UNIT 2 : DESIGN OF COMBINATIONAL CIRCUITS

Introduction to Combinational circuits - Analysis and design procedures - Half Adder, Full Adder-Half Subtractor, Full Subtractor- Parallel binary Adder, Parallel binary Subtractor- Carry look ahead Adder- BCD Adder-Decoders- Encoders-Priority Encoder-Multiplexers- MUX as universal combinational modules- Demultiplexers- Code convertors- Magnitude Comparator.

2.1 Introduction to Combinational circuits

Combinational Logic Circuits are made from the basic and universal gates. The output is defined by the logic and it is depend only the present input states not the previous states.

Inputs and output(s) : logic 0 (low) or logic 1 (high).

Fig. Block diagram of a combinational circuits

Analysis and design procedures

The following are the basic steps to design a combinational circuits

1. Define the problem.
2. Determine the number of input and output variables.
3. Fix a letter symbols to the input and the outputs. (eg. A,B,C ,w, x, Y,F, etc)
4. Get the relationship between input and output from the truth table.
5. By using K-map obtain the simplified Boolean expression for the outputs.
6. Draw the logic diagram using gates.

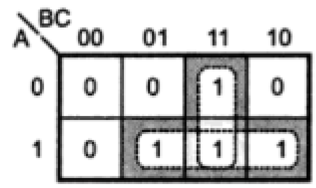
Example : Design a combinational logic circuit with three inputs , the output is at logic 1 when more than one inputs are at logic 1.

Solution: Assume A, B, C are inputs and Y is output .

Truth table

Inputs			**Output**
A	**B**	**C**	**Y**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

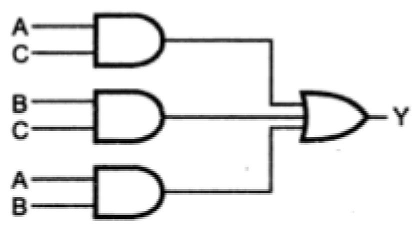
K map Simplification



Boolean Expression

Y=AC + BC + AB

Logic Diagram



2.2 Adder

The Basic operation in digital computer is binary addition. The circuit which perform the addition of binary bits are called as Adder.

The logic circuit which perform the addition of two bit is called Half adder and three bit is called Full adder.

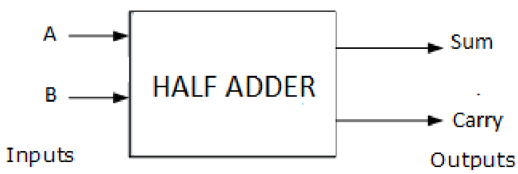
Rules for two bit addition

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
$1 + 1 = 10_2$

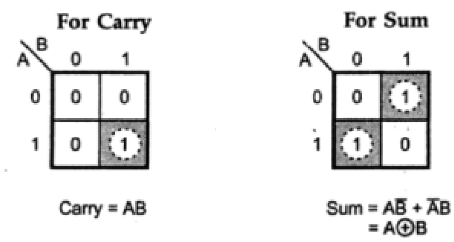
2.2.1 Half Adder

The two inputs of the half adders are augend and addend, the outputs are sum and carry.

Block diagram of Half adder



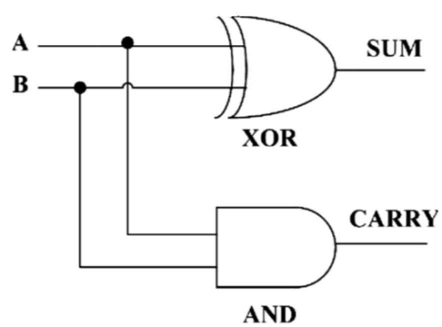
K-map simplification



Truth table of Half adder

Inputs		Outputs	
A	**B**	**Carry**	**Sum**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

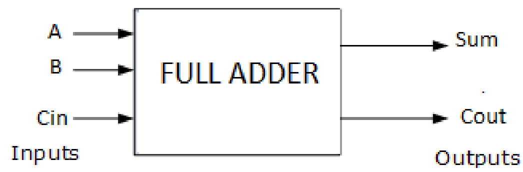
Logic diagram



2.2.2 Full Adder

The three inputs of the full adders are augend , addend and the carry input from the previous addition, the outputs are sum and carry

Block diagram of Full adder

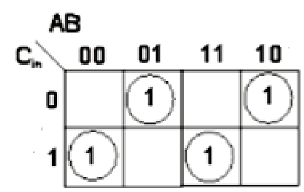


Truth table

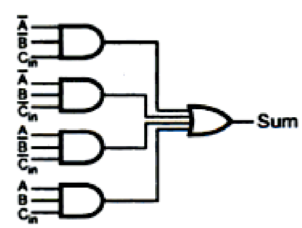
Inputs			Outputs	
A	**B**	**Cin**	**Cout**	**Sum**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-map simplifications

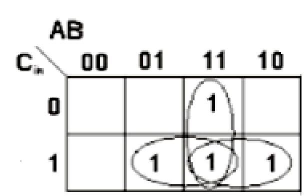
For Sum



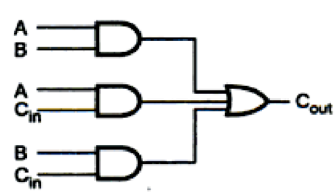
$$Sum = \bar{A}\bar{B}Cin + \bar{A}B\overline{Cin} + A\bar{B}\overline{Cin} + ABCin$$



For Cout



$$Cout = AB + BCin + ACin$$



The Full Adder can be implement using Two Half Adders and OR gates

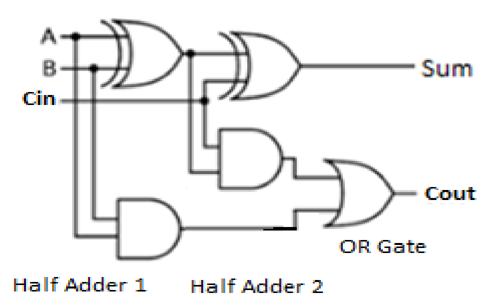
The expression for sum is

$$\begin{aligned}
\text{Sum} &= \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in} \\
&= C_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}\,(\overline{A}\,B + A\,\overline{B}) \\
&= C_{in}\,(A \cdot B) + \overline{C}_{in}\,(A \oplus B) \\
&= C_{in}\,(\overline{A \oplus B}) + \overline{C}_{in}\,(A \oplus B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}$$

The Expression for carry is

$$\begin{aligned}
C_{out} &= AB + A\,C_{in} + BC_{in} \\
&= AB + A\,C_{in} + BC_{in}\,(A + \overline{A}) \\
&= ABC_{in} + AB + A\,C_{in} + \overline{A}\,BC_{in} \\
&= AB\,(C_{in} + 1) + A\,C_{in} + \overline{A}\,BC_{in} \\
&= AB + AC_{in} + \overline{A}\,BC_{in} \\
&= AB + A\,C_{in}\,(B + \overline{B}) + \overline{A}\,B\,C_{in} \\
&= AB\,C_{in} + AB + A\,\overline{B}C_{in} + \overline{A}\,BC_{in} \\
&= AB\,(C_{in} + 1) + A\,\overline{B}C_{in} + \overline{A}\,BC_{in} \\
&= AB + A\,\overline{B}C_{in} + \overline{A}\,BC_{in} \\
&= AB + C_{in}\,(A\,\overline{B} + \overline{A}\,B) \\
&= AB + C_{in}\,(A \oplus B)
\end{aligned}$$

Logic Diagram



2.3 Subtractor

Subtractor is the logic circuit which is used to subtract two binary number (digit) and provides Difference and Borrow as a output. In digital electronics we have two types of subtractor, Half Subtractor and Full Subtractor.

Rules for two bit addition

0 - 0 = 0
0 - 1 = 1 with borrow 1
1 - 0 = 1
1 - 1 = 0

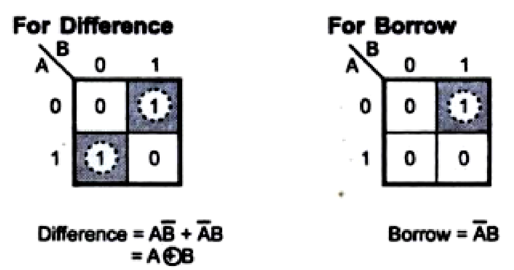
2.3.1 Half Subtractor

Half Subtractor is used for subtracting one single bit binary digit from another single bit binary digit.The truth table of Half Subtractor is shown below.

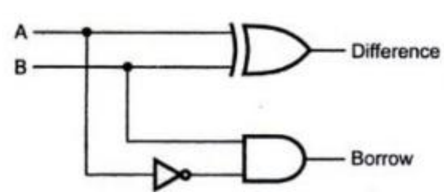
Truth table of Half adder

Inputs		Outputs	
A	**B**	**Difference**	**Borrow**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map for Difference and Borrow



Logic Diagram



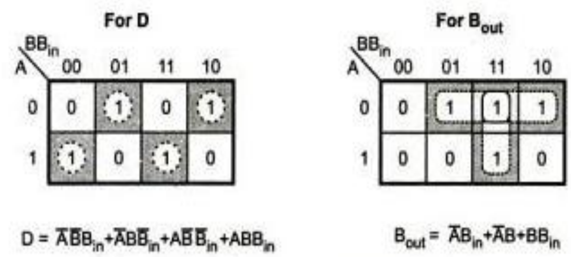
2.3.2 Full Subtractor

A logic Circuit Which is used for Subtracting Three Single bit Binary digit is known as Full Subtractor.The inputs are A,B, Bin and the outputs are D and Bout.

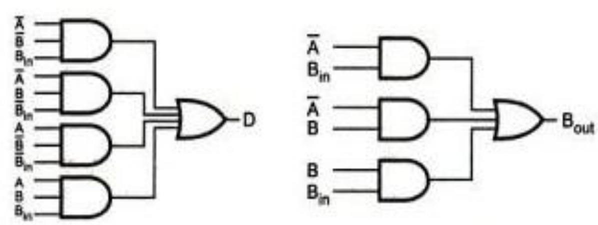
Truth table

Inputs			Outputs	
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

K-map for D and Bout



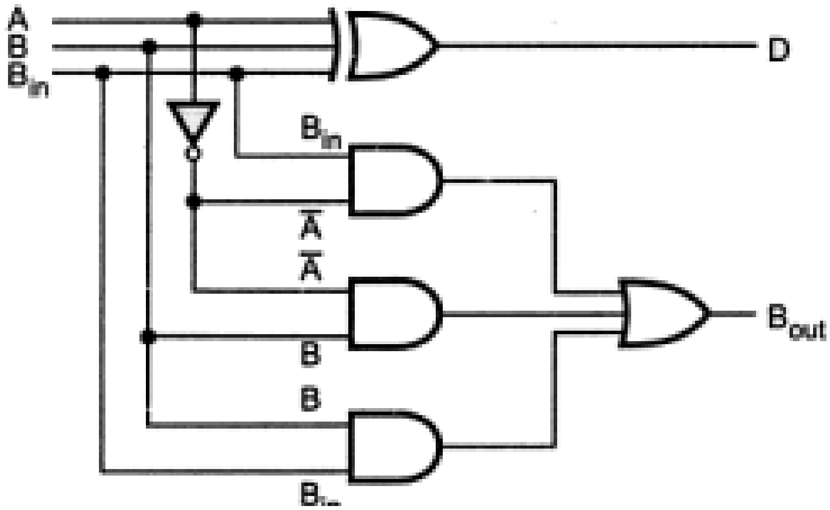
Logic Diagram



We can further simplify the function of the Difference (D)

$$\begin{aligned} D &= \overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B\,\overline{B}_{in} + A\,\overline{B}\,\overline{B}_{in} + A\,B\,B_{in} \\ &= B_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{B}_{in}\,(\overline{A}\,B + A\,\overline{B}) \\ &= B_{in}\,(A \odot B) + \overline{B}_{in}\,(A \oplus B) \\ &= B_{in}\,(\overline{A \oplus B}) + \overline{B}_{in}\,(A \oplus B) \\ &= B_{in} \oplus (A \oplus B) \end{aligned}$$

Simplified Logic diagram



2.4 Parallel Adder – Subtractor

2.4.1 Four bit Parallel binary Adder

In practical situations it is required to add two data each containing more than one bit. Two binary numbers each of *n* bits can be added by means of a full adder circuit. Consider the example that two 4-bit binary numbers $B_4B_3B_2B_1$ and $A_4A_3A_2A_1$ are to be added with a carry input C_1. This can be done by cascading four full adder circuits. The least significant bits A_1, B_1, and C_1 are added to the produce sum output S_1 and carry output C_2. Carry output C_2 is then added to the next significant bits A_2 and B_2 producing sum output S_2 and carry output C_3. C_3 is then added to A_3 and B_3 and so on. Thus finally producing the four-bit sum output $S_4S_3S_2S_1$ and final carry output Cout.

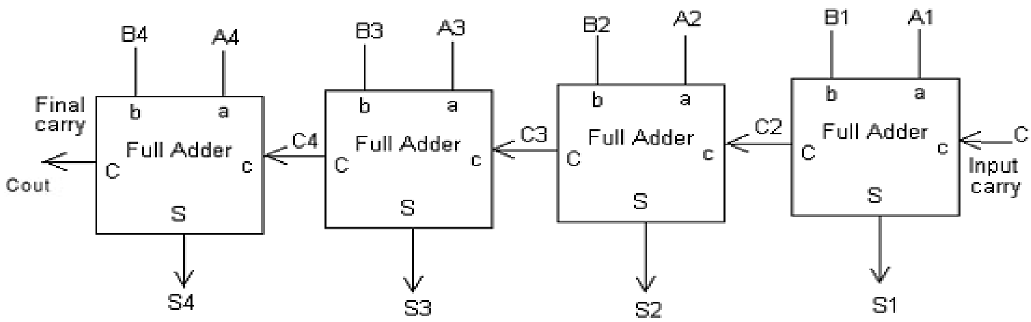


Fig.Block diagram of 4 bit binary parallel Adder

2.4.2 Four Bit Parallel Binary Subtractor

We can design a four bit parallel subtractor by connecting three full subtractors and one half subtractor. In the figure $A = A_3 A_2 A_1 A_0$ is minuend $B = B_3 B_2 B_1 B_0$ is subtrahend giving the difference $D = D_3 D_2 D_1 D_0$.

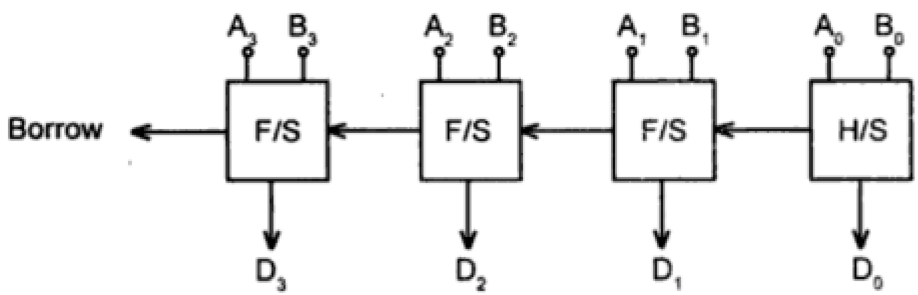


Fig.Block diagram of 4 bit binary parallel Subtractor

The subtraction operation can be performed using 1's and 2's complement addition, so we can design Full subtractor using Full Adder.

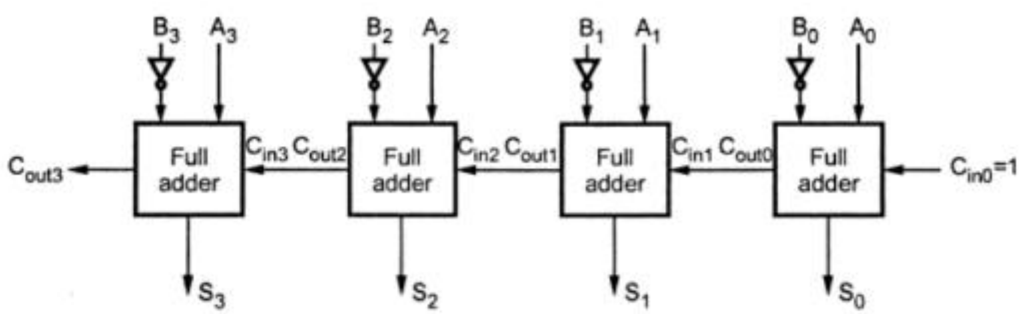


Fig.Four bit binary subtractor using Full Adder

2.4.3 Parallel binary Adder – Subtractor

The addition and subtraction operations can be perform using a common adder circuit, where a EX-OR gate is connected in the second input along with the mode selection bit M. if M=0 the circuit act as a adder, M=1 then substractor. If M=0 then output of the EX-OR gate is B act as adder, if M=1 then B' act as a subtractor.

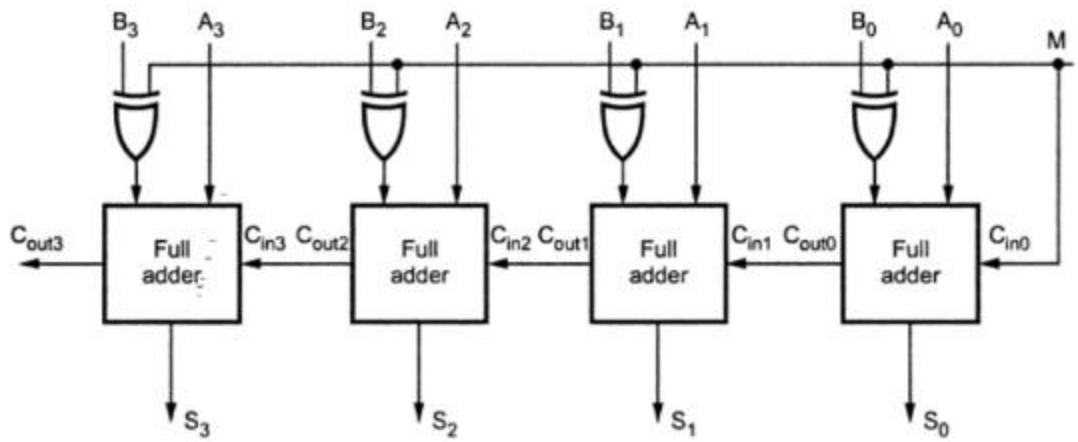


Fig.Parallel binary Adder – Subtractor

2.4.4 Carry look ahead Adder

In the parallel adder the carry input of each stage is depends on the carry output of the previous stage. This processes leads to time delay in addition.This delay is called propagation delay. The process can be speeding up by eliminating the inter stage carry delay called look ahead carry addition. In uses two functions carry generate and carry propagate.

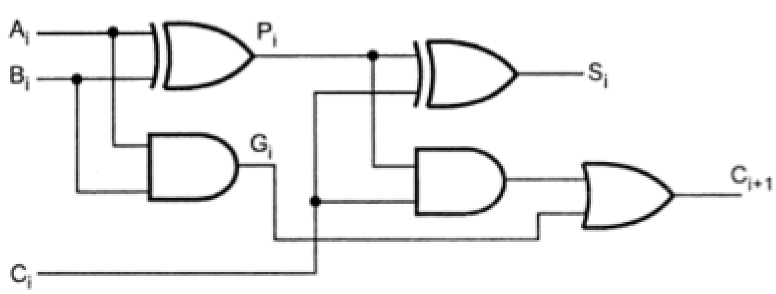


Fig.Full Adder Circuit

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

The output sum and carry can be expressed as

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

Gi is called carry generate and Pi is called carry propagate.

The Boolean function for the carry output of each stage can be

$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2 (G_1 + P_1C_1)$$
$$= G_2 + P_2G_1 + P_2 P_1C_1$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1C_1)$$
$$= G_3 + P_3G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

From the above functions it can be seen that C_4 does not have to wait for C_3 and C_2. All the carries are propagating at the same time.

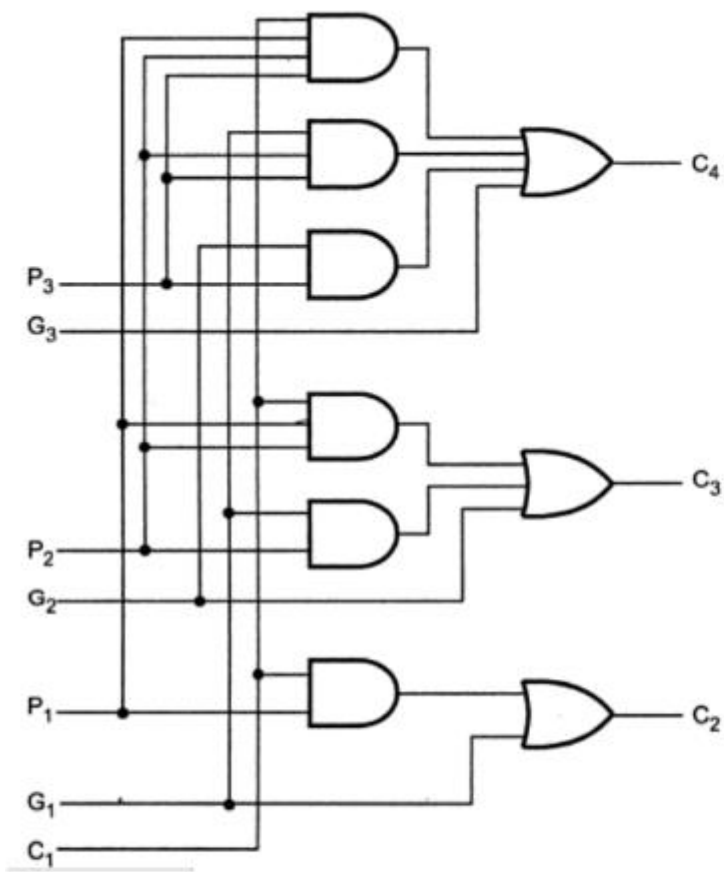


Fig.Logic diagram of a look-ahead carry generator

2.5 BCD Adder

In digital system, the decimal number is represented in the form of binary coded decimal (BCD).The ten digit (0-9) decimal numbers are represented by the binary digits. The circuit which add the two BCD number is called BCD adder. The BCD cannot be greater than 9. The representation of the BCD number as follows, consider the 526 it can be expressed as

$$\begin{array}{ccc} 5 & 2 & 6 \\ \downarrow & \downarrow & \downarrow \\ 0101 & 0010 & 0110 \end{array}$$

There are three different cases in BCD Addition

i)Sum is less than or equal to 9 with carry 0

Consider the addition of two BCD numbers 6 and 3, The addition is performed as normal binary addition

$$\begin{array}{rr} 6 & 0\;1\;1\;0 \\ +\;3 & 0\;0\;1\;1 \\ \hline 9 & 1\;0\;0\;1 \end{array}$$

ii)Sum is greater than 9 with carry 0

consider the number 6 and 8 in BCD

The sum is invalid BCD number, Add the sum with correction number 6

$$\begin{array}{rrrl} & 6 & 0\;1\;1\;0 & \\ & +\;8 & 1\;0\;0\;0 & \\ \hline & 14 & 1\;1\;1\;0 & \leftarrow \text{Invalid BCD number} \\ & & +\;0\;1\;1\;0 & \leftarrow \text{Add 6 for correction} \\ \hline \underbrace{0\;0\;0\;1}_{1} & & \underbrace{0\;1\;0\;0}_{4} & \leftarrow \text{BCD for 14} \end{array}$$

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

iii) Sum equals 9 or less with carry 1

Consider the addition of 8 and 9 in BCD.

The result 0001 0001 is valid BCD number but it is incorrect. Add 6 to get correct number.

```
   8                  1 0 0 0
 + 9                  1 0 0 1
 ----  --------------------------
  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 for17
```

The procedure for BCD addition is

1. Add two BCD numbers using ordinary binary addition.
2. If four bit sum is less than or equal to zero, then correction is needed.
3. If the four bit sum is greater than 9 or if carry is generated then add 0110.

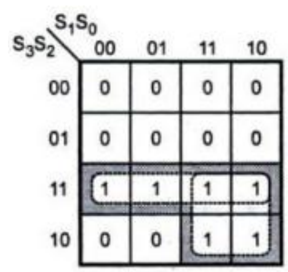
Implementation of BCD Adder

We require 4-bit binary adder for initial addition, Logic circuit to detect sum greater than 9, and second 4 bit binary adder to add 0110.

The following truth table is used to design a circuit for the sum, which is greater than 9

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 for carry (Y) identification



$Y = S_3S_2 + S_3S_1$

If Y=1 add 0110 using binary adder

Block diagram of Binary Adder

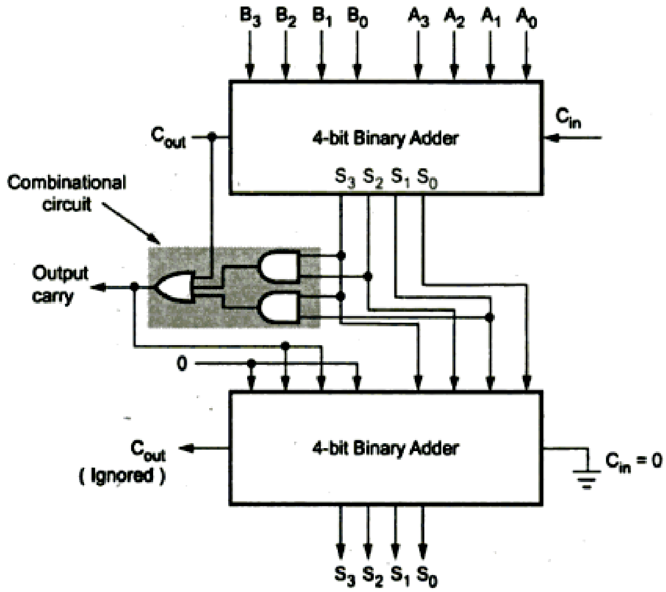


Fig.Block diagram of BCD adder

The binary adder add two BCD numbers, ifcarry is ‘0’ nothing to be added. If carry is ‘1’ add 0110 with the sum, consider the overall carry from the first stage of the addition.

Example : Design an8-bit BCD adder using IC 74283.

Solution: Use two 4-bit BCD adder to design 8-bit binary adder.

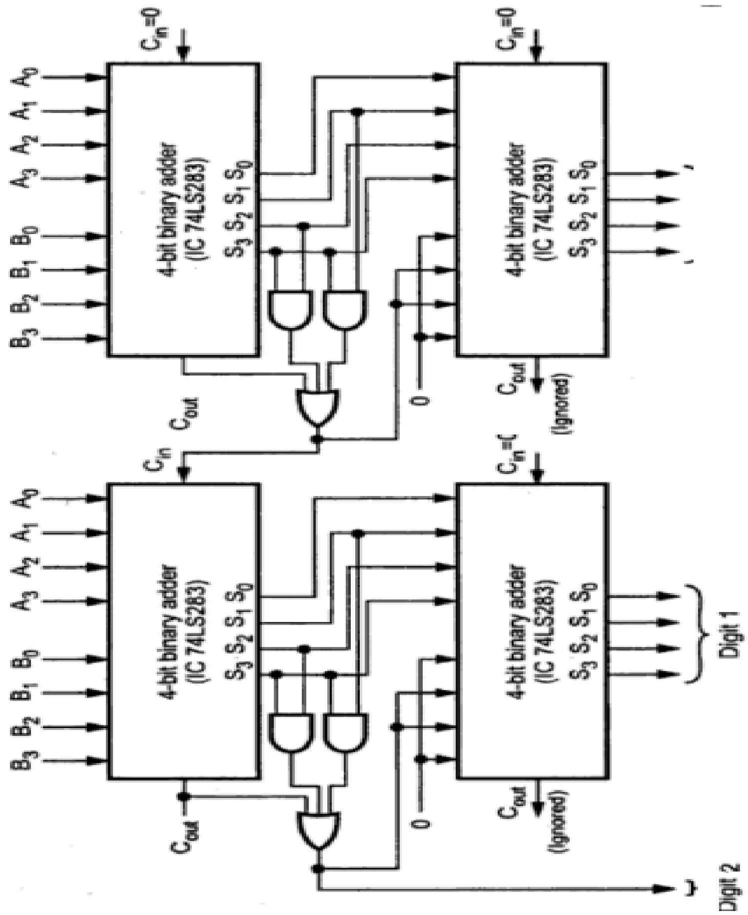


Fig. 8- bit BCD Adder using IC 74283

2.6 Decoder

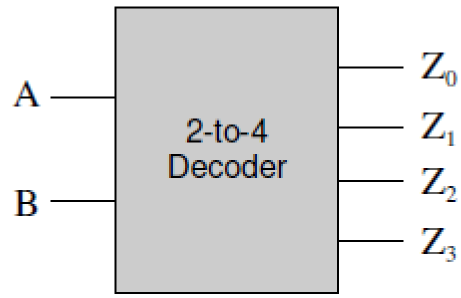
Decoder is a combinational circuit.

It has N inputs and 2^N outputs.

2 to 4 Decoder

It has 2 inputs and $2^2 = 4$ outputs.

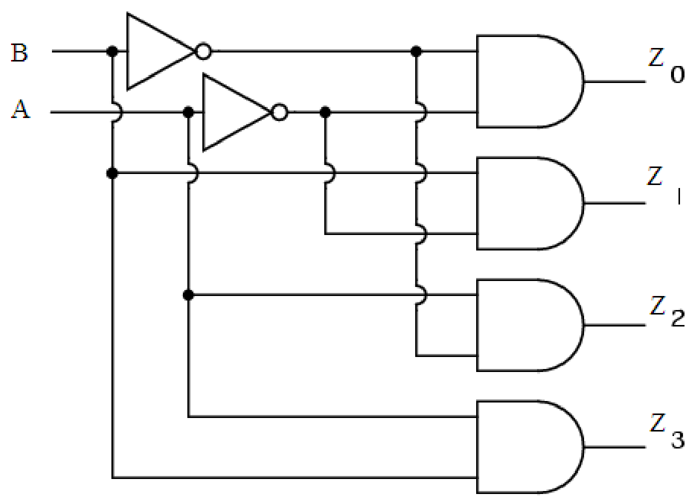
Circuit Diagram



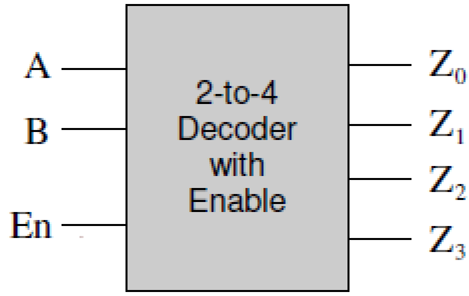
Truth Table

A	B	Z_0	Z_1	Z_2	Z_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Logic Diagram



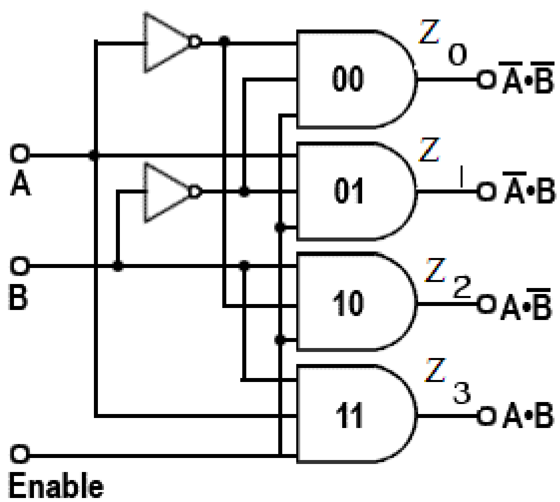
2 to 4 Decoder with Enable input



Truth Table

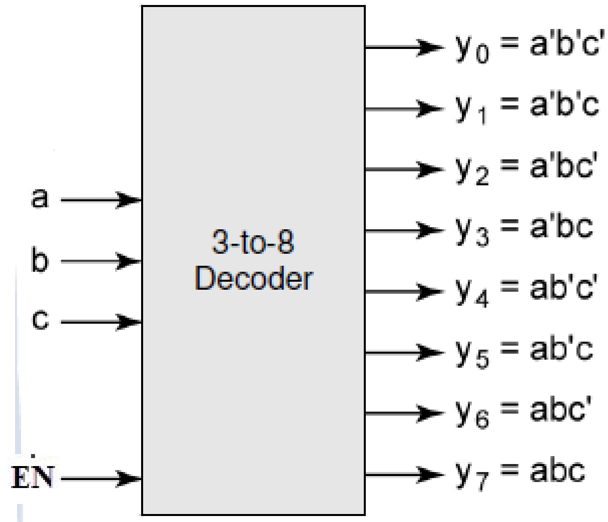
	En	A	B	Z_0	Z_1	Z_2	Z_3
enabled	1	0	0	1	0	0	0
	1	0	1	0	1	0	0
	1	1	0	0	0	1	0
	1	1	1	0	0	0	1
disabled	0	x	x	0	0	0	0

Logic Diagram



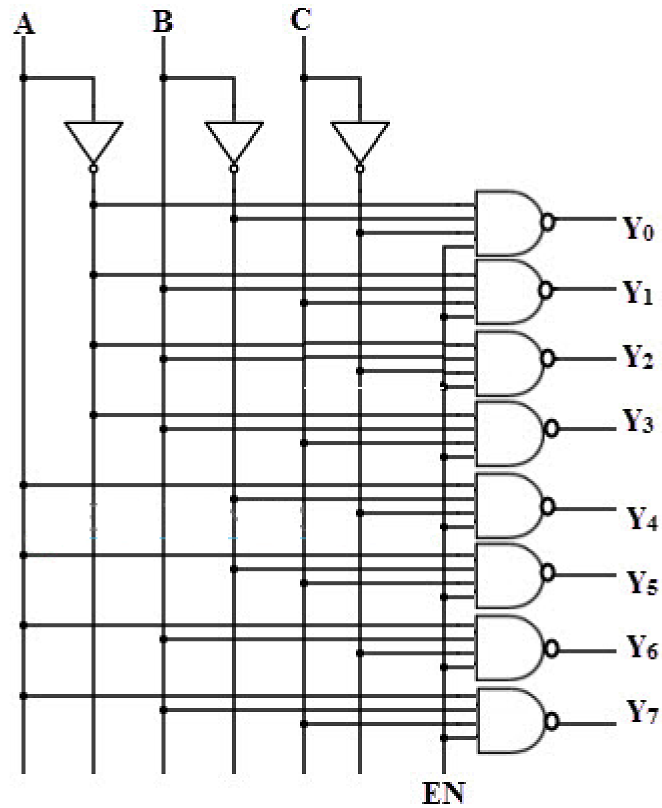
3 to 8 Decoder

It has 3 inputs and $2^3 = 8$ outputs.



Inputs				Outputs							
EN	A	B	C	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

Logic Diagram

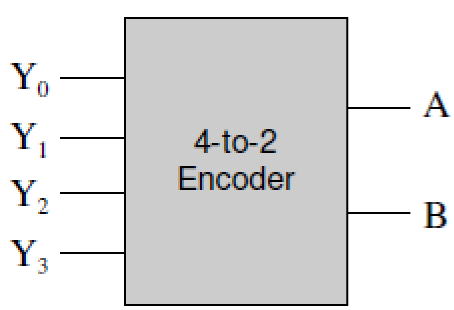


2.7 Encoders

Encoders is a combinational circuit which takes 2^N inputs and gives out N outputs, the enable pin should be kept 1 for enabling the circuit.

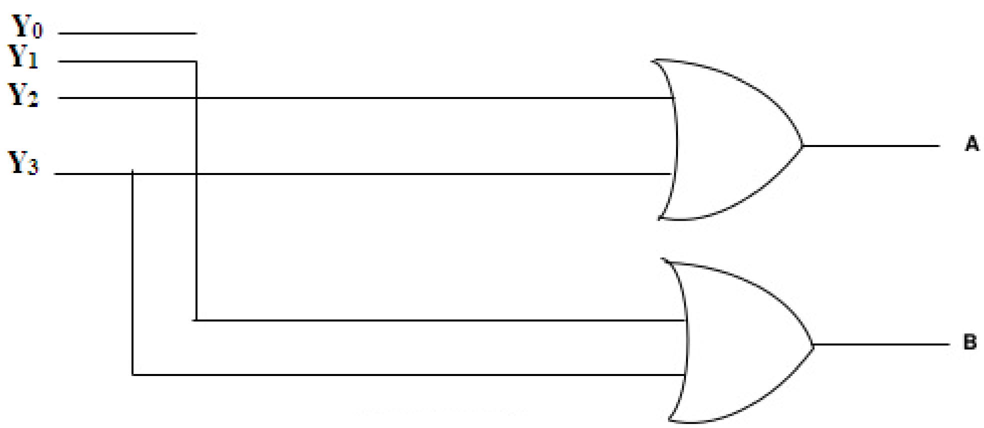
4 to 2 Encoder

It has 2^2 inputs and 2 outputs.



Truth Table

Y_0	Y_1	Y_2	Y_3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



2.7.1 Priority Encoders

A Priority Encoder works opposite of the decoder circuit. If more than one input is active, the higher order input has priority.

4 to 2 Priority Encoders

D0-D3 - inputs

A1,A0 – outputs

Active (A)– Valid indicator. It indicates the output is valid or not

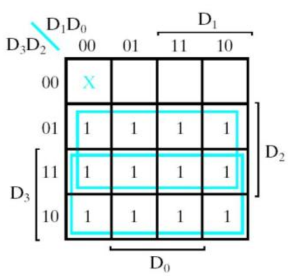
Output is invalid when no inputs are active .i.e, A=0

Output is valid when at least one input is active .i,e, A=1

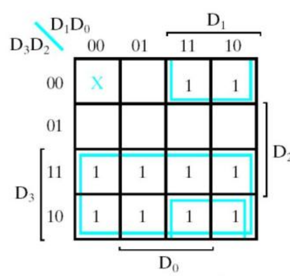
Truth Table

D3	D2	D1	D0	A1	A0	Active
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

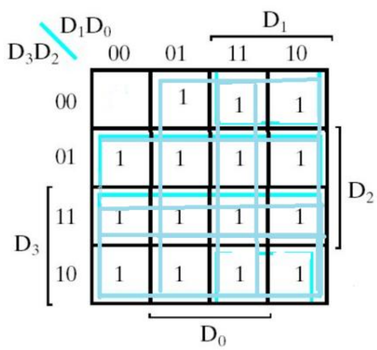
K-map simplification



$A_1 = D_2 + D_3$

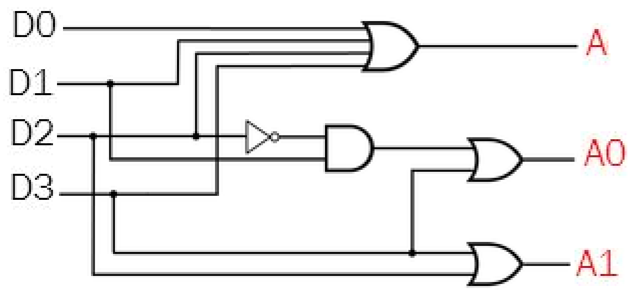


$A_0 = D_3 + D_1\overline{D}_2$

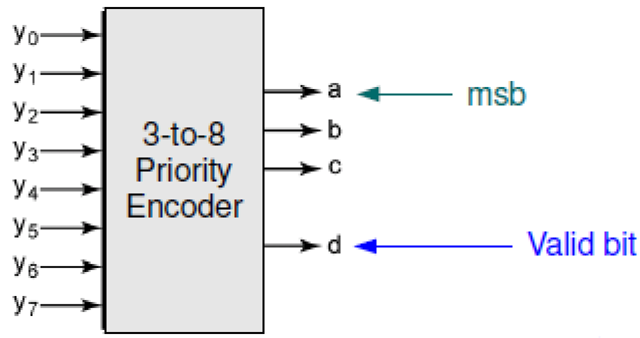


$A = D_2 + D_3 + D_1 + D_0$

Logic Diagram



3 to 8 Priority Encoder



y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

2.8 Mutliplexer (Mux)

Multiplexer is a combinational circuit that selects binary information from one of many inputs and directs it into single output.

The selection of particular input is controlled by a set of selection line

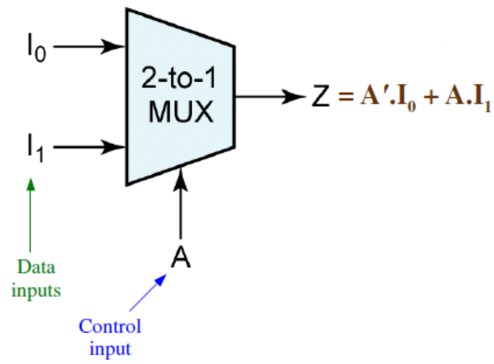
Mutliplexer has 2^n inputs, n select line (control input) and one output

It also called as Data selector

2 to 1 Multiplexer

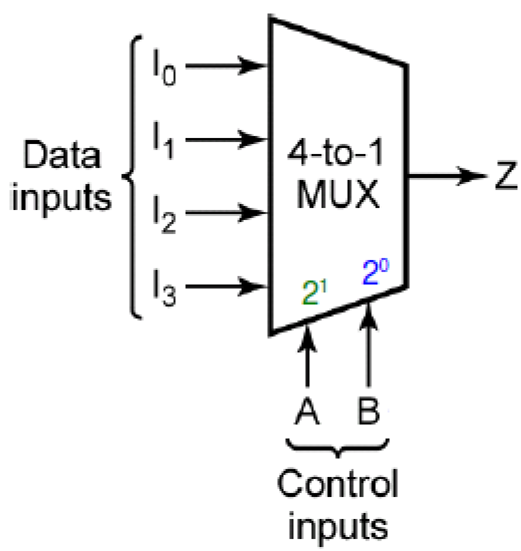
has 2^1 inputs, 1 select line and one output

Circuit diagram



4 to 1 MUX

4 to 1 MUX has 2^2 = 4 inputs, 2 select line and one output

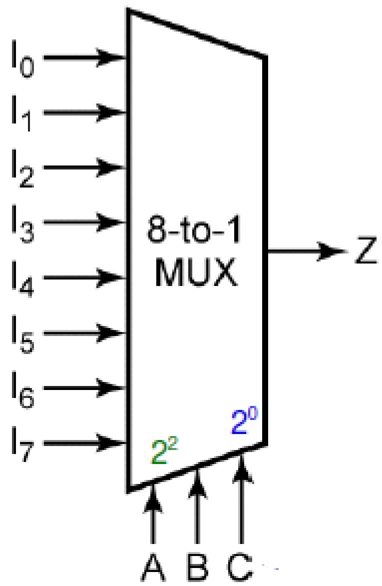


A	B	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Z = A'.B'.I_0 + A'.B.I_1 + A.B'.I_2 + A.B.I_3$$

8 to1 MUX

8 to1 MUX has $2^3 = 8$ inputs, 3 select line and one output



A	B	C	Z
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

$$Z = A'.B'.C'.I_0 + A'.B'.C.I_1 + A'.B.C'.I_2 + A'.B.C.I_3 +$$
$$A.B'.C'.I_0 + A.B'.C.I_1 + A'.B.C'.I_2 + A.B.C.I_3$$

$$Z = \Sigma\, m_i . I_i$$

2.8.1 MUX as universal combinational modules

Each minterm of the function can be mapped to a data input of the multiplexer.
For each row in the truth table, where the output is 1, set the corresponding data input of the mux to 1.Set the remaining inputs of the mux to 0.

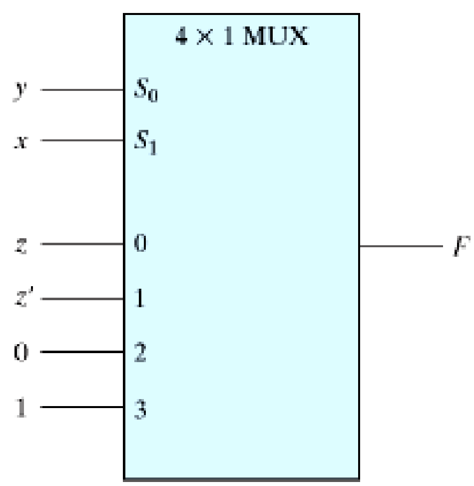
Example 1: Implement the following Boolean function using 4:1 MUX

F(x,y,z) = Σm(1, 2, 6, 7)

Truth Table

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	

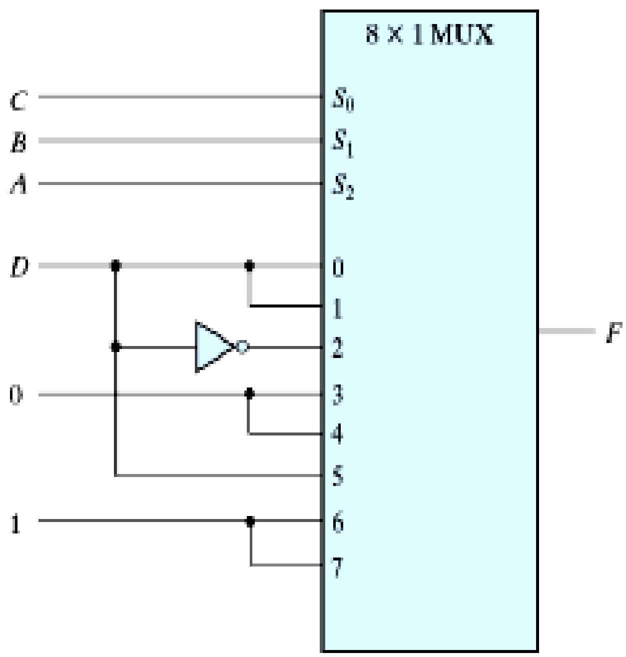
<u>Multiplexer Implementation</u>



Example 2: Implement the following Boolean function using 8:1 MUX

$$F(A,B,C,D) = \Sigma m(1, 3, 4, 11, 12\text{-}15)$$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



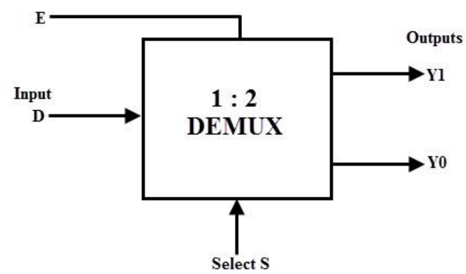
2.9 Demultiplexer (DEMUX)

Demultiplexer has 2^n outputs , n select lines, one input.

A **demultiplexer** is also called a data distributor.

<u>1-to-2 demultiplexer</u>

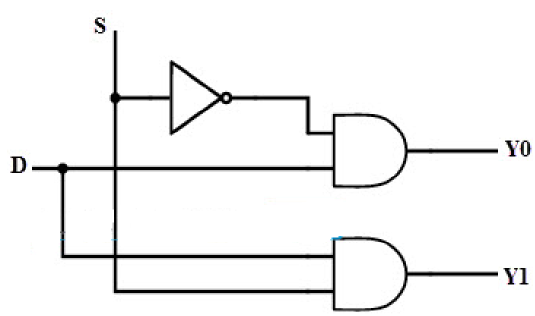
has 2^2 outputs , 2 select lines, one input.



The truth table

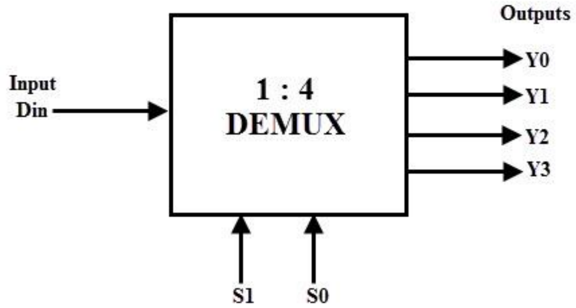
Select	Input	Outputs	
S	D	Y_1	Y_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Logic diagram



1-to-4 Demultiplexer

It has one input,2 select lines,4 outputs



The truth table

Data Input	Select Inputs		Outputs			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

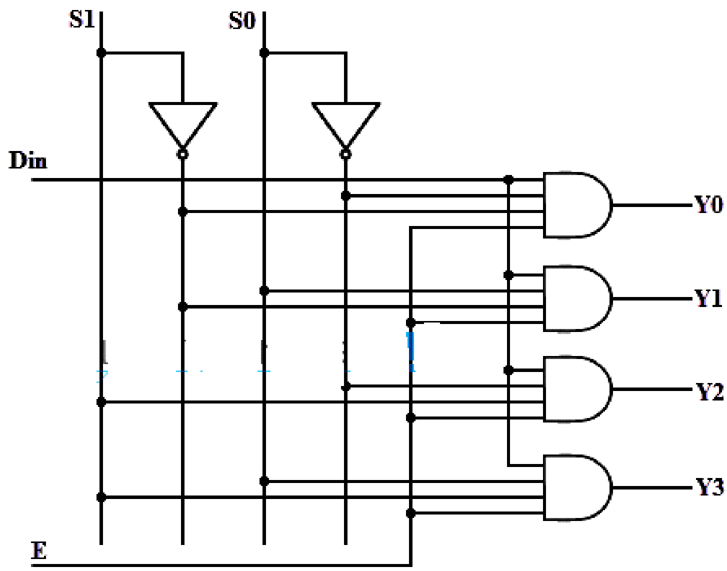
$$Y0 = \overline{S1}\ \overline{S0}\ D$$

$$Y1 = \overline{S1}\ S0\ D$$

$$Y2 = S1\ \overline{S0}\ D$$

$$Y3 = S1\ S0\ D$$

Logic Diagram

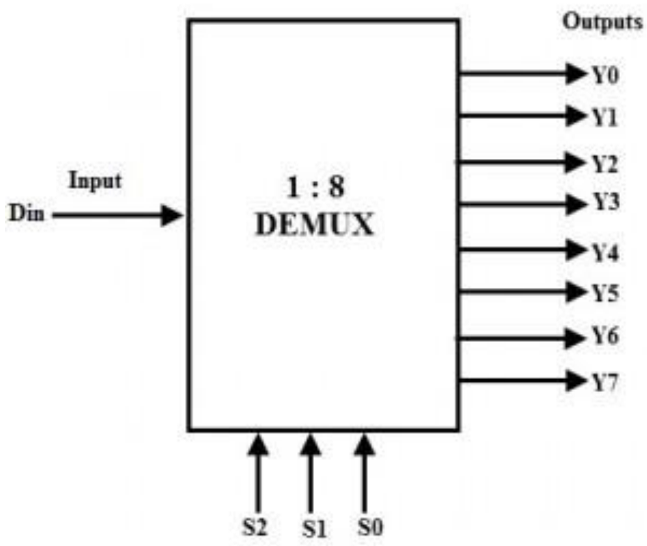


1-to-8 Demultiplexer

Has one input

3-select lines

8-outputs



The truth table

Data Input	Select Inputs			Outputs							
D	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	0	0	0	0	0	D
D	0	0	1	0	0	0	0	0	0	D	0
D	0	1	0	0	0	0	0	0	D	0	0
D	0	1	1	0	0	0	0	D	0	0	0
D	1	0	0	0	0	0	D	0	0	0	0
D	1	0	1	0	0	D	0	0	0	0	0
D	1	1	0	0	D	0	0	0	0	0	0
D	1	1	1	D	0	0	0	0	0	0	0

$$Y0 = D\,\overline{S2}\,\overline{S1}\,\overline{S0}$$

$$Y1 = D\,\overline{S2}\,\overline{S1}\,S0$$

$$Y2 = D\,\overline{S2}\,S1\,\overline{S0}$$

$$Y3 = D\,\overline{S2}\,S1\,S0$$

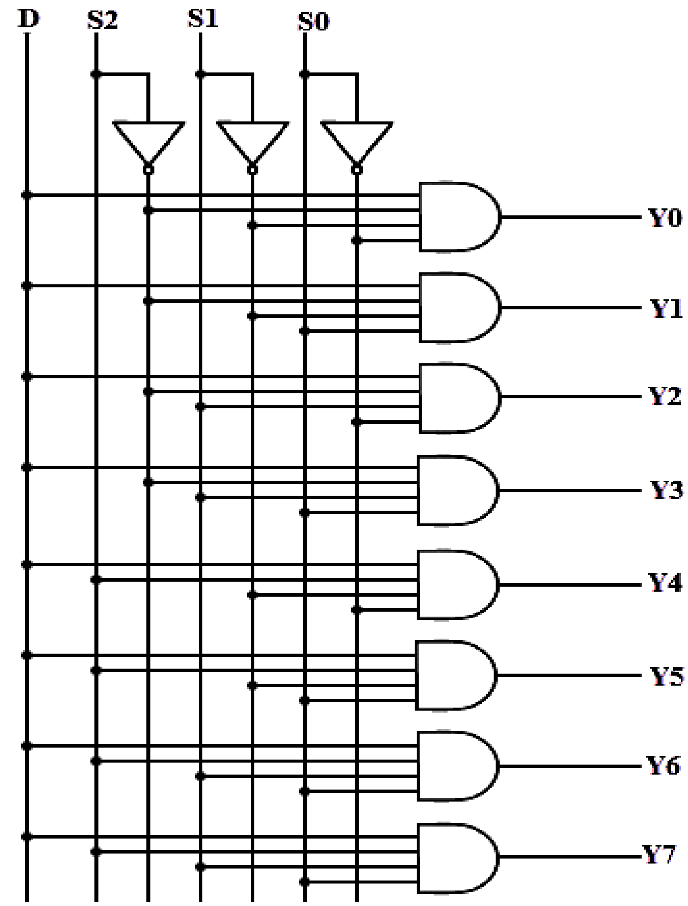
$$Y4 = D\,S2\,\overline{S1}\,\overline{S0}$$

$$Y5 = D\,S2\,\overline{S1}\,S0$$

$$Y6 = D\,S2\,S1\,\overline{S0}$$

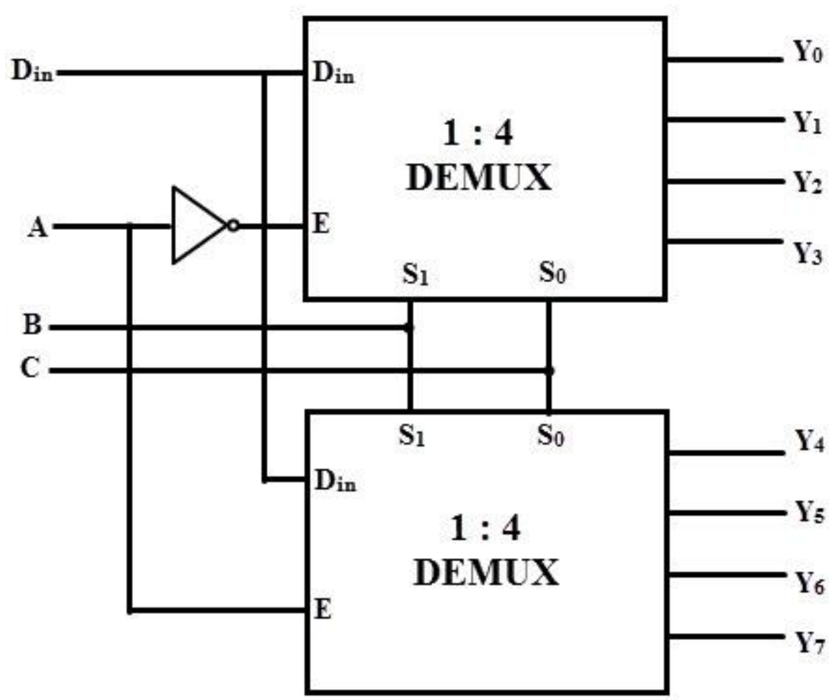
$$Y7 = D\,S2\,S1\,S0$$

Logic Diagram


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

1-to-8 DEMUX using Two 1-to- 4 Demultiplexers

1-to-8 demultiplexer can be implemented by using two 1-to-4 demultiplexers with a proper cascading.



In the above figure, the highest significant bit A of the selection inputs are connected to the enable inputs such that it is complemented before connecting to one DEMUX and to the other it is directly connected.By this configuration, when A is set to zero, one of the output lines from Y0 to Y3 is selected based on the combination of select lines B and C. Similarly, when A is set to one, based on the select lines one of the output lines from Y4 to Y7 will be selected.

2.9.1 Applications of Demultiplexer

- Synchronous data transmission systems
- Boolean function implementation (as we discussed full subtractor function above)
- Data acquisition systems
- Combinational circuit design
- Automatic test equipment systems

- Security monitoring systems (for selecting a particular surveillance camera at a time), etc.

2.10 CODE CONVERTORS

Numbers are usually coded in one form or another so as to represent or use it as required. For instance, a number 'nine' is coded in decimal using symbol (9)d. Same is coded in natural-binary as (1001)b. While digital computers all deal with binary numbers, there are situations wherein natural-binary representation of numbers in in-convenient or in-efficient and some other (binary) code must be used to process the numbers.

One of these other code is gray-code, in which any two numbers in sequence differ only by one bit change. This code is used in K-map reduction technique. The advantage is that when numbers are changing frequently, the logic gates are turning ON and OFF frequently and so are the transistors switching which characterizes power consumption of the circuit; since only one bit is changing from number to number, switching is reduced and hence is the power consumption.

Let's discuss the conversion of various codes from one form to other.

2.10.1 BINARY-TO-GRAY

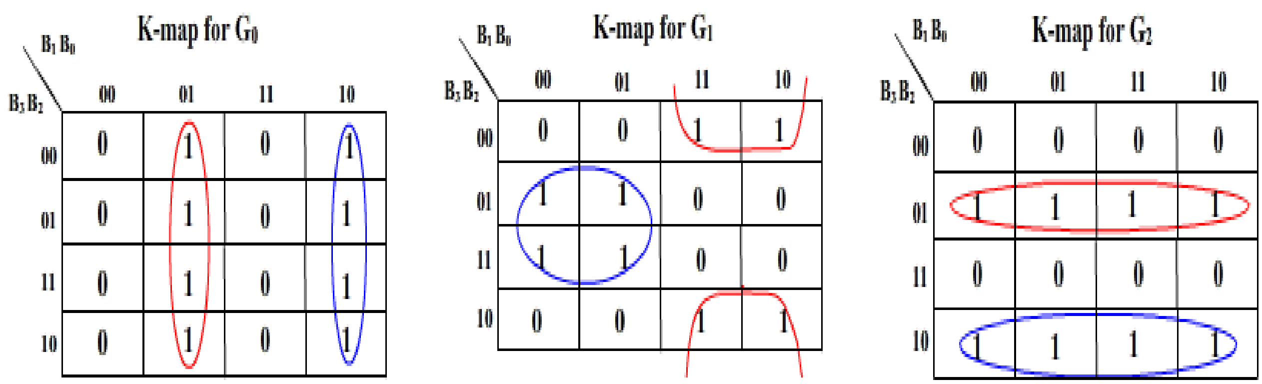
The table that follows shows natural-binary numbers (upto 4-bit) and corresponding gray codes.

Natural-binary code				Gray code			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Looking at gray-code (G3G2G1G0), we find that any two subsequent numbers differ in only one bit-change.

The same table is used as truth-table for designing a logic circuitry that converts a given 4-**bit** natural binary number into gray number. For this circuit, B3 B2 B1 B0 are inputs while G3 G2 G1 G0 are outputs.

K-map for the outputs:

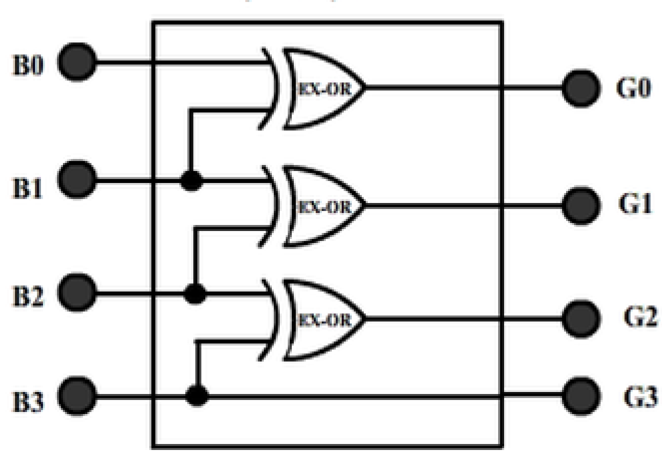


$G_0 = B_1'B_0 + B_1B_0'$ $\qquad$ $G_1 = B_1'B_2 + B_1B_2'$ $\qquad$ $G_2 = B_3'B_2 + B_3B_2'$

$G_0 = B_0 \oplus B_1$ $\qquad$ $G_2 = B_1 \oplus B_2$ $\qquad$ $G_2 = B_2 \oplus B_3$

And **G3 = B3**

Binary to Gray Converter



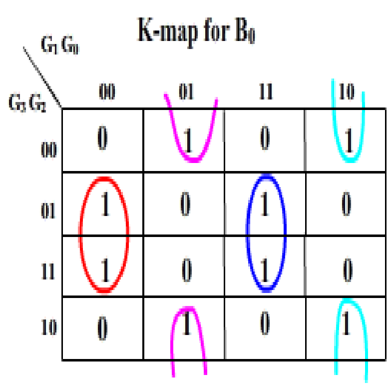
So that's a simple three EX-OR gate circuit that converts a 4-bit input binary number into its equivalent 4-bit gray code. It can be extended to convert more than 4-bit binary numbers.

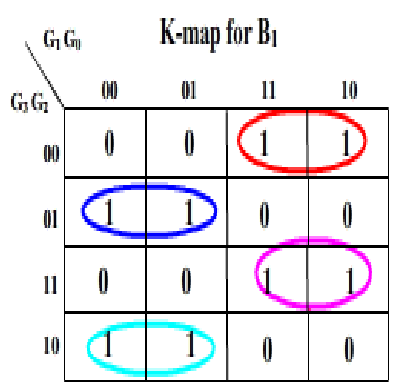
2.10.2 Gray-to-Binary

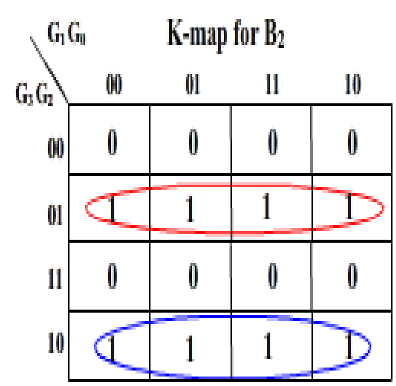
Truth-table:

Gray code				Natural-binary code			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

Then the K-maps:







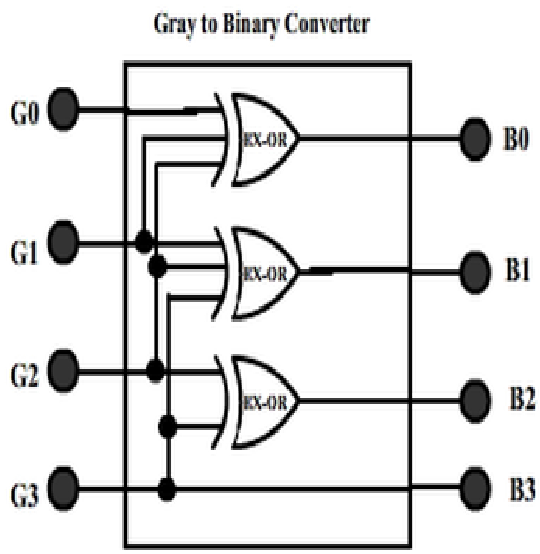
$$B_0 = G_2G_1'G_0' + G_2'G_1G_0' + G_2'G_1'G_0 + G_2G_1G_0$$
$$= G_0'(G_1'G_2 + G_1G_2') + G_0(G_1G_2 + G_1'G_2')$$
$$= G_0'(G \oplus G_2) + G_0(G_1 \oplus G_2)' = G_0 \oplus G_1 \oplus G_2$$

$$B_1 = G_3'G_2'G_1 + G_3'G_2G_1' + G_3G_2G_1 + G_3G_2'G_1'$$
$$= G_3'(G_2 \oplus G_1) + G_3(G_2 \oplus G_1)'$$
$$= G_1 \oplus G_2 \oplus G_3$$

$$B_2 = G_3'G_2 + G_3G_2'$$
$$= G_3 \oplus G_2$$

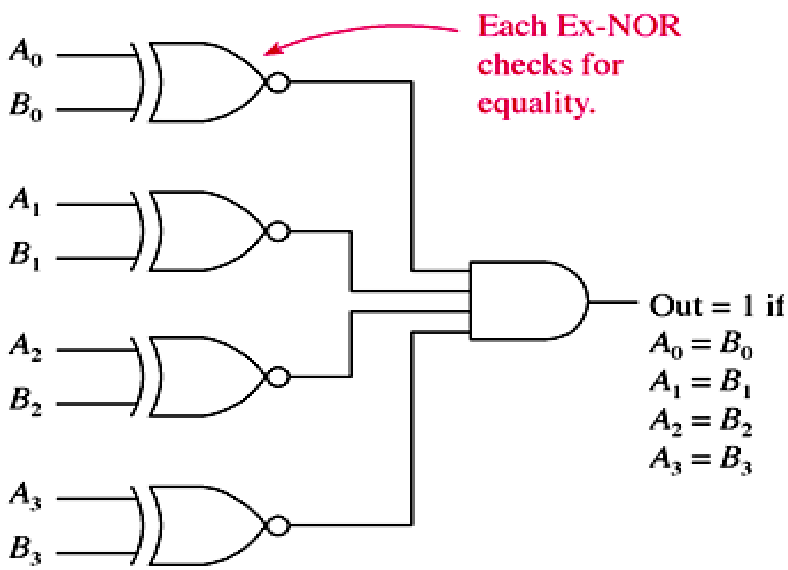
And B3 = G3

The realization of Gray-to-Binary converter is

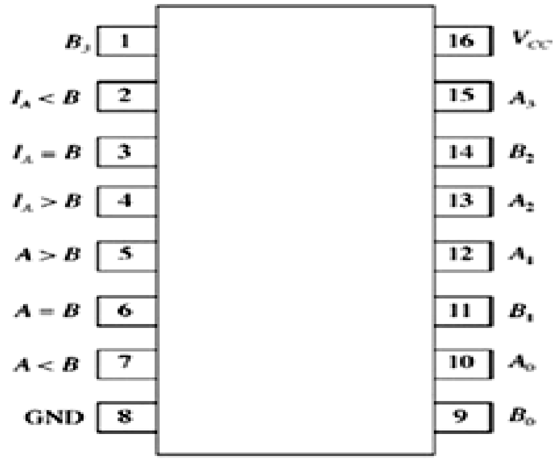


2.11 Comparators

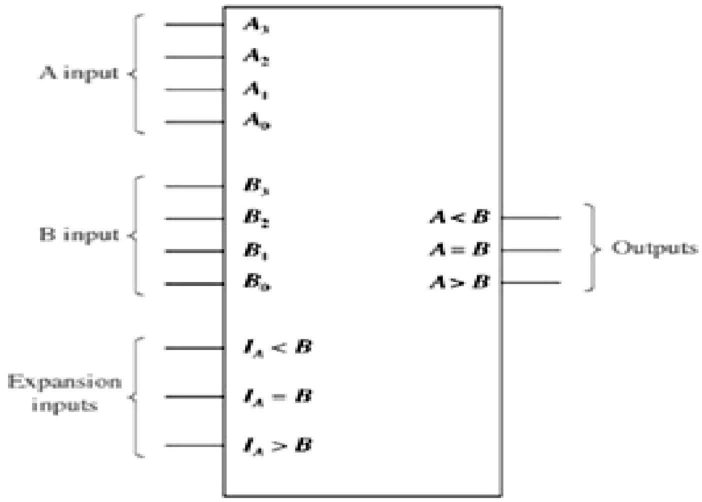
- A comparator will evaluate two binary strings and output a 1 if the two strings are exactly the same.
- The Exclusive-NOR (Equality gate) is used to perform the comparison.
- One Exclusive-NOR is used per pair of Binary bits and the outputs of all Exclusive-NORS are ANDed together.



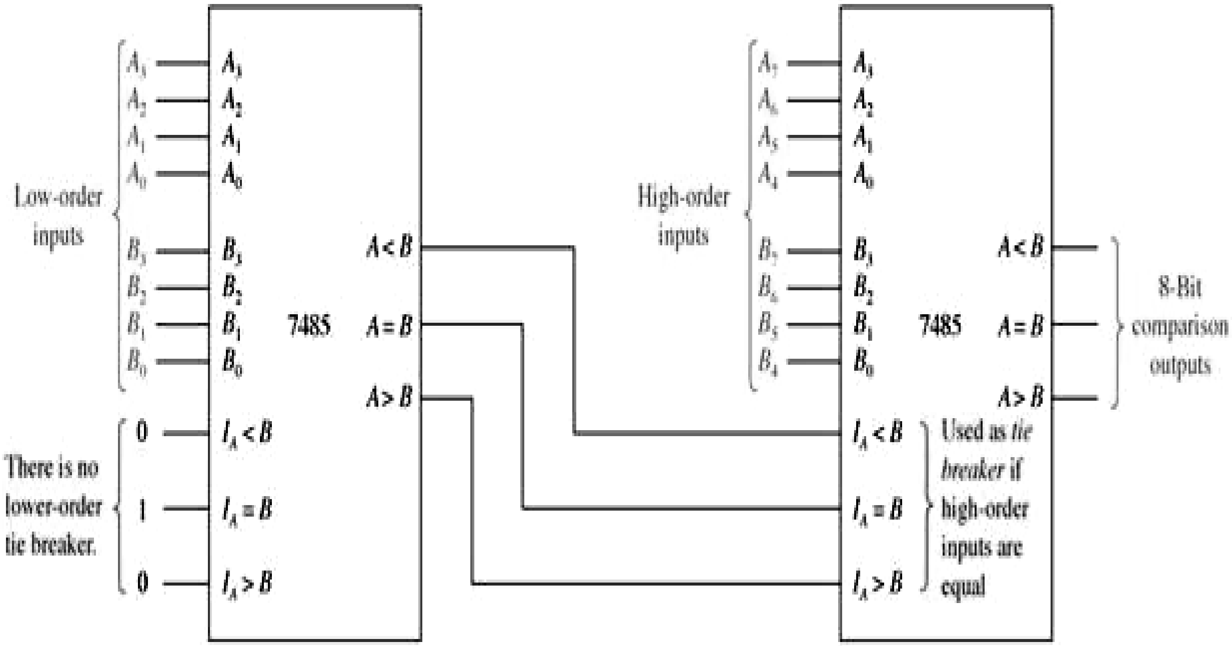
- The 7485 is a 4-bit magnitude comparator.
  A magnitude comparator will determine if A = B, A > B or A < B.



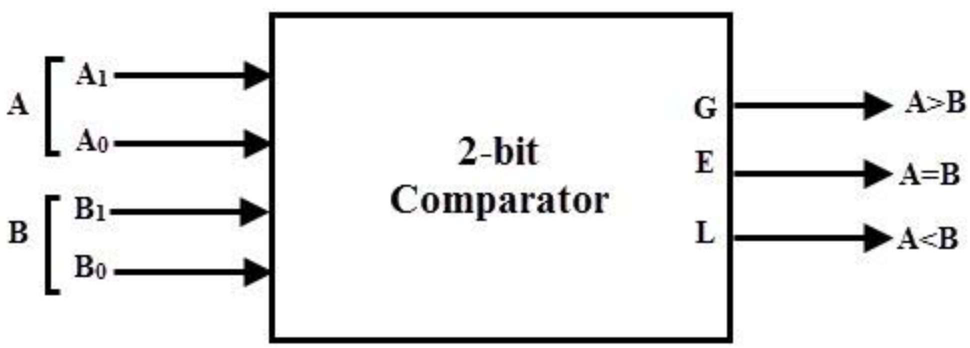
(a)



- Expansion inputs are provided on the 7483 so that word sizes larger then 4-bits may be compared.



Magnitude Comparator
Definition
A magnitude comparator is a combinational circuit that compares two numbers A & B to determine whether:
A > B, or
A = B, or
A < B
2-bit magnitude comparator



Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

4-bit magnitude comparator

Inputs: 8-bits (A ⇒ 4-bits , B ⇒ 4-bits)

A and B are two 4-bit numbers

_ Let A = A3A2A1A0 , and

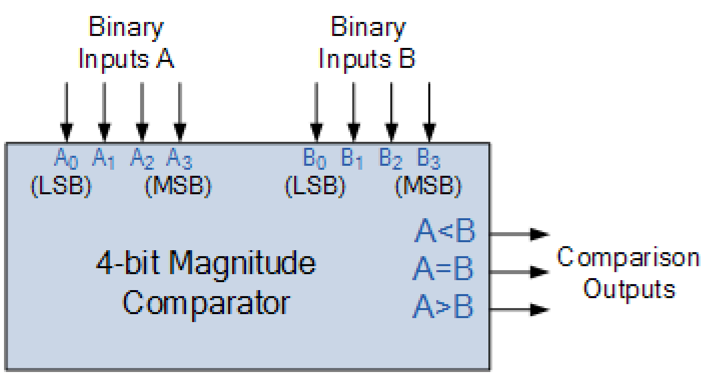
_ Let B = B3B2B1B0

_ Inputs have *28* (*256*) possible combinations

_ Not easy to design using conventional techniques

The circuit possesses certain amount of regularity⇒ *can be designed algorithmically*.

Design of the EQ output (A = B) in 4-bit magnitude comparator



INPUTS								OUTPUTS		
A3	A2	A1	A0	B3	B2	B1	B0	A > B	A = B	A < B
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	1
0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	1	0	1	0	0	1
0	0	0	0	0	1	1	0	0	0	1
0	0	0	0	0	1	1	1	0	0	1
0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	1	0	0	1	0	0	1
0	0	0	0	1	0	1	0	0	0	1
0	0	0	0	1	0	1	1	0	0	1
0	0	0	0	1	1	0	0	0	0	1
0	0	0	0	1	1	0	1	0	0	1
0	0	0	0	1	1	1	0	0	0	1
0	0	0	0	1	1	1	1	0	0	1
0	0	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0	1	0
0	0	0	1	0	0	1	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1
⋮			⋮				⋮			⋮
1	1	1	1	1	1	1	1	0	1	0

Assign $X_n = \overline{A_n \oplus B_n}$

$(A > B) \rightarrow A_3\overline{B_3} + X_3A_2\overline{B_2} + X_3X_2A_1\overline{B_1} + X_3X_2X_1A_0\overline{B_0}$

$(A < B) \rightarrow \overline{A_3}B_3 + X_3\overline{A_2}B_2 + X_3X_2\overline{A_1}B_1 + X_3X_2X_1\overline{A_0}B_0$

$(A = B) \rightarrow X_3X_2X_1X_0$