  $$C_{i+1}= G_i+ P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \ldots\ldots\ldots P_iP_{i-1}\ldots.P_2P_1G_0 + P_iP_{i-1} \ldots. P_1P_0C_0.$$

- This is a two level circuit



Carry Look-Ahead Generator



Full Adder with Look-Ahead Carry

Total 4 gate delay: One gate delay for P_i and G_i generator, two gate delay for Carry generator and one gate delay for Sum generator.

Advantages:

- CLA Adders generate the carry-in for each full adder simultaneously, by using simplified equations involving P_i, G_i, and C_{in}.
- This system reduces the propagation delay.
- This is because the output carry at any stage is dependent only on the first Carry signal given at the input.
- It is the fastest adder when compared to other addition mechanisms.

Disadvantages:

- The carry look-ahead adder circuit gets more complicated as the number of variables increase.
- The circuit for a carry look-ahead adder is expensive as it involves more hardware.
- As the number of variables increases, the circuit implements more hardware.
- Thus, when the carry look-ahead adder is implemented as an IC, the area is bound to increase.

Ripple Carry Adder vs. Carry Look Ahead Adder

Ripple Carry Adder	Carry Look Ahead Adder
The Carry bit passes through a long logic chain through the entire circuit.	The Carry bit enters in the system only at the input.
As the full adder blocks are dependent on their predecessor blocks' carry value, the entire system works a little slow.	Since the entire system depends on the first carry input, the computations are very quick, making it the fastest adder.
It has a simple repetitive design.	Has a slightly complicated design with many logic gates
The system design is cheap to manufacture.	The manufacturing process is expensive as compared to other systems.
The ripple carry adder chips have a considerable size and area.	The chip area increases, as there are many components in the circuit.

Combinational Circuits

Overview

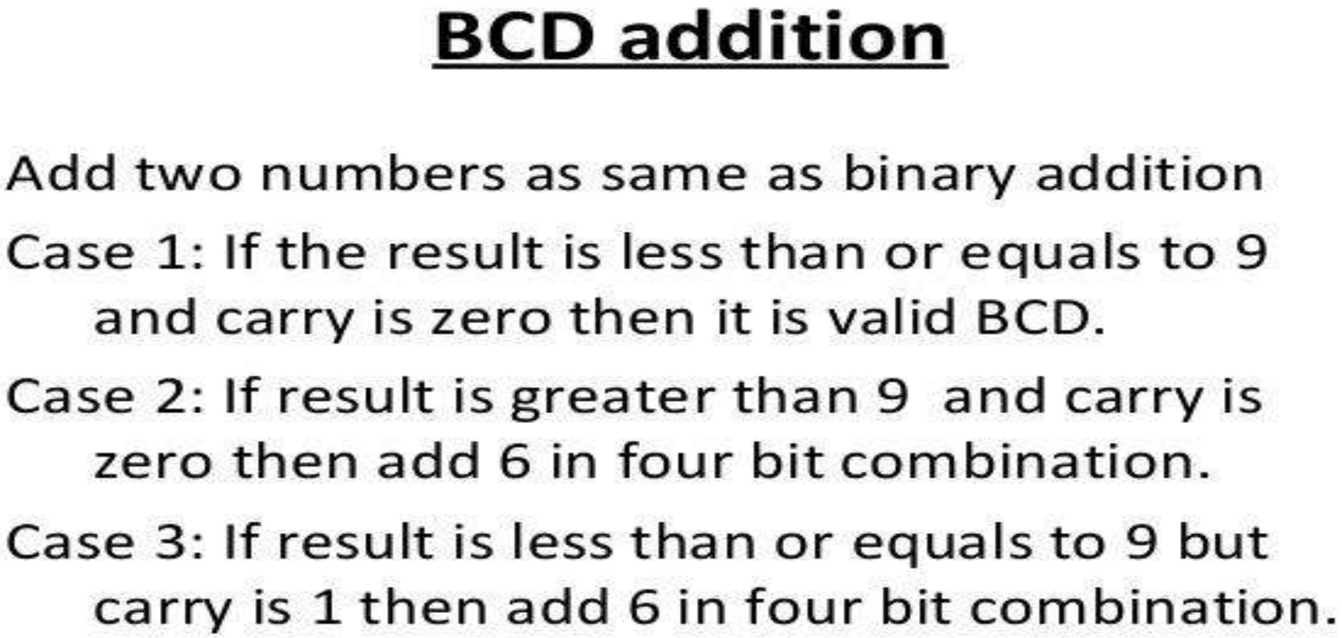
- **BCD Adder**
- **BCD Subtractor**
- **Comparator**
- **Error detection and correction codes**

BCD

Decimal Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- Inputs: $A_3A_2A_1A_0,\ B_3B_2B_1B_0, C_{in}$ from previous decade.
- Output: C_{out} (carry to next decade), $Z_3Z_2Z_1Z_0$.
- Idea: Perform regular binary addition and then apply a corrective procedure.

BCD Addition Rules



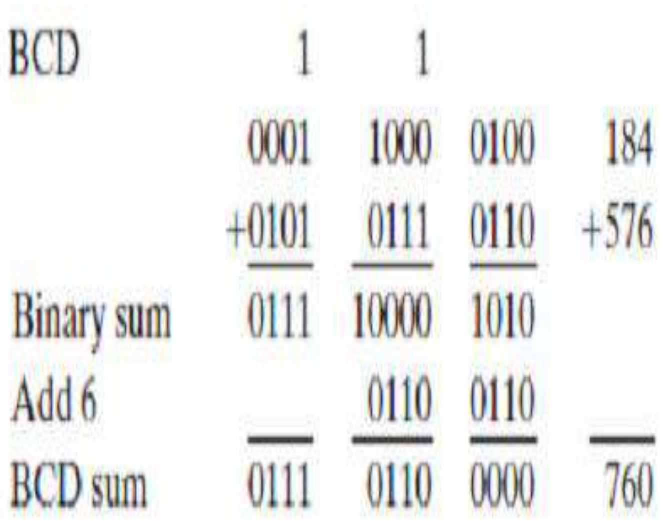
BCD addition

Add two numbers as same as binary addition

Case 1: If the result is less than or equals to 9 and carry is zero then it is valid BCD.

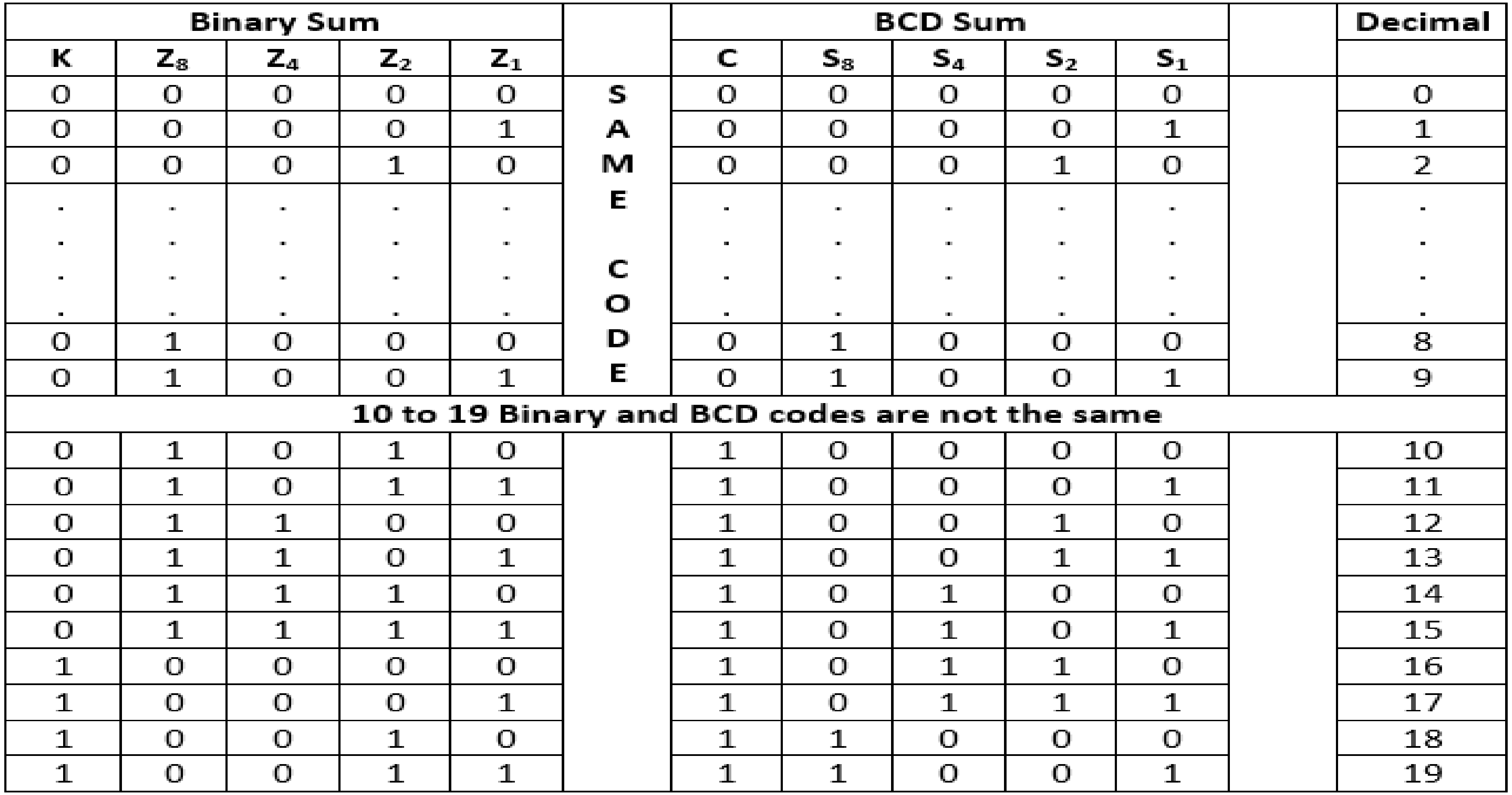
Case 2: If result is greater than 9 and carry is zero then add 6 in four bit combination.

Case 3: If result is less than or equals to 9 but carry is 1 then add 6 in four bit combination.



BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

Comparing Binary and BCD Sums



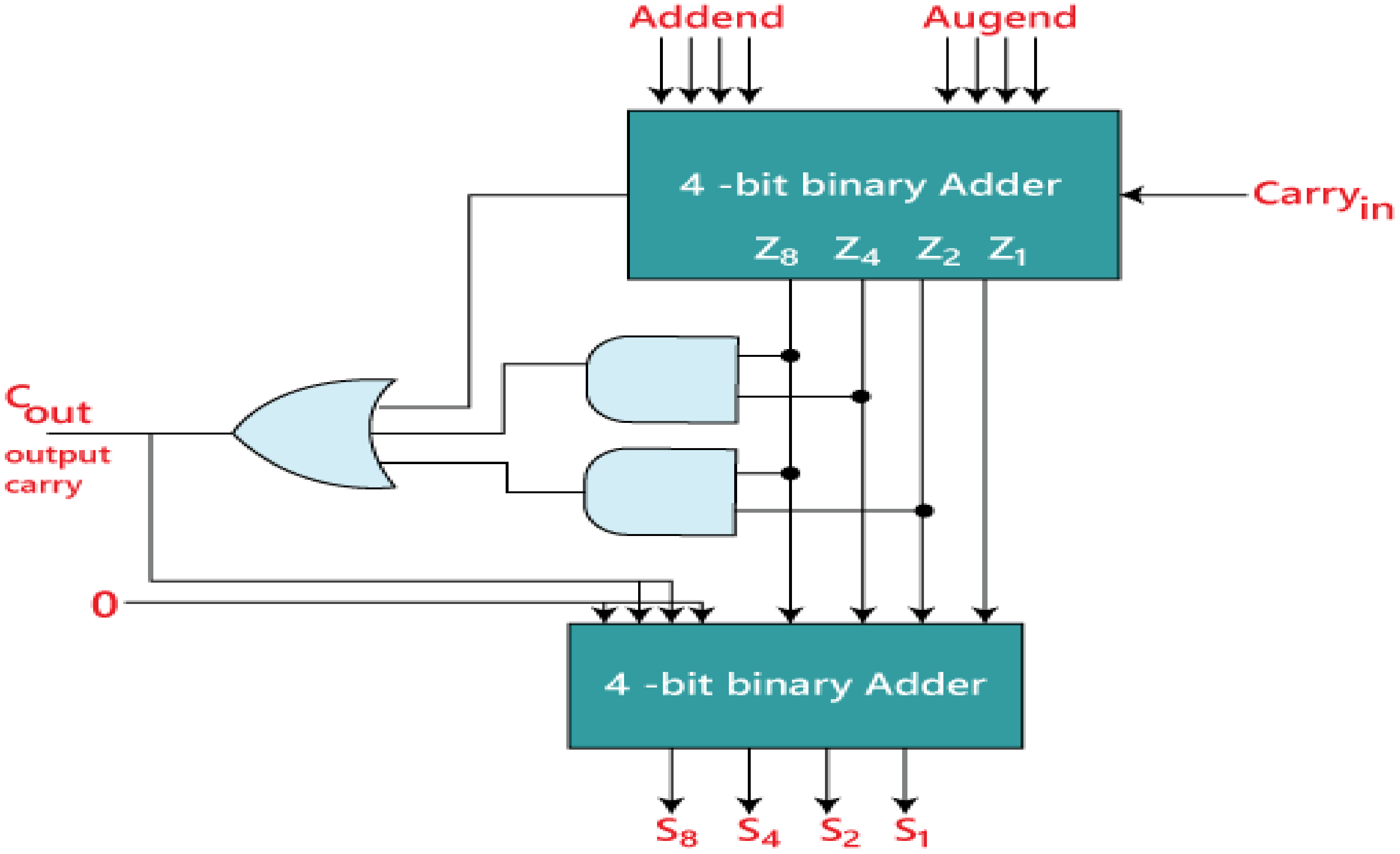
Binary Sum						BCD Sum						Decimal
K	Z_8	Z_4	Z_2	Z_1		C	S_8	S_4	S_2	S_1		
0	0	0	0	0	SAME CODE	0	0	0	0	0		0
0	0	0	0	1		0	0	0	0	1		1
0	0	0	1	0		0	0	0	1	0		2
. . . .	. . . .	. . . .	. . . .	. . . .		. . . .	. . . .	. . . .	. . . .	. . . .		. . . .
0	1	0	0	0		0	1	0	0	0		8
0	1	0	0	1		0	1	0	0	1		9
10 to 19 Binary and BCD codes are not the same												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

BCD Adder

- In the previous table Decimal sum from **0 to 9**, the Binary sum same as BCD sum. So, no conversion is needed.
- Apply correction if the Decimal sum is between **10-19**.
  - The correction is needed (Decimal sum **16-19**)when the binary sum has an output carry $K = 1$
  - The correction is needed (Decimal sum **10-15**)when $Z_8 = 1$ and either $Z_4 = 1$ or $Z_2 = 1$.
- So, the condition for a correction and an output carry can be expressed by the Boolean function:

$$C = K + Z_8Z_4 + Z_8Z_2$$

- When $C = 1$, it is necessary to add 0110 to the binary sum to get BCD sum and provide an output carry for the next stage.



Black diagram of a BCD adder

Cascading of BCD Adders

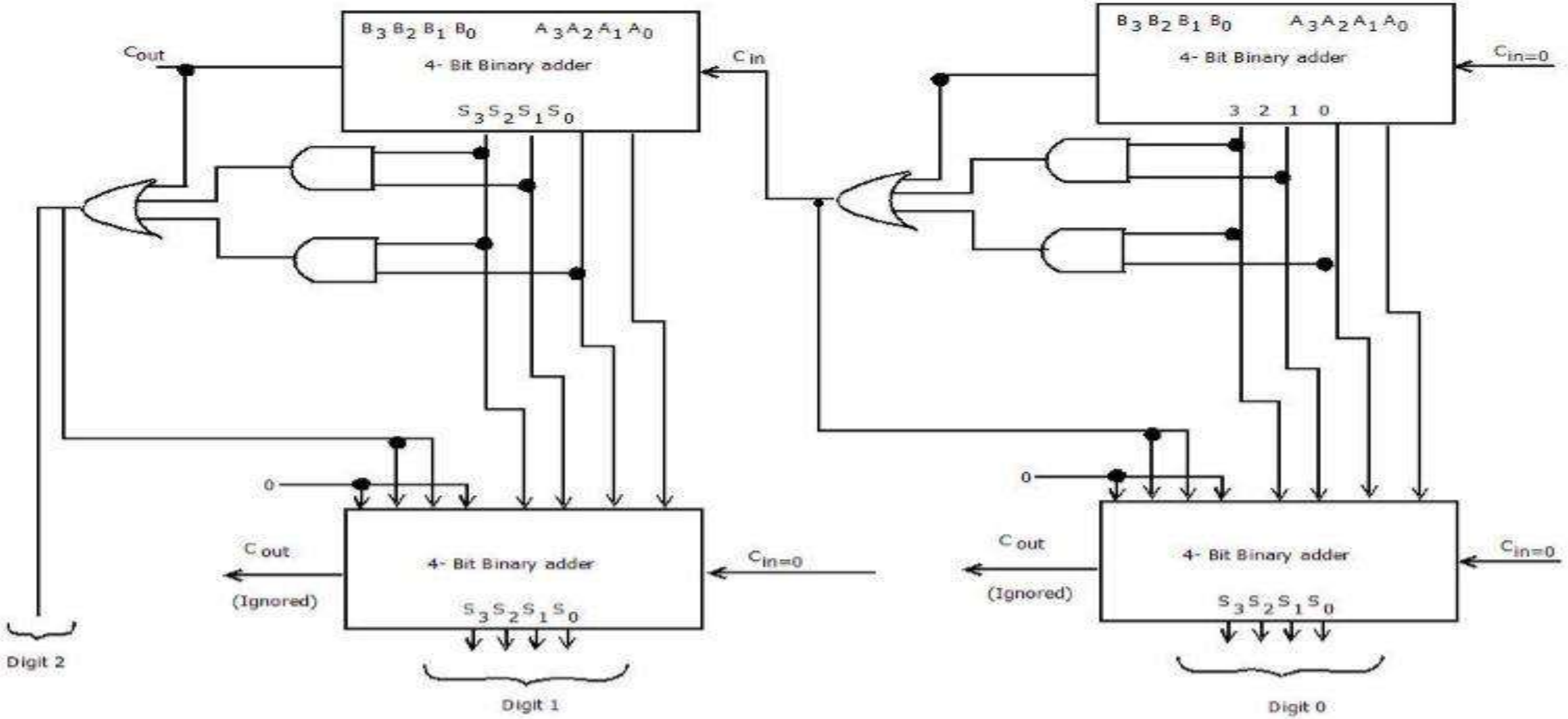


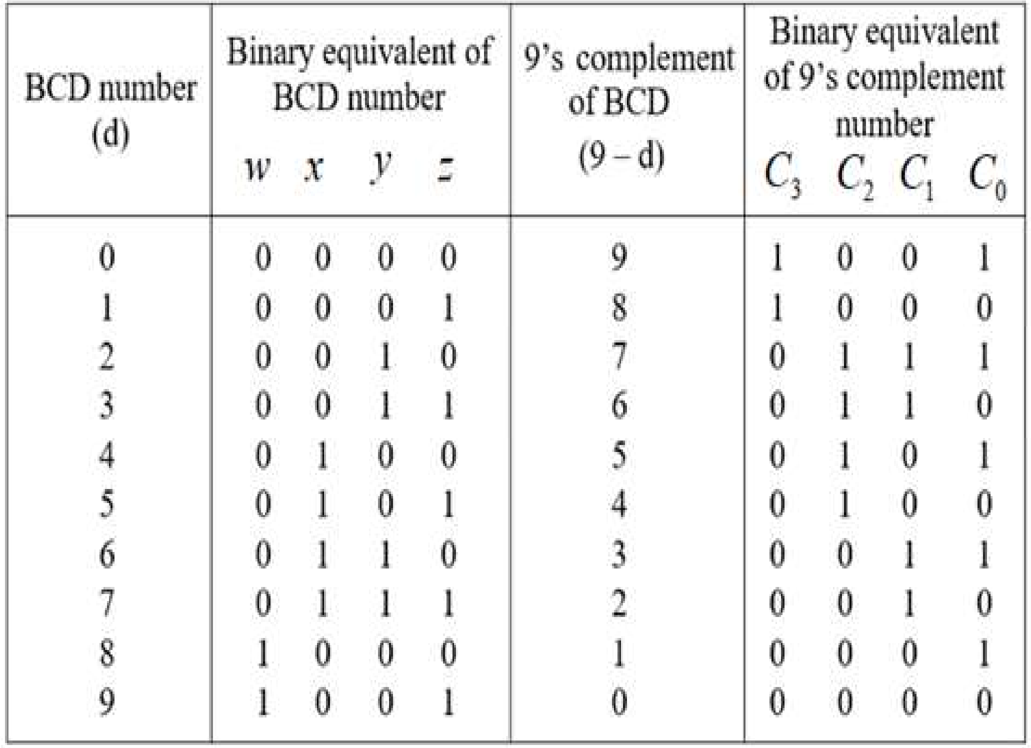
Fig : 8 - Bit BCD Adder

BCD Subtraction Rules

Let two BCD numbers are A and B.
B to be subtracted from A.

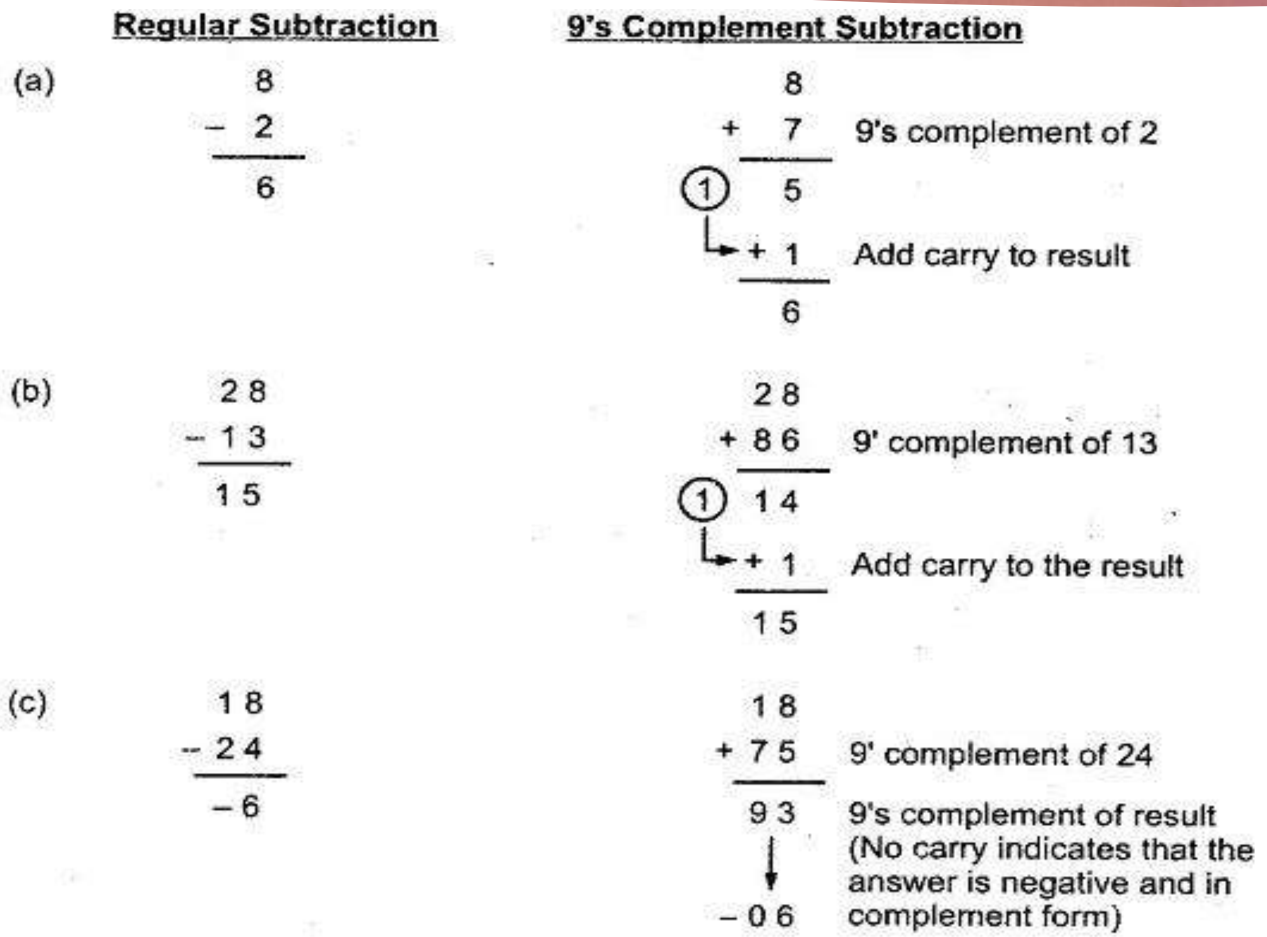
RULES:

- Add 9’s Complement of B to A
- If result > 9, Correct by adding 0110
- If carry is generated at most significant position then the result is positive and the End around carry must be added
- If carry is not generated at most significant position then the result is negative and the result is 9’s complement of original result



BCD number (d)	Binary equivalent of BCD number				9’s complement of BCD (9 – d)	Binary equivalent of 9’s complement number			
	w	x	y	z		C_3	C_2	C_1	C_0
0	0	0	0	0	9	1	0	0	1
1	0	0	0	1	8	1	0	0	0
2	0	0	1	0	7	0	1	1	1
3	0	0	1	1	6	0	1	1	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	4	0	1	0	0
6	0	1	1	0	3	0	0	1	1
7	0	1	1	1	2	0	0	1	0
8	1	0	0	0	1	0	0	0	1
9	1	0	0	1	0	0	0	0	0

Example

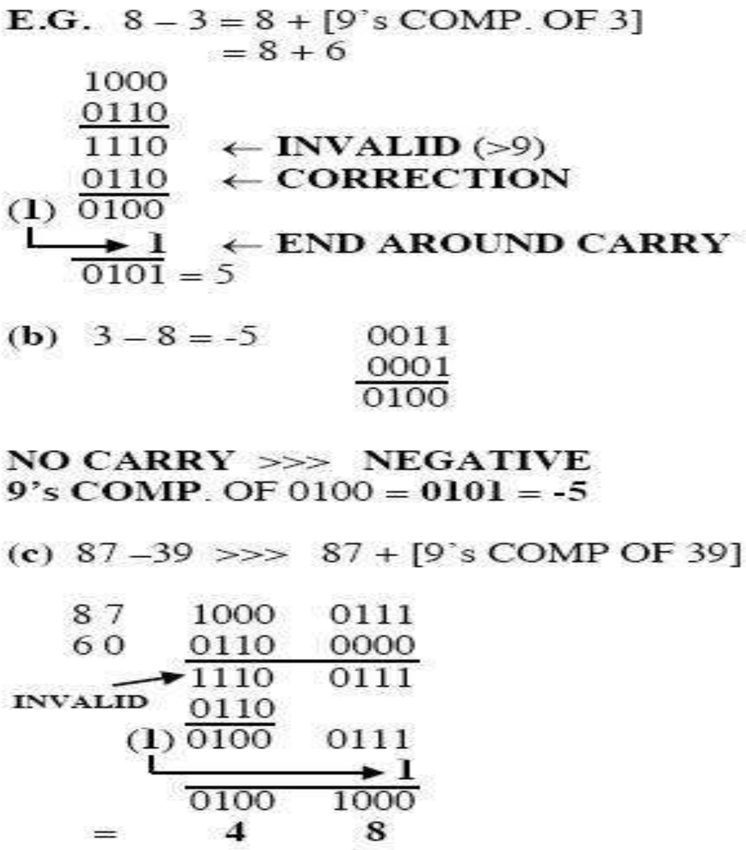


```
      Regular Subtraction        9's Complement Subtraction

(a)        8                          8
         − 2                        + 7    9's complement of 2
         ───                        ───
           6                   (1)    5
                                └─► + 1    Add carry to result
                                    ───
                                      6

(b)       2 8                        2 8
        − 1 3                      + 8 6   9' complement of 13
        ─────                      ─────
          1 5                  (1)   1 4
                                └─► +  1   Add carry to the result
                                   ─────
                                     1 5

(c)       1 8                        1 8
        − 2 4                      + 7 5   9' complement of 24
        ─────                      ─────
          − 6                        9 3   9's complement of result
                                      ↓    (No carry indicates that the
                                           answer is negative and in
                                     − 0 6 complement form)
```



```
E.G.  8 – 3 = 8 + [9's COMP. OF 3]
            = 8 + 6
      1000
      0110
      ────
      1110    ← INVALID (>9)
      0110    ← CORRECTION
      ────
  (1) 0100
   └────► 1   ← END AROUND CARRY
      ────
      0101 = 5

(b)  3 – 8 = -5        0011
                       0001
                       ────
                       0100

NO CARRY  >>>  NEGATIVE
9's COMP. OF 0100 = 0101 = -5

(c)  87 –39  >>>  87 + [9's COMP OF 39]

     8 7     1000    0111
     6 0     0110    0000
            ─────────────
INVALID ──► 1110     0111
             0110
             ────
         (1) 0100    0111
          └─────────────► 1
             ────    ────
             0100    1000
   =           4       8
```

9's Complement Circuit

- 9'complement of 2 is 7
- Binary equivalent of 2 is 0010
- 1's complement of 0010 is 1101
- Then, 1101

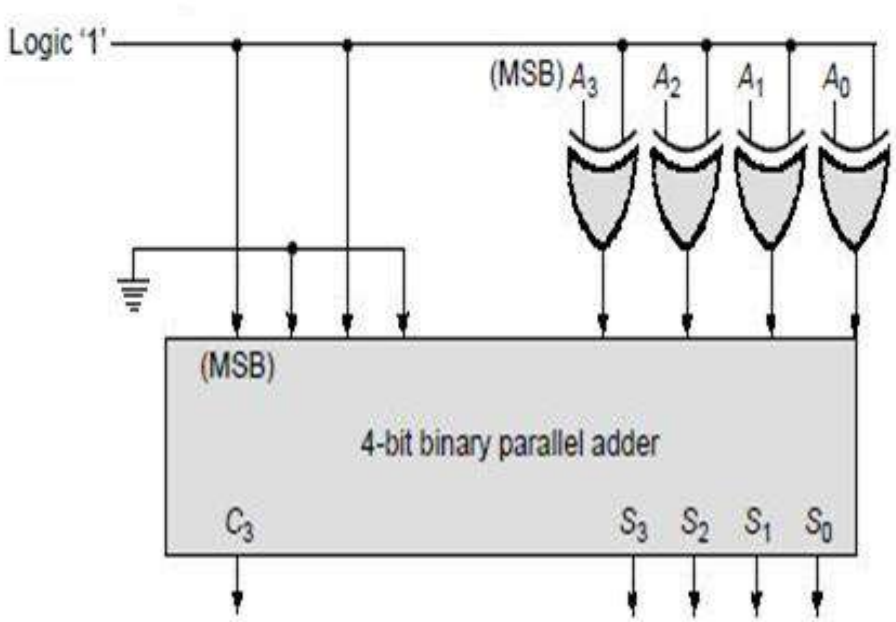
  $\underline{+\ 1010}$

  $=$ 0111 which is Binary equivalent of 7
- If carry discard it.

- 9'complement of 3 is 6
- Binary equivalent of 3 is 0011
- 1's complement of 0011 is 1100
- Then, 1100

  $\underline{+\ 1010}$

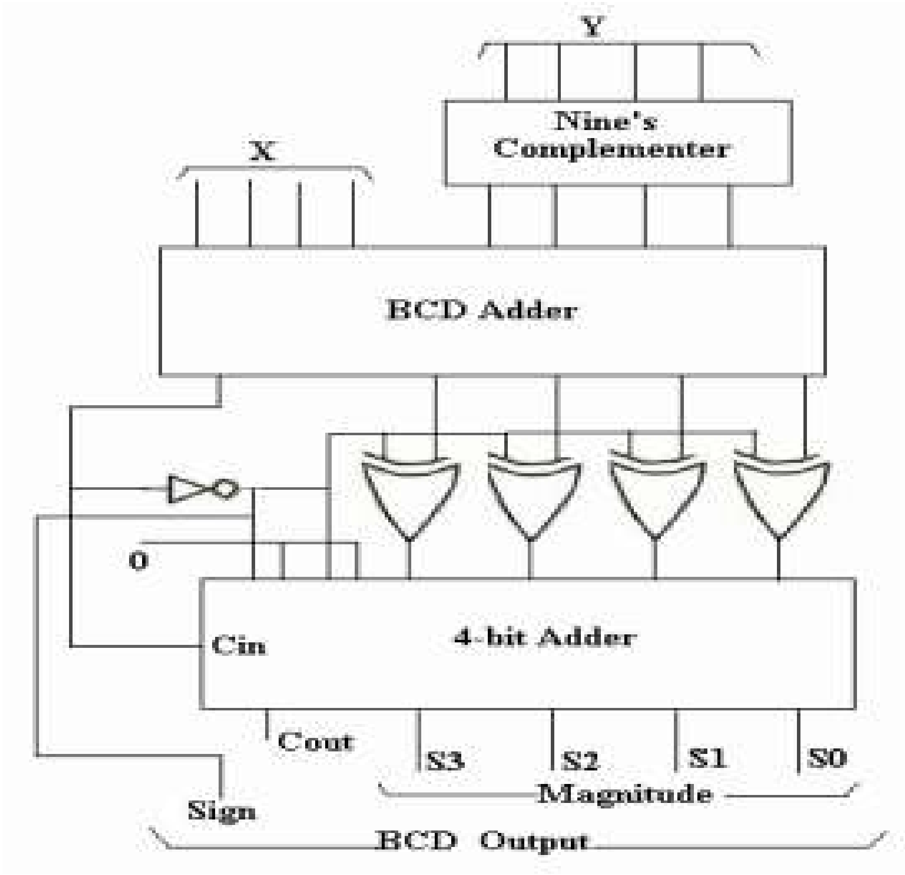
  $=$ 0110 which is Binary equivalent of 6
- If carry discard it.



BCD Subtractor Circuit

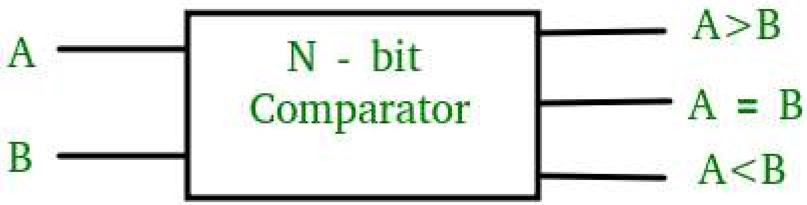
RULES:

- Add 9’s Complement of B to A
- If result > 9, Correct by adding 0110
- If carry is generated at most significant position then the result is positive and the End around carry must be added
- If carry is not generated at most significant position then the result is negative and the result is 9’s complement of original result



Comparator

- A magnitude digital 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 other for B and have three output terminals, one for $A > B$ condition, one for $A = B$ condition and one for $A < B$ condition.
- A comparator makes use of a cascade connection of identical sub networks similar to the case of the parallel adder.

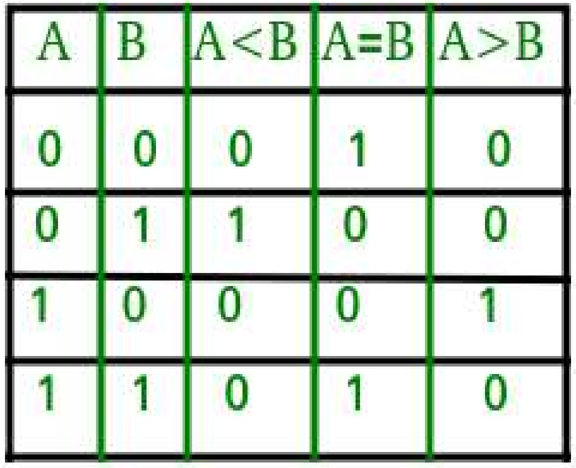


1-Bit Magnitude Comparator

- 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.

From the above truth table logical expressions for each output can be expressed as follows:

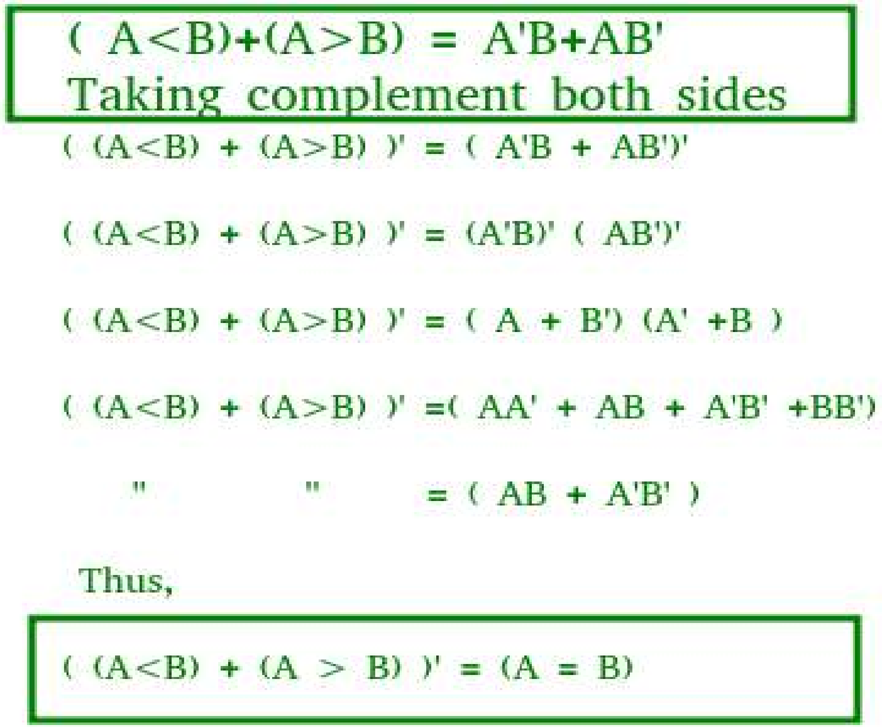
A>B: AB' **A<B: A'B** **A=B: A'B' + AB**



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

From the above expressions we can derive the following formula:



$$(A<B)+(A>B) = A'B+AB'$$

Taking complement both sides

$$((A<B) + (A>B))' = (A'B + AB')'$$

$$((A<B) + (A>B))' = (A'B)'(AB')'$$

$$((A<B) + (A>B))' = (A + B')(A' + B)$$

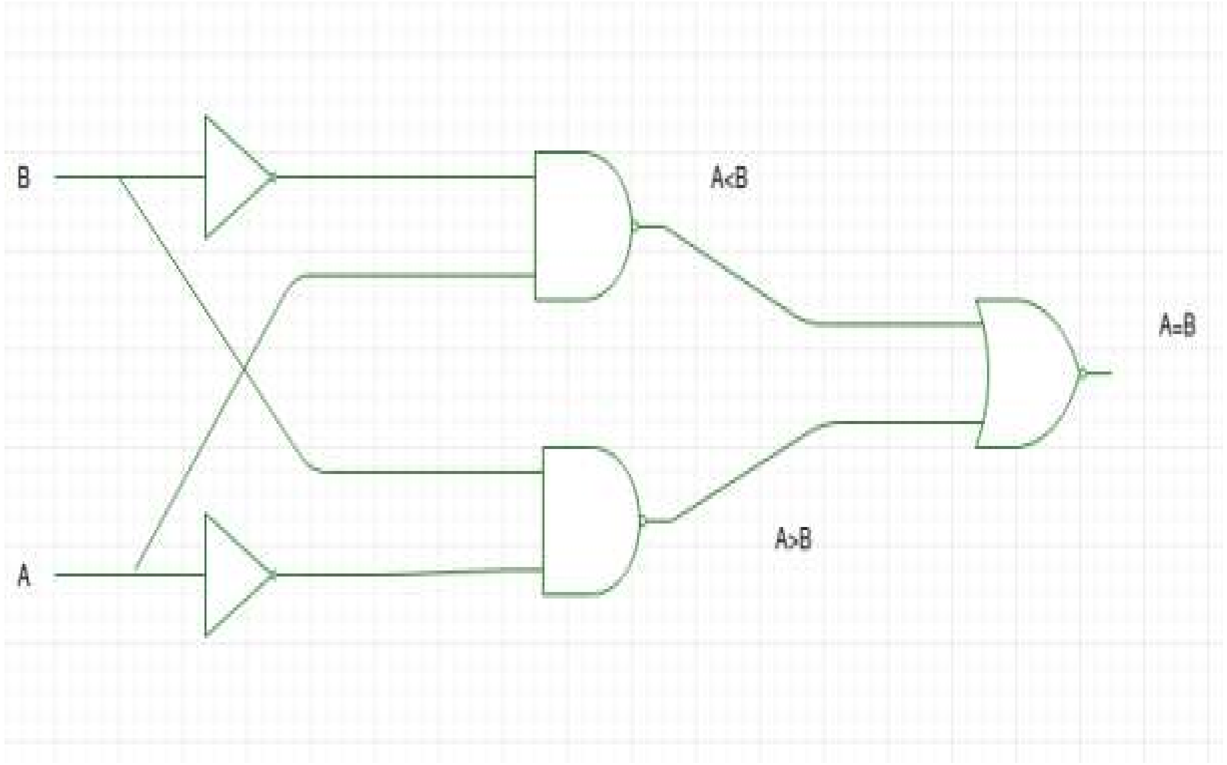
$$((A<B) + (A>B))' = (AA' + AB + A'B' + BB')$$

$$\text{"} \qquad \text{"} \qquad = (AB + A'B')$$

Thus,

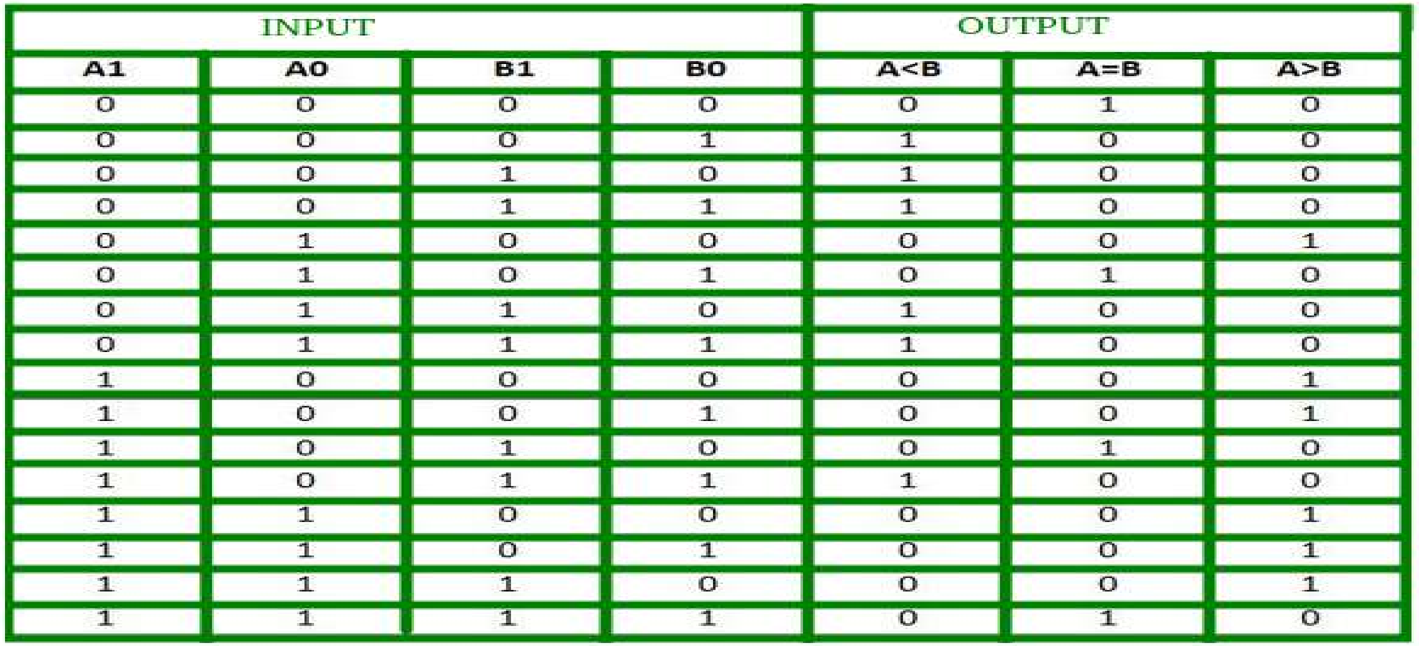
$$\boxed{((A<B) + (A > B))' = (A = B)}$$

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



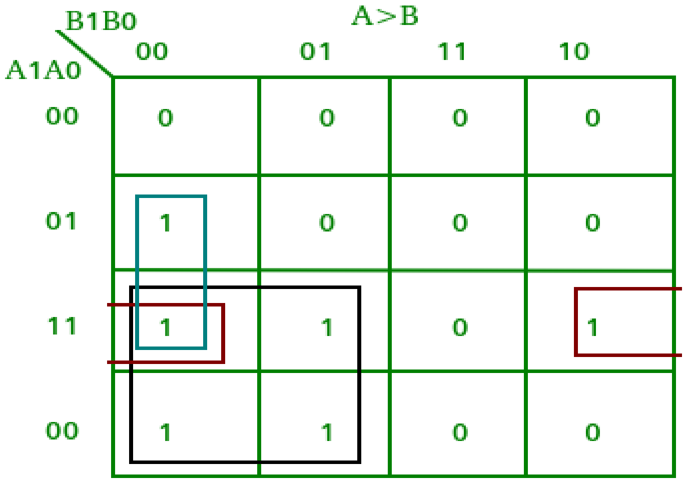
2-Bit Magnitude Comparator

A comparator used to compare two binary numbers 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.



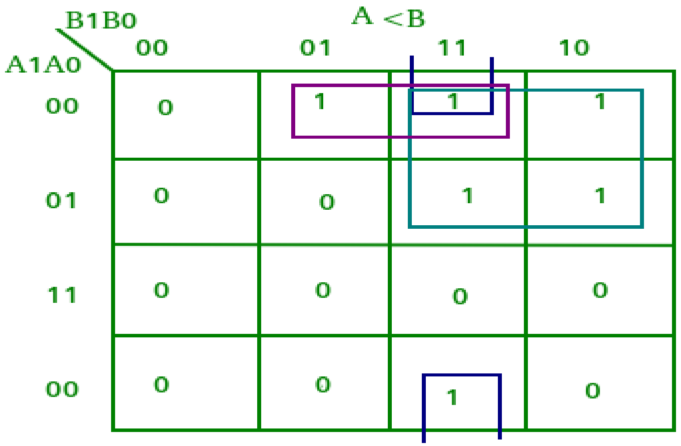
INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

From the Truth Table K-map for each output can be drawn as follows:



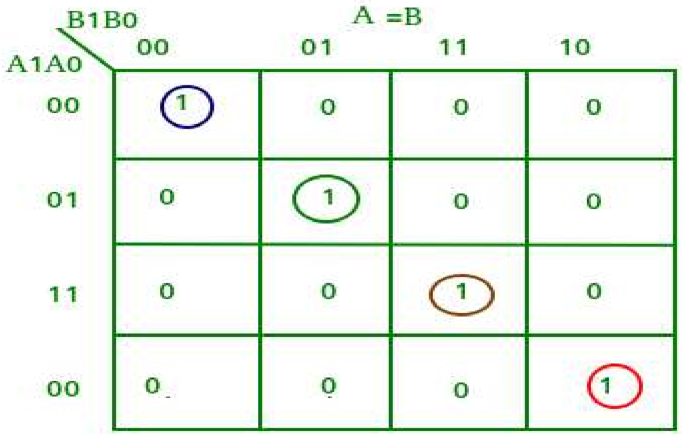
A>B

B1B0 A1A0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
00	1	1	0	0



A <B

B1B0 A1A0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
00	0	0	1	0



A =B

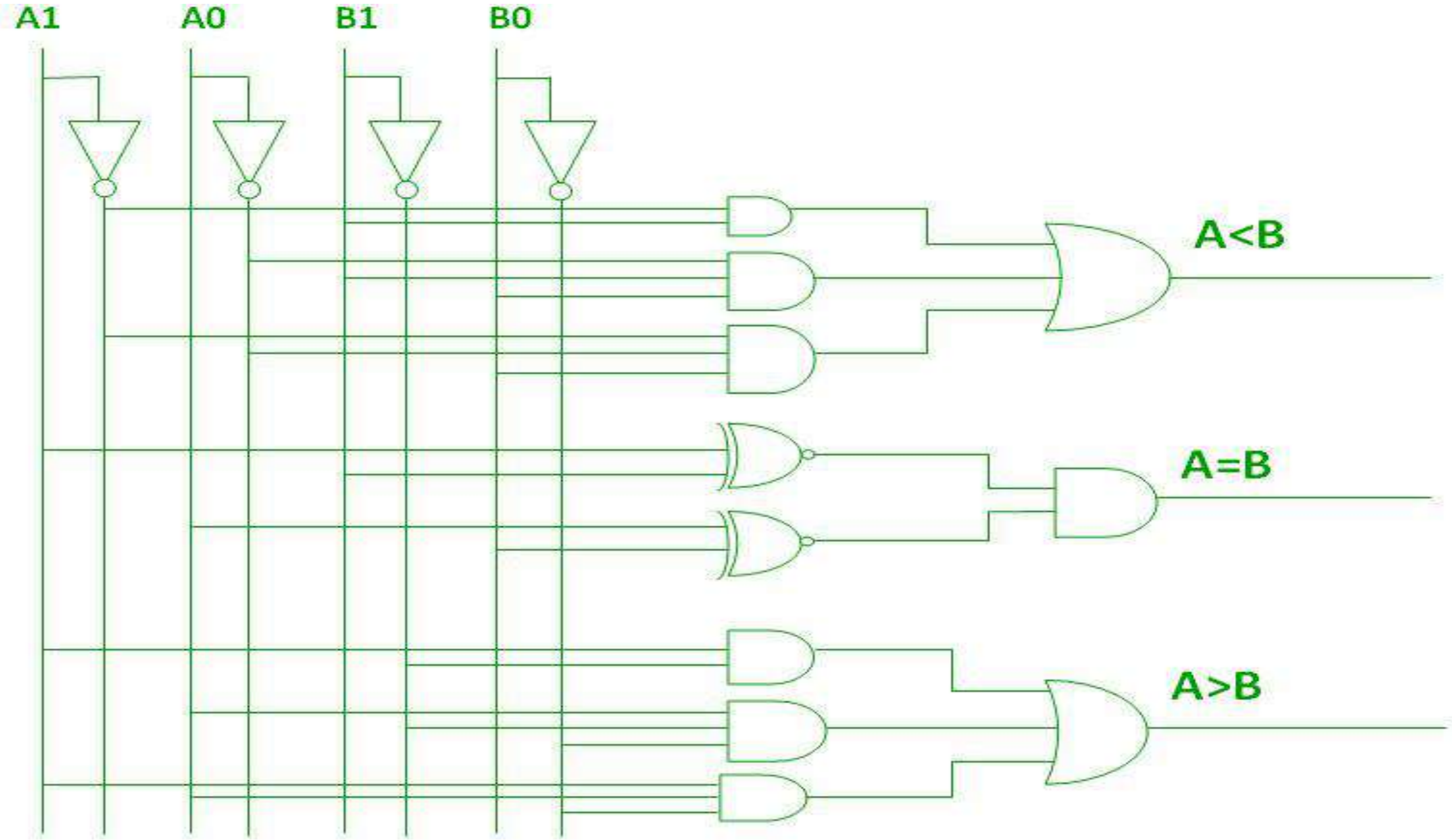
B1B0 A1A0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
00	0	0	0	1

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’)
(A0 Ex-Nor B0) (A1 Ex-Nor B1)

Logic Diagram

By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



4-Bit Magnitude Comparator

•A comparator used to compare two binary numbers each of four bits is called a 4-bit magnitude comparator.
• It consists of eight inputs each for two four bit numbers.
• Three outputs to generate less than, equal to and greater than between two binary numbers.

In a 4-bit comparator the condition of A = B can be possible in the following four cases:

A = B is possible only when all the individual bits of one number exactly coincide with corresponding bits of another number.

If ***A3 = B3 and A2 = B2 and A1 = B1 and A0 = B0***
As the numbers are binary, the digits are either 0 or 1.
The equality relation of each pair of bits can be expressed logically with an equivalence function.

xi = AiBi + Ai'Bi' ***i = 0, 1, 2, 3*** where ***xi = 1*** if the pair of bits in position i are equal.

So,

(A = B) = x3 . x2 . x1. x0

In a 4-bit comparator the condition of A>B can be possible in the following four cases:

If **A3 = 1 and B3 = 0**
If **A3 = B3, A2 = 1 and B2 = 0**
If **A3 = B3, A2 = B2, A1 = 1 and B1 = 0**
If **A3 = B3, A2 = B2, A1 = B1, A0 = 1 and B0 = 0**
The sequential comparison can be expressed logically as:

$$(A>B) = A3B3' + x3\ A2B2' + x3x2\ A1B1' + x3x2x1\ A0B0'$$

In a 4-bit comparator the condition of A<B can be possible in the following four cases:

If **A3 = 0 and B3 = 1**
If **A3 = B3, A2 = 0 and B2 = 1**
If **A3 = B3, A2 = B2, A1 = 0 and B1 = 1**
If **A3 = B3, A2 = B2, A1 = B1, A0 = 0 and B0 =1**
The sequential comparison can be expressed logically as:

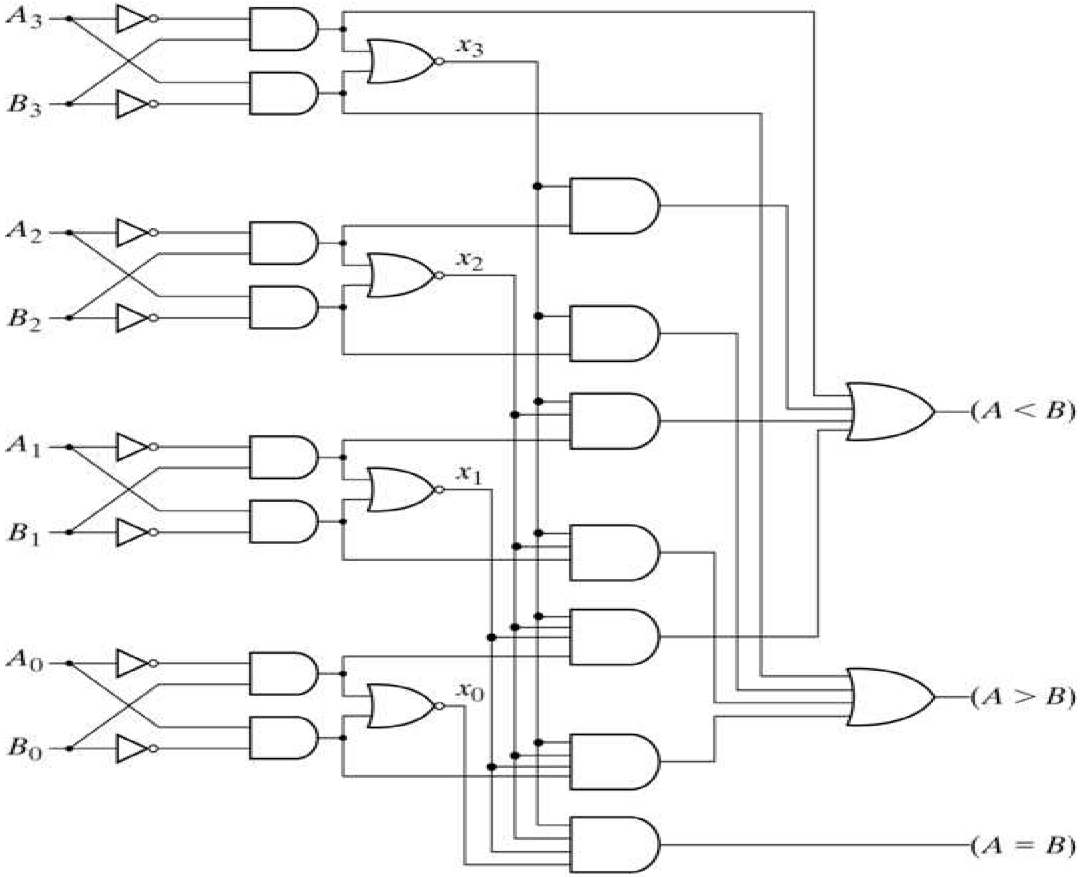
$$(A<B) = A3'B3 + x3\ A2'B2 + x3x2\ A1'B1 + x3x2x1\ A0'B0$$

Logic Diagram

(A = B) = x3 . x2 . x1. x0

(A>B) = A3B3’ + x3 A2B2’ + x3x2 A1B1’ + x3x2x1 A0B0’

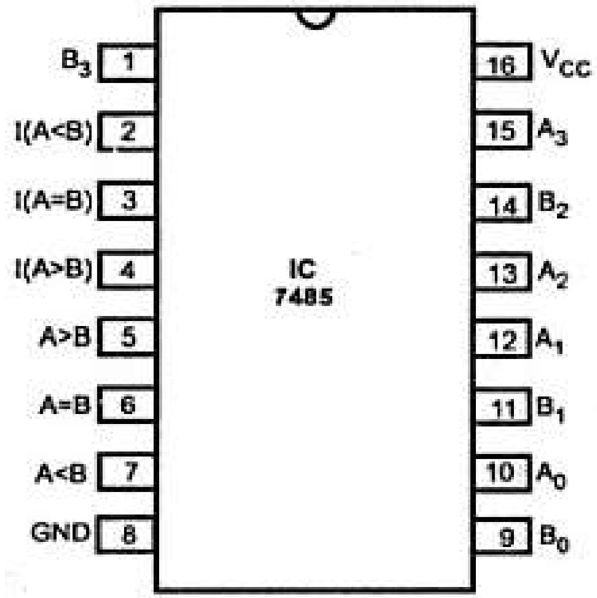
(A<B) = A3’B3 + x3 A2’B2 + x3x2 A1’B1 + x3x2x1 A0’B0



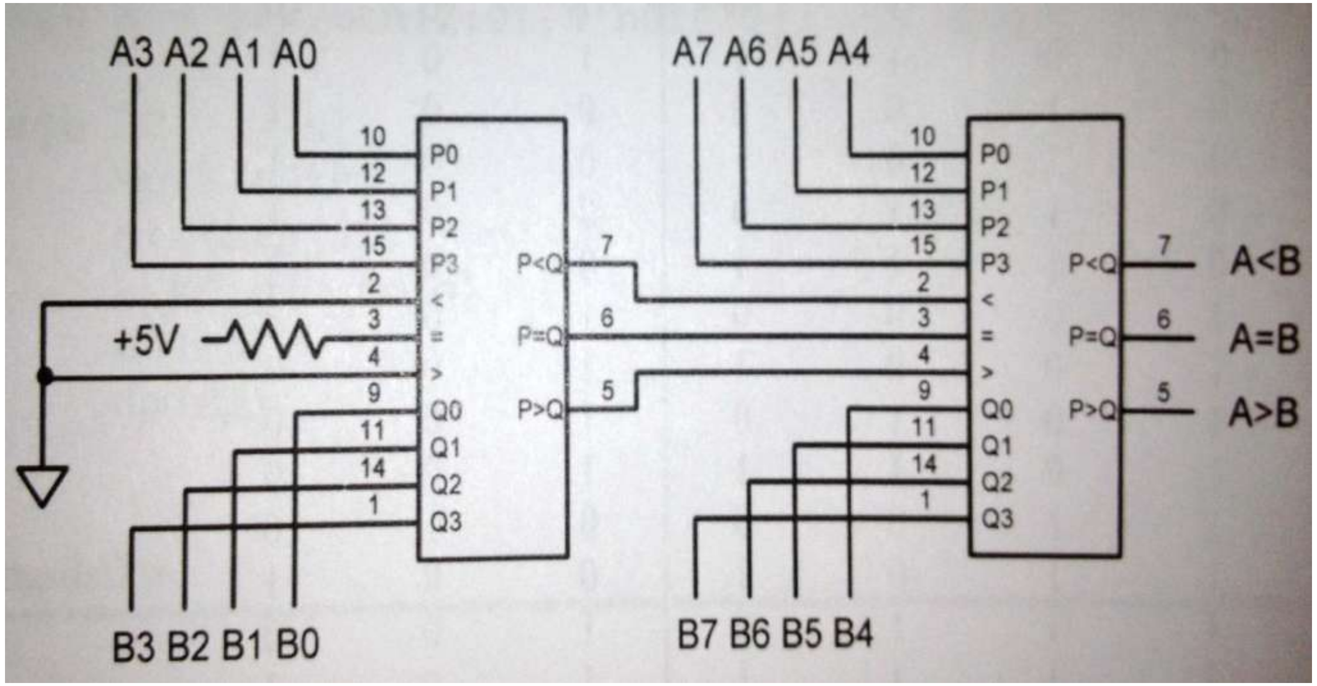
4-Bit Magnitude Comparator

Cascading Comparator

A comparator performing the comparison operation to more than four bits by cascading two or more 4-bit comparators is called cascading comparator.
When two comparators are to be cascaded, the outputs of the **lower-order comparator** are connected to corresponding inputs of the **higher-order comparator**.



Pin diagram (IC 7485)



Applications of Comparators

- Comparators are used in central processing units (CPUs) and microcontrollers (MCUs).
- These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.
- Comparators are also used as process controllers and for Servo motor control.
- Used in password verification and biometric applications.

Error Detection and Correction Codes

- Bits 0 and 1 corresponding to two different range of analog voltages. During transmission of binary data from one system to the other, the noise may also be added. Due to this, there may be errors in the received data at other system.
- That means a bit 0 may change to 1 or a bit 1 may change to 0. We can't avoid the interference of noise. But, we can get back the original data first by detecting whether any errors present and then correcting those errors.
- For this purpose, we can use the following codes.
  - Error detection codes
  - Error correction codes

- **Error detection codes** − are used to detect the errors present in the received data. These codes contain some bits, which are included to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data.

  Example − Parity code, Hamming code, CRC code etc.

- **Error correction codes** − are used to correct the errors present in the received data so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes.

  It also detects the error.

  Example − Hamming code, CRC code etc.

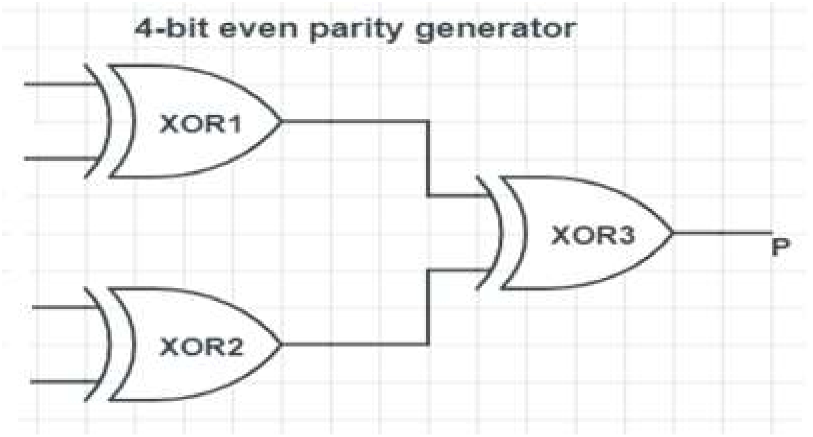
- Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

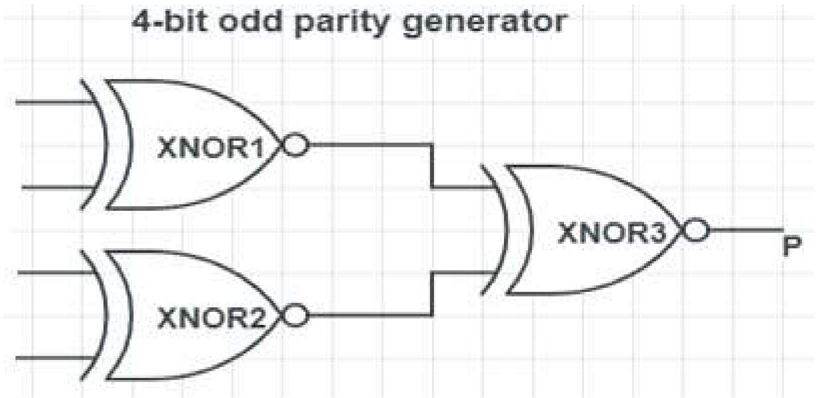
Parity Code Method

- A parity bit is an extra bit included in binary message to make total number of 1’s either odd or even.
- Parity word denotes number of 1’s in a binary string.
- There are two parity system-Even Parity and Odd Parity.
- **In even parity system** 1 is appended to binary string if there is an odd number of 1’s in string otherwise 0 is appended to make total even number of 1’s.
- **In odd parity system**, 1 is appended to binary string if there is even a number of 1’s to make an odd number of 1’s.
- The receiver knows that whether sender is an odd parity generator or even parity generator.
- Suppose if sender is an odd parity generator then there must be an odd number of 1’s in received binary string.
- If an error occurs to a single bit that is either bit is changed to 1 to 0 or 0 to 1, received binary bit will have an even number of 1’s which will indicate an error.

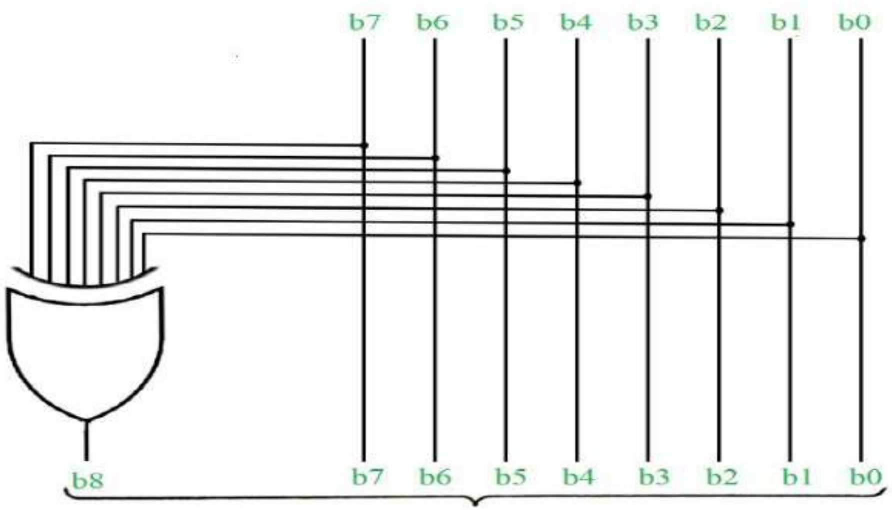
Parity Generator

D_3	D_2	D_1	D_0	Even-parity P	Odd-parity P
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

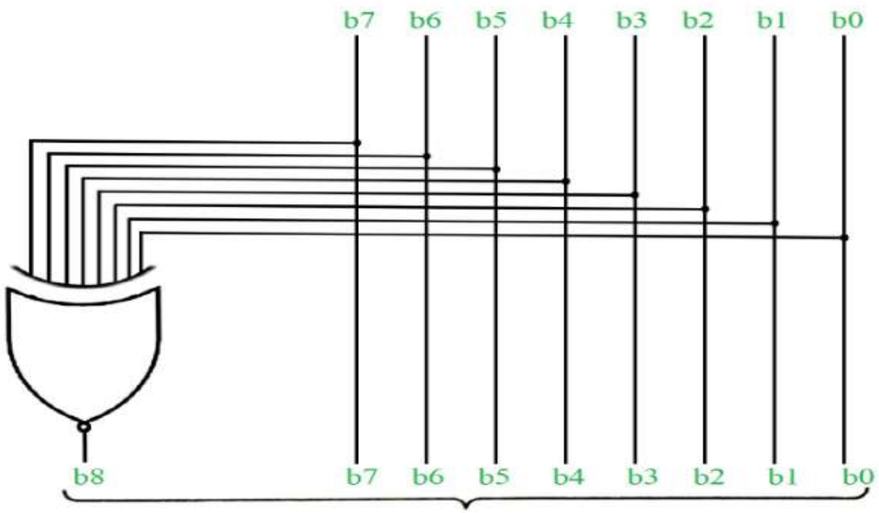




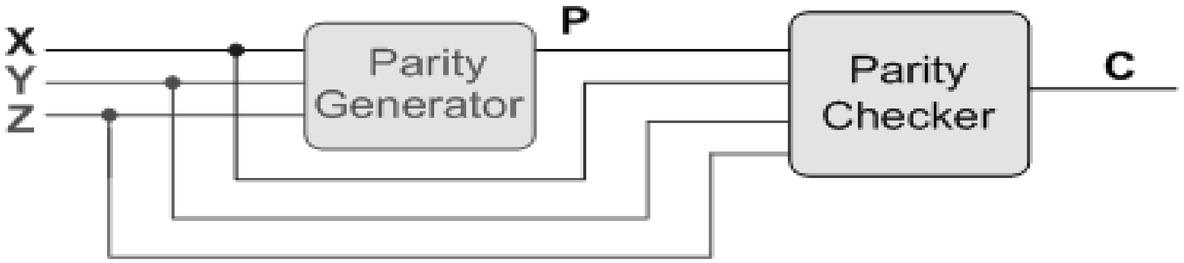
Parity Generator and Checker



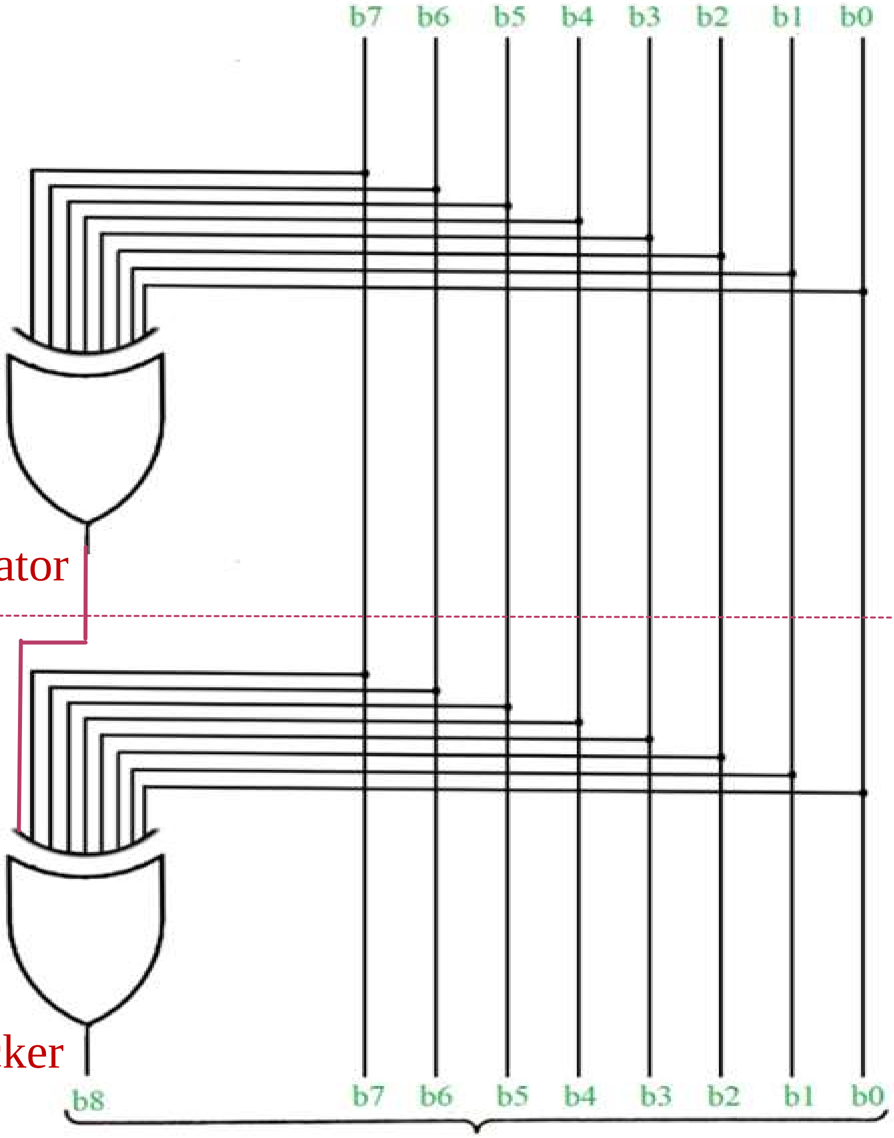
msg transmitted
Even Parity Geneartor



msg transmitted
Odd Parity Generator



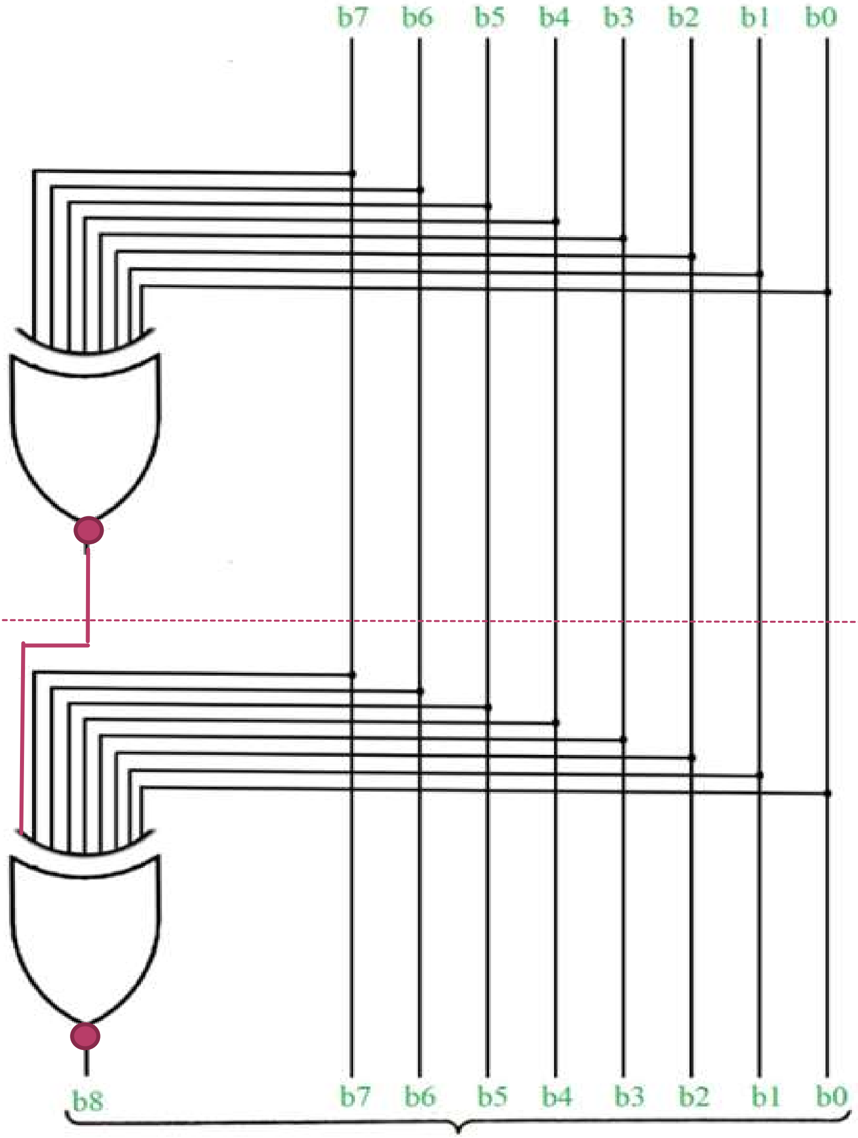
Parity Generator and Checker



Parity Generator

Parity Checker

Even Parity Generator and Checker



Parity Generator

Parity Checker

Odd Parity Generator and Checker

- The limitation of this method is that only error in a single bit would be identified.
- It does not tell which bit is incorrect .
- It also can not correct the incorrect bit.
- To overcome this another code called Hamming Code is used to detect an error.
- It indicates which bit is in error.
- It also correct that error.
- Because of this Hamming Code is called as self correcting code.

Hamming Code

- It was developed by R.W. Hamming for error correction.
- Hamming code is useful for both detection and correction of error present in the received data.
- This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.
- The **minimum value of 'k'** for which the following relation is correct is nothing but the required number of parity bits.

  $2k \geq n + k + 1$ Where, ‘n’ is the number of bits in the binary code, ‘k’ is the number of parity bits

- Therefore, the number of bits in the Hamming code is equal to n + k.
- Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.

- Let us find the Hamming code for 4-bit binary code
- We can find the required number of parity bits by using the following mathematical relation.
- $2k \geq n + k +1$
- Substitute, $n = 4$ in the above mathematical relation.
- $\Rightarrow 2k \geq 4 + k + 1 \Rightarrow 2k \geq 5 + k$
- The minimum value of k that satisfied the above relation is 3. Hence, we require 3 parity bits.
- Therefore, the number of bits in Hamming code will be 7, since there are 4 bits in binary code and 3 parity bits.

- We have to place the parity bits and bits of binary code in the Hamming code as shown below.
- Now the Hamming code word format will be ***d7 d6 d5 p4 d3 p2 p1*****,** where ***'d'*** represents the data bit and ***'p'*** represents the parity bit.
- The parity bit *p1, p2* and *p4* are assigned values by the following three parity relations.
- $p1 = d7 \oplus d5 \oplus d3$ $p2 = d7 \oplus d6 \oplus d3$ $p4 = d7 \oplus d6 \oplus d5$

Example: 1

Construct an even parity seven bit Hamming code for a word 1011.

d7	*d6*	*d5*	*p4*	*d3*	*p2*	*p1*
1	*0*	*1*	*?*	*1*	*?*	*?*

From first relation to have even parity ***p1*** should be 1. From second relation to have even parity ***p2*** should be 0. From third relation to have even parity ***p4*** should be 0. So, the final Hamming code is ***1 0 1 0 1 0 1.***

- For finding the position of error the following relations are to be followed.
- $x = d7 \oplus d5 \oplus d3 \oplus p1$ $y = d7 \oplus d6 \oplus d3 \oplus p2$ $z = d7 \oplus d6 \oplus d5 \oplus p4$
- The parity check may be even parity or odd parity
- If parity relation is satisfied then ***x*** or ***y*** or ***z*** equal to ***0***, otherwise ***1***.

Example: 2

The Hamming code is received 1010001. What was the correct code transmitted.

The code received	*d7*	*d6*	*d5*	*p4*	*d3*	*p2*	*p1*
	1	*0*	*1*	*0*	*0*	*0*	*1*

Applying first parity relation ***x = 1.*** Applying second parity relation ***y = 1.*** Applying third parity relation ***z = 0.***

So, ***z y x = 011,*** which is equal to *3,* that is, third data bit is erroneous one and should be corrected as *1* instead of *0*. Now, the correct code is ***1 0 1 0 1 0 1.***

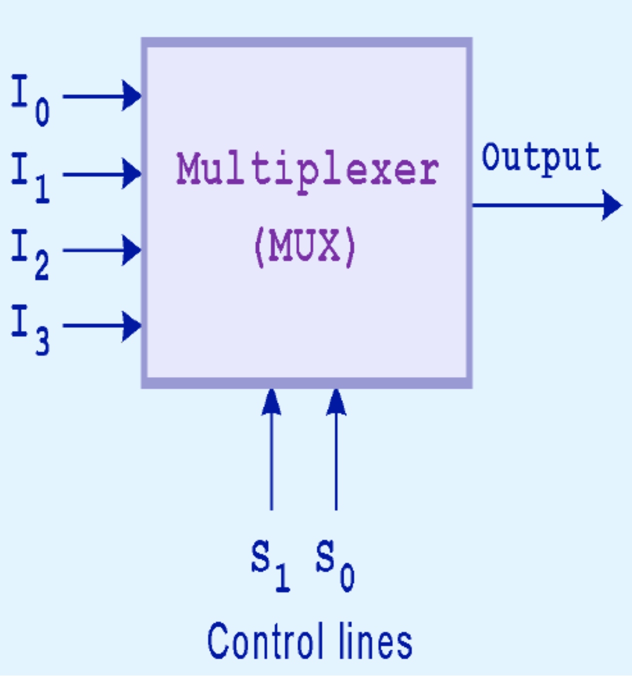
Combinational Circuits

Overview

- **Multiplexer**
- **De-Multiplexer**
- **Decoder**
- **Encoder**
- **Priority Encoder**
- **BCD to Seven Segment Display**

Multiplexer

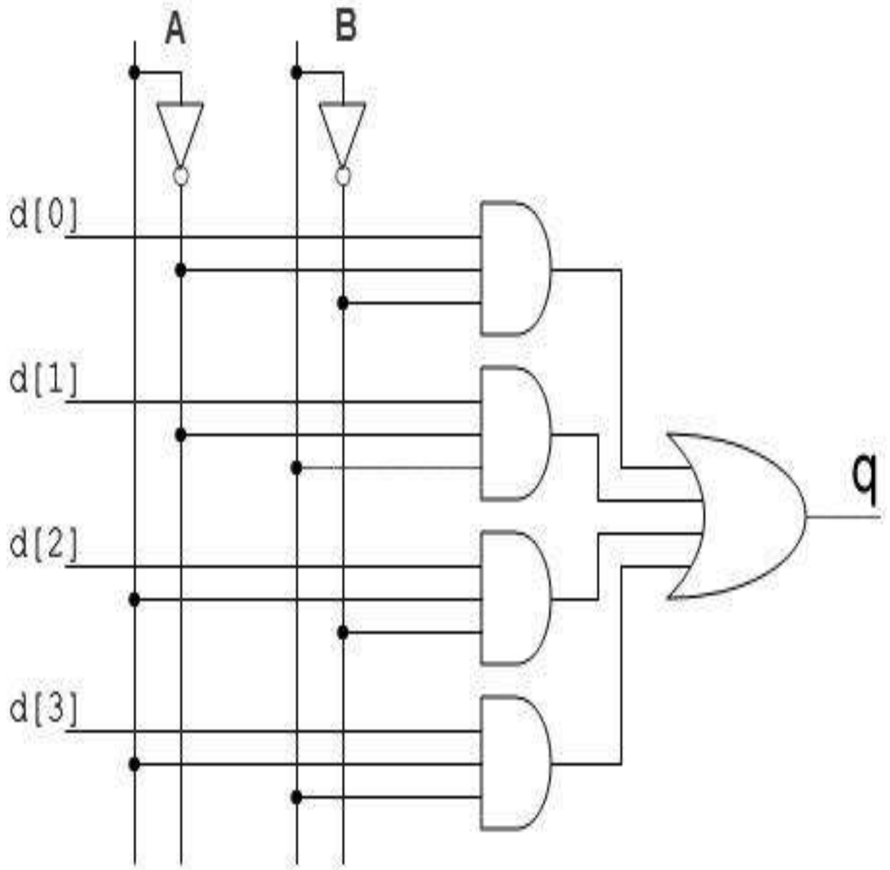
- A Multiplexer or Mux is a device that has many inputs and a single output.
- It selects a single input to the output from several inputs.
- The particular input chosen for output is determined by the value of the multiplexer's control lines.
- To be able to select among *n* inputs, $\log_2 n$ control lines are needed.
- A multiplexer is also called as a data selector.
- The main purpose of Mux is to perform high speed switching.
- In analog applications, these are made up of transistor switches and relays, whereas in digital applications, these are made up of logic gates.



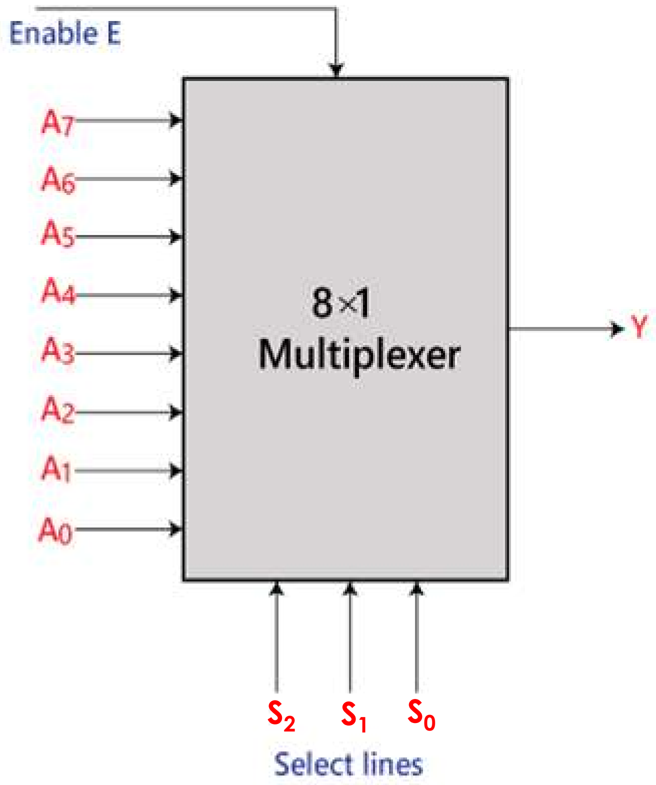
Block diagram of Multiplexer

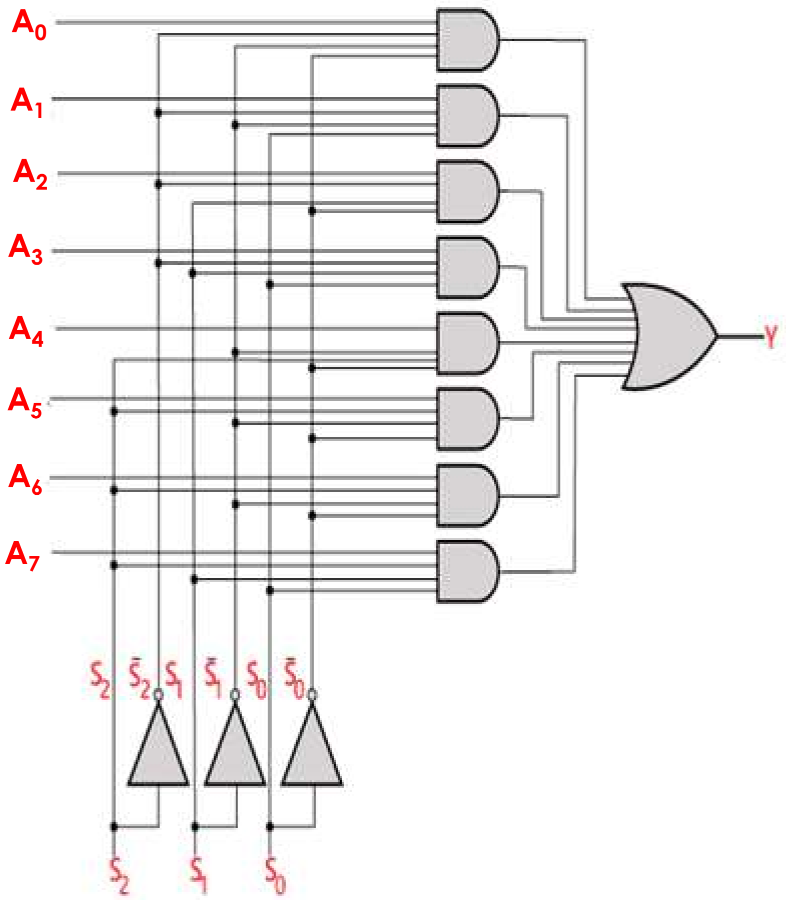
4-to-1 multiplexer

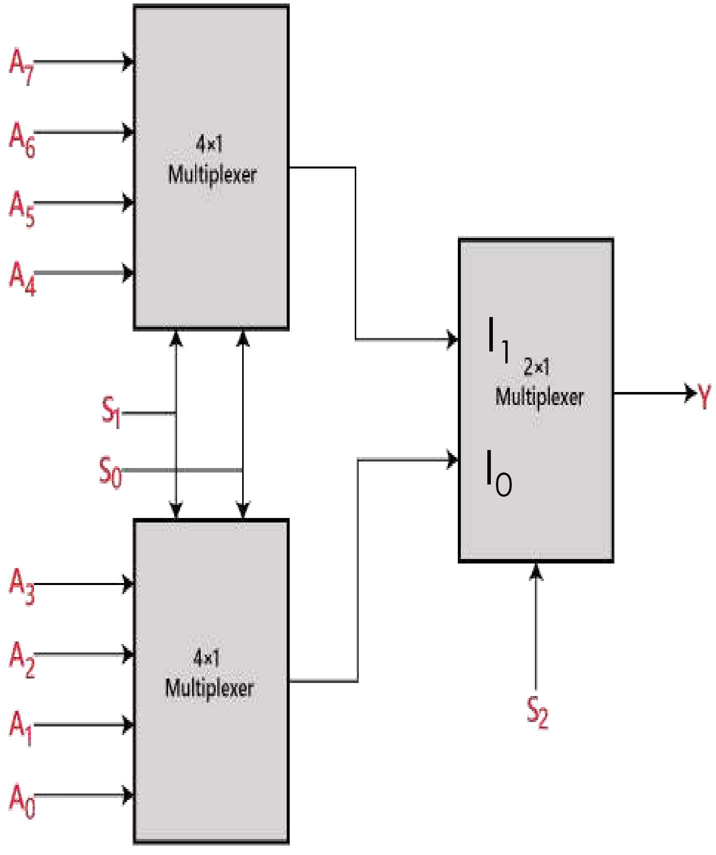
- This is what a 4-to-1 multiplexer looks like on the inside.
- The 4X1 multiplexer comprises 4-input bits, 1-output bit, and 2- control bits.
- The control bit AB decides which of the i/p data bit should transmit the output.
- For example, when the control bits AB =00, then the higher AND gate are allowed while remaining AND gates are restricted. Thus, data input d0 is transmitted to the output 'q"



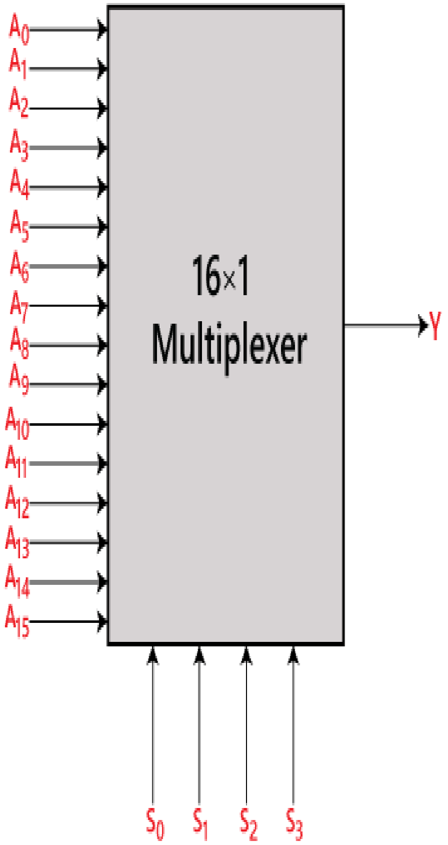
8-to-1 multiplexer

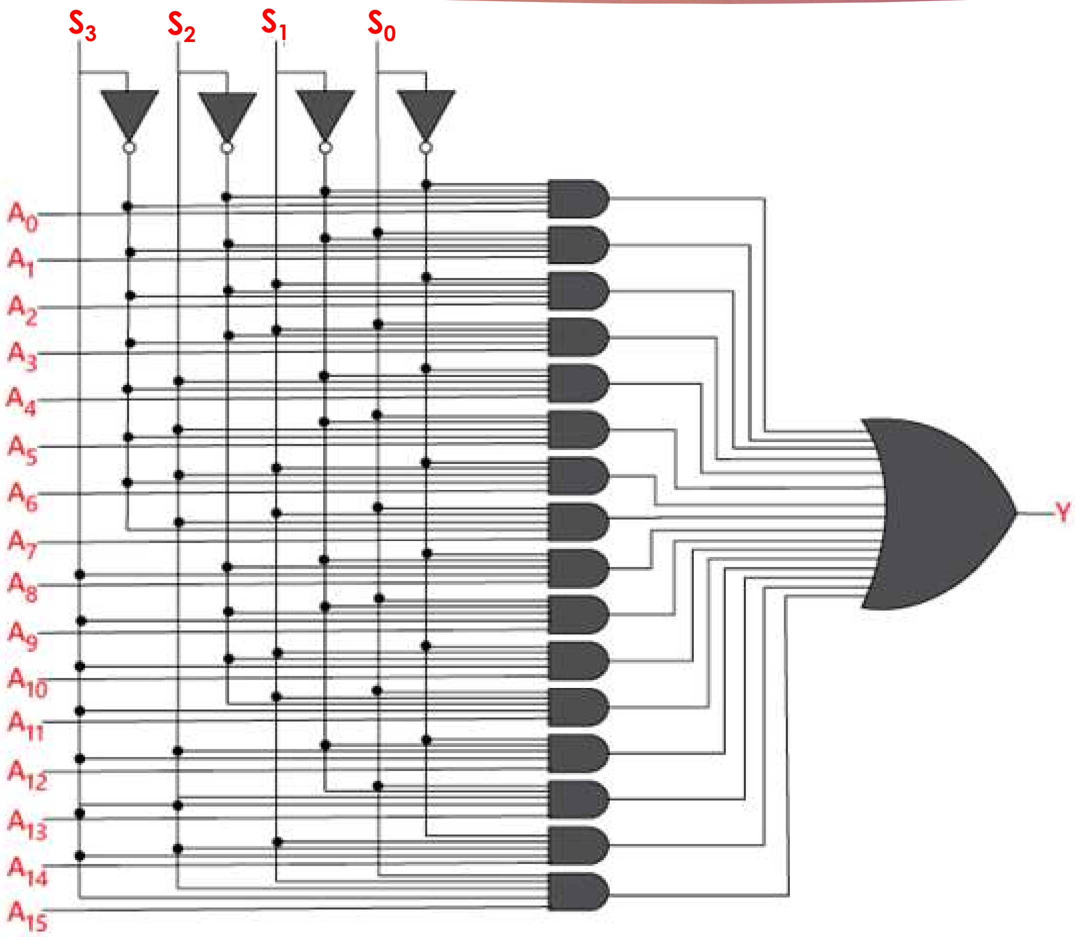


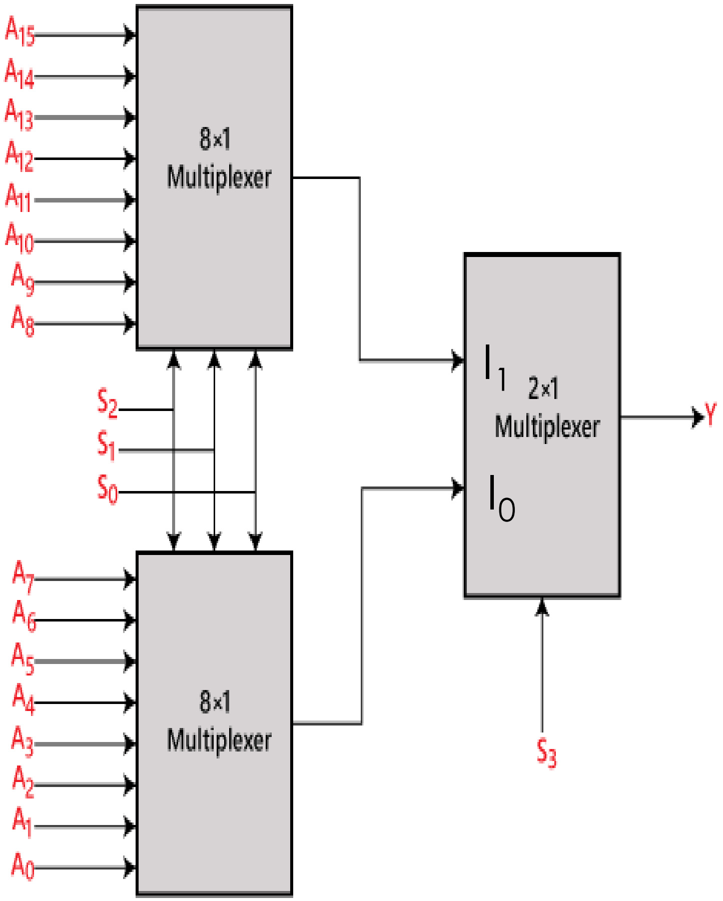




16-to-1 multiplexer







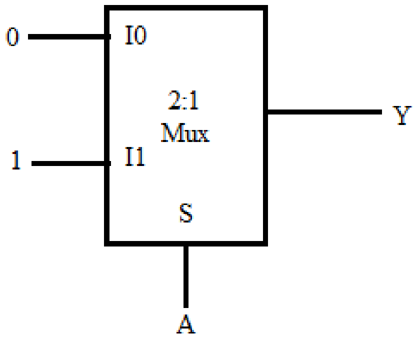
Applications

- A Multiplexer is used in various applications wherein multiple data can be transmitted using a single line.
- A Multiplexer is used to increase the efficiency of the communication system by allowing the transmission of data, such as audio & video data from different channels via cables and single lines.
- A Multiplexer is used in computer memory to decrease the number of copper lines necessary to connect the memory to other parts of the computer.
- A multiplexer is used in telephone networks to integrate the multiple audio signals on a single line of transmission.
- A Multiplexer is used to transmit the data signals from the computer system of a satellite to the ground system by using a GSM (Global System for Mobile communication) communication.

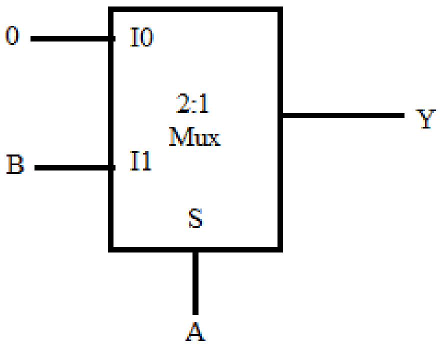
MUX as Universal Logic Circuit

Implemented by MUX + Equation

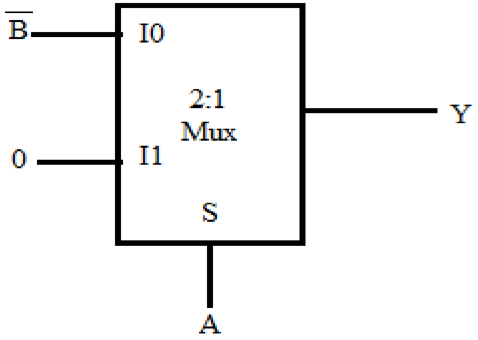
Y=output = A

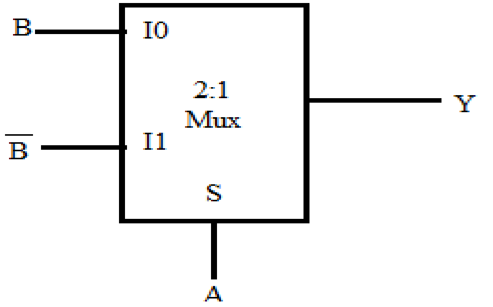


Y=A.B



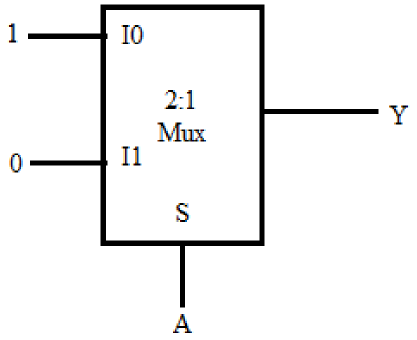
Y=(A+B)'



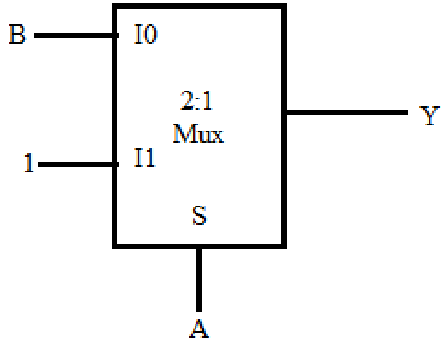


$Y = A \oplus B = \bar{A}B + A\bar{B}$

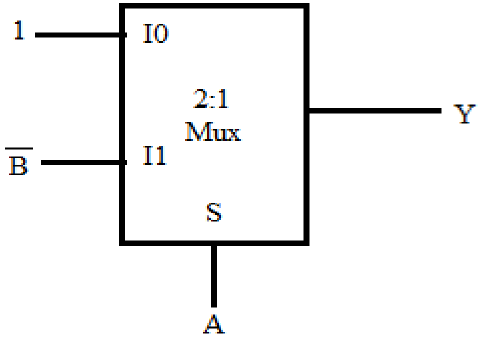
Y=A'

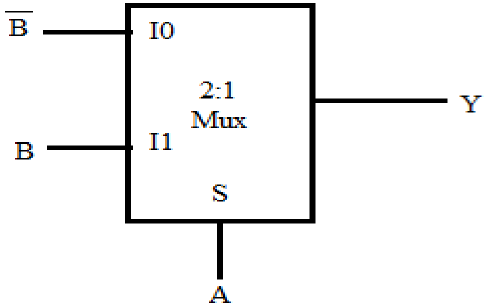


Y=A+B



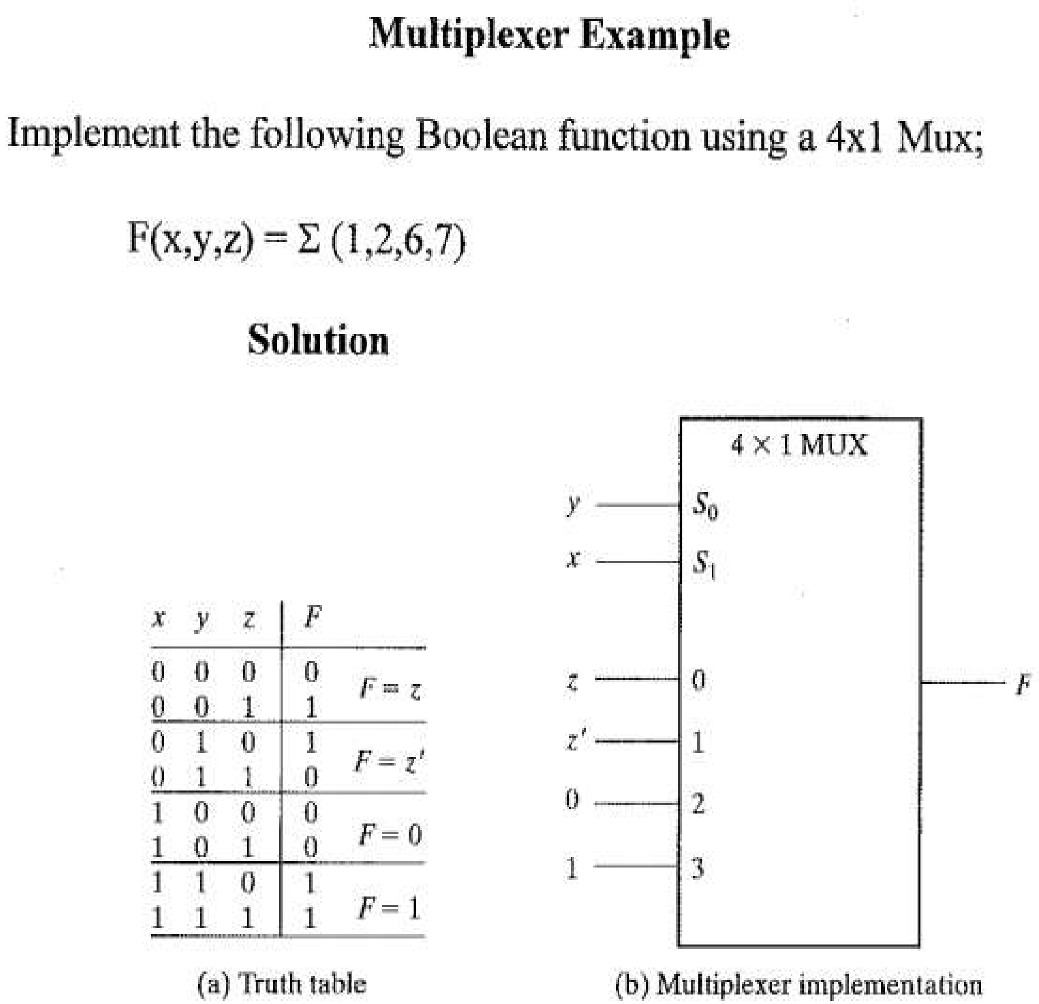
Y=(A.B)'





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

Boolean function implementation using Mux



Multiplexer Example

Implement the following Boolean function using a 4x1 Mux;

$F(x,y,z) = \Sigma\,(1,2,6,7)$

Solution

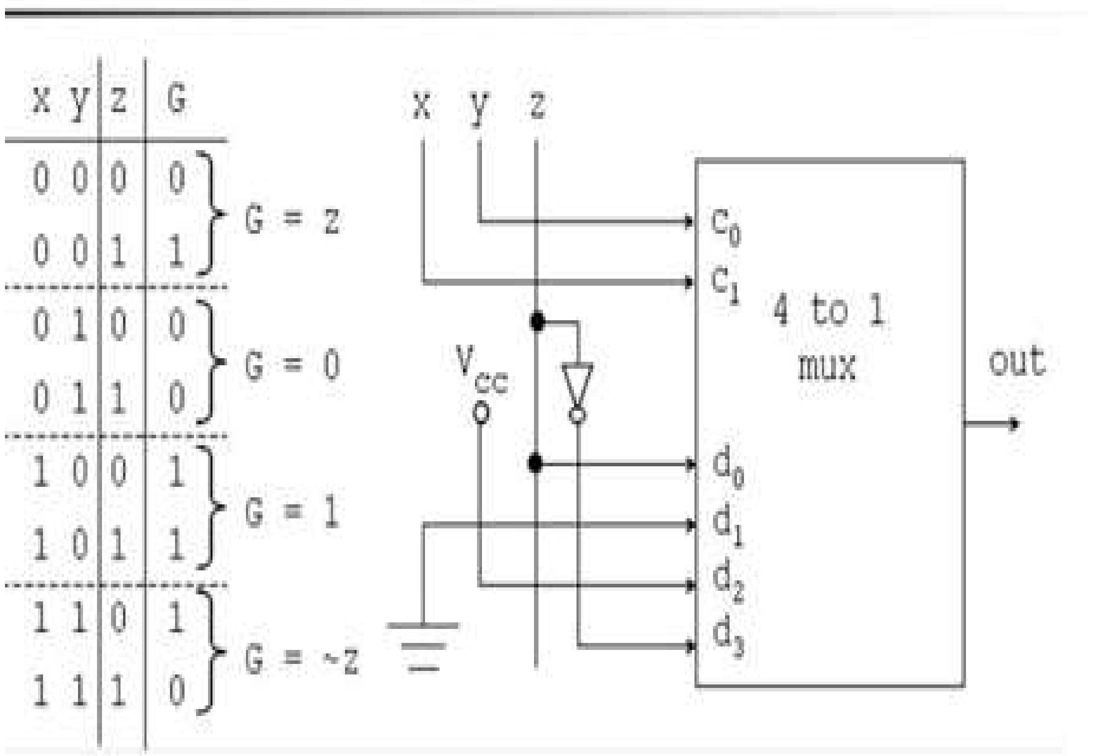
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

4 × 1 MUX
y
S_0
x
S_1
z
0
z'
1
0
2
1
3
F

(b) Multiplexer implementation

$G(x,y,z) = m(1, 4, 5, 6)$



x	y	z	G	
0	0	0	0	G = z
0	0	1	1	
0	1	0	0	G = 0
0	1	1	0	
1	0	0	1	G = 1
1	0	1	1	
1	1	0	1	G = ~z
1	1	1	0	

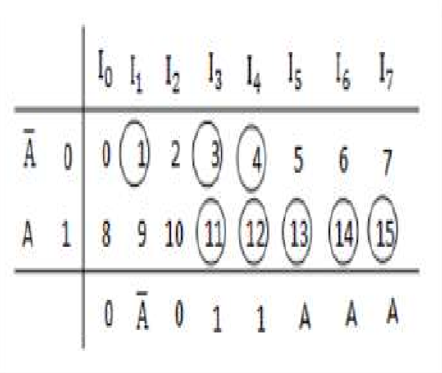
Example: Implementation of given function using 8 to 1 multiplexer

F(A,B,C,D) = Σ (1,3,4,11,12,13,14,15)

Solution.

- Total number of variable n = 4 (A,B,C,D)
- Number of select lines: n-1= 3 (B, C, D)
- The given function has 4 variable, so 16 possible minterms (0 – 15) are entered in the implementation table.
- All the minterms are divided into 2 groups
  - The first group (0-7) minterms are entered in the first row (Variable A =0)
  - The second group (8–15) minterms are entered in the second row (Variable A= 1)
- Circle the minterm number as per function, which you have to implement (in this case it's 1,3,4,11,12,13,14,15)
- Find out the multiplexer input as per above given steps.

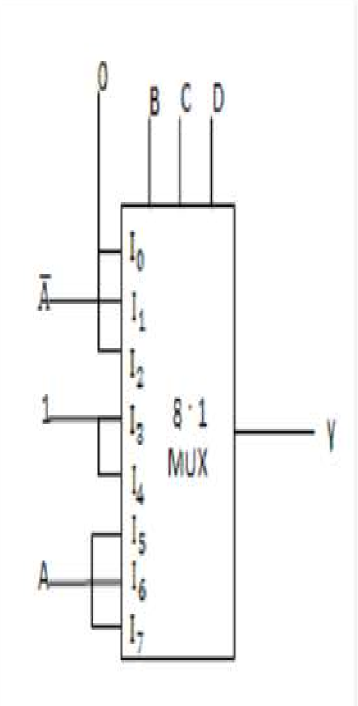
Implementation Table



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

Given multiplexer is 8:1

Logic diagram



Rules:

- If two min-terms are not circled in a coloumn, apply 0 to Mux input.
- If two min-terms are circled in a coloumn, apply 1 to Mux input.
- If bottom one is circled and top one is not circled in a column, apply A to Mux input.
- If bottom one is not circled and top one is circled in a column, apply A' to Mux input.

Example

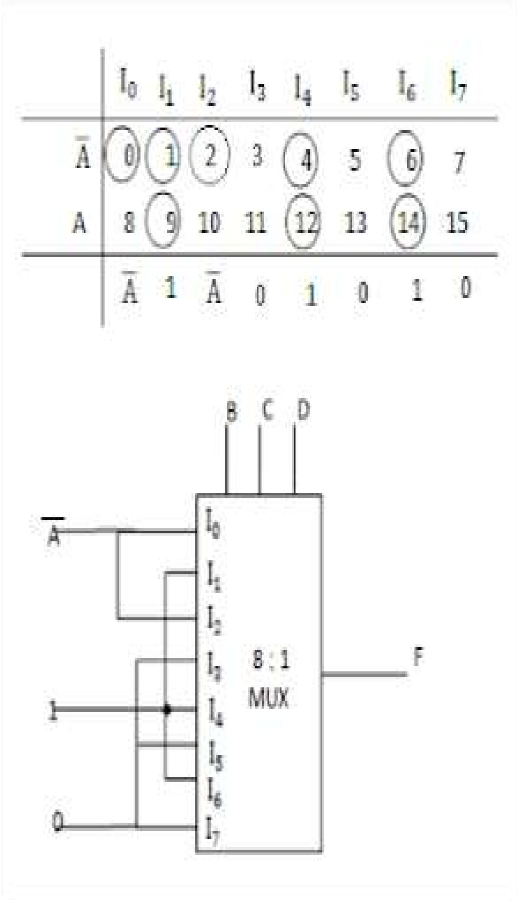
Implement the following Boolean function using 8 : 1 MUX

F(A,B,C,D) = Σ m(0,1,2,4,6,9,12,14)

Solution.

Select lines are B, C and D

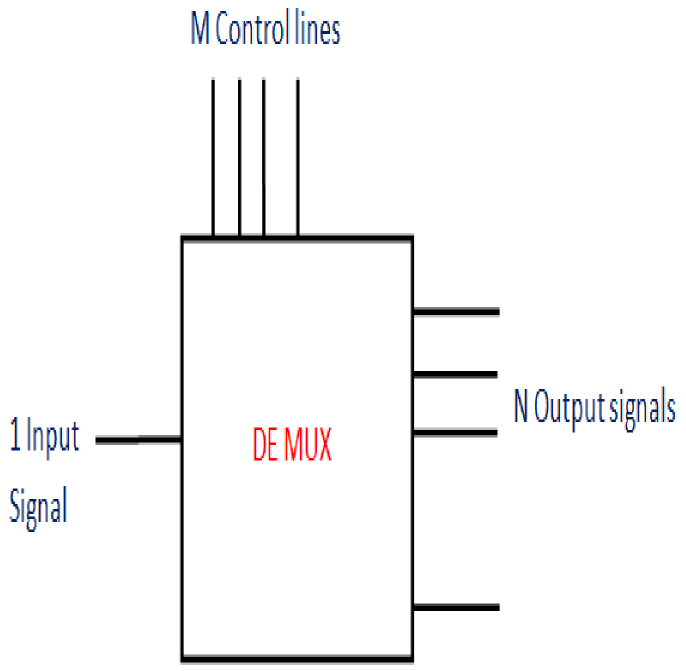
Follow all the steps as per above points.



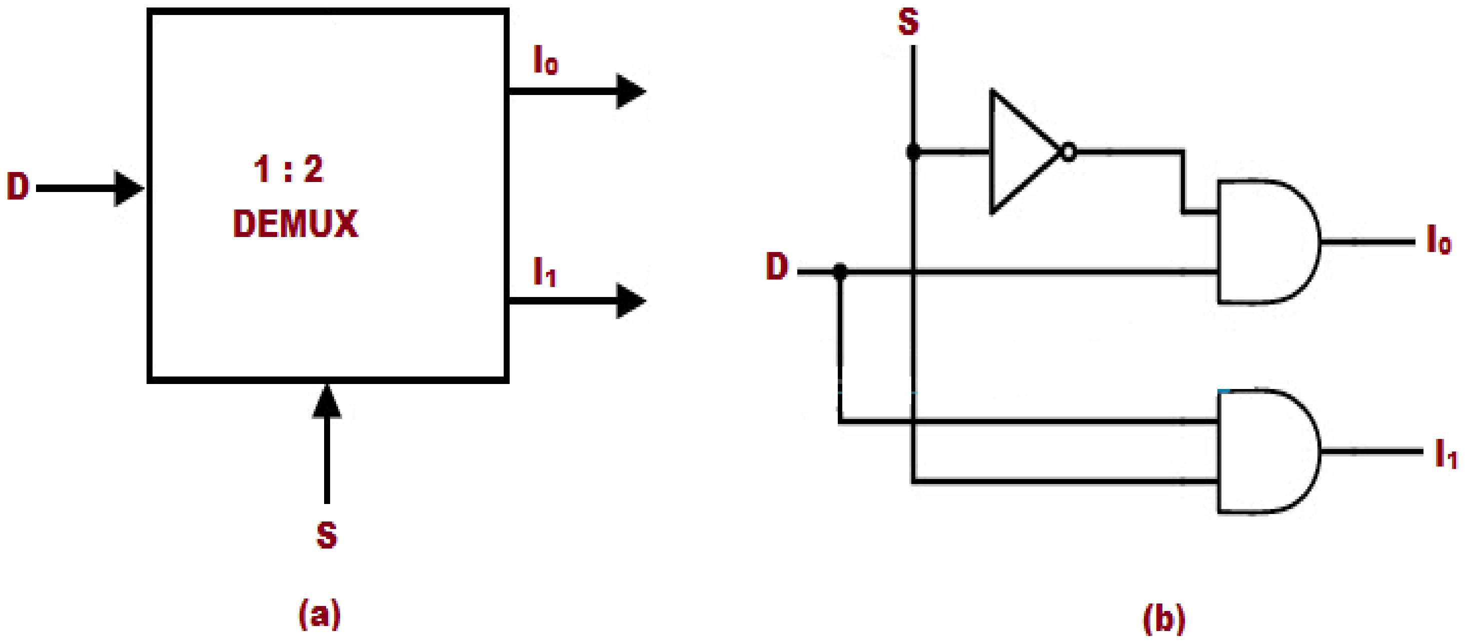
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\bar{A}$	⓪	①	②	3	④	5	⑥	7
A	8	⑨	10	11	⑫	13	⑭	15
	$\bar{A}$	1	$\bar{A}$	0	1	0	1	0

Demultiplexer

- **A Demultiplexer** or **Demux** is a circuit which can distribute or deliver multiple outputs from a single input.
- It can perform as single input many output switch.
- The output lines of demultiplexer are ‘N’ in number, select line number is ‘M’ and N = 2^M.
- The control signal or select input code decides the output line to which the input has to be transmitted.
- It is also called as **Data distributor.**
- There are several types of Demultiplexers
  - 1:2 Demultiplexer or 1-to-2 Demultiplexer
  - 1:4 Demultiplexer
  - 1:8 Demultiplexer
  - 1:16 Demultiplexer

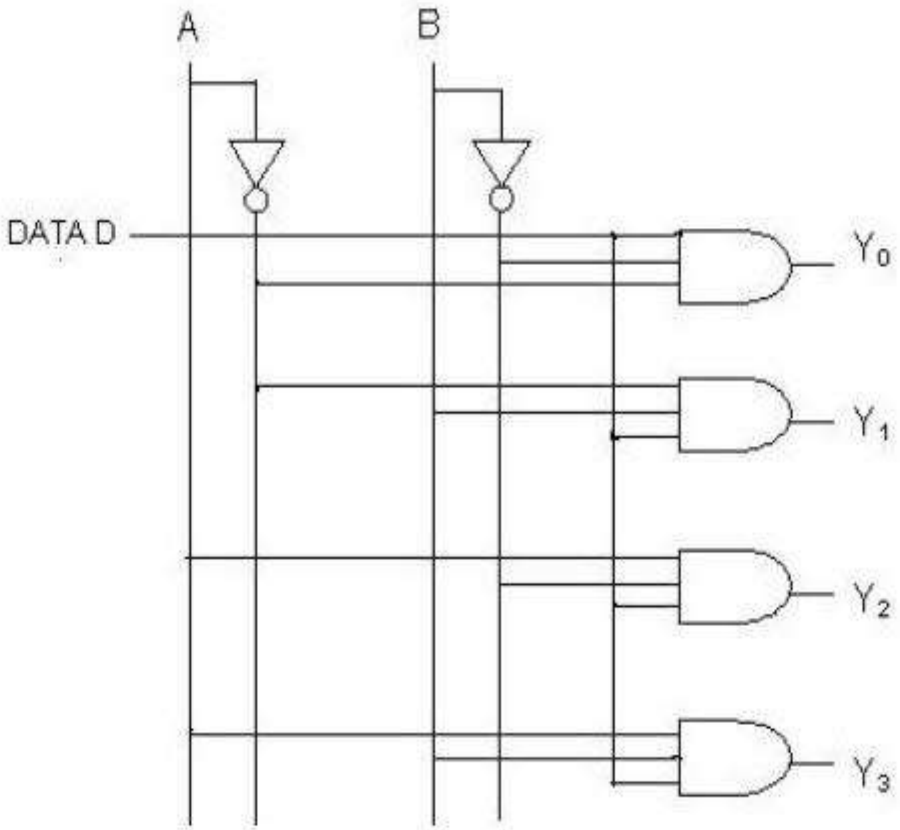


1:2 Demultiplexer

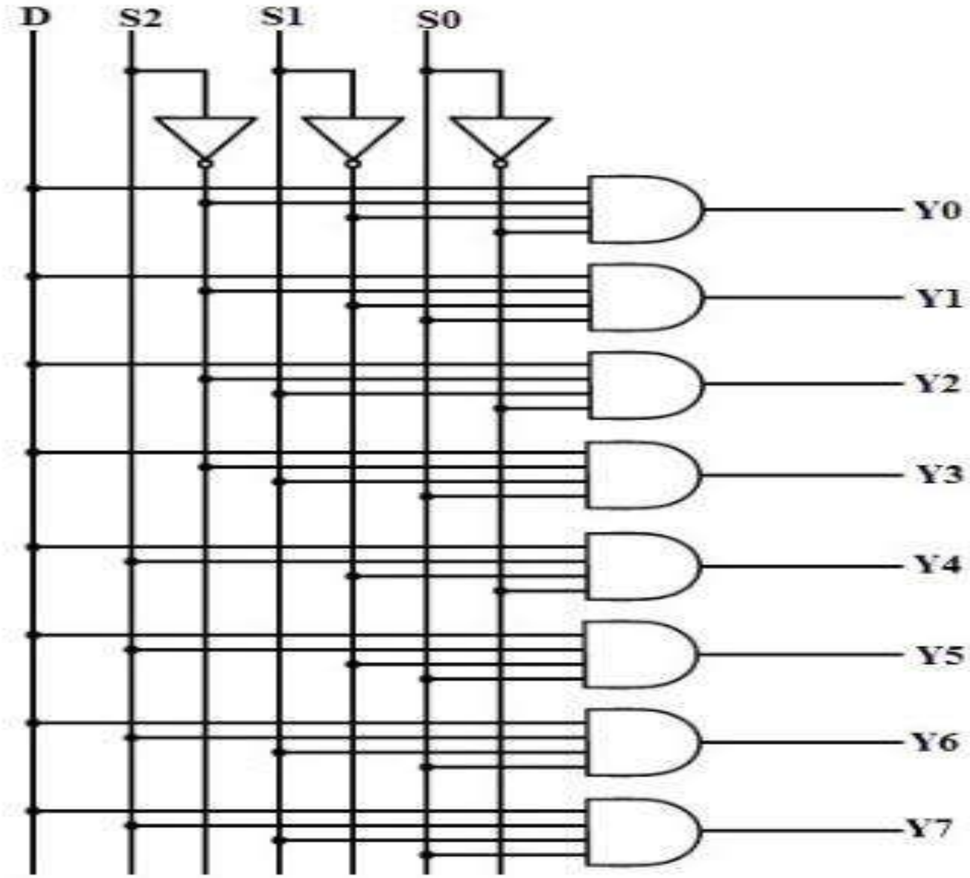


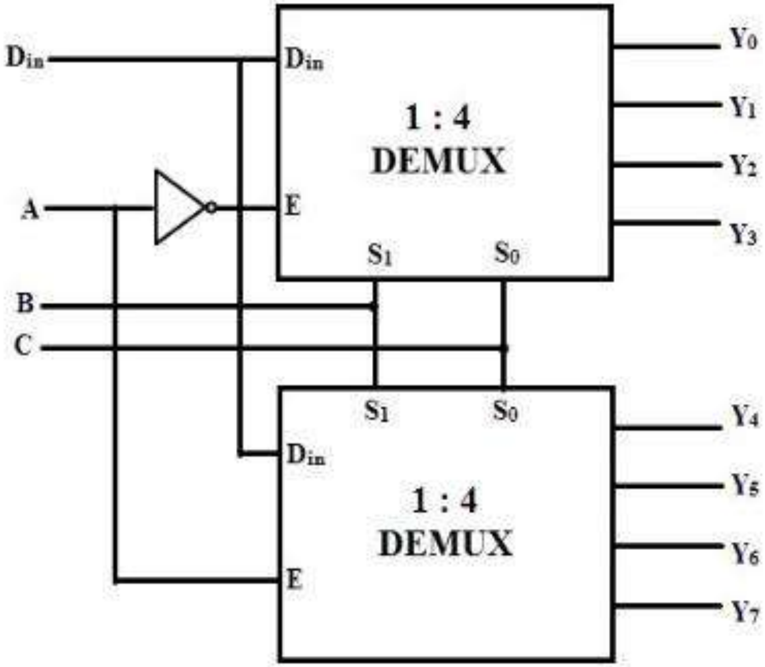
1:4 Demultiplexer

- The input bit is Data D with two select lines A and B.
- The input bit D is transmitted to four output bits Y0, Y1, Y2, and Y3.
- When AB is 00 the upper AND gate is enabled while the other AND gates are disabled. Thus, the data is transmitted to Y0.
- If D is low, then Y0 is low and if D is high, Y0 is high. The value of Y0 depends on the value of D.



1:8 Demultiplexer





Applications of Demultiplexer (Demux)

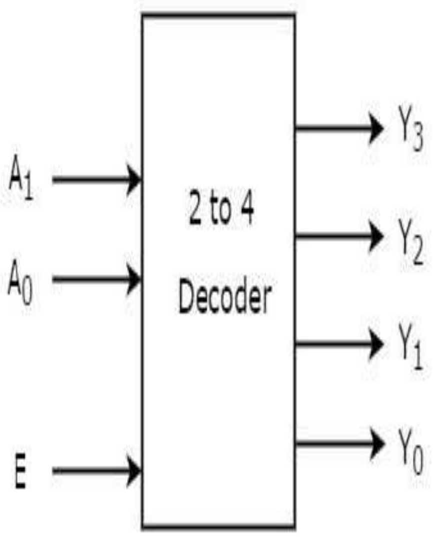
- Demux are widely used in microprocessor, computers and digital electronics.
- Demultiplexer and Multiplexer both are used in communication systems to carry multiple data signals (i.e. audio, video etc) using single line for transmission.
- In Arithmetic logic unit (ALU), the output of ALU can be stored in storage unit (multiple registers) by using Demultiplexer.

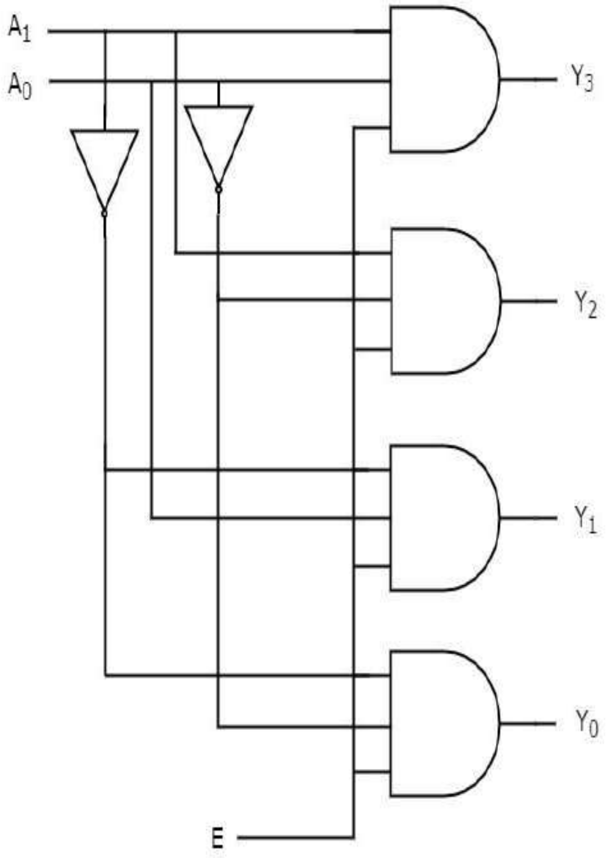
It is also used

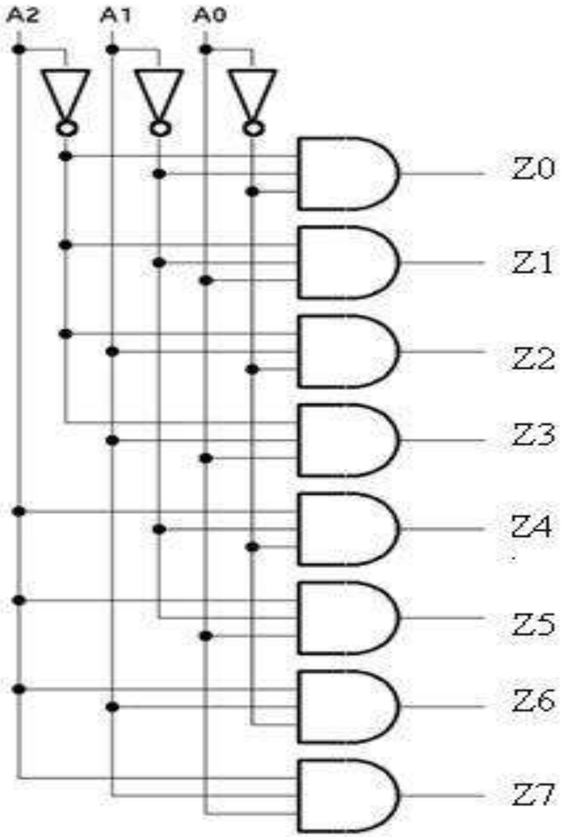
- To enable the different rows of memory chips depends on the address. Also to chose different banks of memory.
- To enable different functional unit in the system
- To select different IO devices for data transfer
- Data acquisition systems
- Automatic test equipment systems
- Security monitoring systems

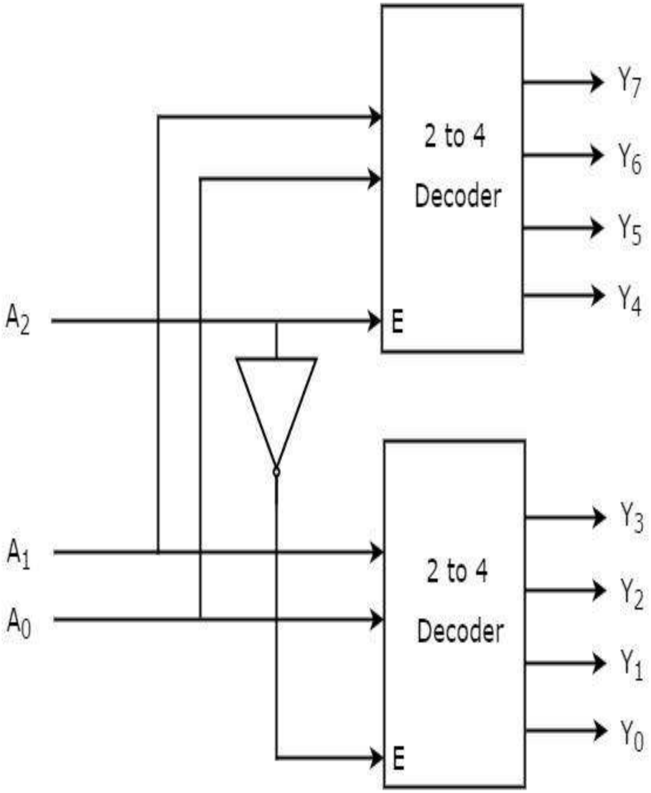
Decoder

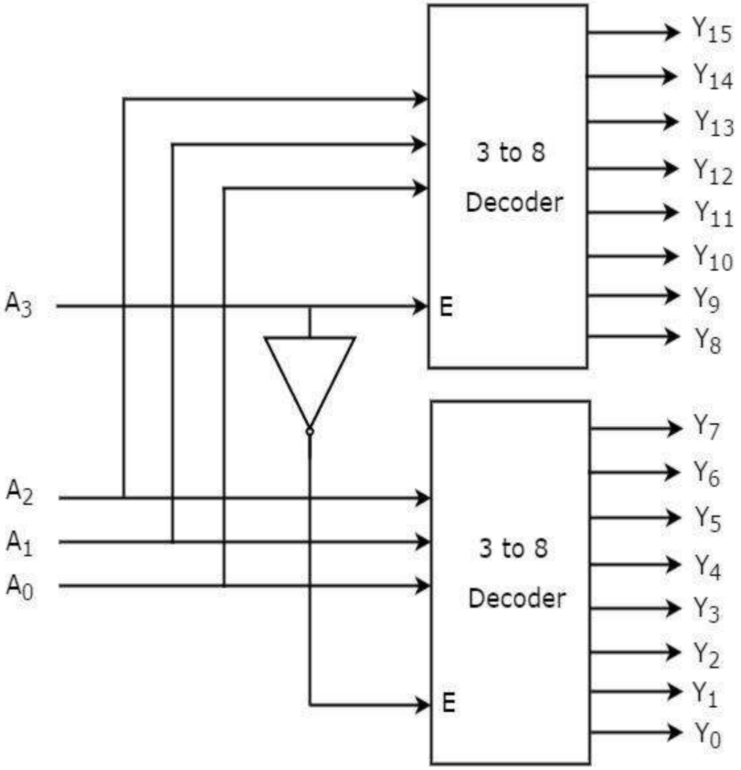
- **Decoder** is a combinational logic circuit whose purpose is to decode the information.
- It is comprised of n number of input lines and 2^n number of output lines.
- In every probable input condition, among the various output signals, only one output signal will produce the logic one.
- So, this is n-to-2^n decoder, where n input lines and 2^n output lines.
- Generally, there are 3 types of line decoders (2-to-4, 3-to-8 and 4-to-16).






A2
A1
A0
Z0
Z1
Z2
Z3
Z4
Z5
Z6
Z7


2 to 4
Decoder
E
Y7
Y6
Y5
Y4
A2
A1
A0
2 to 4
Decoder
E
Y3
Y2
Y1
Y0


3 to 8
Decoder
E
Y15
Y14
Y13
Y12
Y11
Y10
Y9
Y8
A3
A2
A1
A0
3 to 8
Decoder
E
Y7
Y6
Y5
Y4
Y3
Y2
Y1
Y0

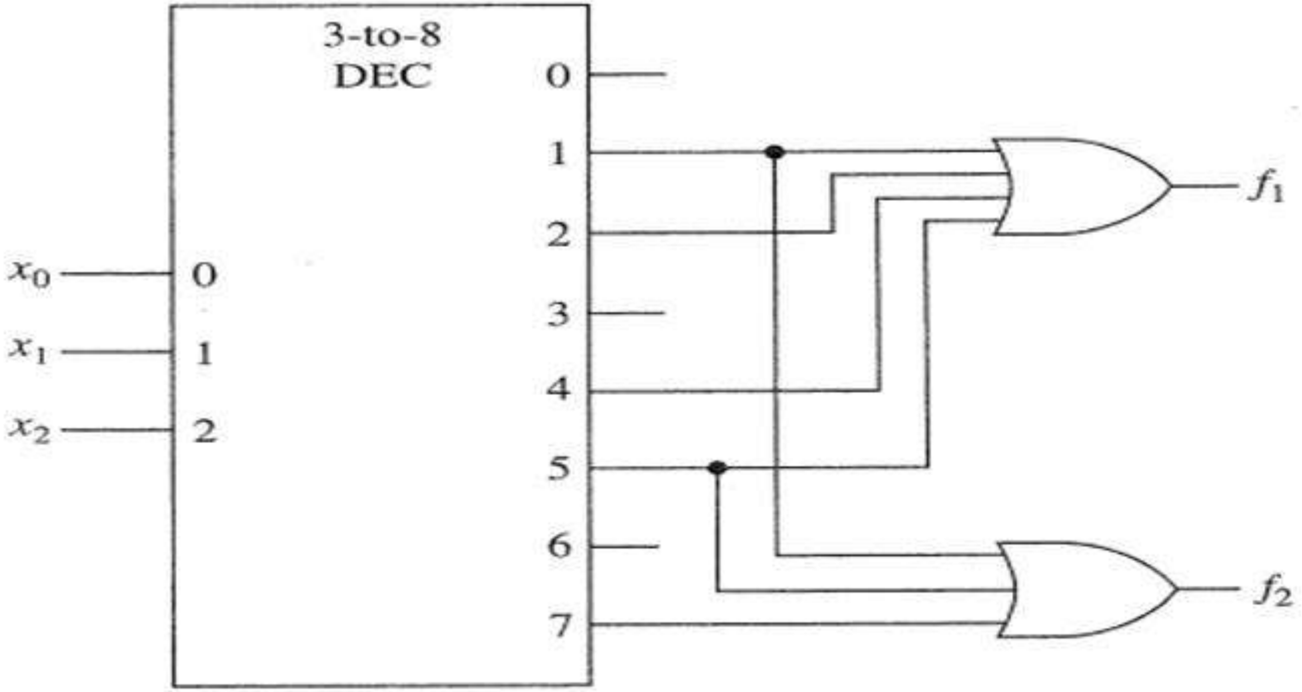
Logic Design Using Decoders

- An n-to-2^n line decoder is a minterm generator.
- By using or-gates in conjunction with an n-to-2^n line decoder, realizations of Boolean functions are possible.
- Do not correspond to minimal sum-of-products.
- Are simple to produce. Particularly convenient when several functions of the same variable have to be realized.

Realization of the Boolean expressions

$f_1(x_2,x_1,x_0) = \Sigma m(1,2,4,5)$ and

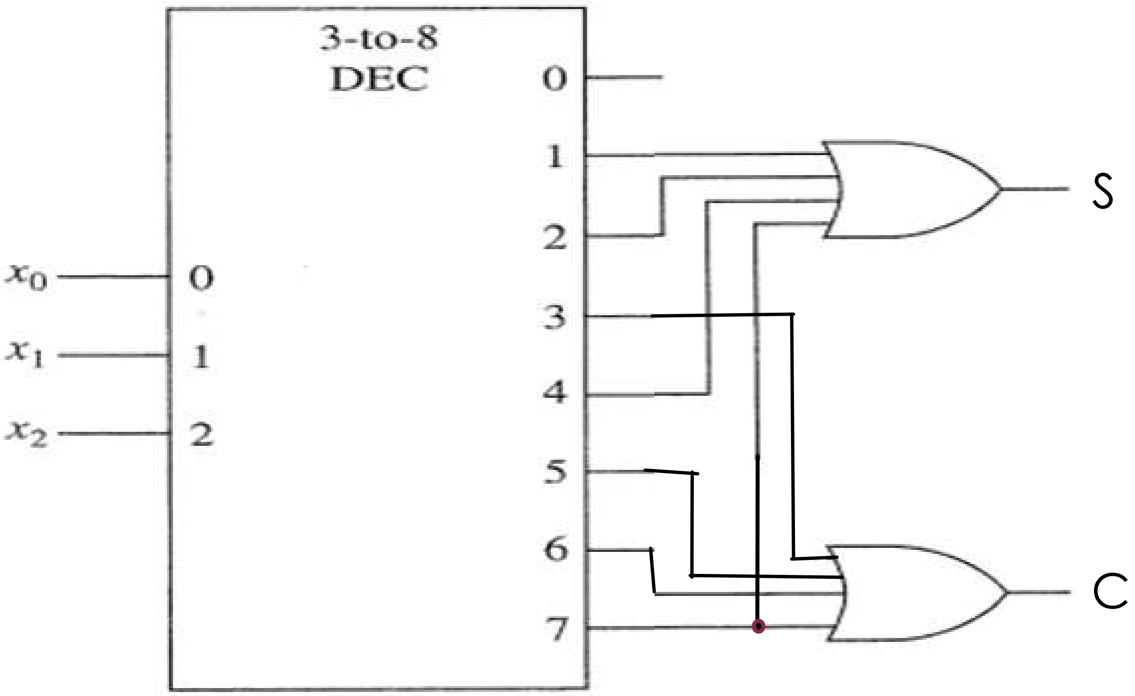
$f_2(x_2,x_1,x_0) = \Sigma m(1,5,7)$



Implementation of a Full Adder circuit using Decoder.

$S(x_0, x_1, x_2) = \sum(1, 2, 4, 7)$

$C(x_0, x_1, x_2) = \sum(3, 5, 6, 7)$

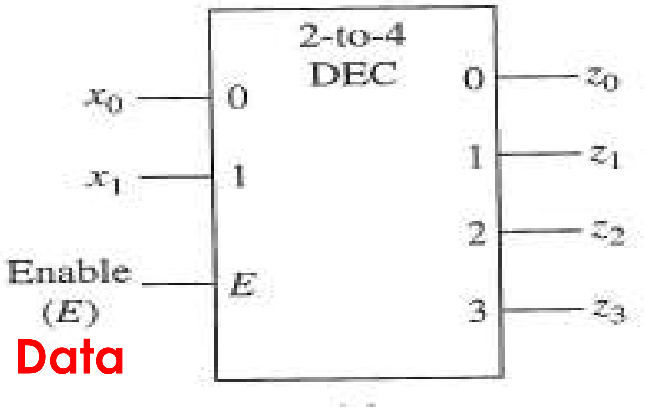


x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Decoders with enable inputs

- When disabled, all outputs of the decoder can either be at logic-0 or logic-1.
- Enable input provides the decoder with additional flexibility.
- Idea: if data is applied to the enable input.
- Process is known as demultiplexing.
- Now Decoder works as Demultiplexer.

$\overline{x}_0\overline{x}_1E$

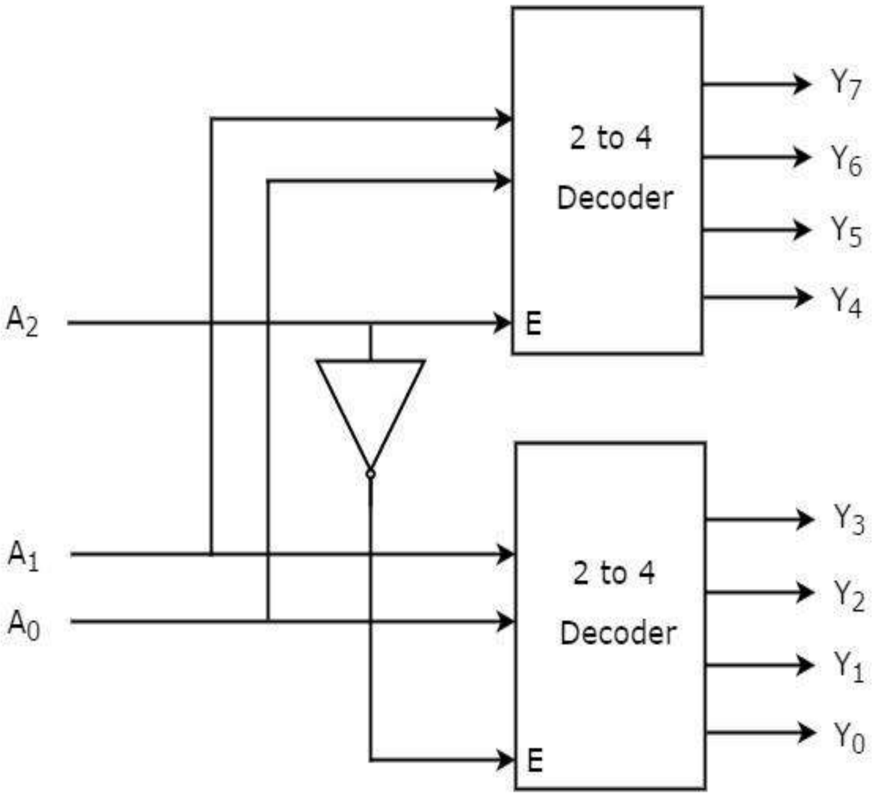


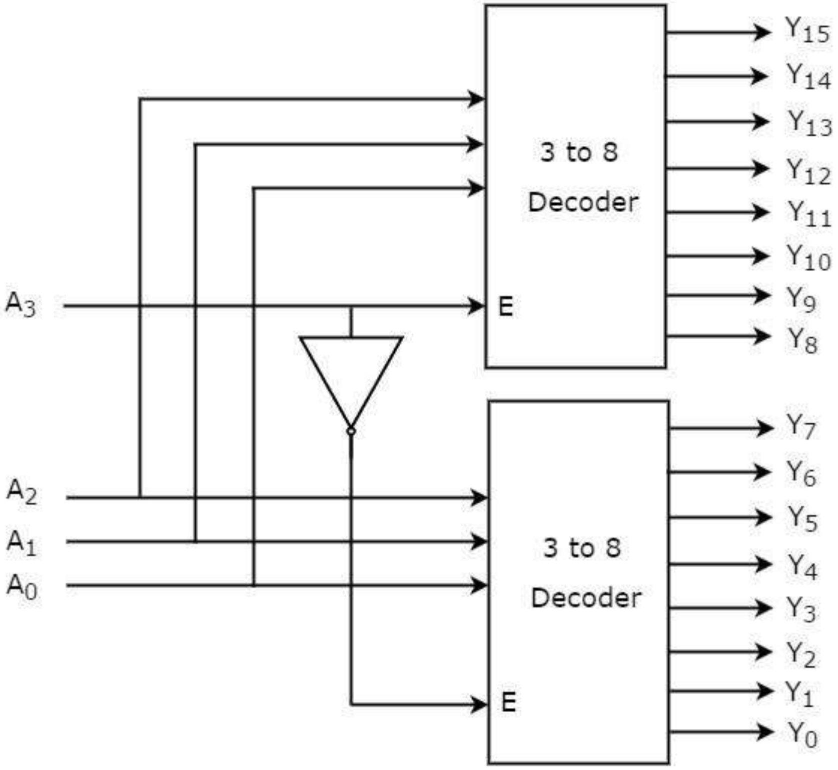
Data

If $x_0 = 0, x_1 = 0$ then data appears on line z_0.

- Enable inputs are useful when constructing larger decoders from smaller decoders.

Larger Decoders from smaller Decoder





Applications

- In digital electronic decoder play an important role. It is used to convert the data from one form to another form.
- Generally, these are frequently used in the communication systems like telecommunication, networking, and transfer the data from one end to the other end.
- In the same way it is also used in the digital domain for easy transmission of data.
- It is also used as

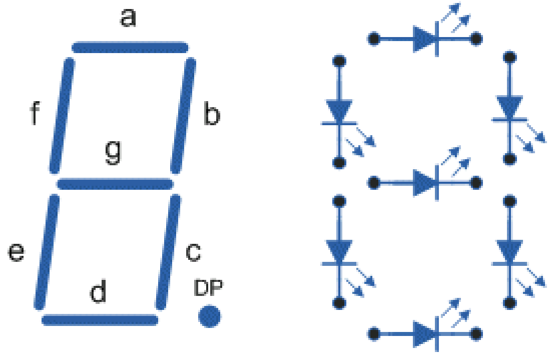
  Binary to Octal converter

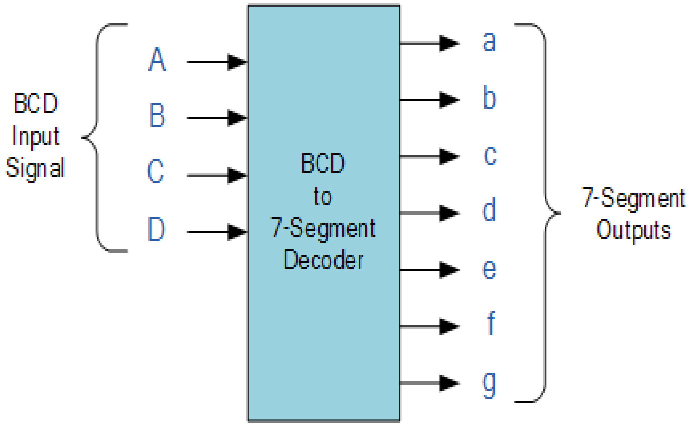
  BCD to Decimal converter

  BCD to Seven Segment Display
- Boolean functions can be implemented using decoder.

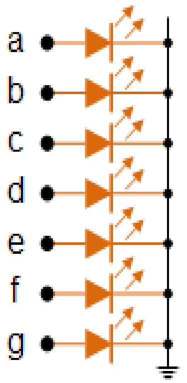
BCD to Seven segment display

- The Seven segment display is most frequently used the digital display in calculators, digital counters, digital clocks, measuring instruments, etc.
- Usually, the displays like LED's as well as LCD's are used to display the characters as well as numerical numbers.
- These displays are frequently driven by the output phases of digital integrated circuits like decade counters as well as latches.
- However, the outputs of these are in the type of 4-bit BCD (Binary Coded Decimal), so not appropriate for directly operating the seven segment display.
- For that, a display decoder can be employed for converting BCD code to seven segment code.
- Generally, it has four input lines as well as seven output lines.
- The Decoder is an essential component in BCD to seven segment display.

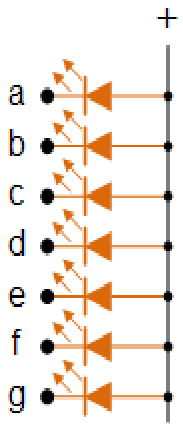




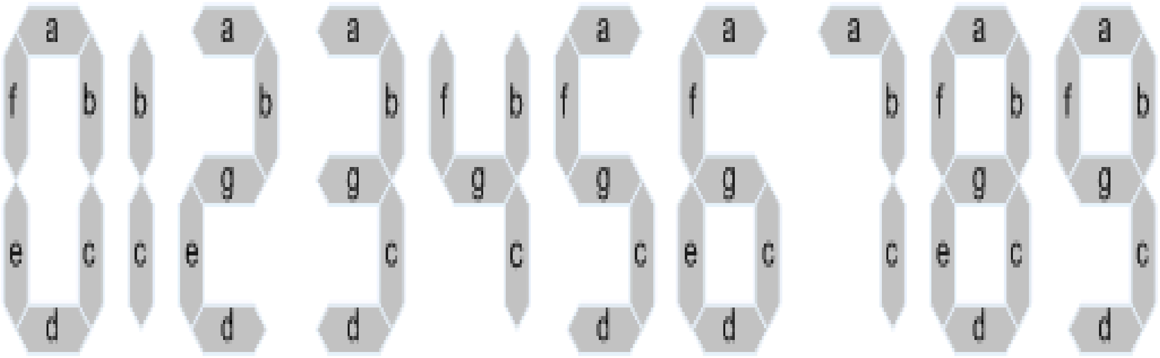
- The circuit design, as well as operation, mainly depends on the concepts of Boolean Algebra as well as logic gates.
- The common terminals are either anode or cathode. So, it may be common cathode type or common anode type.



Common
Cathode

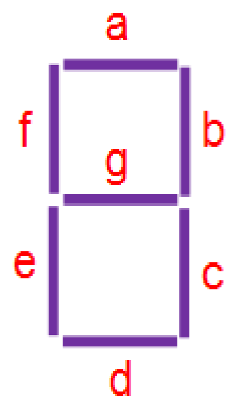


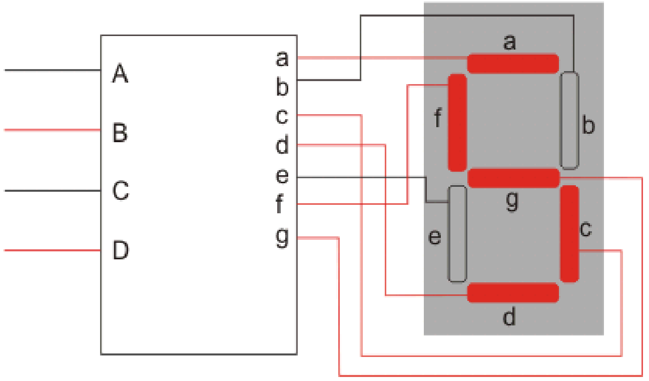
Common
Anode



Truth Table

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9





$a = F1\ (A, B, C, D) = \sum m\ (0, 2, 3, 5, 6, 7, 8, 9)$

$b = F2\ (A, B, C, D) = \sum m\ (0, 1, 2, 3, 4, 7, 8, 9)$

$c = F3\ (A, B, C, D) = \sum m\ (0, 1, 3, 4, 5, 6, 7, 8, 9)$

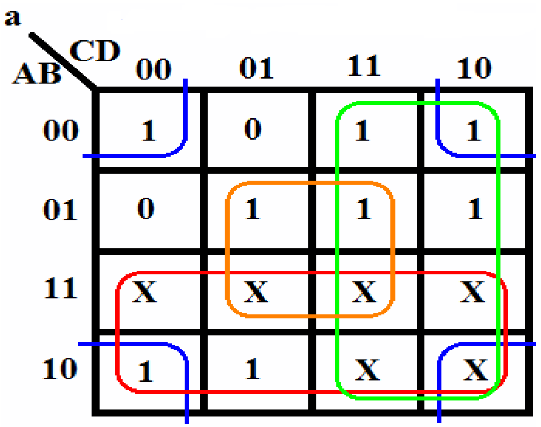
$d = F4\ (A, B, C, D) = \sum m\ (0, 2, 3, 5, 6, 8, 9)$

$e = F5\ (A, B, C, D) = \sum m\ (0, 2, 6, 8)$

$f = F6\ (A, B, C, D) = \sum m\ (0, 4, 5, 6, 8, 9)$

$g = F7\ (A, B, C, D) = \sum m\ (2, 3, 4, 5, 6, 8, 9)$

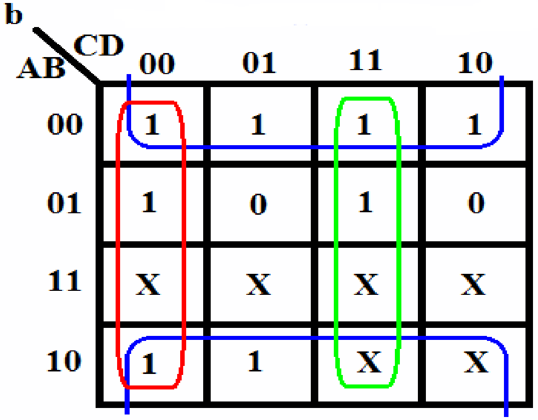
K-Map



a

AB\CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

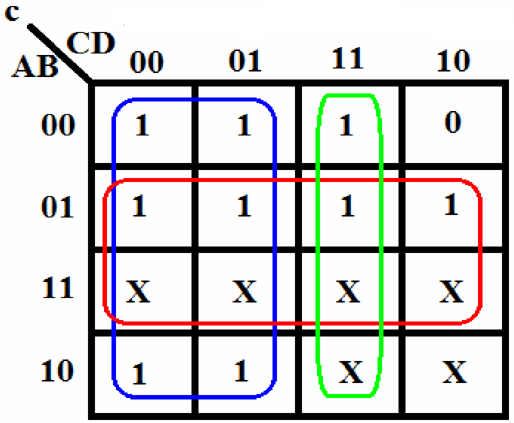
$a = A + C + BD + \overline{B}\overline{D}$



b

AB\CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	X	X	X	X
10	1	1	X	X

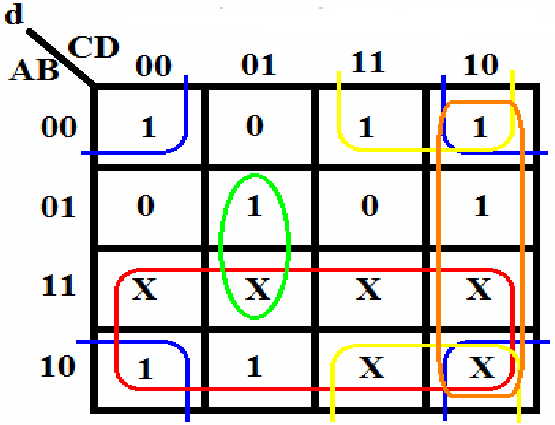
$b = \overline{B} + \overline{C}\overline{D} + CD$



c

AB\CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

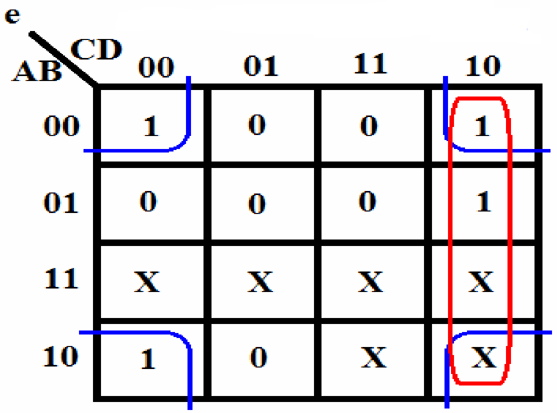
$c = B + \overline{C} + D$



d

AB\CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	X	X	X	X
10	1	1	X	X

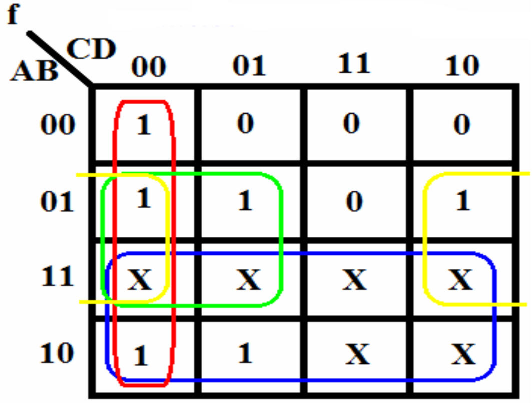
$d = A + \overline{B}\overline{D} + \overline{B}C + C\overline{D} + B\overline{C}D$



e

AB\CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

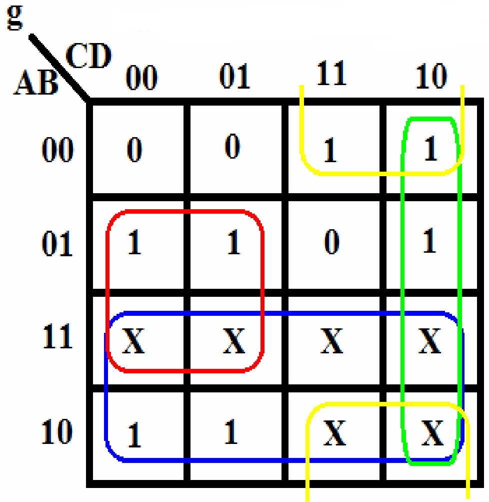
$e = \overline{B}\overline{D} + C\overline{D}$



f

AB\CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

$f = A + B\overline{C} + B\overline{D} + \overline{C}\overline{D}$

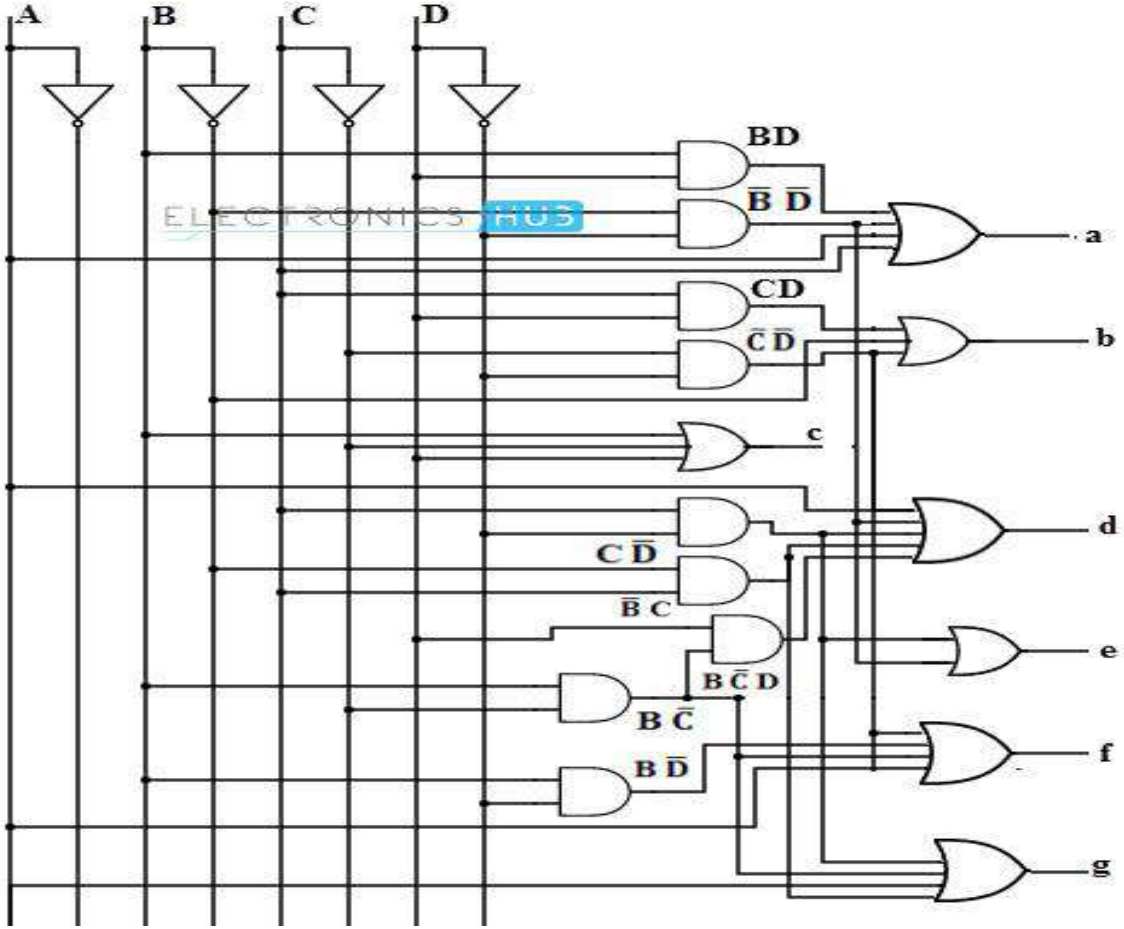


g

AB\CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

$g = A + B\overline{C} + \overline{B}C + C\overline{D}$

Logic Circuit



$a = A + C + BD + \bar{B}\,\bar{D}$

$b = \bar{B} + \bar{C}\,\bar{D} + CD$

$c = B + \bar{C} + D$

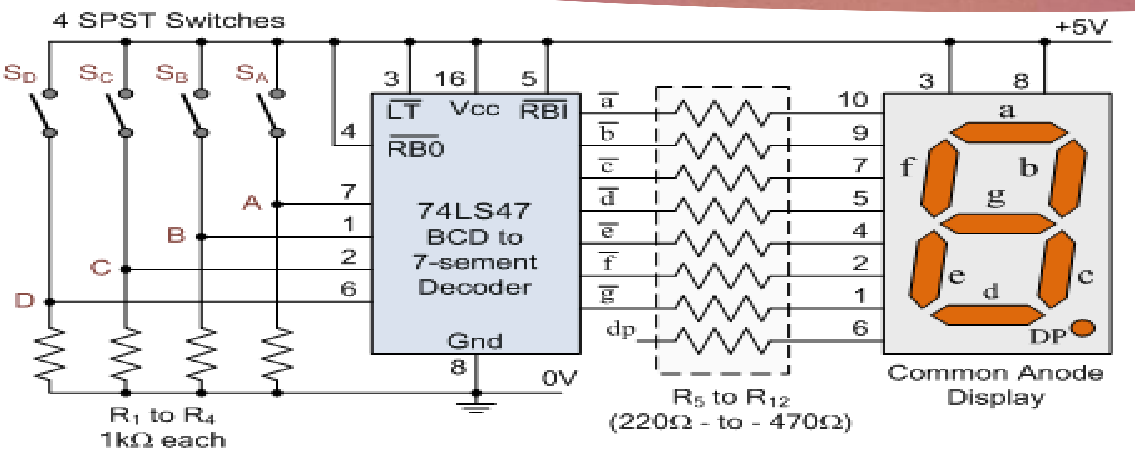
$d = \bar{B}\,\bar{D} + C\,\bar{D} + B\,\bar{C}\,D + \bar{B}\,C + A$

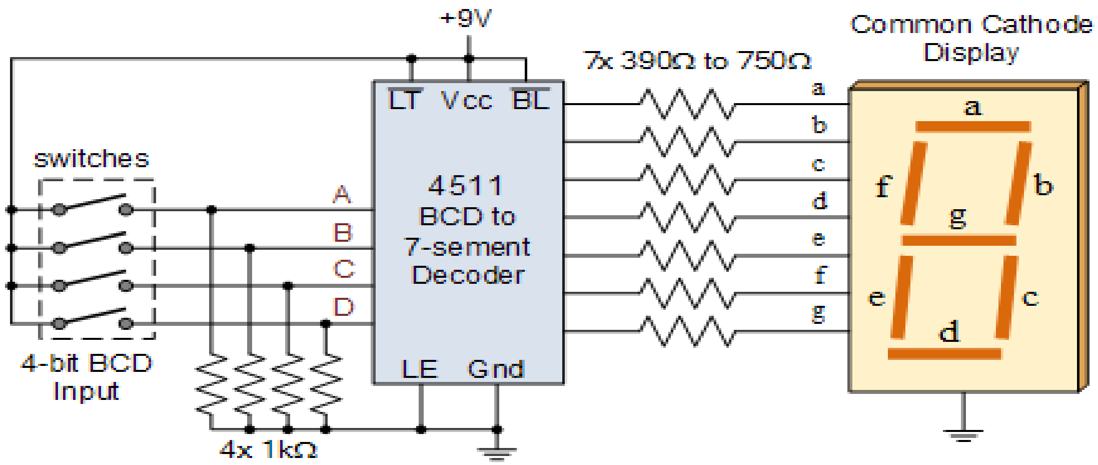
$e = \bar{B}\,\bar{D} + C\,\bar{D}$

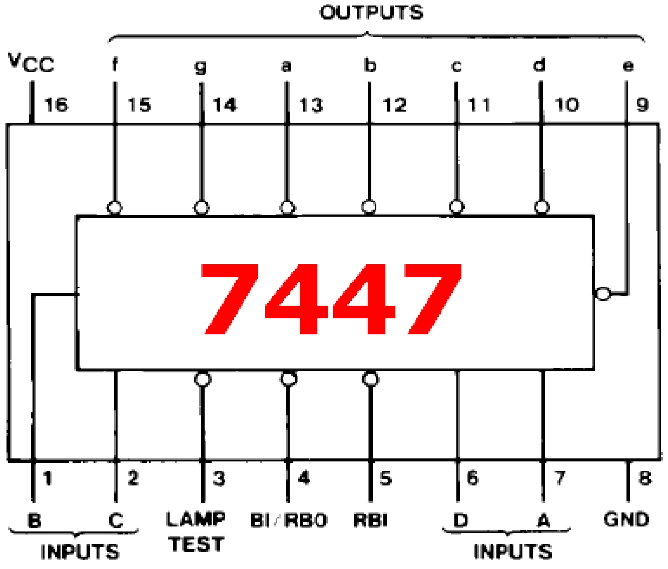
$f = A + \bar{C}\,\bar{D} + B\,\bar{C} + B\,\bar{D}$

$g = A + B\,\bar{C} + \bar{B}\,C + C\,\bar{D}$

IC and Connection Diagram

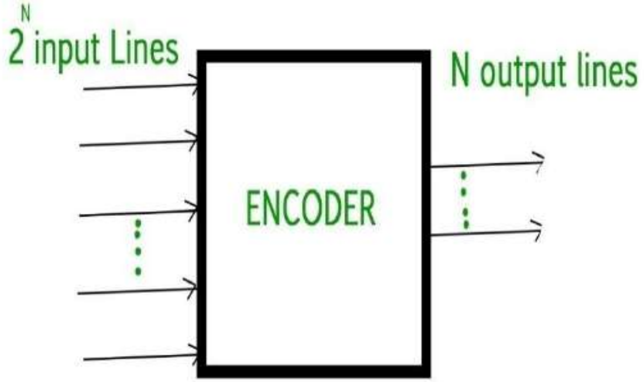




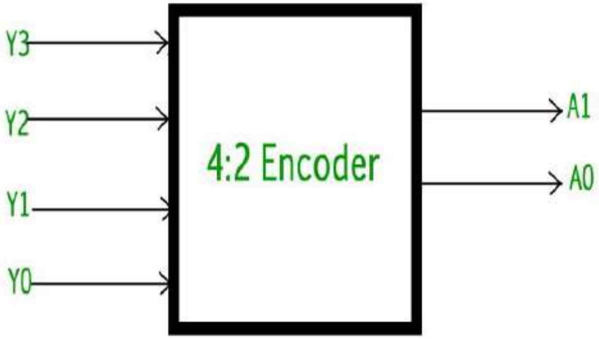


Encoder

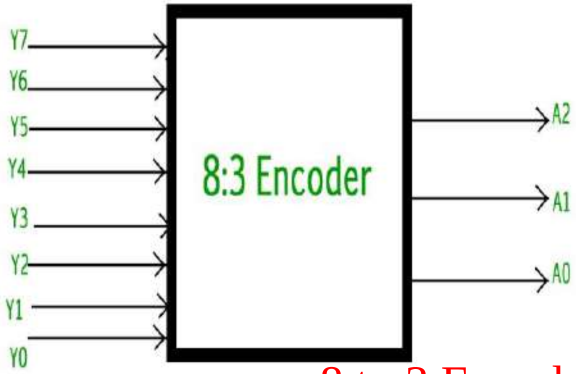
- An Encoder is a combinational circuit that performs the reverse operation of Decoder.
- It has maximum of **2^N input lines** and **‘N’ output lines**, hence it encodes the information from 2^N inputs into an N-bit code.
- It will produce a binary code equivalent to the input.



- The 4 to 2 Encoder consists of **four inputs Y3, Y2, Y1, Y0 and two outputs A1 and 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 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.



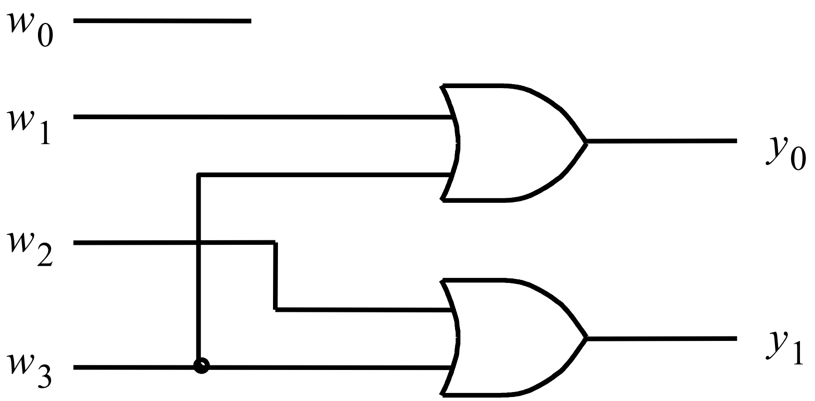
4 to 2 Encoder



8 to 3 Encoder

4-to-2 Binary Encoder

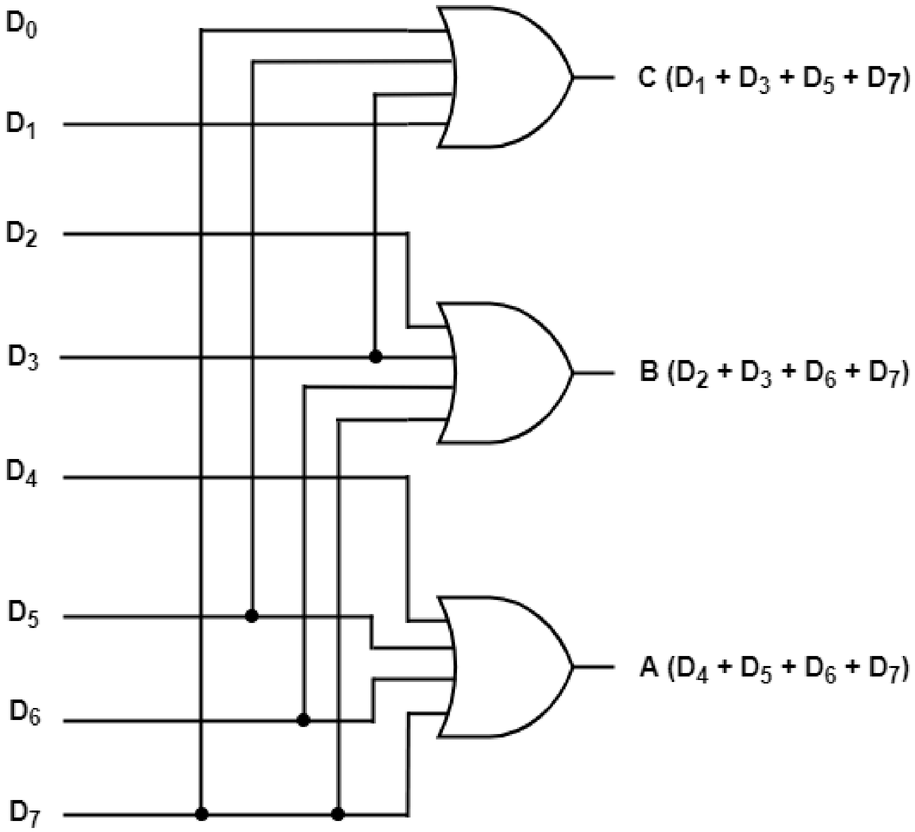
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



8-to-3 Binary Encoder

At any one time, only
one input line has a value of 1.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	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


D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
C ($D_1 + D_3 + D_5 + D_7$)
B ($D_2 + D_3 + D_6 + D_7$)
A ($D_4 + D_5 + D_6 + D_7$)

Priority Encoder

- One of the main disadvantages of standard digital encoder is that they can generate the wrong output code when there is more than one input present at logic level “1”.
- One simple way to overcome this problem is to “Prioritize” the level of each input pin.
- If there is more than one input at logic level “1” at the same time, the actual output code would only correspond to the input with the highest designated priority.
- This type of digital encoder is known as **Priority Encoder** or **P-Encoder** for short.
- The **Priority Encoder** solves the problems by allocating a priority level to each input.
- The *priority encoders* output corresponds to the currently active input which has the highest priority.
- So, when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

4-to-2 Priority Encoder

Truth Table

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

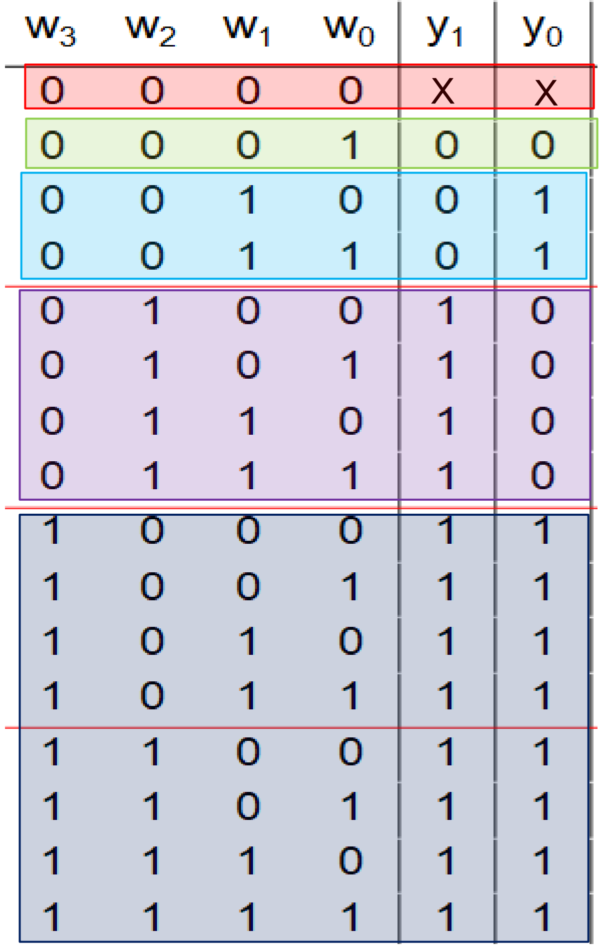
0 0 0 0

0 0 0 1

0 0 1 x

0 1 x x

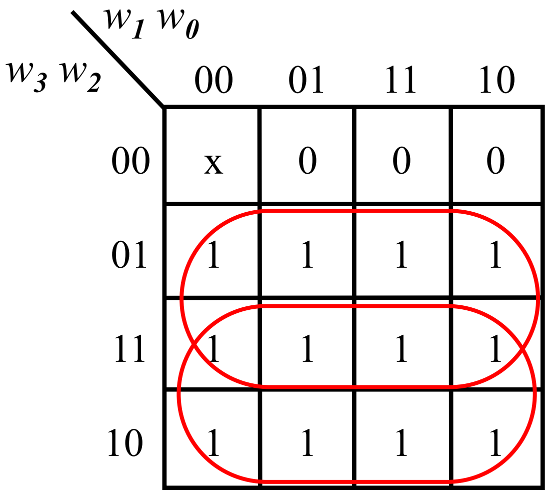
1 x x x



w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

K-Map

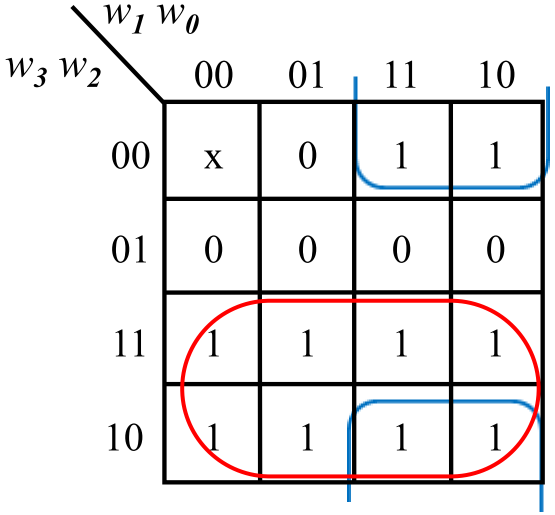
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1



$w_3 w_2$ \ $w_1 w_0$	00	01	11	10
00	x	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$y_1 = w_3 + w_2$

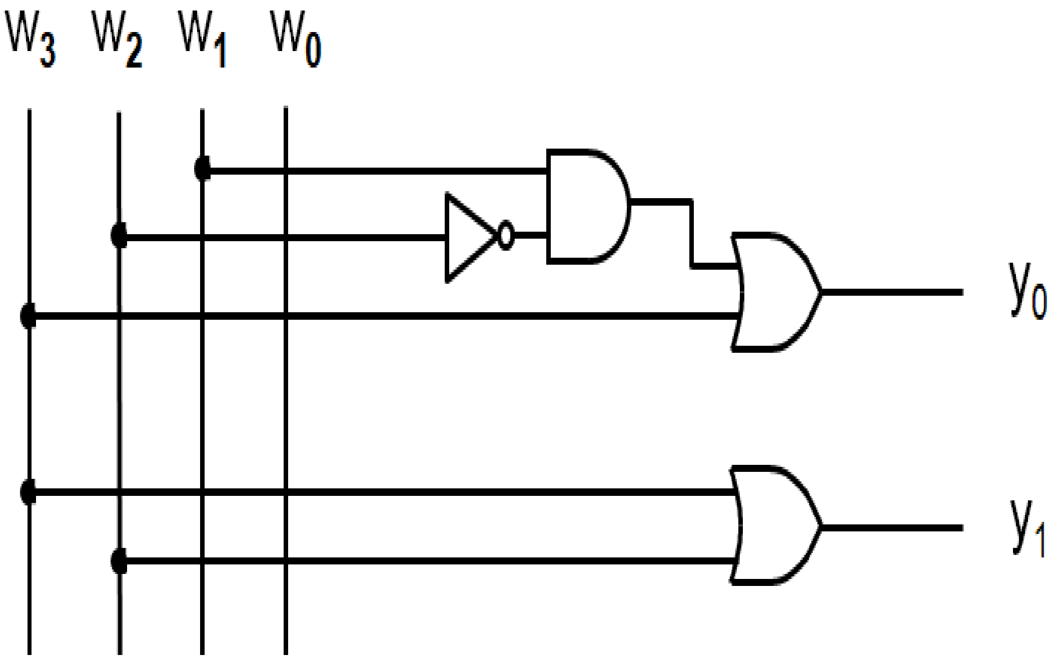
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1



	00	01	11	10
00	x	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$y_0 = w_3 + w_1 \overline{w_2}$$

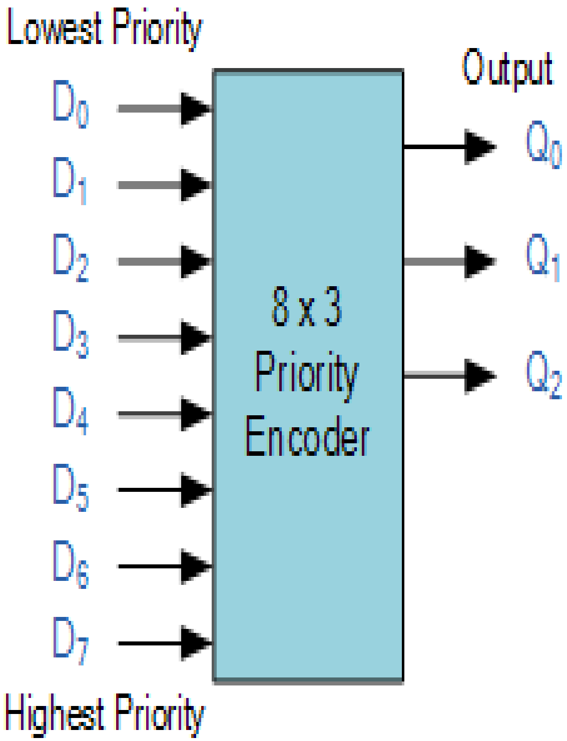
Circuit for the 4-to-2 priority encoder

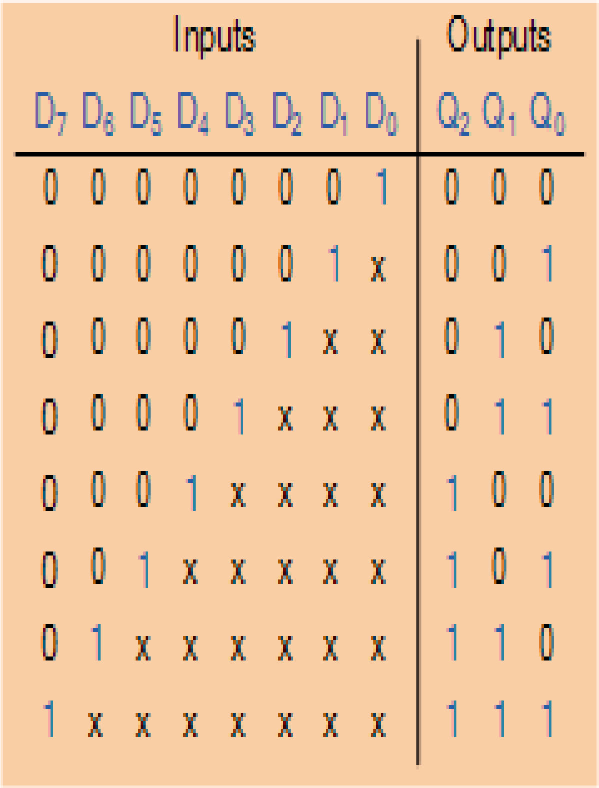


$y_0 = w_3 + w_1\overline{w_2}$

$y_1 = w_3 + w_2$

8-to-3 Priority Encoder





Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

- From the truth table of the Priority Encoder, the Boolean expression with data inputs D_0 to D_7 and outputs Q_0, Q_1, Q_2 is given as:

$$Q_0 = \sum\left(\bar{D}_6\left(\bar{D}_4\bar{D}_2D_1 + \bar{D}_4D_3 + D_5\right) + D_7\right)$$

$$Q_1 = \sum\left(\bar{D}_5\bar{D}_4\left(D_2 + D_3\right) + D_6 + D_7\right)$$

$$Q_2 = \sum\left(D_4 + D_5 + D_6 + D_7\right)$$

Applications

- Keyboard Encoder
- Interrupt Requests
- Octal to Binary Encoder
- Decimal to Binary Encoder
- Decimal to BCD Encoder