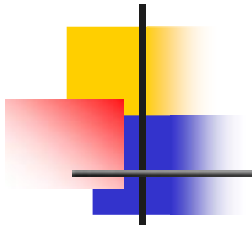


Unit-III

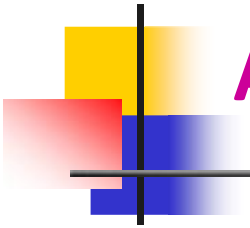
Combinational Logic Design

Digital Electronics

(Second Year B. Tech program in Computer Engineering)

Unit-III Combinational Logic Design — 08 Hrs.

Introduction: Half and Full Adder, Half and Full Subtractor, 4-bit Ripple Adder, Look-Ahead Carry Adder, 4-bit Adder and Subtractor, 1- digit BCD Adder, Multiplexers, Multiplexer Tree, Demultiplexers, Demultiplexer Tree, Encoders, Priority Encoder, Decoders. Comparators: 1-bit, 2-bit, 4-bit Magnitude Comparators, ALU IC 74181.



Adder

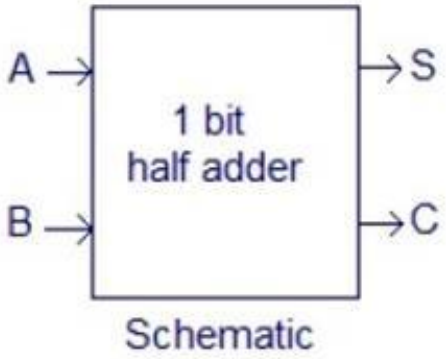
1. An adder circuit is one of the important digital circuits used in computers, calculators, digital processing units, etc.
2. There are two types adder circuits named half-adder and full-adder.
3. Both the half adder and the full adder circuits are used to perform addition and also widely used for performing various arithmetic functions in the digital circuits.

Half Adder

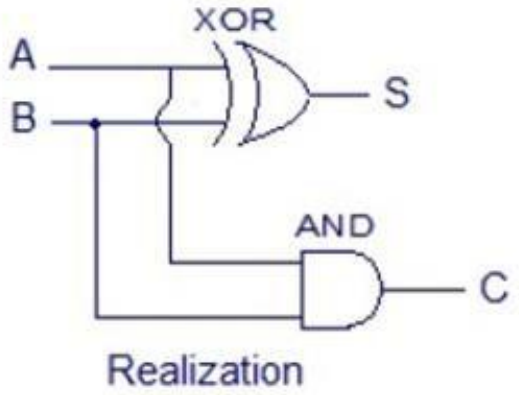
1. A combinational logic circuit which is designed to add two binary digits is known as half adder.
2. The half adder provides the output along with a carry value (if any).
3. The half adder circuit is designed by connecting an EX-OR gate and one AND gate. It has two input terminals and two output terminals for sum and carry.

Adder

Half Adder :



Schematic

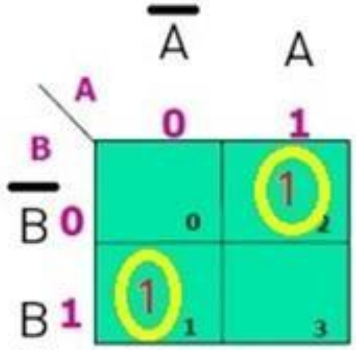


Realization

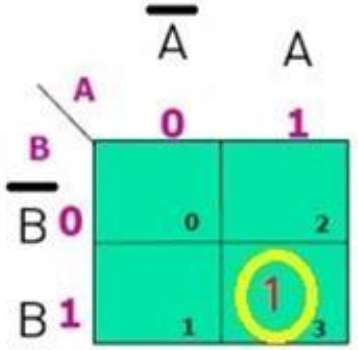
Dec No	Inputs		Outputs	
	A	B	S	C
0	0	0	0	0
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1

Truth table

KMAP for Sum



KMAP for Carry



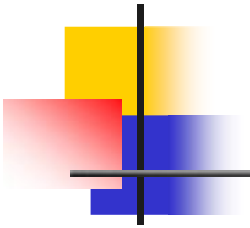
Sum = A'B + AB'

= A ⊕ B

Carry = AB

Drawbacks:

1. It is used for addition of single bit numbers only.
2. It doesnt considers a carry generated from previous additions.

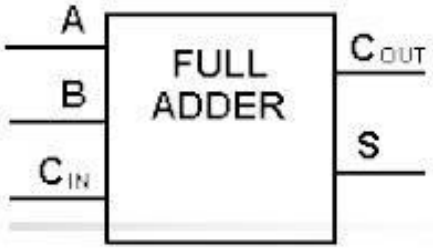


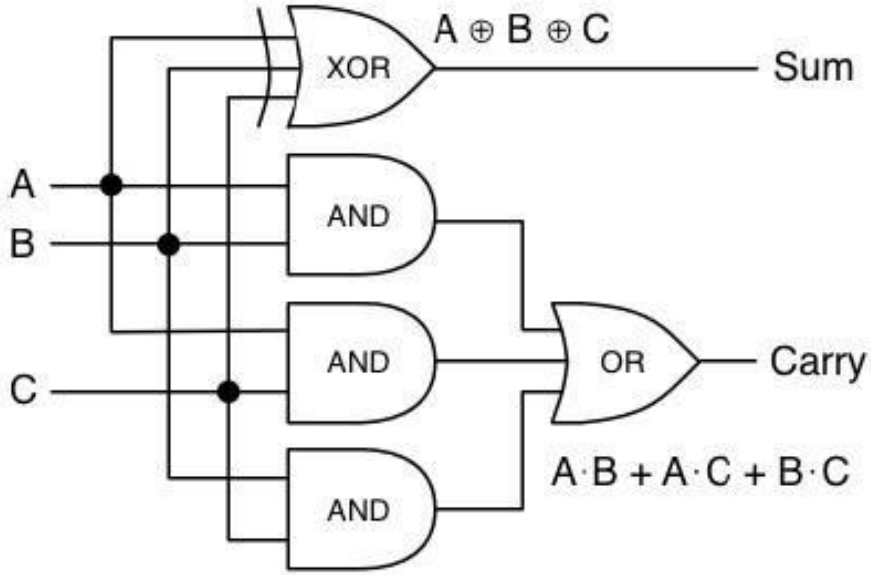
Full Adder

1. The half adder is used to add only two numbers. To overcome this problem, the full adder was developed.
2. A combinational circuit which is designed to add three binary digits and produce two outputs is known as full adder.
3. The full adder circuit adds three binary digits, where two are the inputs and one is the carry forwarded from the previous addition.

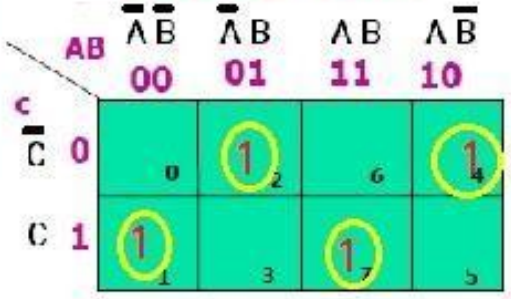
Full Adder :

A	B	Carry In	Sum	Carry 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



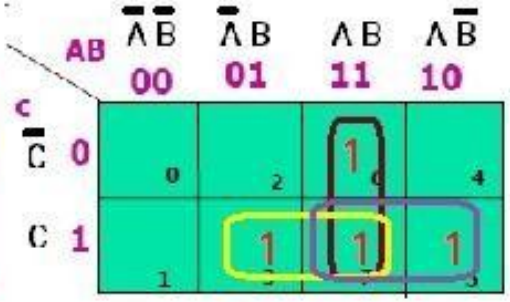


KMAP for Sum

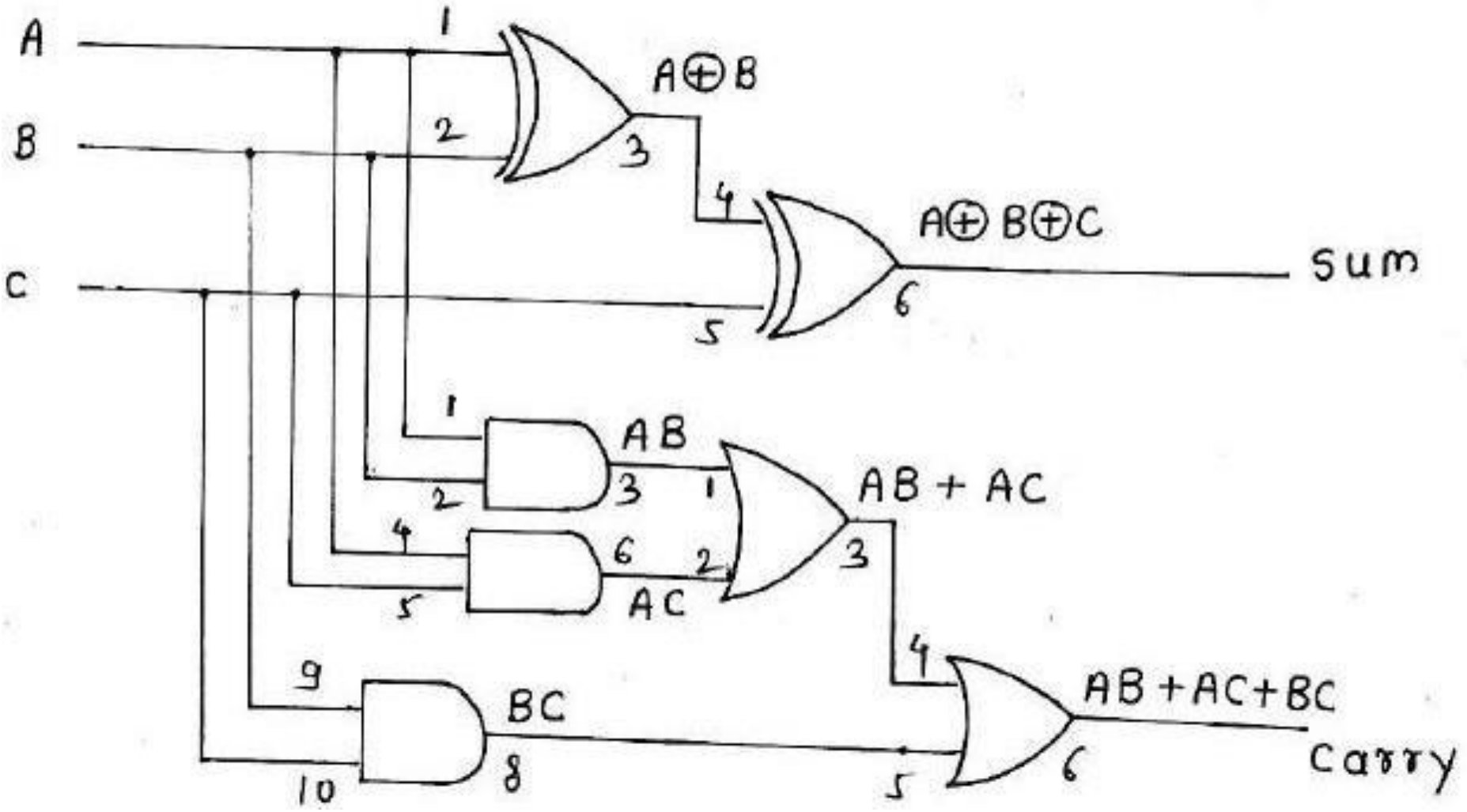


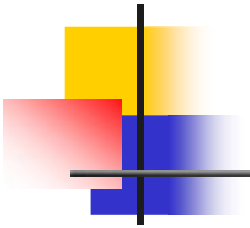
Sum = A'B'C+A'BC'+ABC+AB'C'
= A ⊕ B ⊕ C

KMAP for Carry



Carry = AB + BC + AC


A
B
C
1
2
3
A⊕B
4
5
6
A⊕B⊕C
Sum
1
2
3
AB
1
2
3
AB + AC
4
5
6
AC
9
10
8
BC
4
5
6
AB+AC+BC
Carry



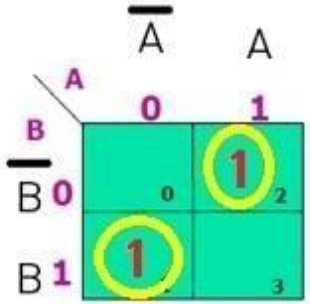
Half Subtractor

1. The half subtractor is also a building block for subtracting two binary numbers. It has two inputs and two outputs.
2. This circuit is used to subtract two single bit binary numbers A and B.
3. The 'diff' and 'borrow' are two output states of the half subtractor.

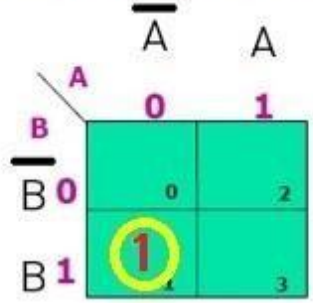
Half Substractor :

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

Kmap for Diff



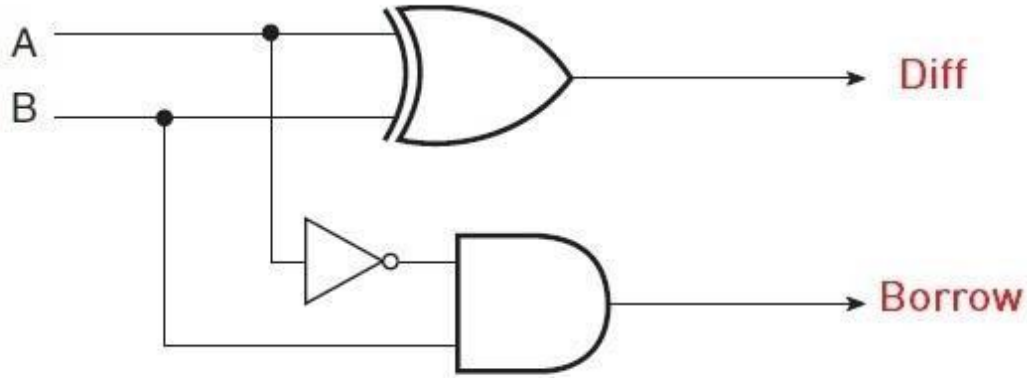
Kmap for Borrow

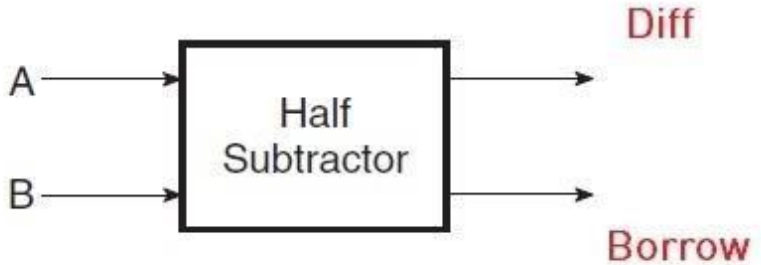


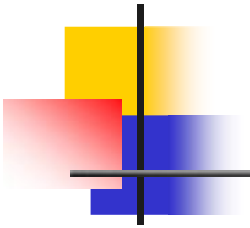
Diff = A'B + AB'

= A ⊕ B

Borrow = A'B







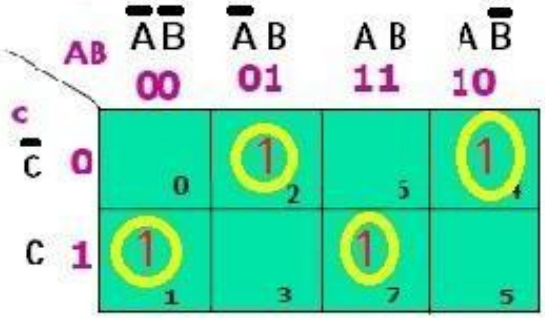
Full Subtractor

1. The Half Subtractor is used to subtract only two numbers.
2. To overcome this problem, a full subtractor was designed.
3. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively.
4. The full subtractor has three input states and two output states i.e., diff and borrow.

Full Substractor :

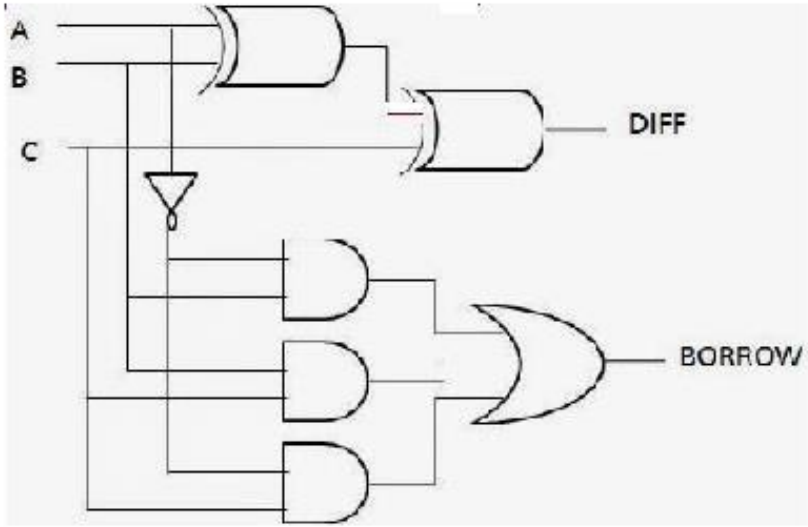
	Inputs			Outputs	
Dec No	A	B	C	Diff	Borrow
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

Kmap for Diff

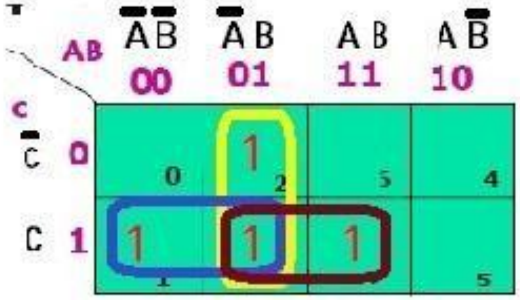


Diff = A'B'C+A'BC'+ABC+AB'C'

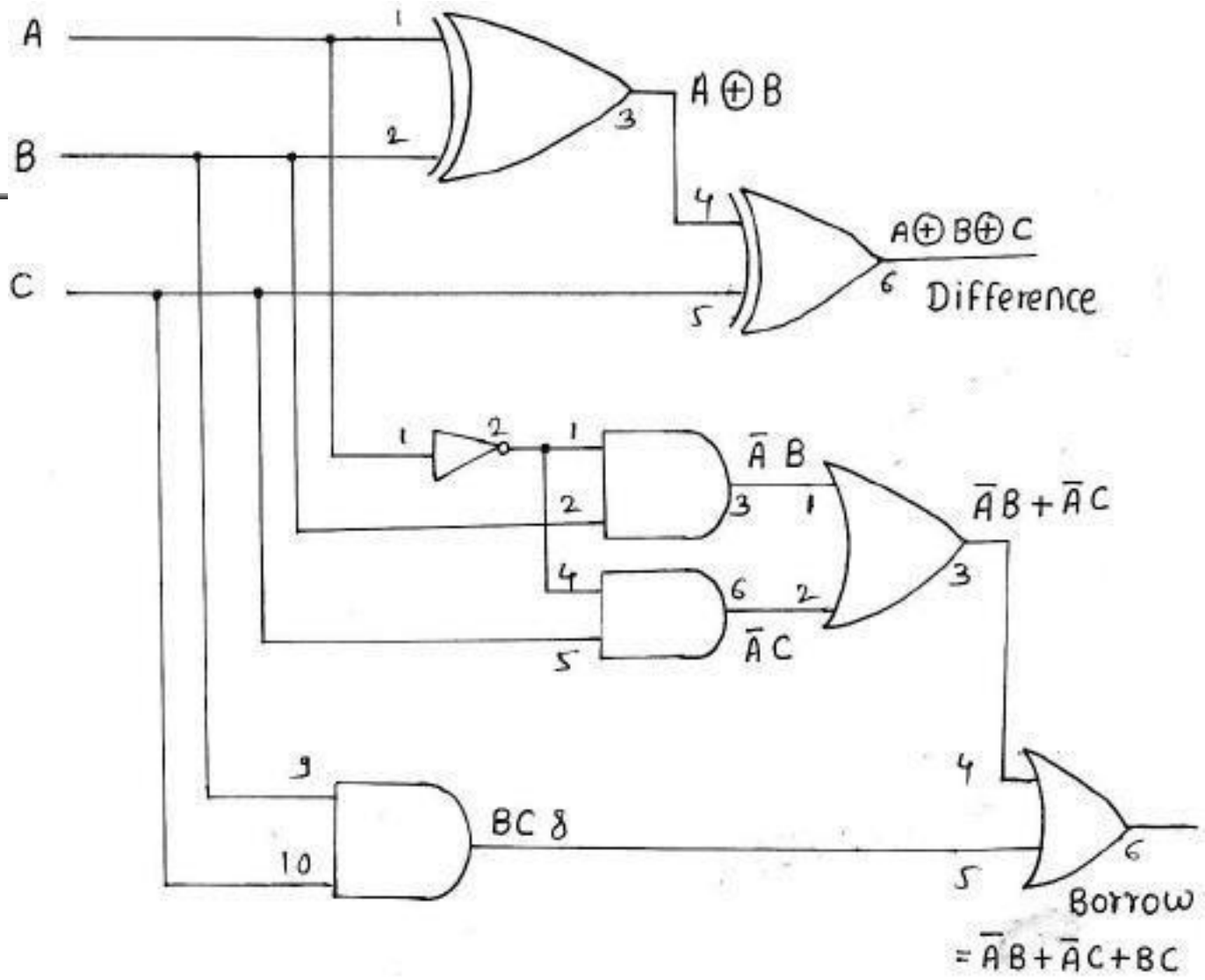
= A ⊕ B ⊕ C



Kmap for Borrow



Borrow =A'B+A'C +BC


A
B
C
A ⊕ B
A ⊕ B ⊕ C
Difference
Ā B
Ā B + Ā C
Ā C
BC
Borrow
= Ā B + Ā C + BC

Equations for Half Adder

Sum = A'B + AB'

= A ⊕ B

Carry = AB

Equations for Half substractor

Diff = A'B + AB'

= A ⊕ B

Borrow = A'B

Equations for Full Adder

Sum = A'B'C+A'BC'+ABC+AB'C'

= A ⊕ B ⊕ C

Carry = AB + BC + AC

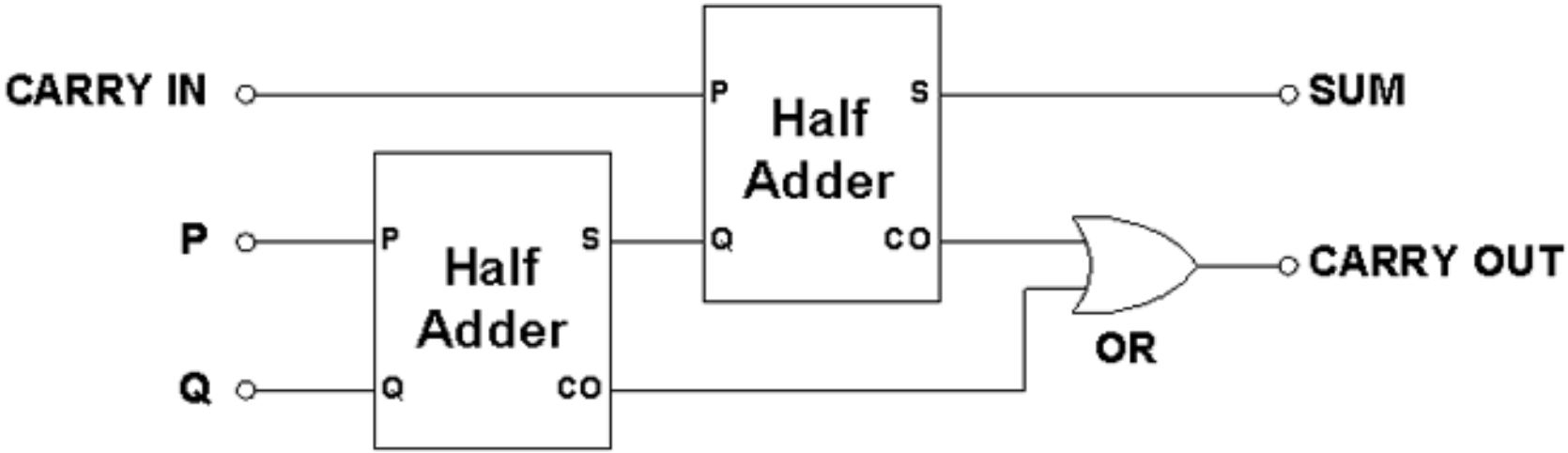
Equations for Full Substrator

Diff = A'B'C+A'BC'+ABC+AB'C'

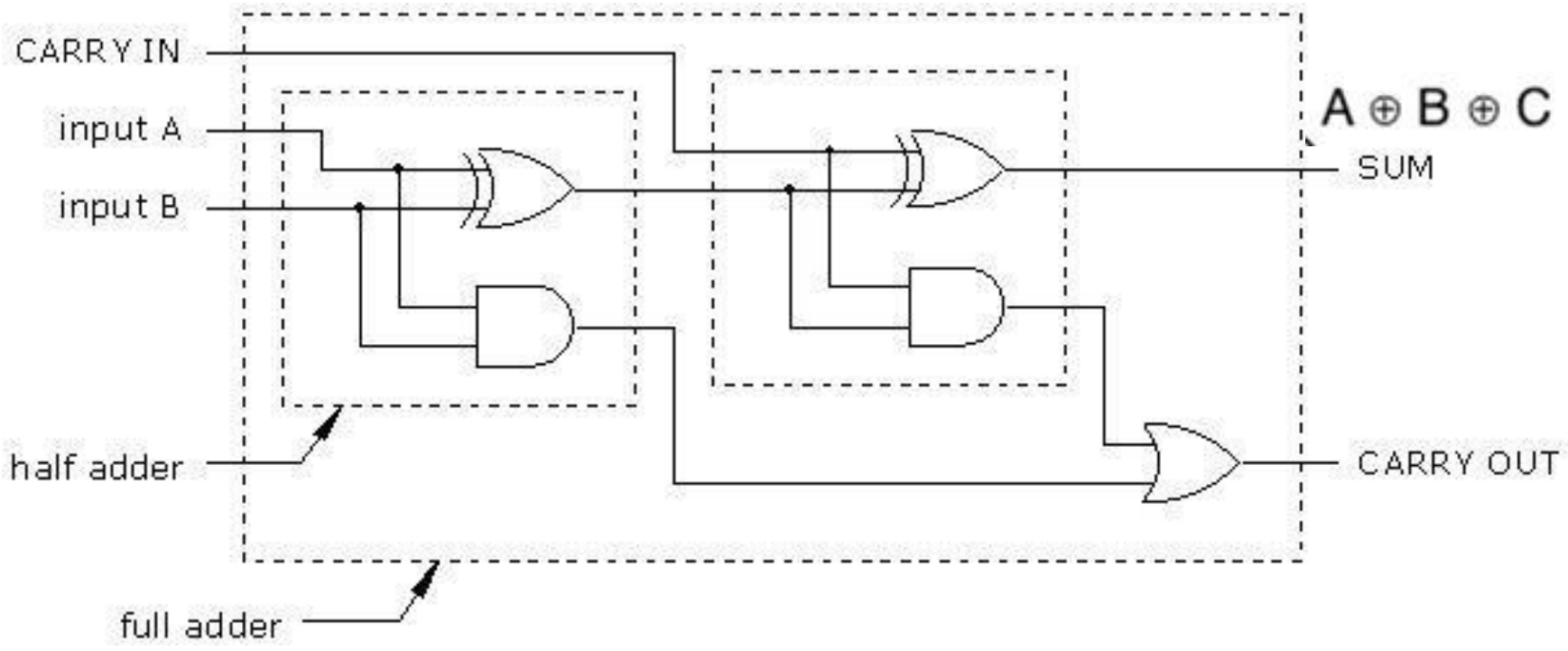
= A ⊕ B ⊕ C

Borrow =A'B+A'C +BC

Carry Look Ahead Adder/ Fast adder/ Full Adder using two half adder & one OR gate :



Full Adder using two half adder & one or gate



Sum - $A \oplus B \oplus C$ Carry - $A \oplus B \cdot C + A \cdot B$

Full Substractor using two half subtractor & one OR gate :

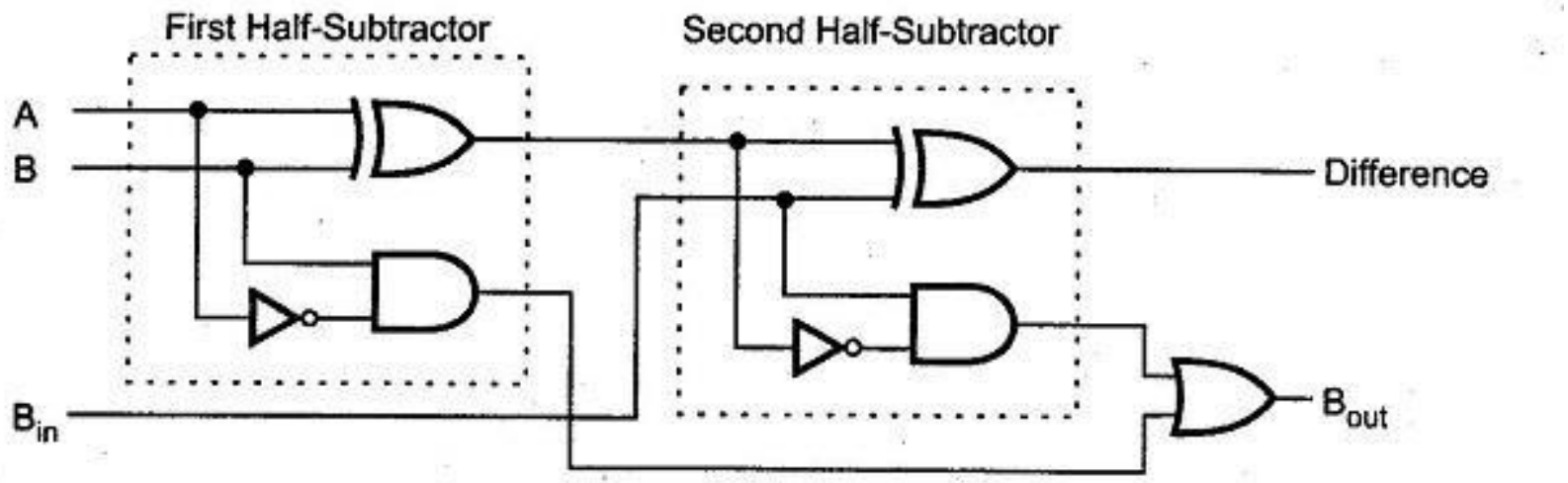
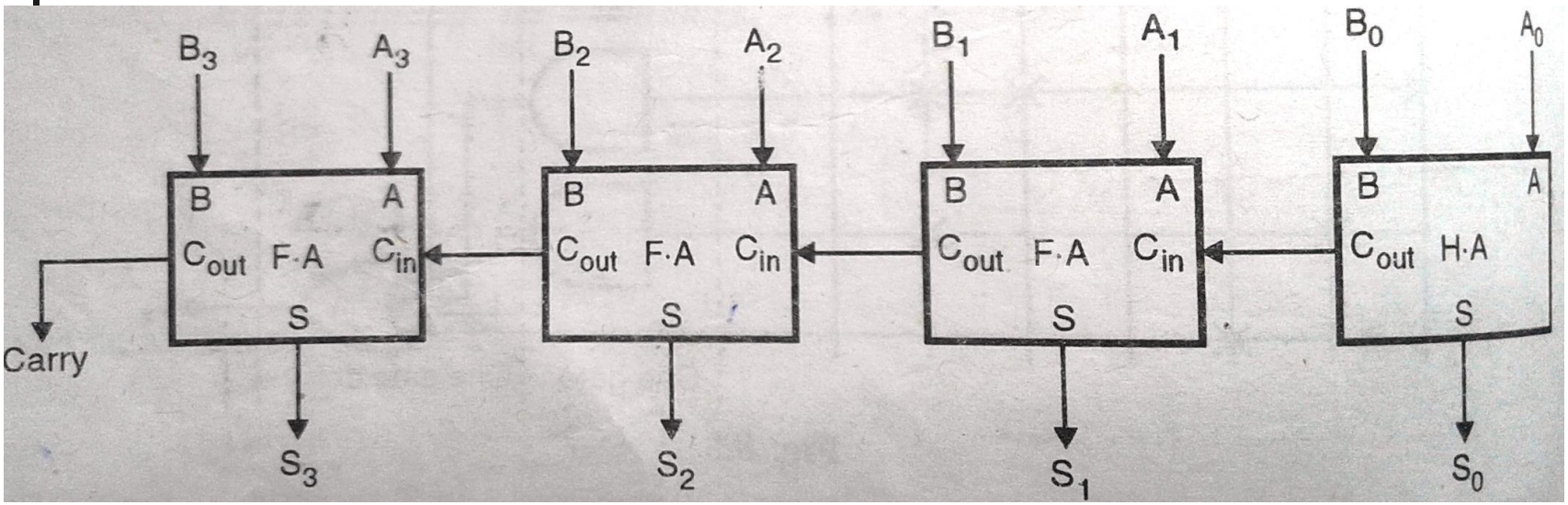
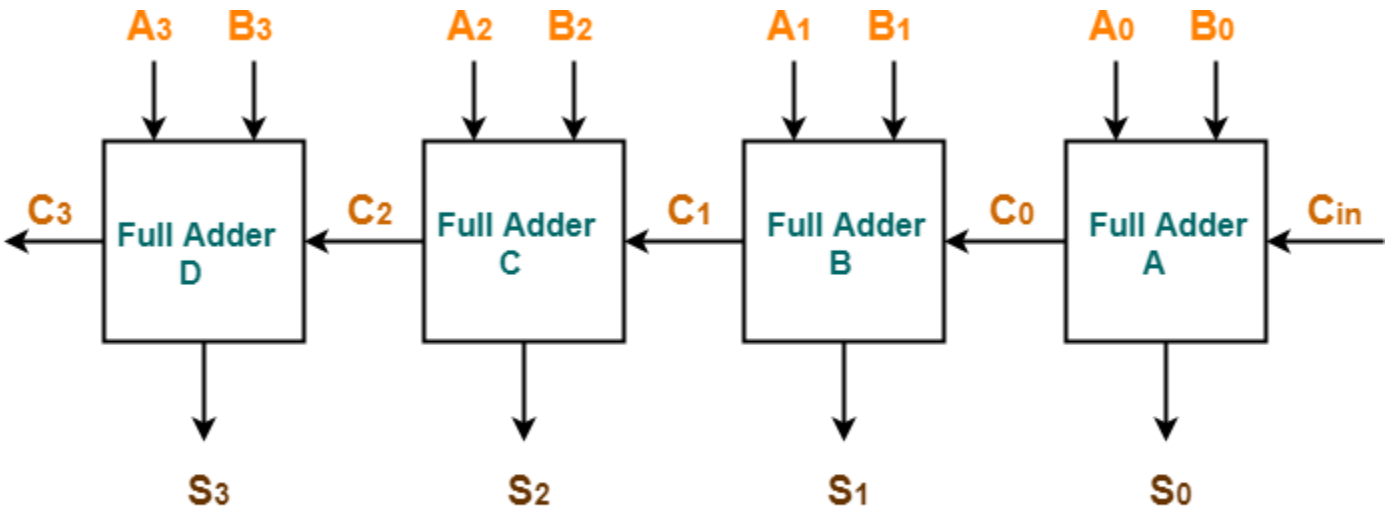


Fig. 3.24 Implementation of a full-subtractor with two half-subtractors and an OR gate

Full Adder using 3 full adder & 1 half adder

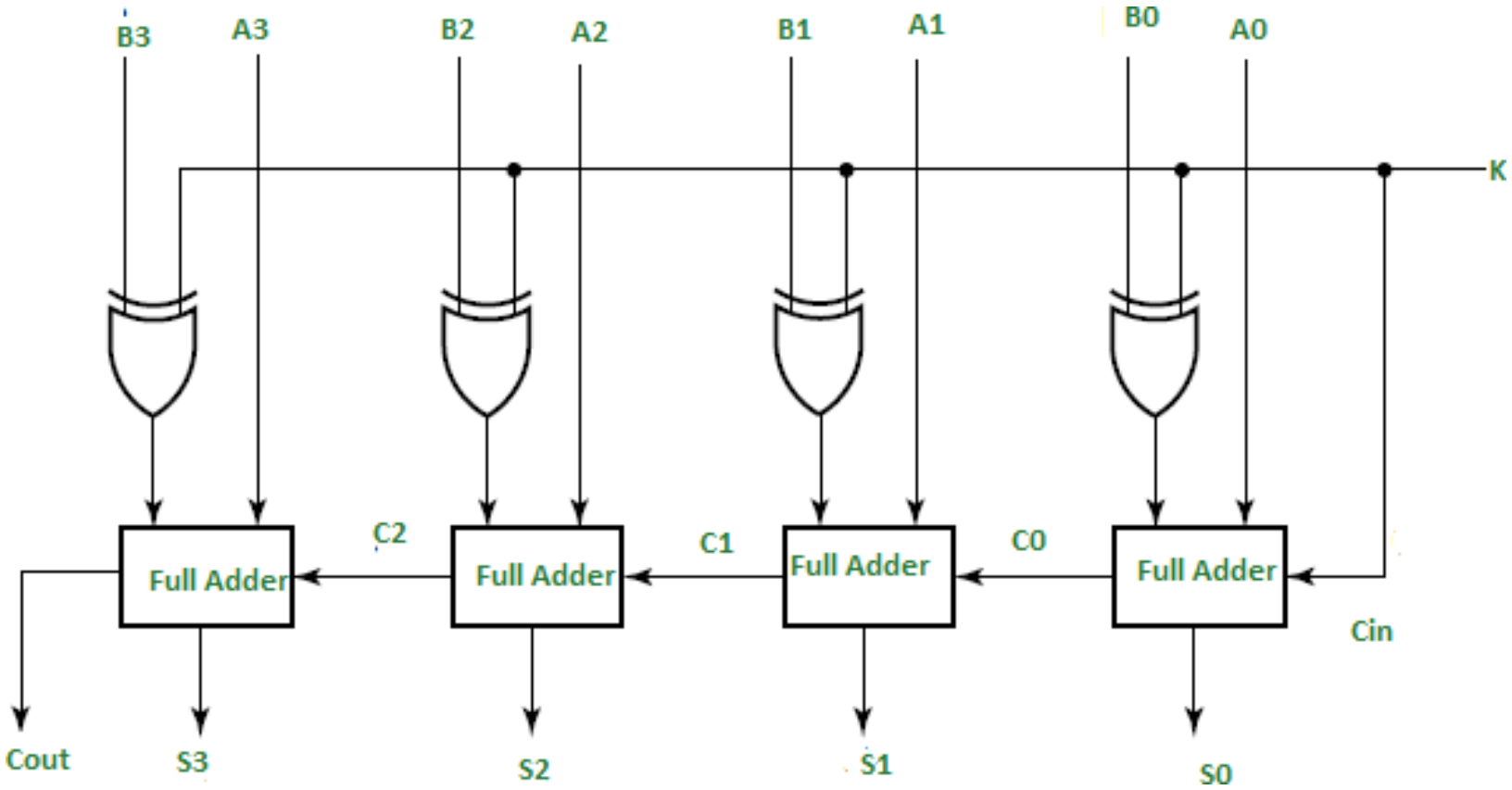


Full Adder using 4 full adder/ Ripple Carry Adder/ Parallel Adder



4-bit Ripple Carry Adder

4 bit Adder Subtractor



Carry =0, Adder and Carry =1, Subtractor

https://www.youtube.com/watch?v=9R_lPPAwht0

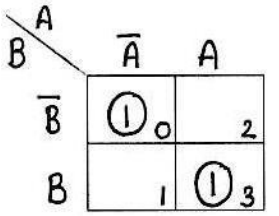
1 bit Comparator:

Truth Table:

Input		Output		
A	B	$Y_{A=B}$	$Y_{A>B}$	$Y_{A<B}$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

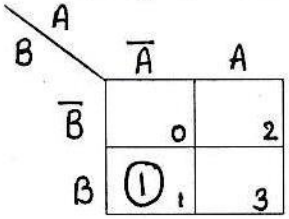
Kmap :

Kmap for $Y_{A=B}$



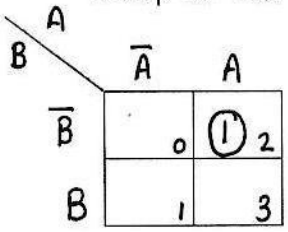
$Y_{A=B} = \bar{A}\bar{B} + AB$

Kmap for $Y_{A<B}$



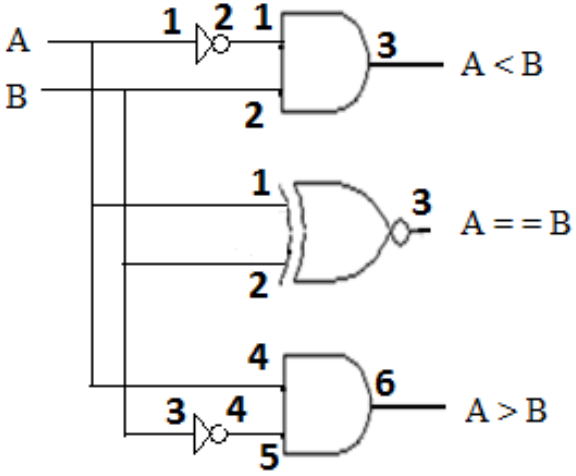
$Y_{A<B} = \bar{A}B$

Kmap for $Y_{A>B}$



$Y_{A>B} = A\bar{B}$

Circuit Diagram



1-bit magnitude comparator

2 bit Comparator:

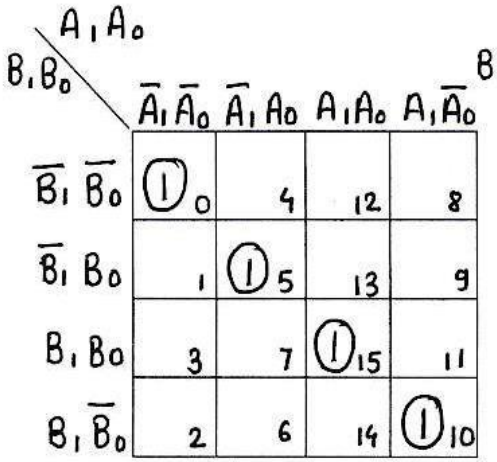
Truth Table:

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

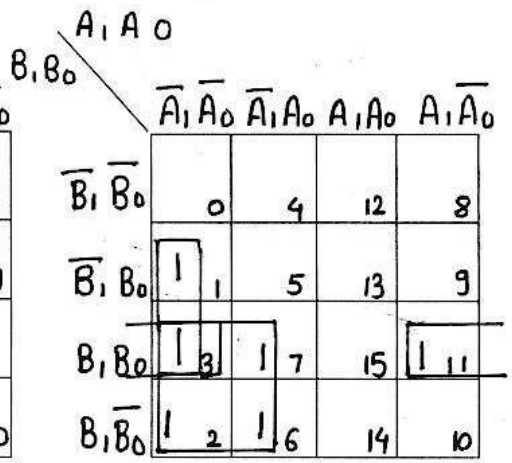
2 bit Comparator:

Kmap :

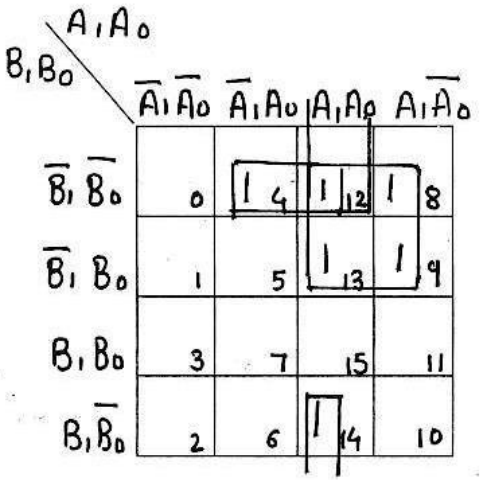
Kmap for $Y_{A=B}$



Kmap for $Y_{A<B}$



Kmap for $Y_{A>B}$



$$Y_{A<B} = \bar{A_1} \cdot B_1 + \bar{A_1}\,\bar{A_0} B_0 + \bar{A_0} B_0 B_1$$

$$Y_{A>B} = A_1 \bar{B_1} + A_0 \bar{B_0} \bar{B_1} + A_1 \bar{B_1} \bar{B_0}$$

$$Y_{A=B} = \bar{A_1}\,\bar{A_0}\,\bar{B_1}\,\bar{B_0} + \bar{A_1} A_0 \bar{B_1} B_0 + A_1 A_0 B_1 B_0 + A_1 \bar{A_0} B_1 \bar{B_0}$$

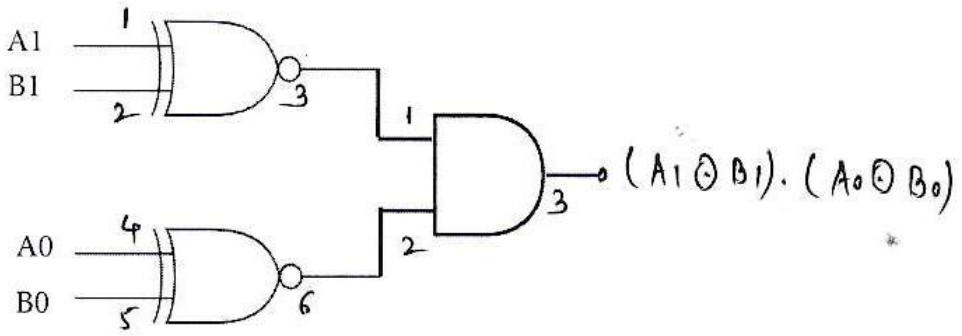
$$= \bar{A_1}\,\bar{B_1}(\bar{A_0}\bar{B_0} + A_0 B_0) + A_1 B_1 (A_0 B_0 + \bar{A_0}\bar{B_0})$$

$$= \bar{A_1}\,\bar{B_1}(A_0 \odot B_0) + A_1 B_1 (A_0 \odot B_0)$$

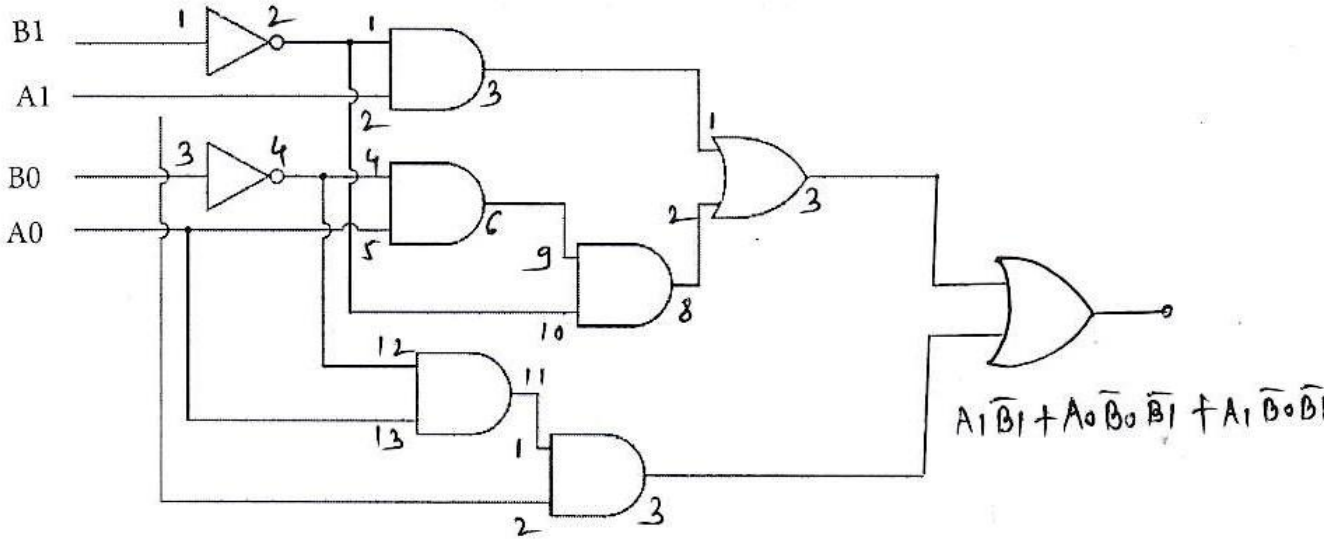
$$= (A_1 \odot B_1) \cdot (A_0 \odot B_0)$$

2 bit Comparator:

Circuit Diagram :



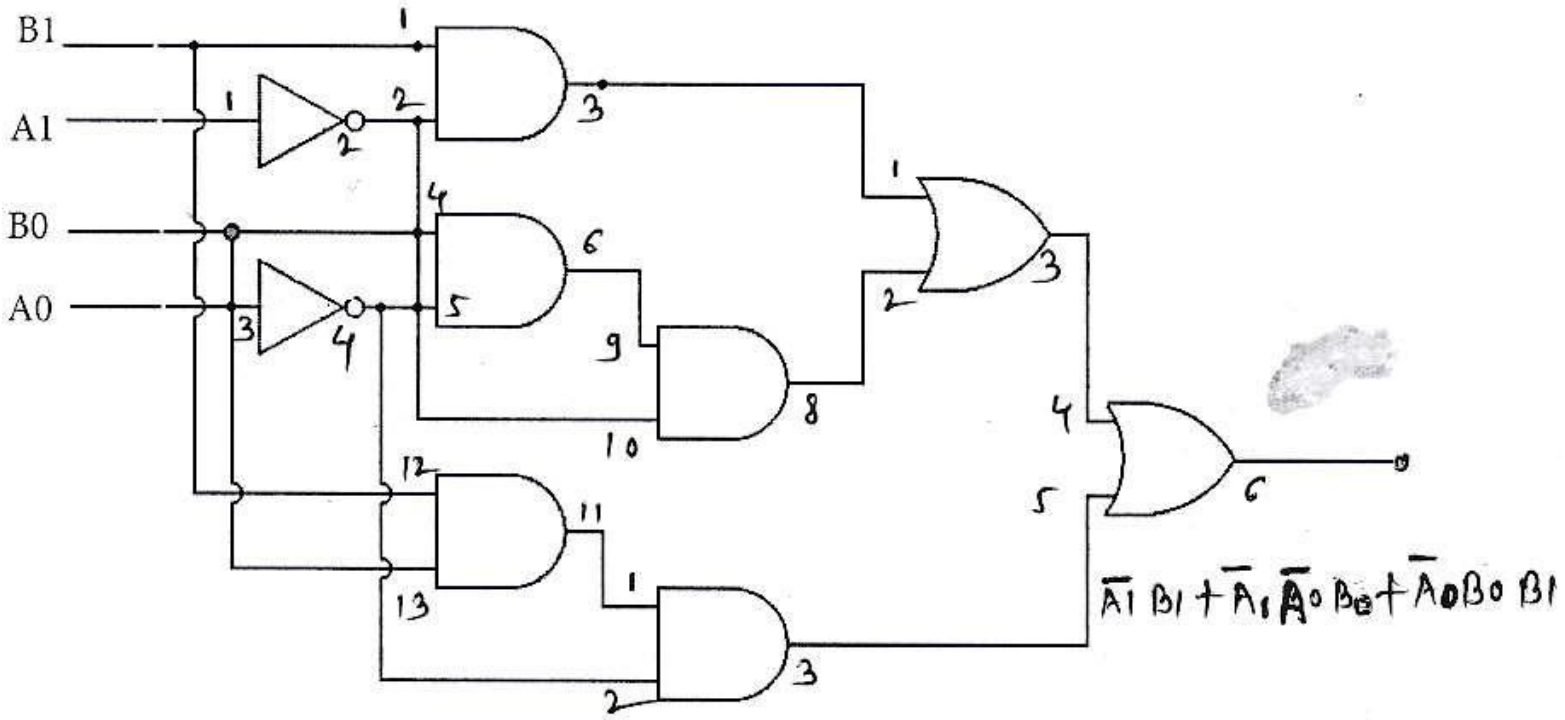
Circuit diagram for $Y_{A=B}$



Circuit diagram for $Y_{A>B}$

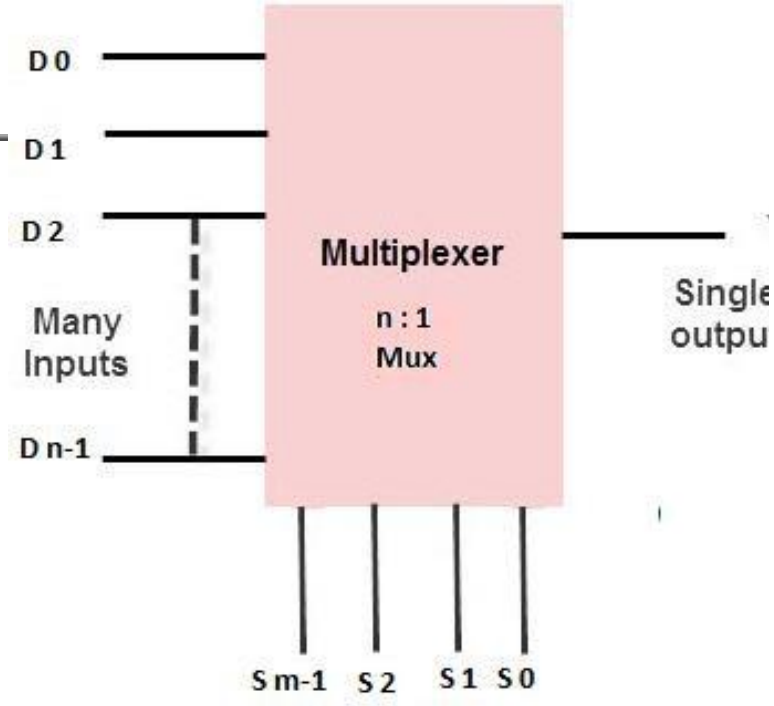
2 bit Comparator:

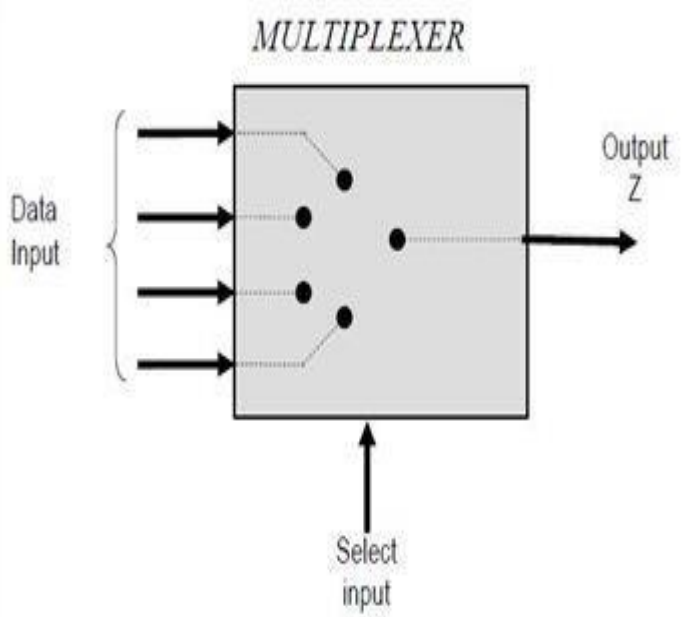
Circuit Diagram :



Circuit diagram for $Y_{A<B}$

Multiplexer (Data Selector) :

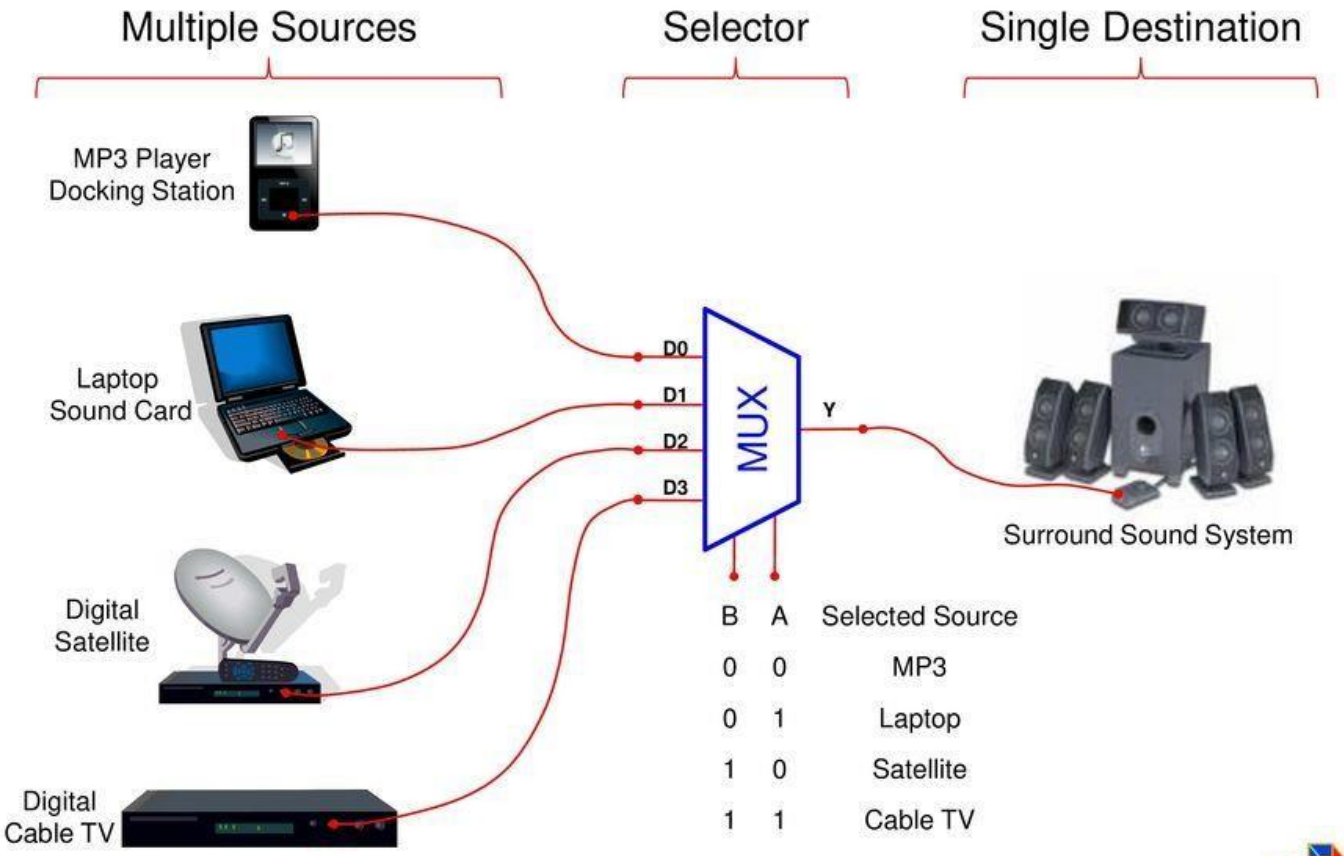


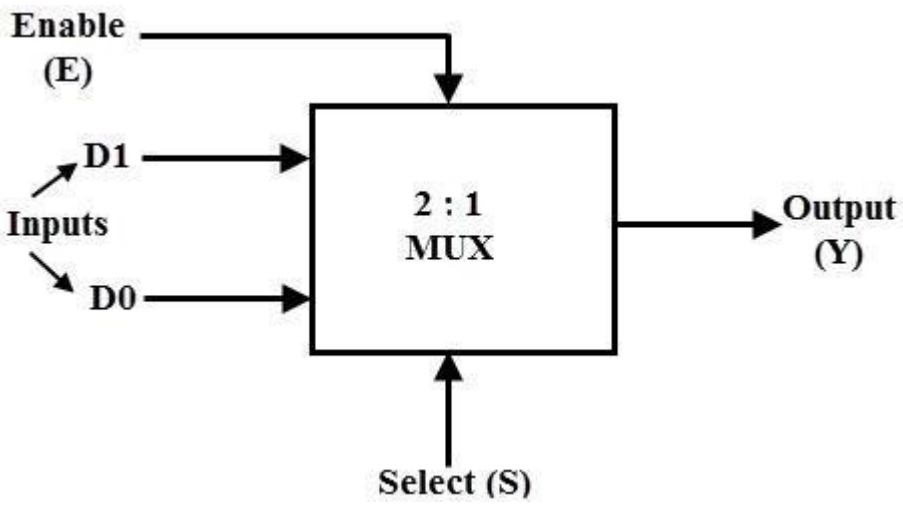


Logic circuit that selects binary information from one of its input line & direct it into a single output line.

Applications: Communication System, Computer Memory, Telephone Network , Transmission from the Computer System of a Satellite etc.

Typical Application of a MUX

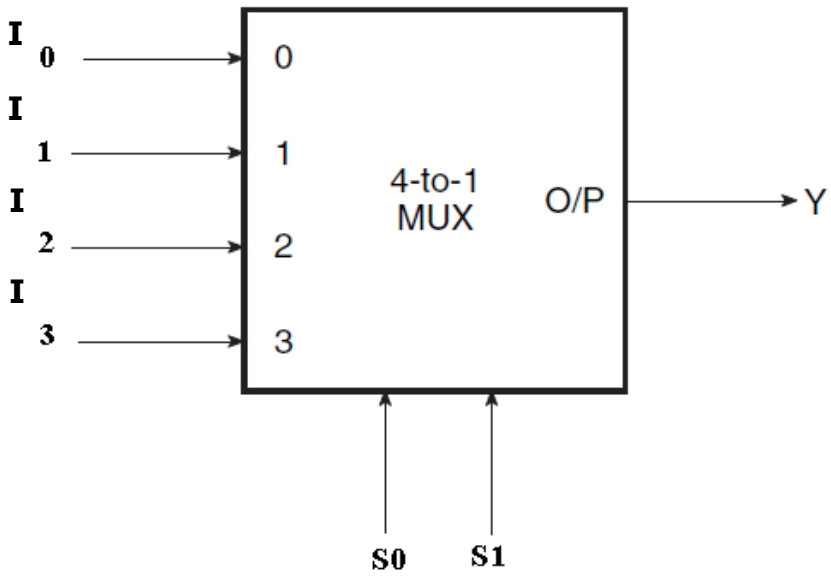




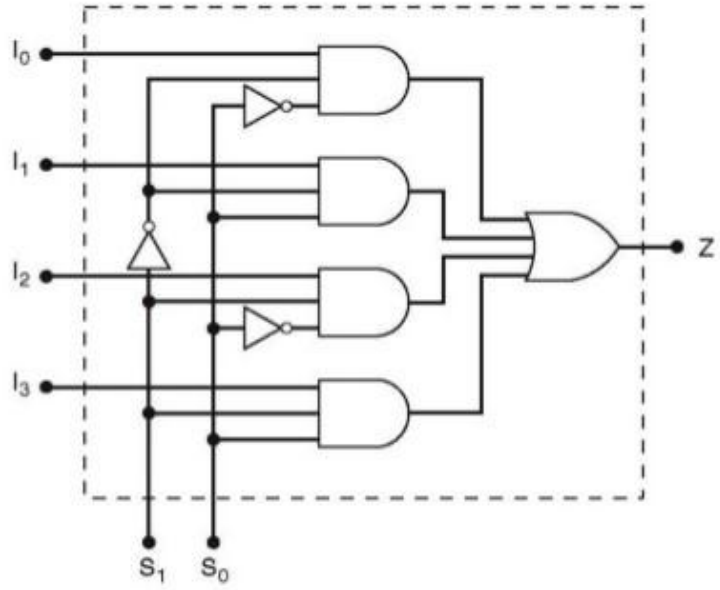
I/P S_0	O/P Y
0	D_0
1	D_1

$$Y = \overline{S}_0 D_0 + S_0 D_1$$

Multiplexer (Data Selector) :

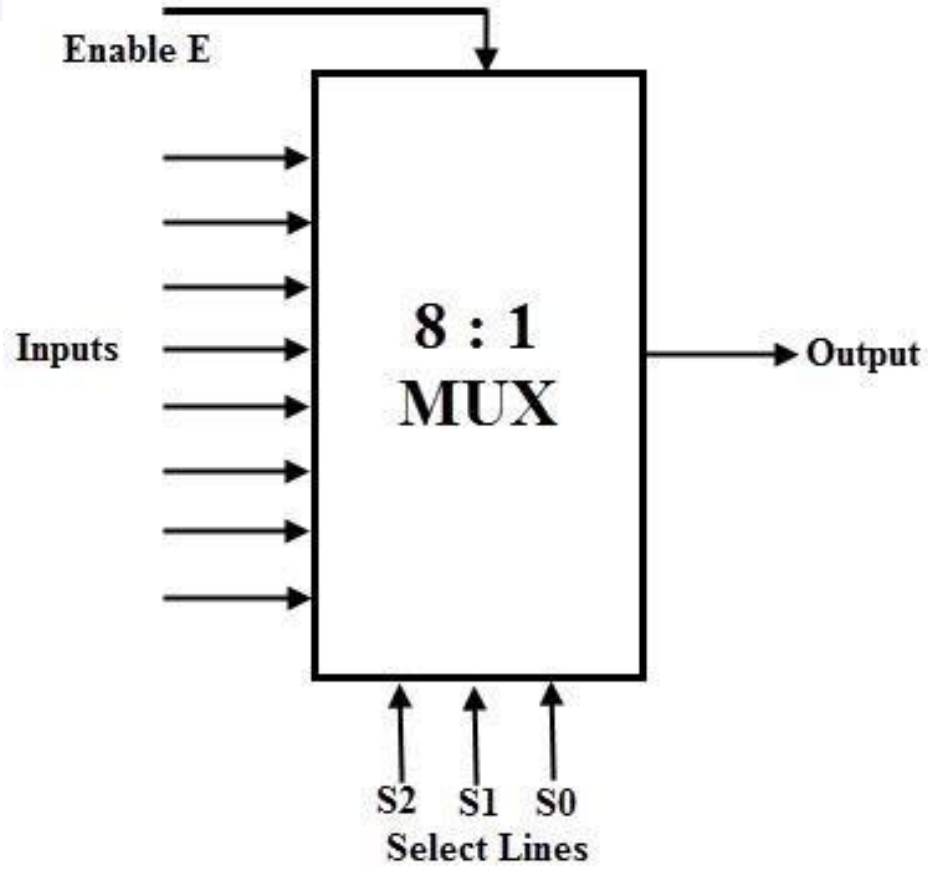


4 : 1 Multiplexer

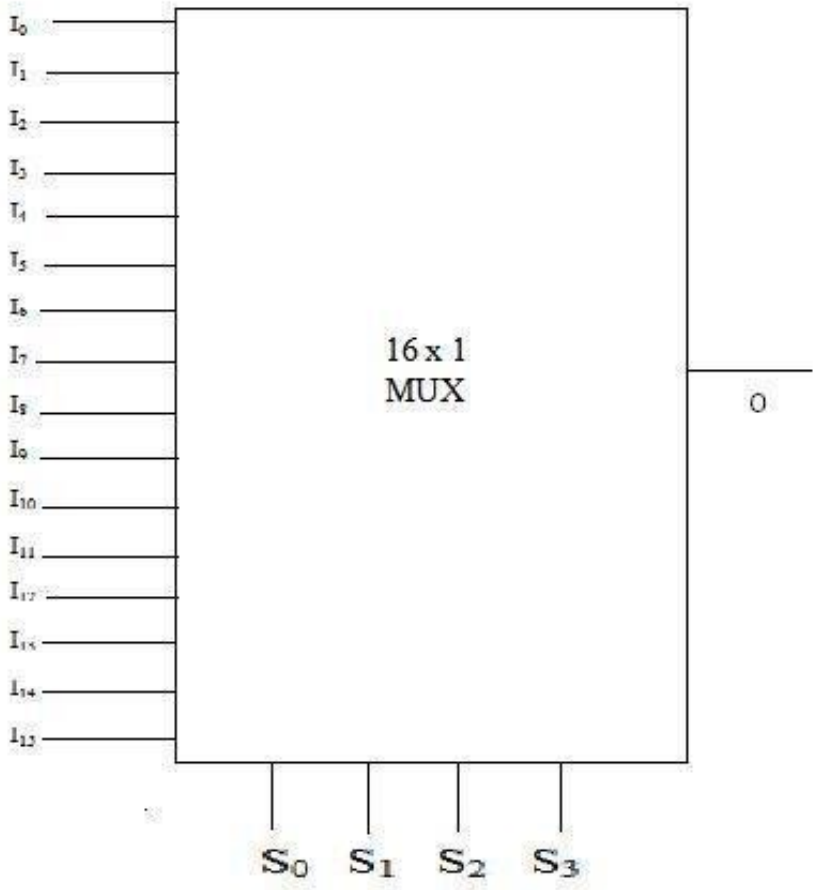


S_0	S_1	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Multiplexer (Data Selector) :

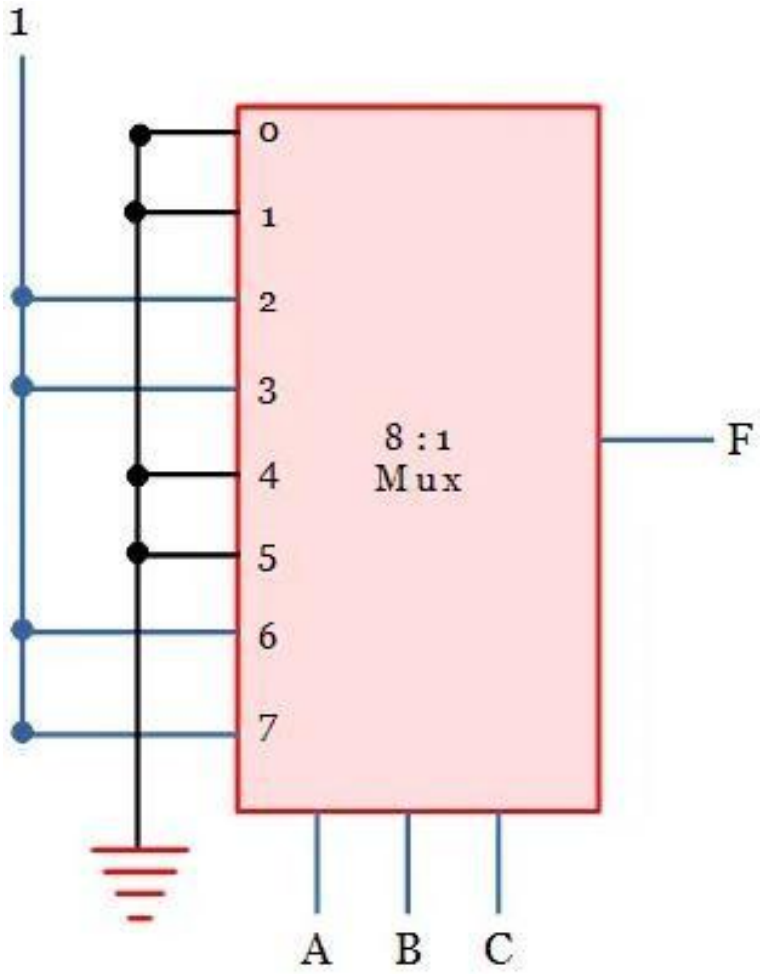


Multiplexer (Data Selector) :



Multiplexer Numerical :

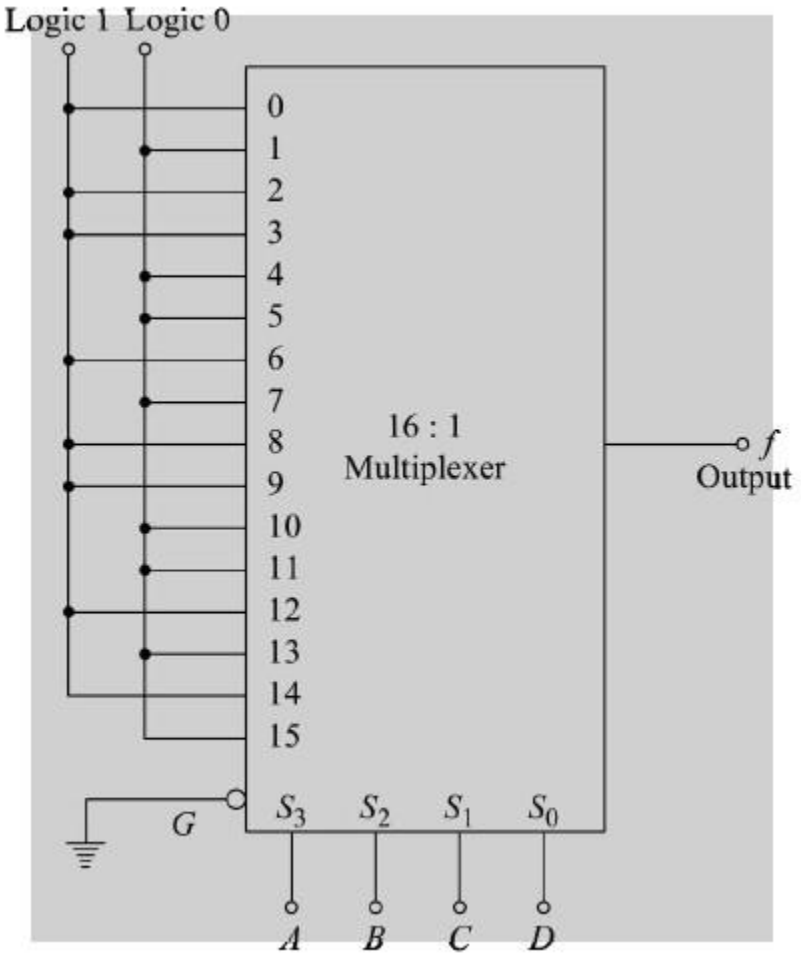
Implement given function using Multiplexer F = m (2,3,6,7)



Multiplexer Numerical :

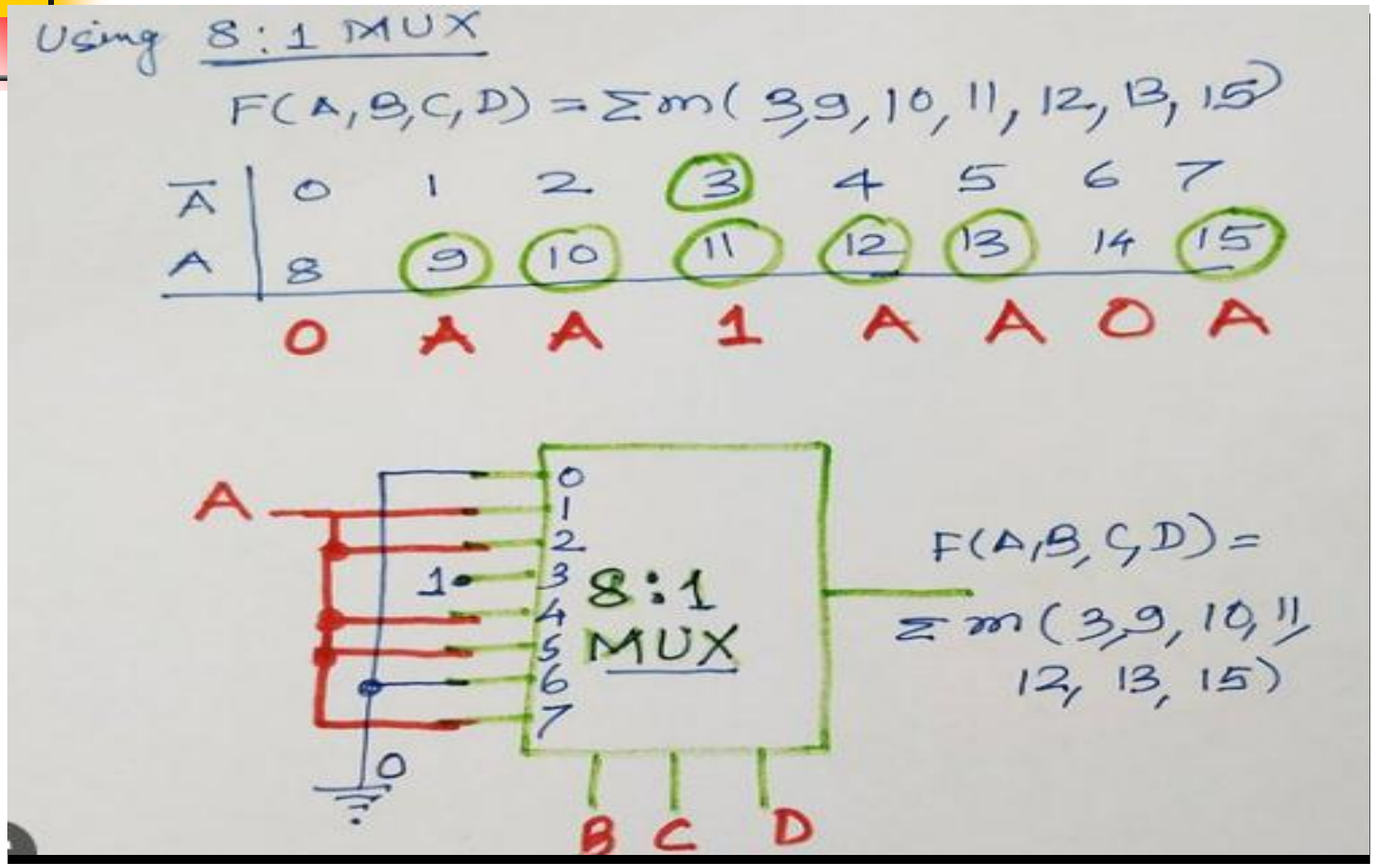
Implement the expression using a multiplexer.

$$f(A, B, C, D) = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$$



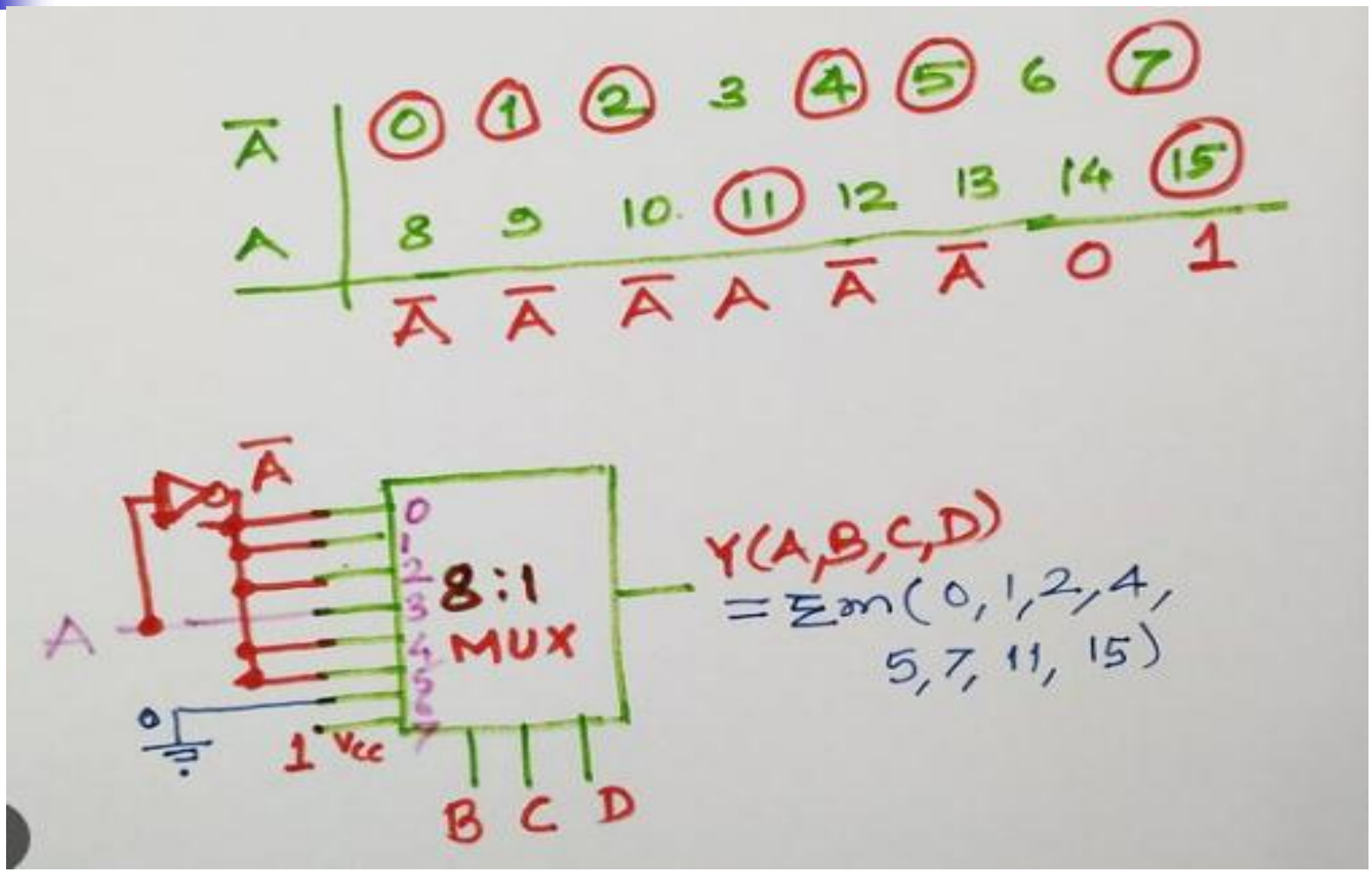
Implementation of Logic Expression

Example: Implement the expression using 8:1 Multiplexer



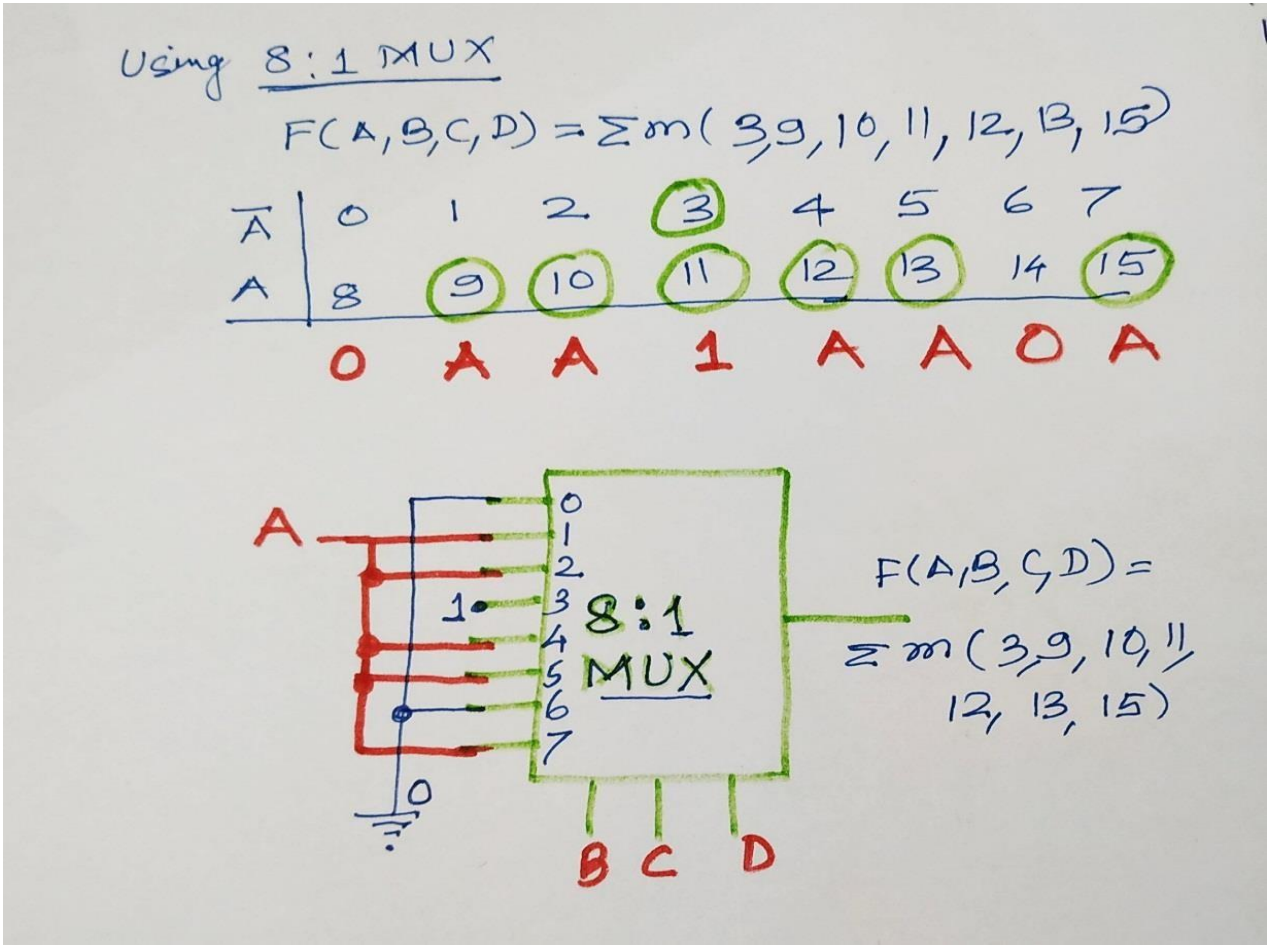
Implement given function using Multiplexer

F(A,B,C,D) = m (0,1,2,4,5,7,11,15)



Multiplexer Numerical :

Implement given function using 8:1 Mux F = m (3, 9,10,11,12,13,15)



Multiplexer Numerical :

Implement given function using 8:1 Mux F = m (0,3,4,7,8,9,13,14)

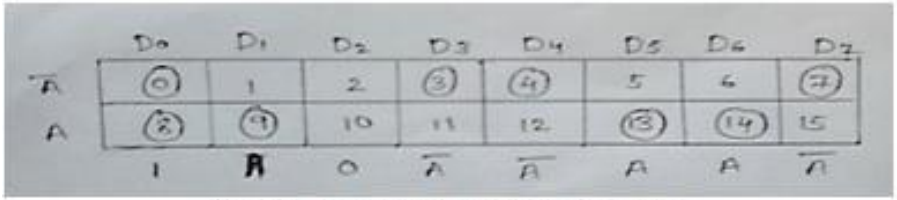


Fig11. K-map for given function

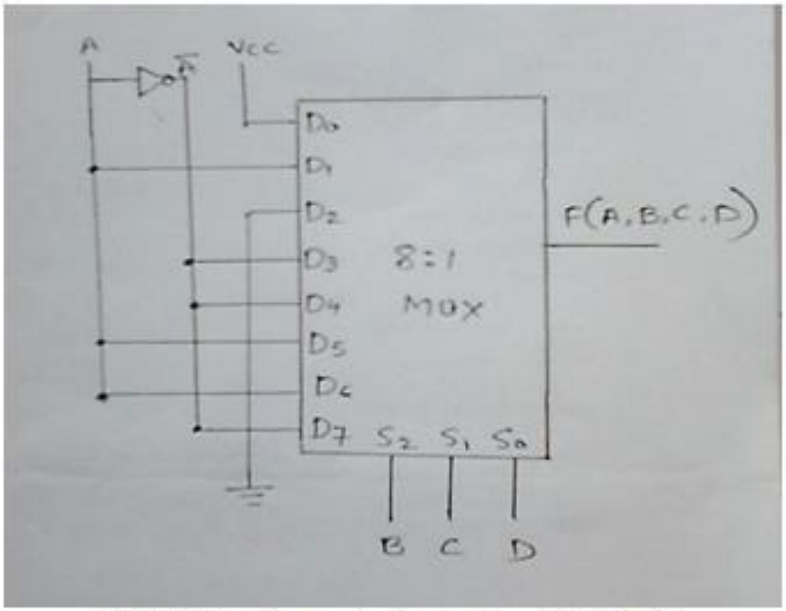
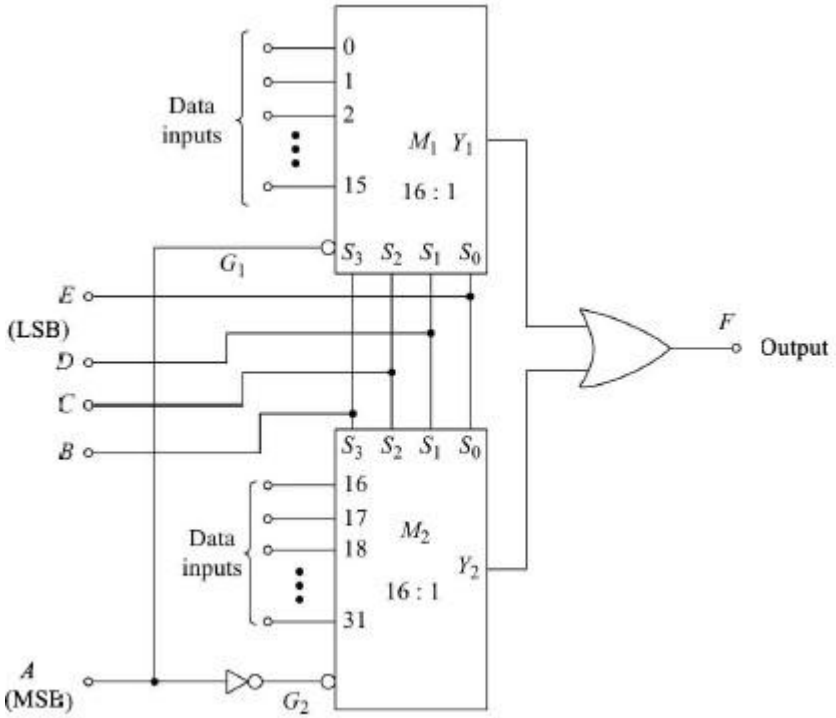


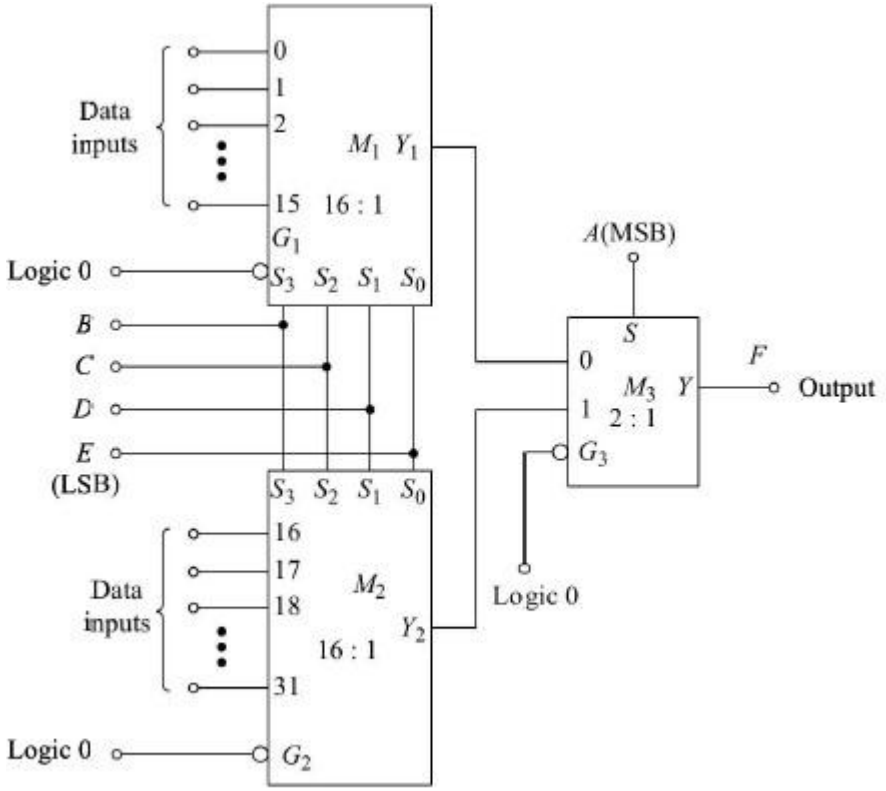
Fig12. Implementation using 8:1 MUX

Multiplexer Tree

Implementing higher order mux using lower order mux



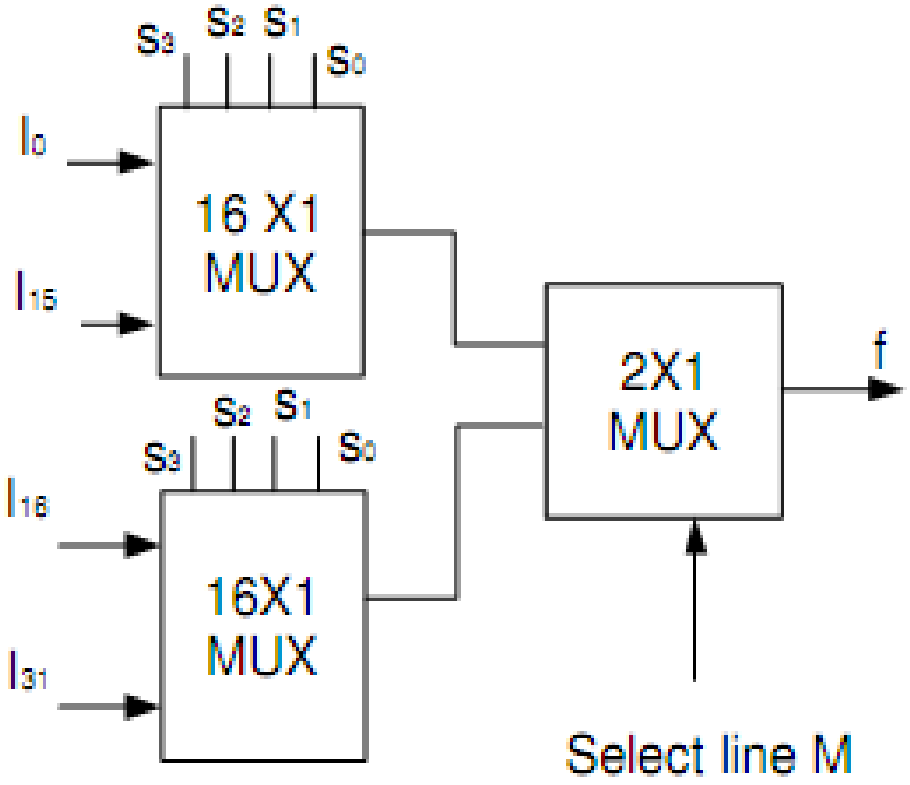
32:1 Multiplexer Using Two 16:1 Multiplexers and One OR Gate



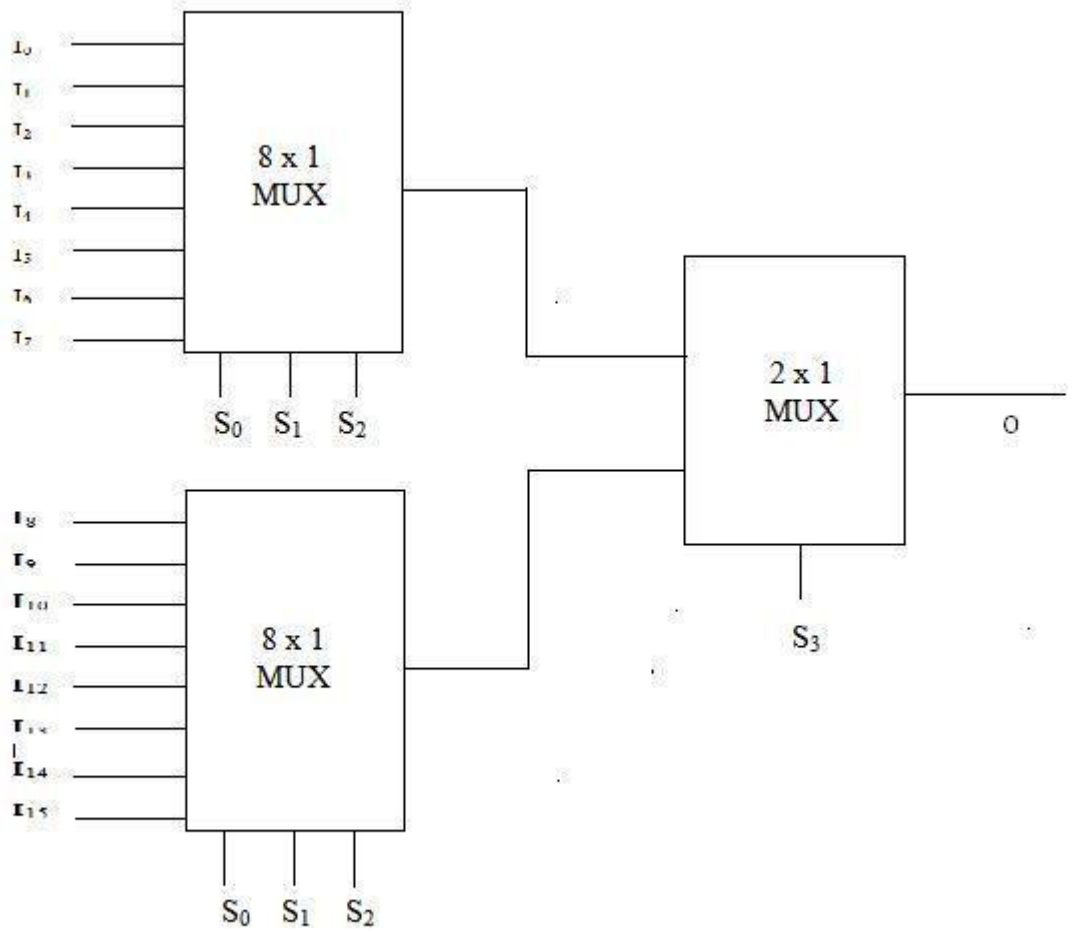
32.1 Multiplexer Using Two 16:1 Multiplexers and One 2:1 Multiplexer

Multiplexer Tree

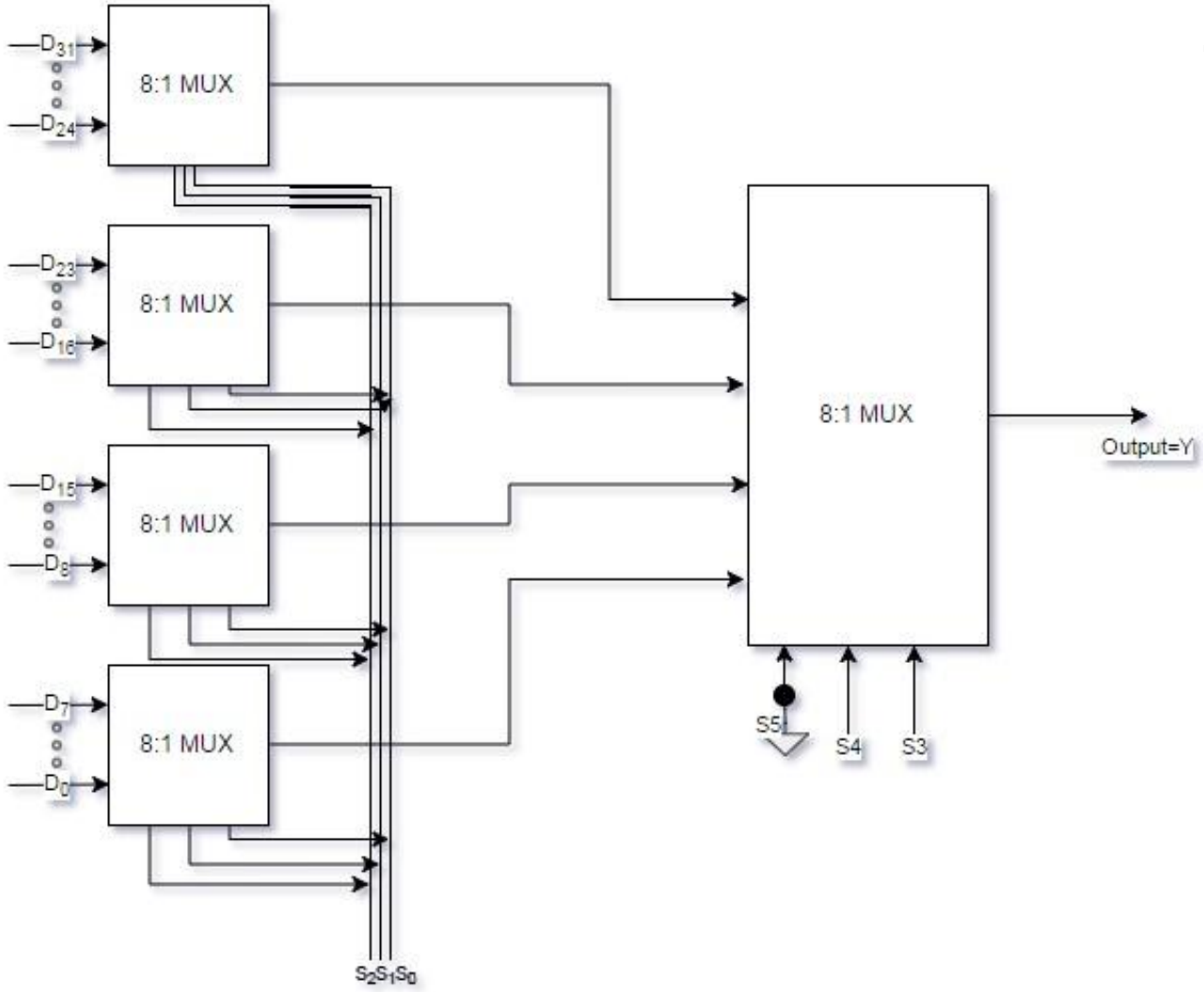
Implementing higher order mux using lower order mux



Multiplexer Tree : 16:1 Mux using 8:1 Mux



Multiplexer Tree



Multiplexer Tree

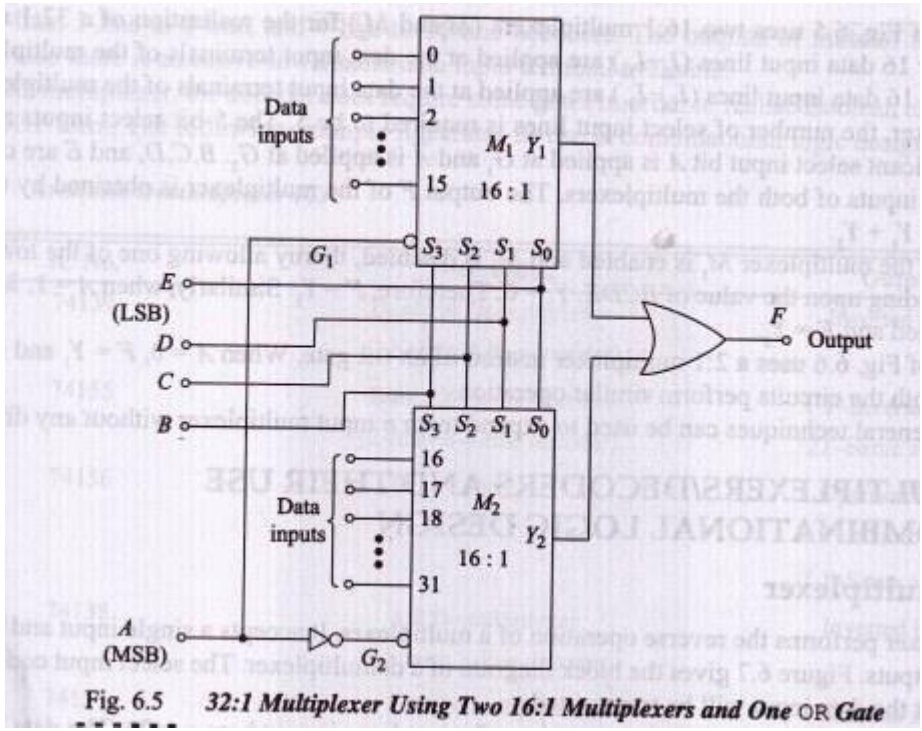


Fig. 6.5 *32:1 Multiplexer Using Two 16:1 Multiplexers and One* OR *Gate*

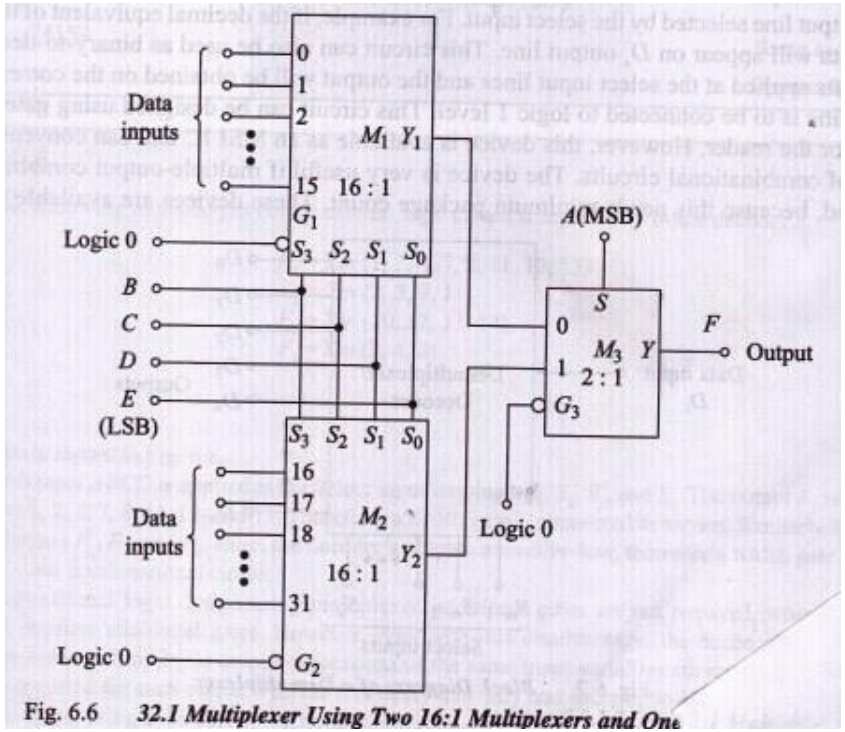
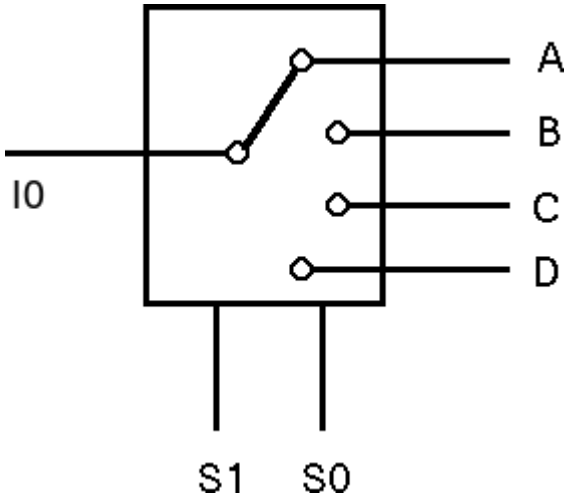
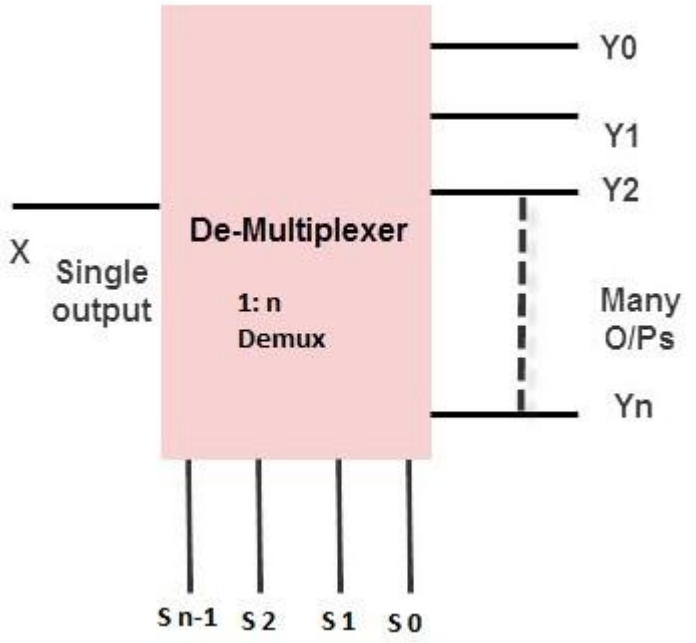


Fig. 6.6 *32.1 Multiplexer Using Two 16:1 Multiplexers and One*

Demultiplxer (Data Distributor) :



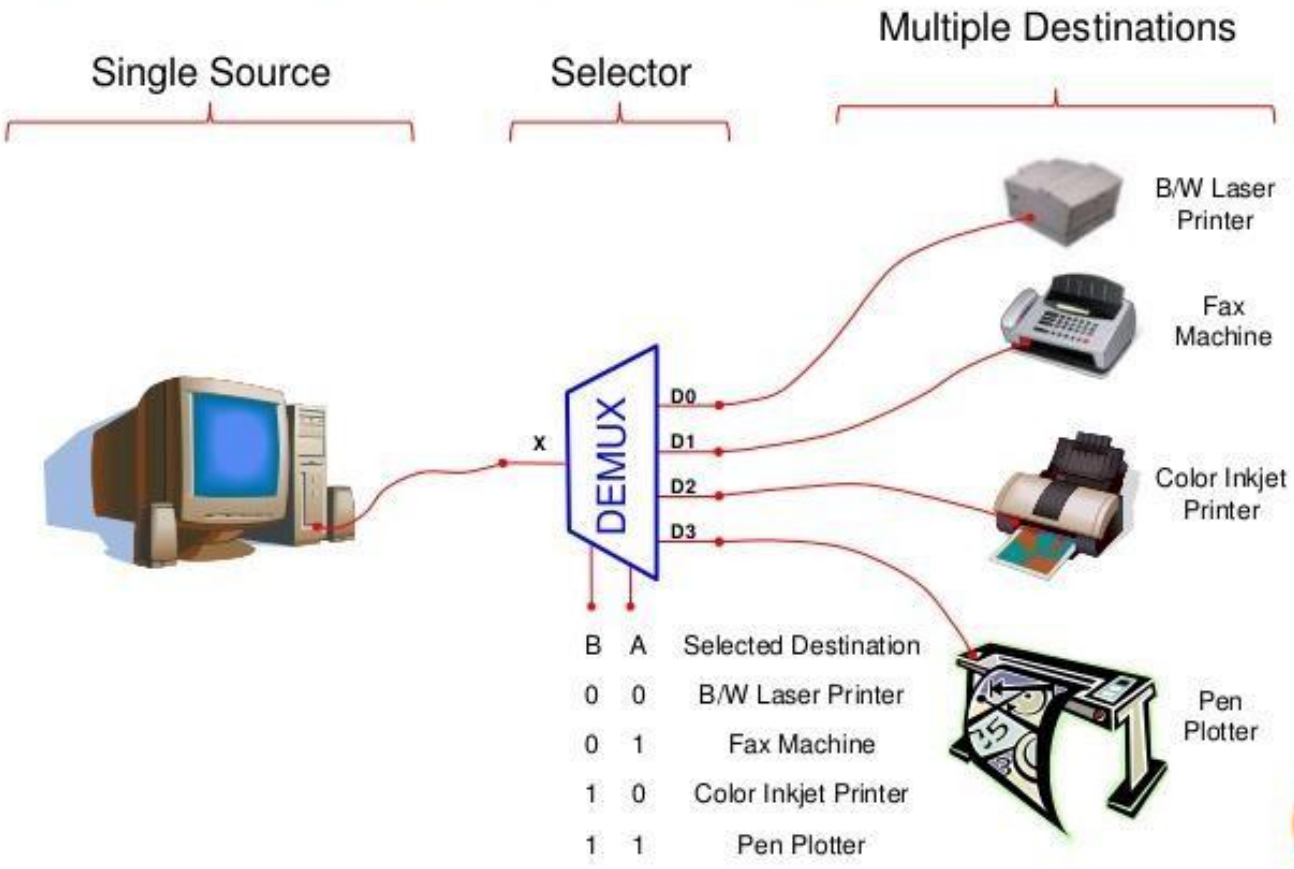


Logic circuit that accepts single input & distributes it over several outputs.

Applications:

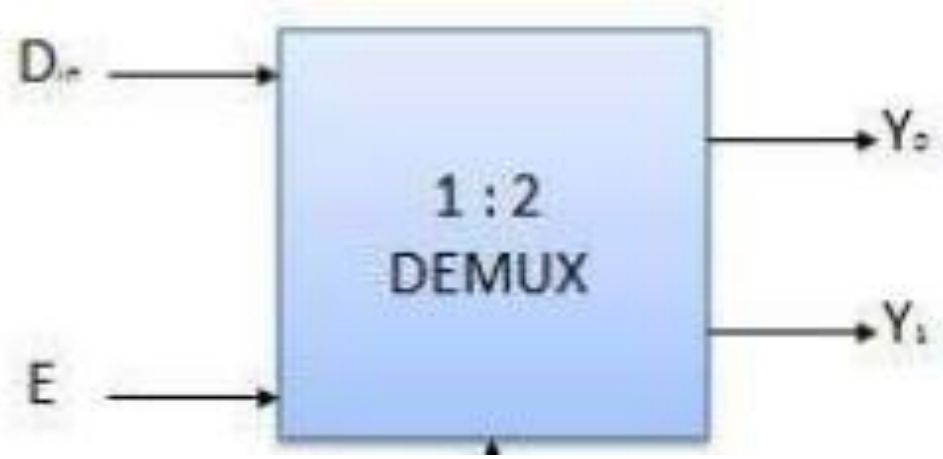
- Communication System
- ALU (Arithmetic Logic Unit)
- Serial to parallel converter

TYPICAL APPLICATION OF A DEMUX

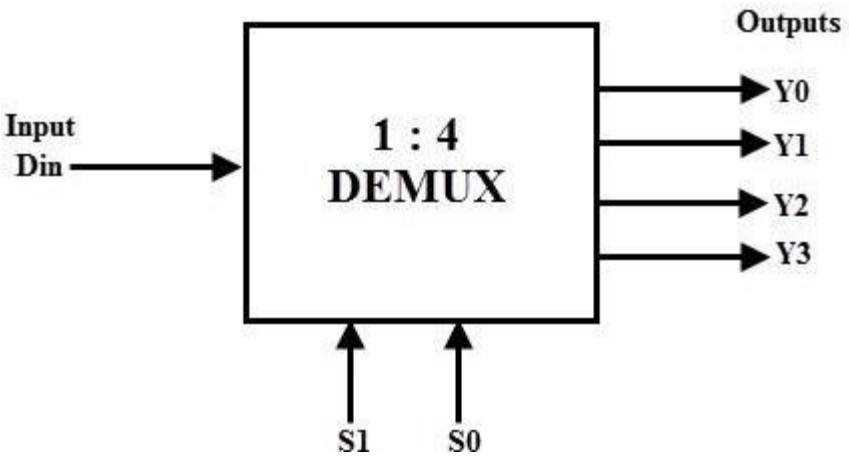


Demultiplxer (Data Distributor) :

- 1 : 2 demultiplexer
- 1 : 4 demultiplexer
- 1 : 16 demultiplexer
- 1 : 32 demultiplexer



Demultiplxer (Data Distributor) :



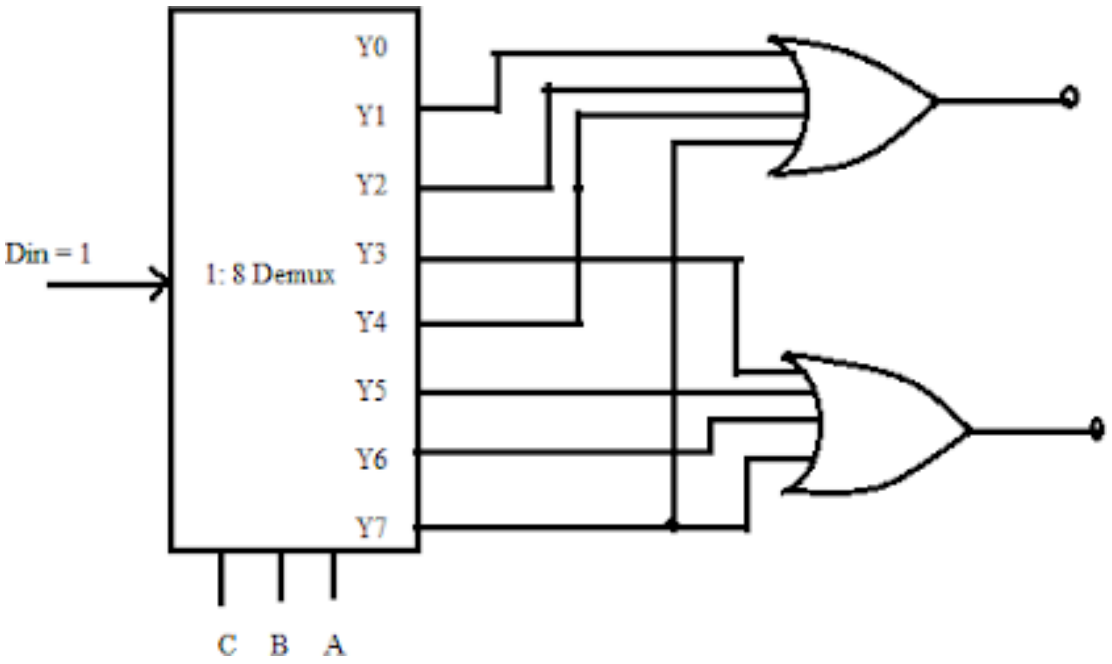
Implement Full Adder using Demux

Truth table of 1 bit full adder:

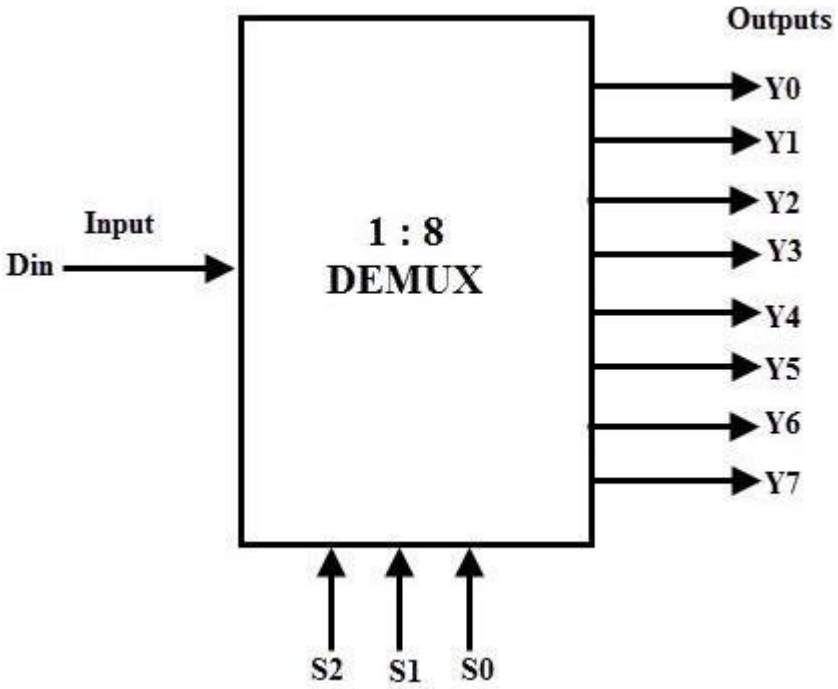
	A	B	C_{in}	Sum	$Carry_{out}$
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Sum = ∑m(1,2,4,7)

Carry = ∑m(3,5,6,7)

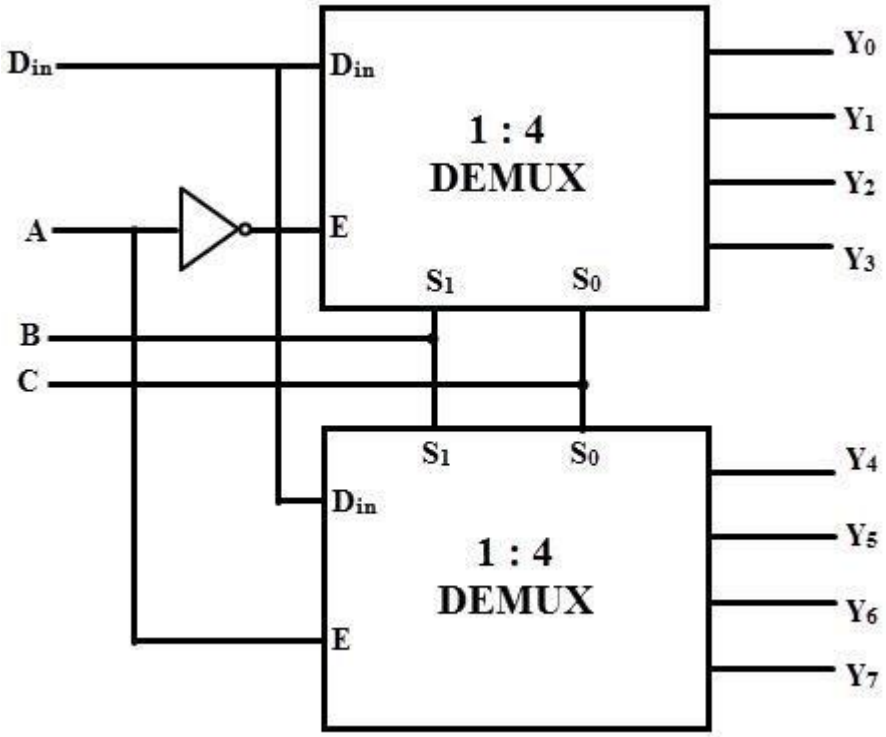


Demultiplxer (Data Distributor) :



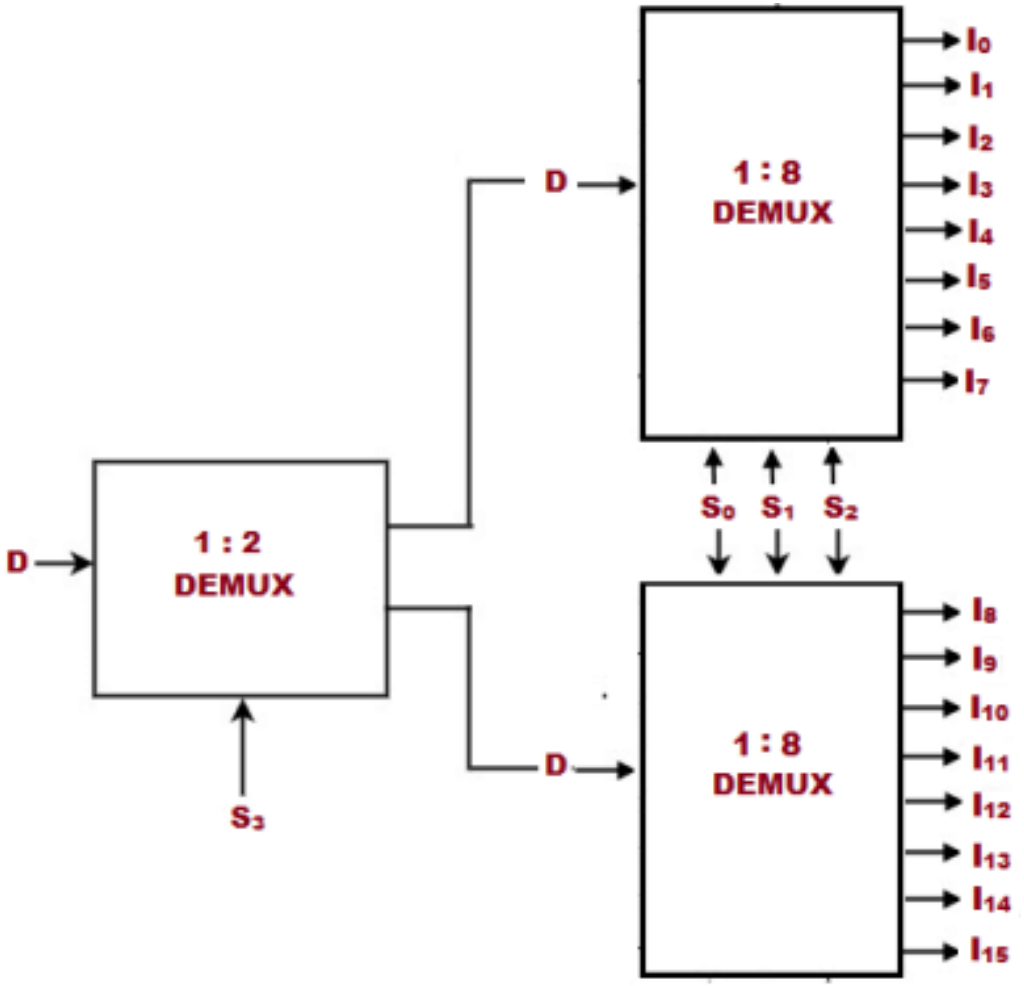
Demultiplxer Tree

1:8 Demux using 1:4 Demux

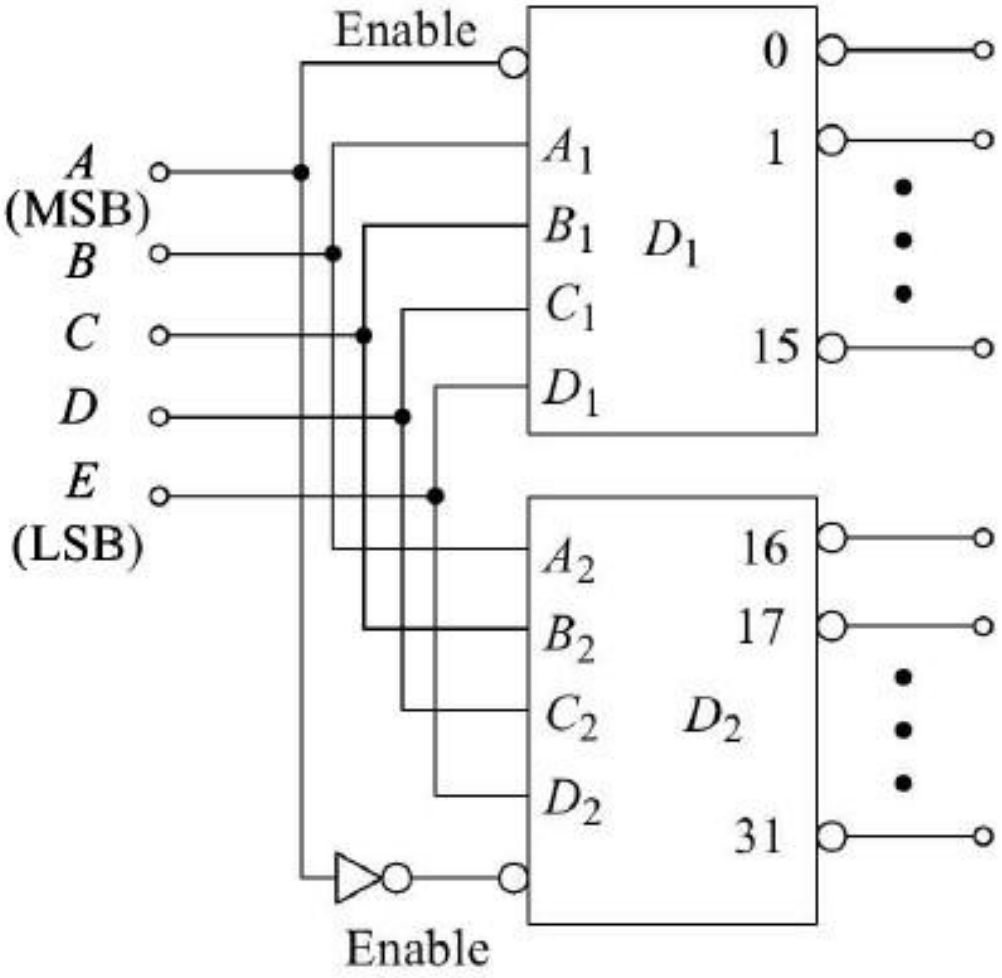


Demultiplxer Tree

1:16 Demux using 1:8 Demux & 1:2 Demux



Demultiplxer Tree



5-Line-to-32-Line Decoder Using Two 4-line-to-16-Line Decoders

Multiplexer Numerical :

Realize logic function of truth table using Multiplexer

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

Realize the logic function of truth table given in below table

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

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	$\bar{D}$
0	1	0	$\bar{D}$
0	1	1	1
1	0	0	D
1	0	1	1
1	1	0	$\bar{D}$
1	1	1	D

The second method can also be used if the logic expression is specified.

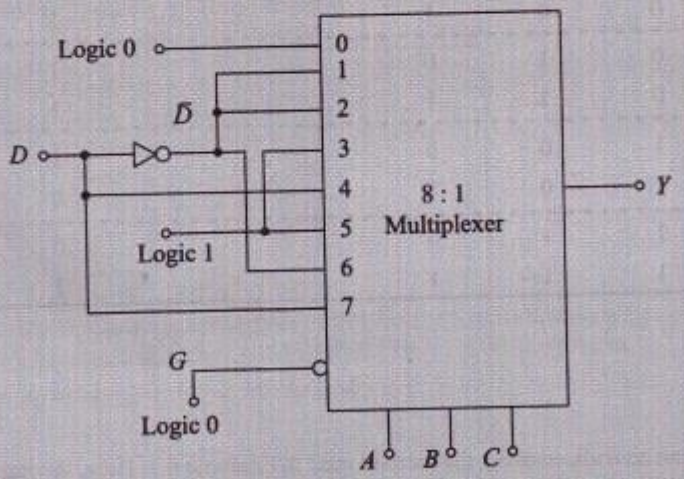
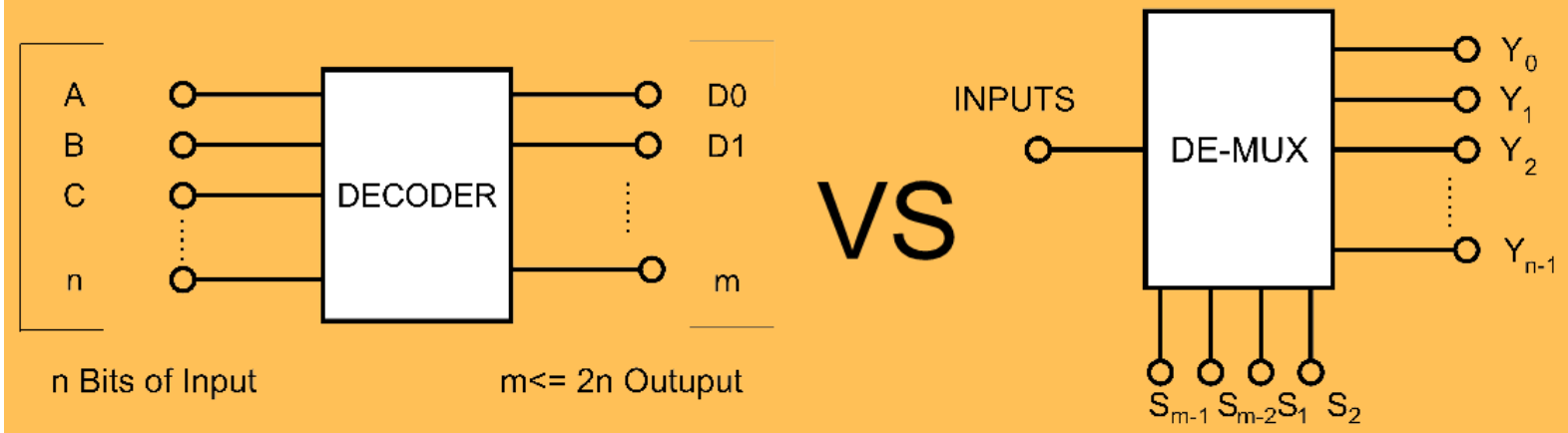


Fig. 6.4 *Realisation of 4-variable Truth Table Using 8:1 Multiplexer*

Decoder

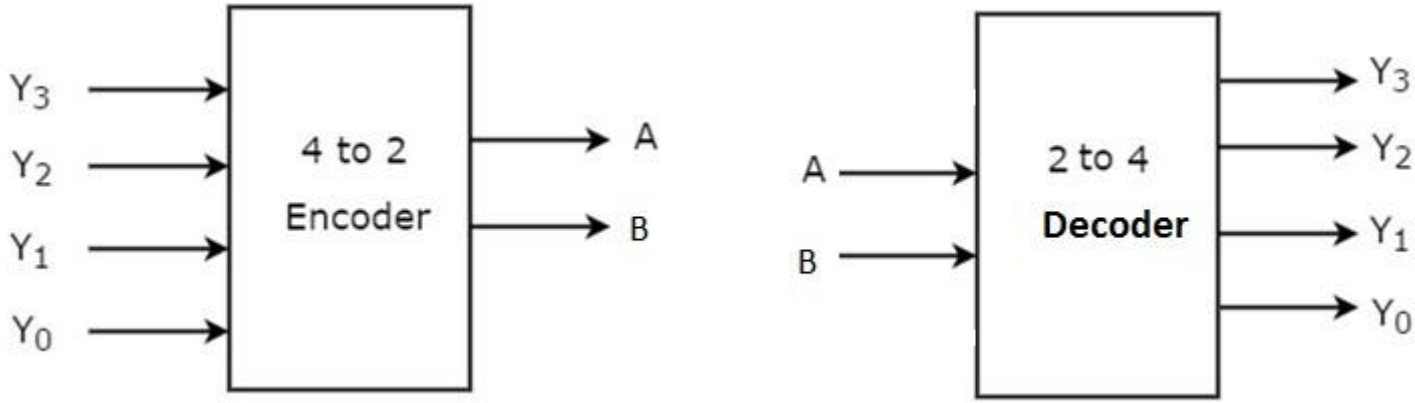
Decoder & De-Multiplexer



Decoder

	Demultiplexer	Decoder
Definition	1 data input 2^n outputs	It has n inputs 2^n outputs It has n control inputs
Characteristic	Connects the data input to the data output	Selects one of the 2^n outputs by decoding the binary value on the basis of n inputs
Reverse of	Multiplexer	Encoder

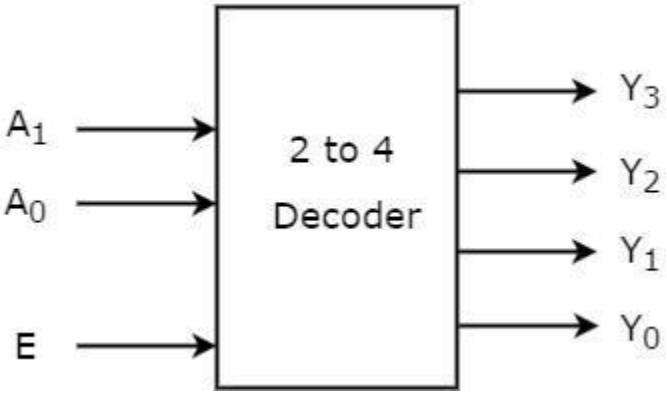
Encoder and Decoder

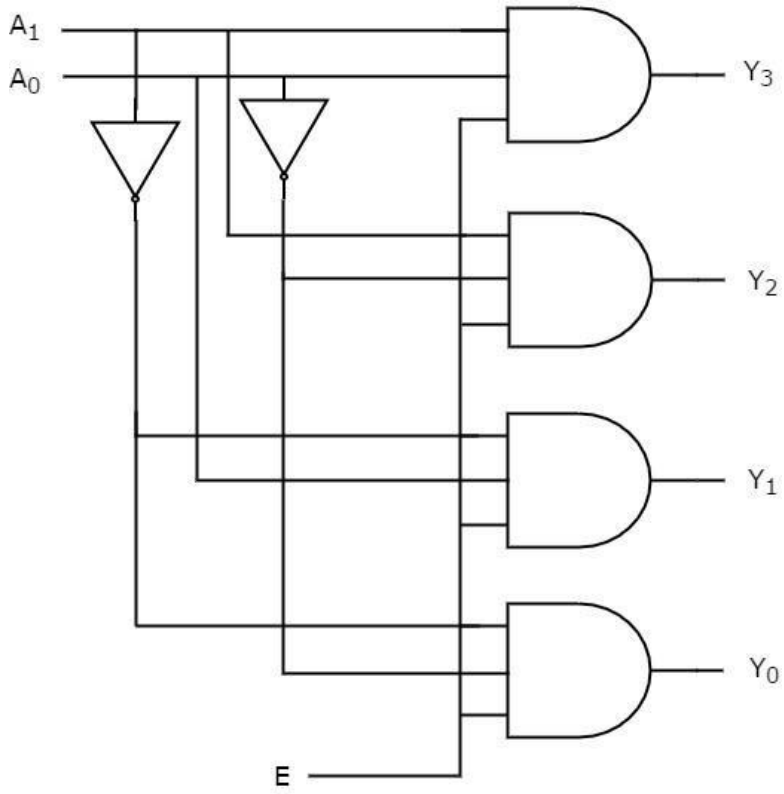


Encoder and Decoder

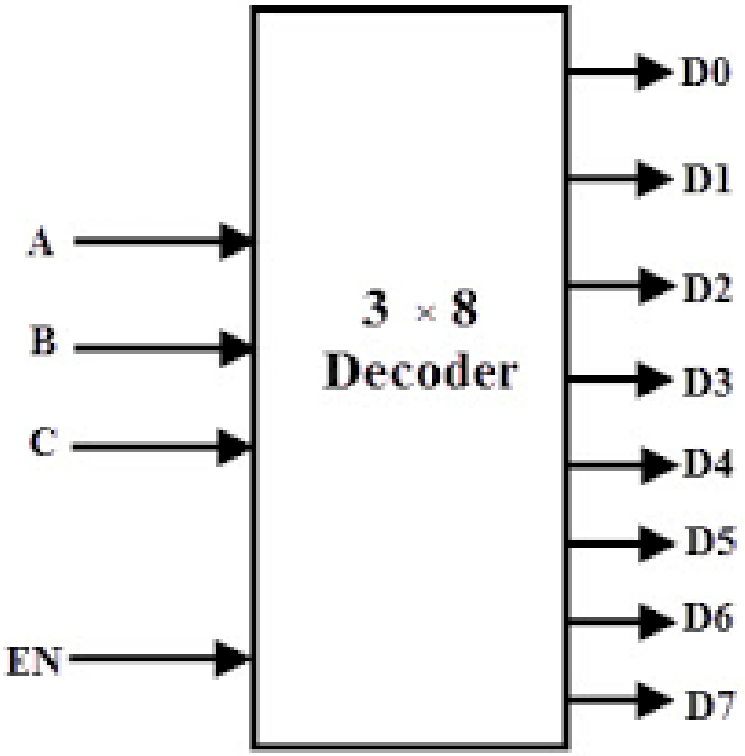
PARAMETER	ENCODER	DECODER
Input applied	Original message signal	Coded binary input
Output generated	Coded binary output	Original message
Input lines	2^n	n
Output lines	n	2^n
Operation	Simple	Complex
Basic logic element	OR gate	AND gate along with NOT gate
Applications	E-mail , video encoders etc.	Microprocessors, memory chips etc.

Decoder





Decoder



Decoder

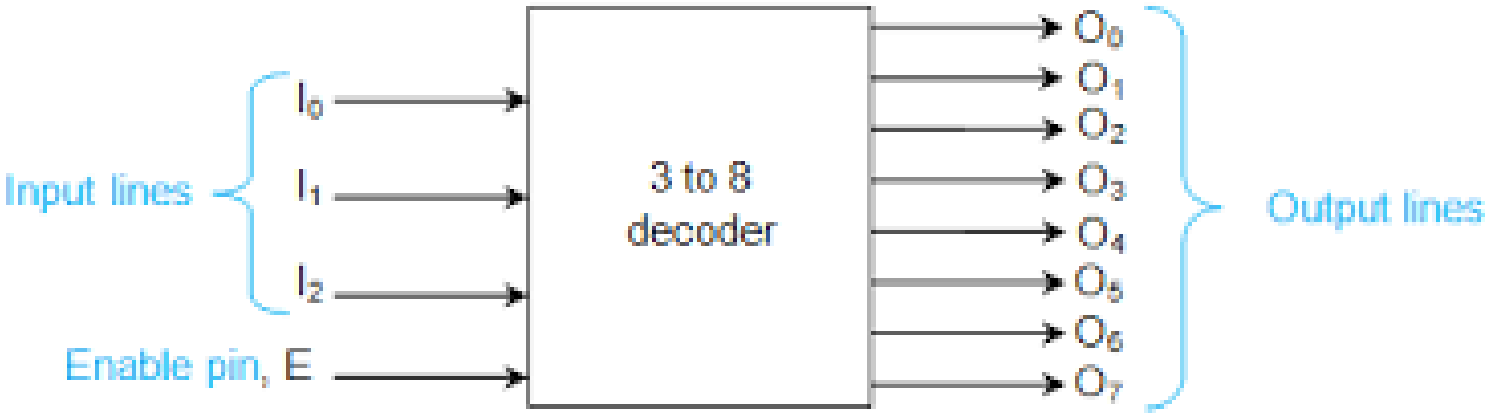


Figure 1 3 to 8 binary decoder

Implement following function using 4 to 16 line decoder

$F_1 = \Sigma m\,(1, 2, 4, 7, 8, 11, 12, 13)$
$F_2 = \Sigma m\,(2, 3, 9, 11)$
$F_3 = \Sigma m\,(10, 12, 13, 14)$
$F_4 = \Sigma m\,(2, 4, 8)$

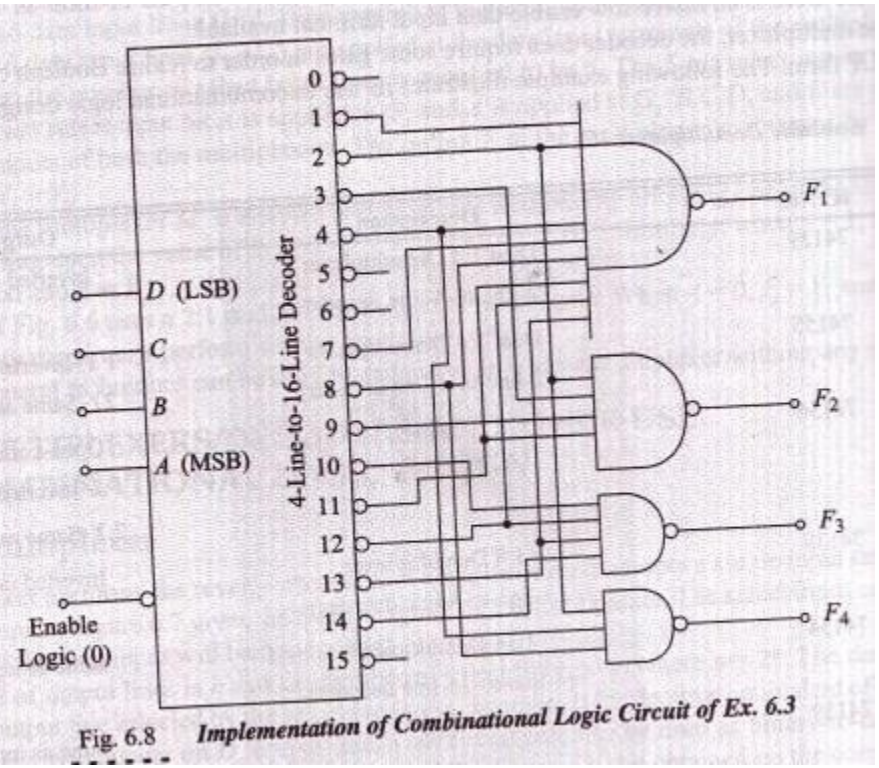
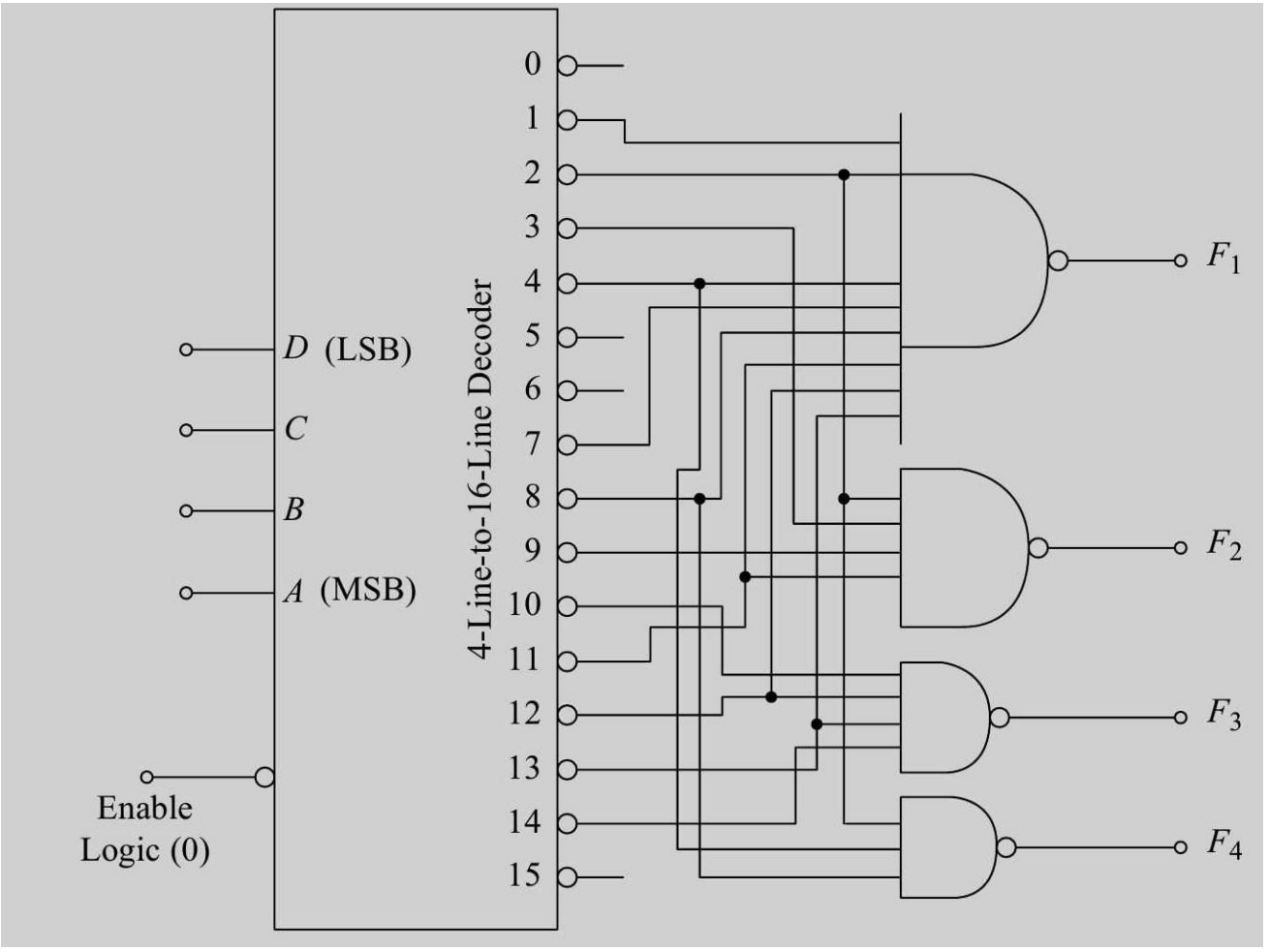
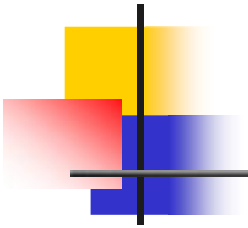


Fig. 6.8 *Implementation of Combinational Logic Circuit of Ex. 6.3*

Implement the following multi-output combinational logic circuit using a 4-to-16-line decoder.

$$F_1 = \Sigma m\,(1, 2, 4, 7, 8, 11, 12, 13)$$
$$F_2 = \Sigma m\,(2, 3, 9, 11)$$
$$F_3 = \Sigma m\,(10, 12, 13, 14)$$
$$F_4 = \Sigma m\,(2, 4, 8)$$

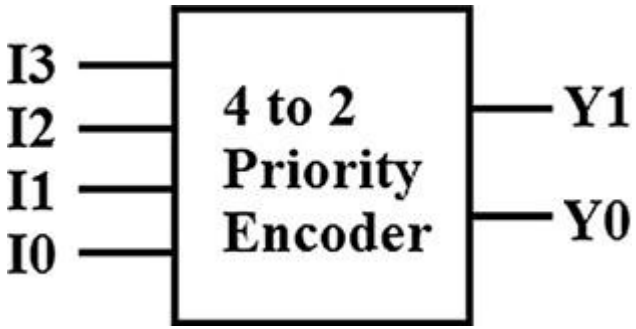




Priority Encoder

- It gives priority from highest to lowest
- We assume I3 has highest priority and I0 has lowest priority
- When I3 is 1 then, we don't need to look in to I2, I1 and I0 (No matter what is the input to I2, I1 and I0), Here I3 will get encoded with the output Y1=1 and Y0=1
- When I2 is 1 then, we don't need to look in to I1 and I0 (No matter what is the input to I1 and I0), Here I2 will get encoded with the output Y1=1 and Y0=1
- When I1 is 1 then, we don't need to look in to I0 (No matter what is the input to I0), Here I1 will get encoded with the output Y1=0 and Y0=1
- When I0 is 1 then, Here I0 will get encoded with the output Y1=0 and Y0=0

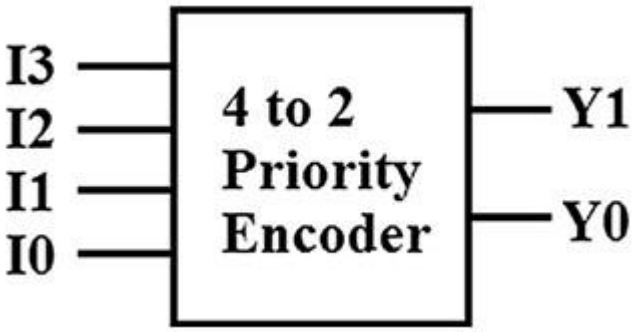
Priority Encoder



Input				Output	
I3	I2	I1	I0	Y1	Y0
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

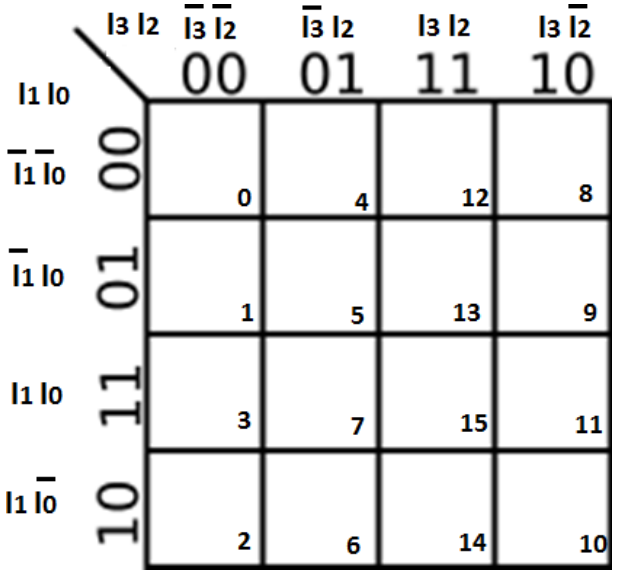
https://www.youtube.com/watch?v=jbghfcdj5Dc

Priority Encoder

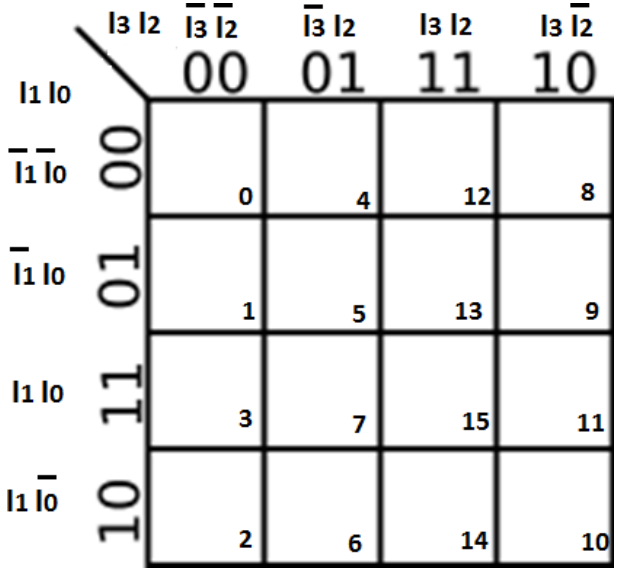


Input				Output	
I3	I2	I1	I0	Y1	Y0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Kmap for Y1

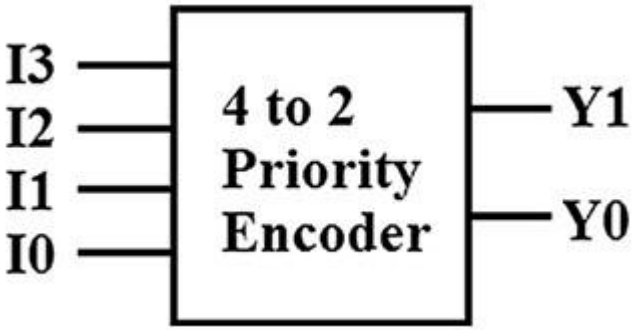


Kmap for Y2



https://www.youtube.com/watch?v=jbghfcdj5Dc

Priority Encoder



Input				Output	
I3	I2	I1	I0	Y1	Y0
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

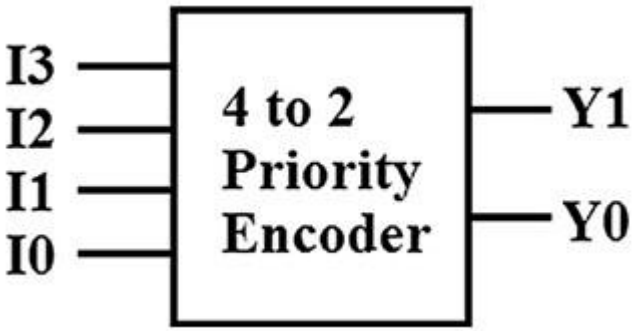
Kmap for Y1

$I_1 I_0$ \ $I_3 I_2$	$\bar{I}_3\bar{I}_2$ (00)	$\bar{I}_3 I_2$ (01)	$I_3 I_2$ (11)	$I_3\bar{I}_2$ (10)
$\bar{I}_1\bar{I}_0$ (00)	x 0	1 4	1 12	1 8
$\bar{I}_1 I_0$ (01)	0 1	1 5	1 13	1 9
$I_1 I_0$ (11)	0 3	1 7	1 15	1 11
$I_1\bar{I}_0$ (10)	0 2	1 6	1 14	1 10

Kmap for Y2

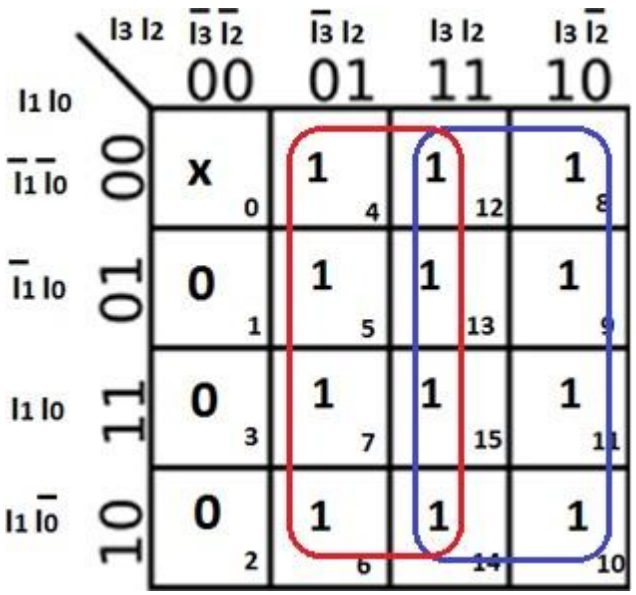
$I_1 I_0$ \ $I_3 I_2$	$\bar{I}_3\bar{I}_2$ (00)	$\bar{I}_3 I_2$ (01)	$I_3 I_2$ (11)	$I_3\bar{I}_2$ (10)
$\bar{I}_1\bar{I}_0$ (00)	x 0	0 4	1 12	1 8
$\bar{I}_1 I_0$ (01)	0 1	0 5	1 13	1 9
$I_1 I_0$ (11)	1 3	0 7	1 15	1 11
$I_1\bar{I}_0$ (10)	1 2	0 6	1 14	1 10

Priority Encoder



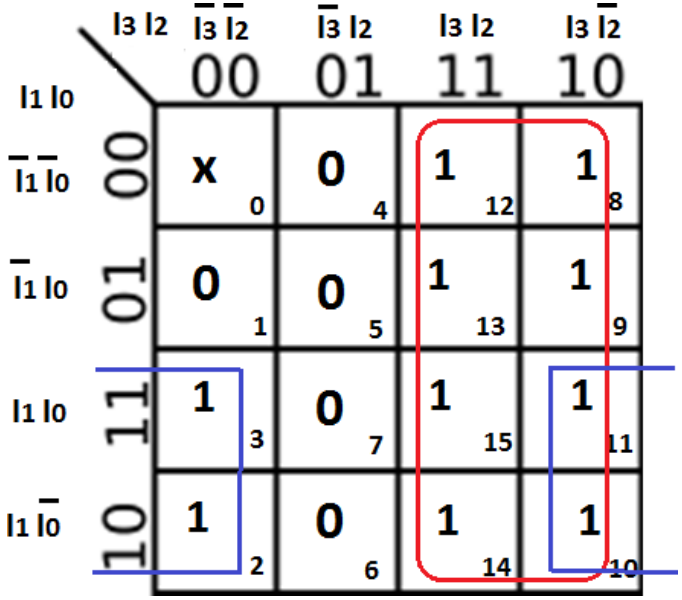
Input				Output	
I3	I2	I1	I0	Y1	Y0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Kmap for Y1



Y1 = I3 + I2

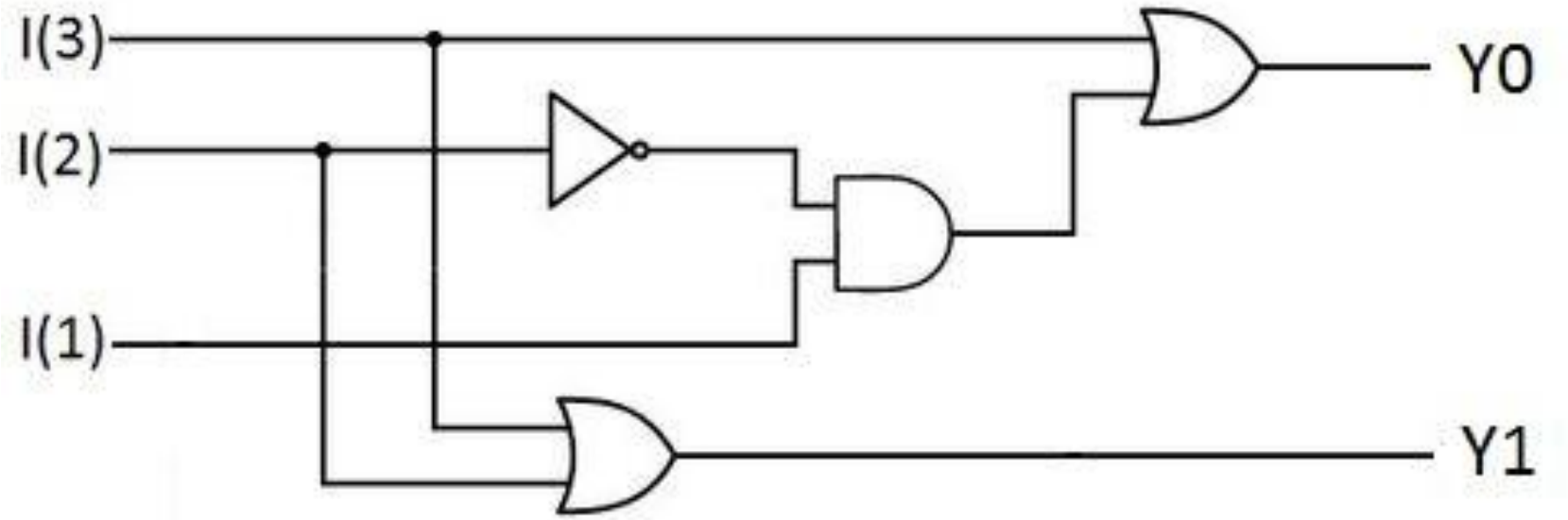
Kmap for Y2



Y2 = I3 + I1 $\overline{I2}$

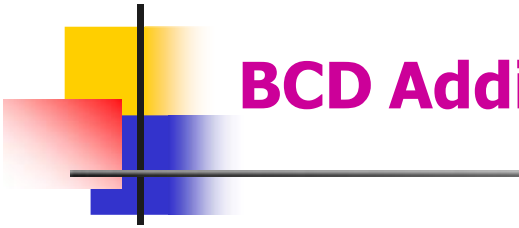
Priority Encoder

Circuit Diagram:



BCD Arithmetic

BCD Addition



BCD numbers can be added as long as certain rules are followed – these are.

if the addition produces a carry
or if it creates an invalid BCD number
then correct the digit by adding six where the error occurred.

BCD Addition: Example

$(7)_{10} + (5)_{10}$ BCD ADDITION

```
  1 1 1
  0111  = 7     sum > 9
+ 0101  = 5     carry = 0
-------
  1100
+ 0110
-------
 10010
```

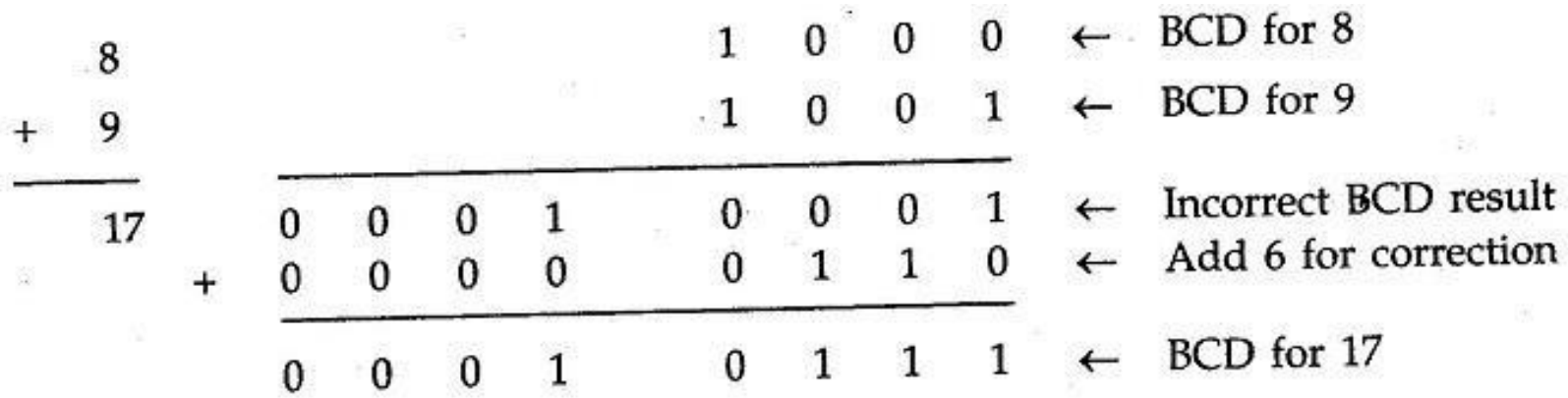
BCD Addition: Example

Carry			1		
BCD for 9		1	0	0	1
BCD for 5		0	1	0	1 +
Carry	1	1	1		
Invalid BCD, sum>9		1	1	1	0
Add 0110 for correction		0	1	1	0
Valid output	1	0	1	0	0

Sum of 9 and 5 is 14.

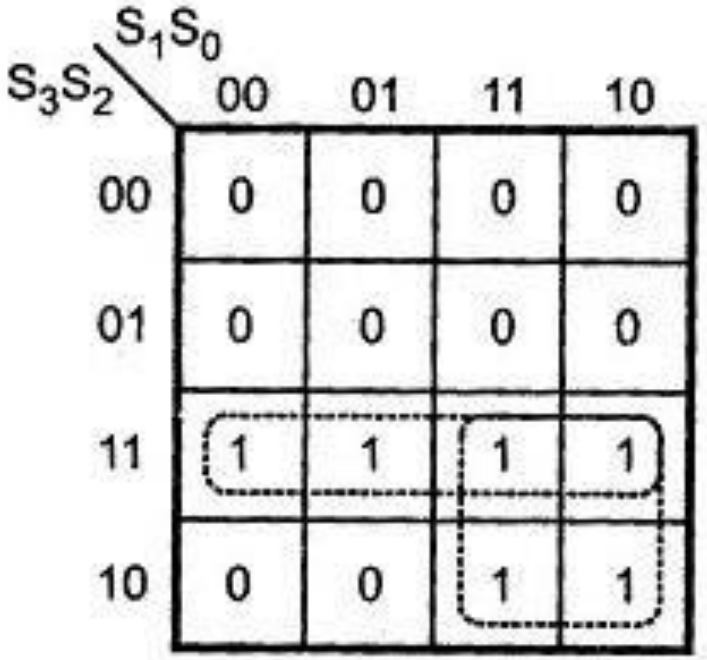
0	0	0	1	0	1	0	0
1				4			

BCD Addition: Example



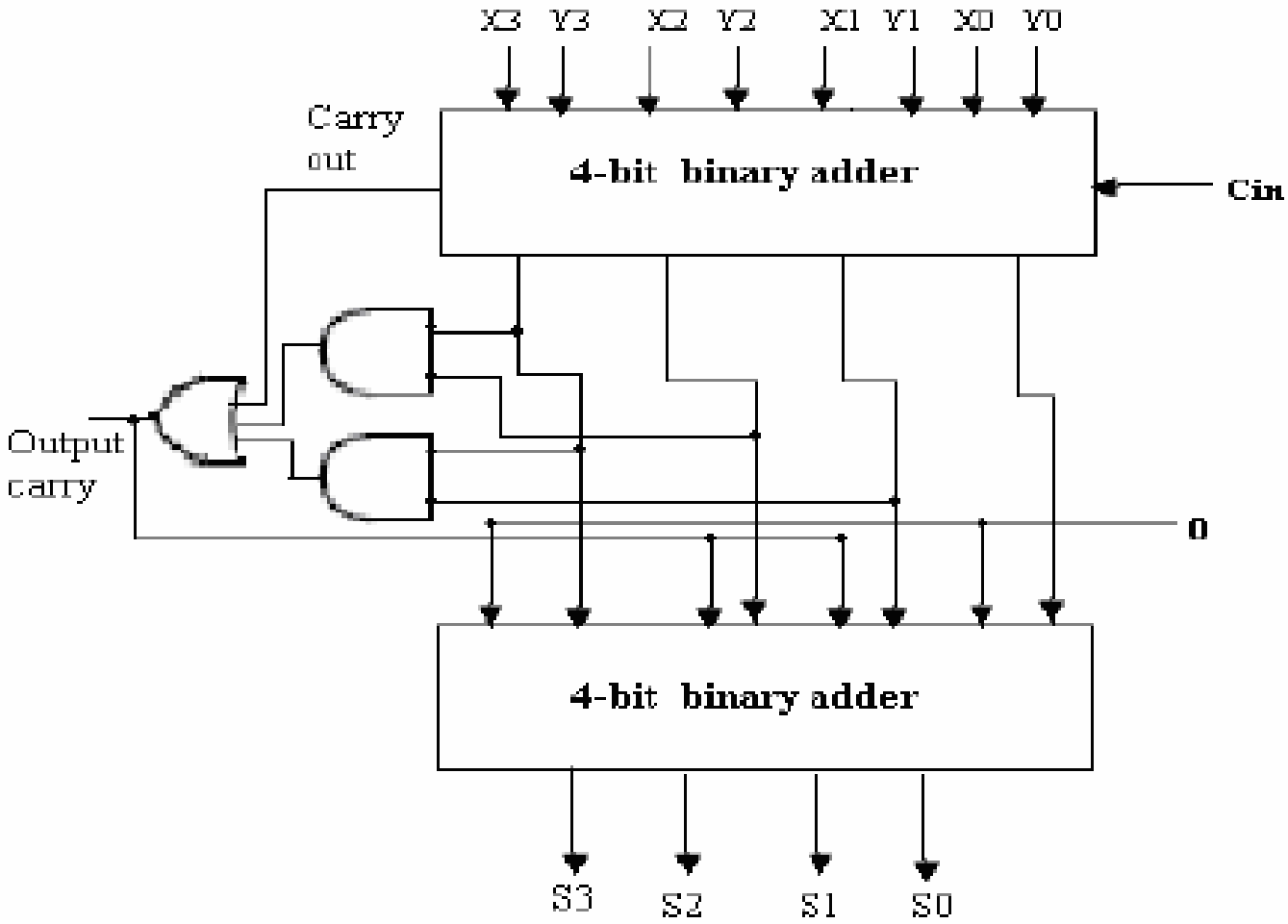
BCD Adder

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



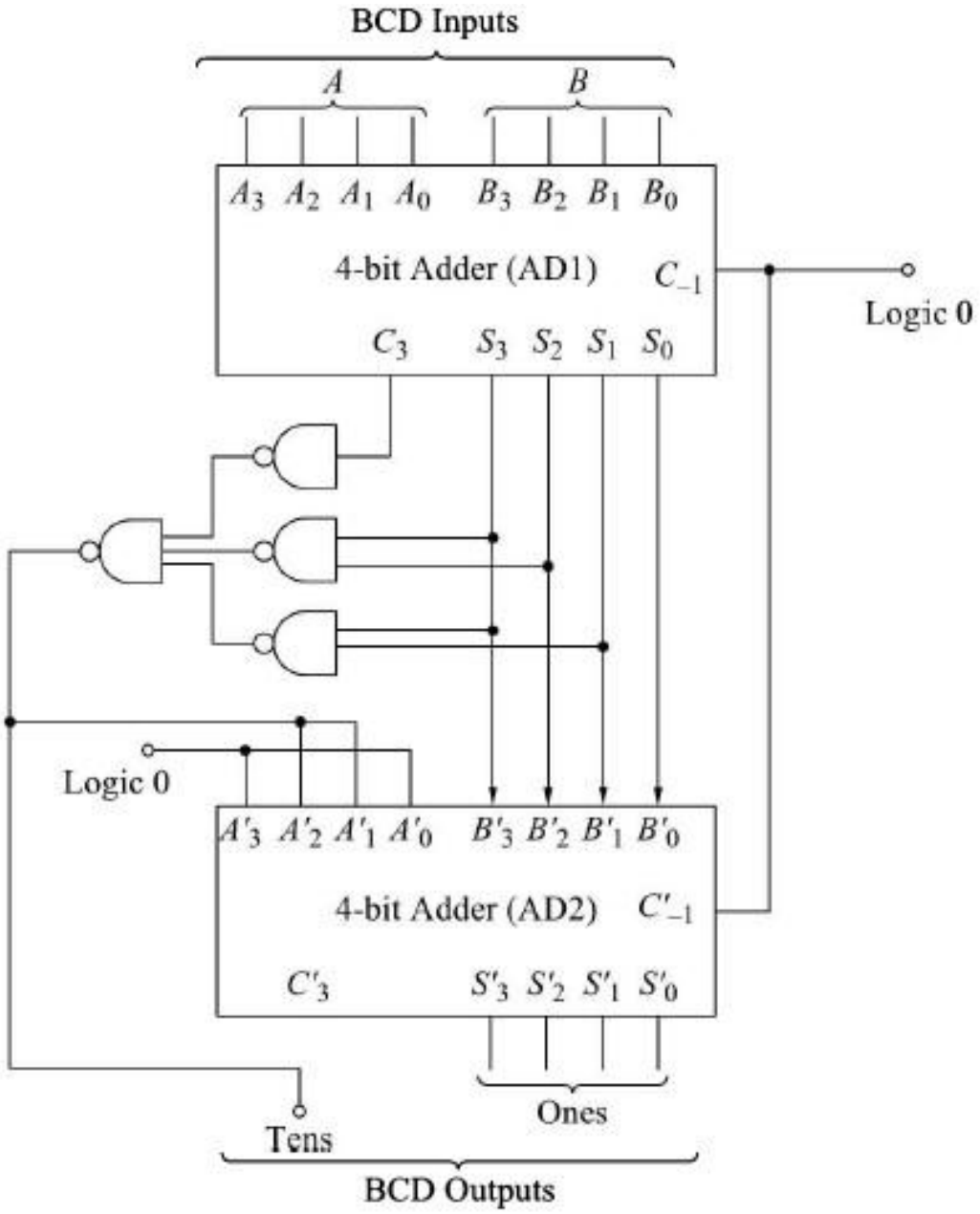
$$Y = S_3S_2 + S_3S_1$$

BCD Adder



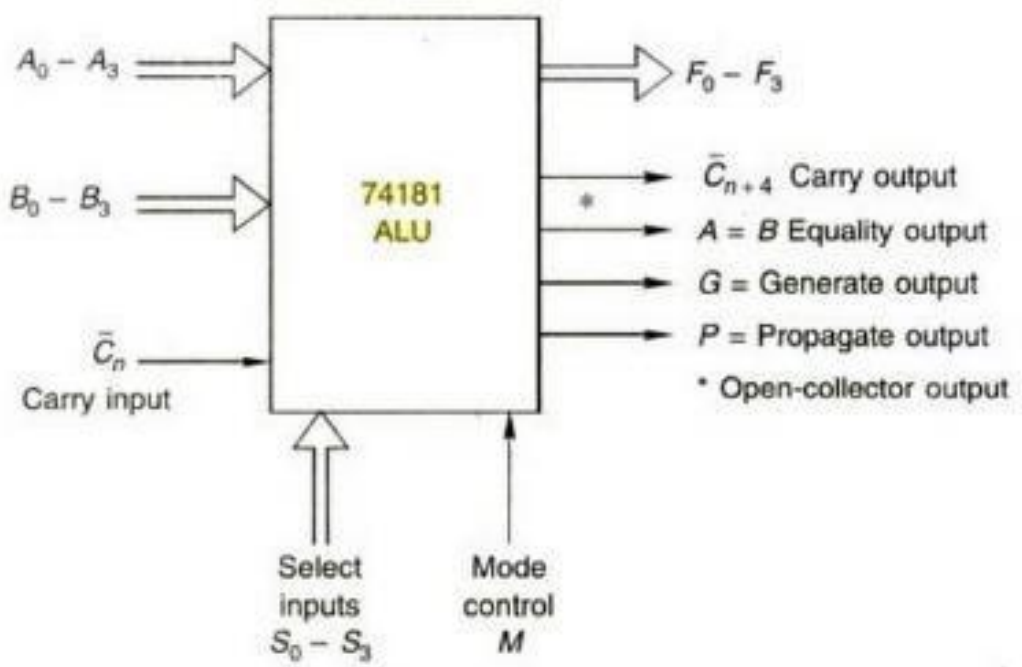
https://www.youtube.com/watch?v=wLHFJEQWEng

BCD Adder



One Digit BCD Adder

ALU Design



A and B – 4 bit binary data input

C_n – Carry input

F – 4 bit binary data output

C_{n+4} – Carry output

A=B – Logic 1 on this line indicates A=B

G – Carry generate

P – Propagate output

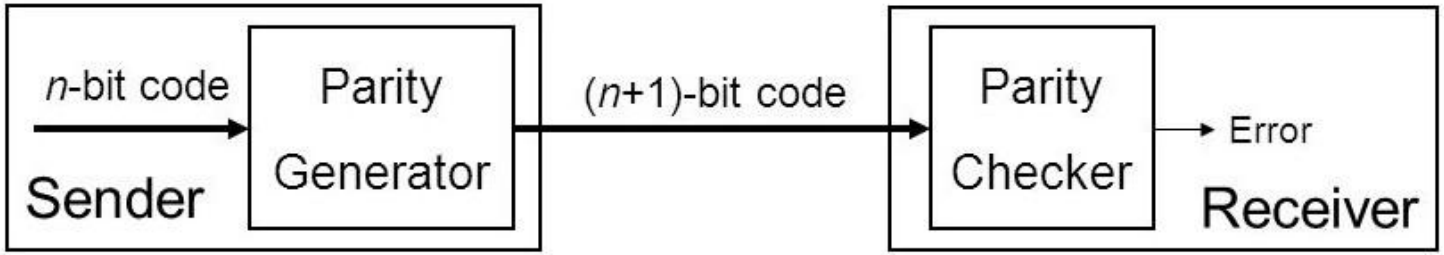
S – Selects one of the operation

M – indicates arithmetic or logical operation

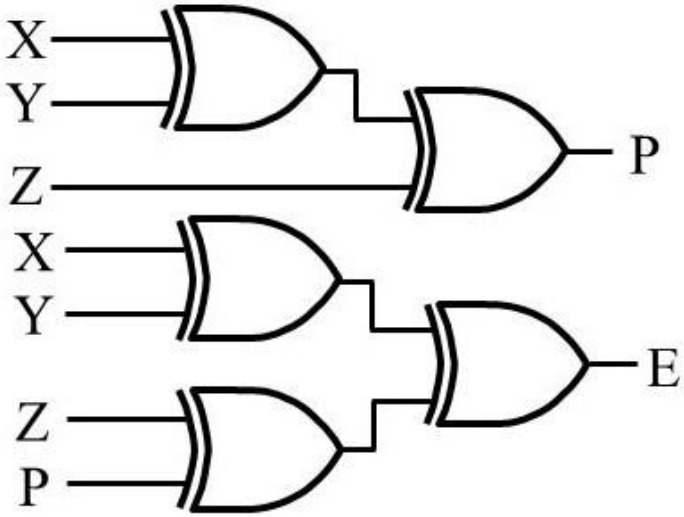
ALU Elementary Design: Function Table

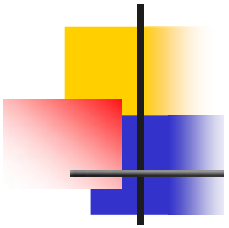
Line	Selection S_3	S_2	S_1	S_0	Active high data: $M = 1$; Logic functions $C_n = X$	$M = 0$; Arithmetic operations: $C_n = 1$ (no carry)	$C_n = 0$ (with carry)
0	0	0	0	0	$F = \overline{A}$	$F = A$	$F = A$ PLUS 1
1	0	0	0	1	$F = \overline{A + B}$	$F = A + B$	$F = (A + B)$ PLUS 1
2	0	0	1	0	$F = \overline{A}.B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
3	0	0	1	1	$F = 0$	$F =$ MINUS 1 (2's COMPL)	$F =$ ZERO
4	0	1	0	0	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
5	0	1	0	1	$F = B$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
6	0	1	1	0	$F = A \oplus B$	$F = A$ MINUS B MINUS 1	$F = A$ MINUS B
7	0	1	1	1	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
8	1	0	0	0	$F = \overline{A} + B$	$F = A$ PLUS AB	$F = A$ PLUS AB PLUS 1
9	1	0	0	1	$F = \overline{A \oplus B}$	$F = A$ PLUS B	$F = A$ PLUS B PLUS 1
10	1	0	1	0	$F = B$	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
11	1	0	1	1	$F = AB$	$F = AB$ MINUS 1	$F = AB$
12	1	1	0	0	$F = 1$	$F = A$ PLUS A	$F = A$ PLUS A PLUS 1
13	1	1	0	1	$F = A + \overline{B}$	$F = (A + B)$ PLUS A	$F = (A + B)$ PLUS A PLUS 1
14	1	1	1	0	$F = A + B$	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
15	1	1	1	1	$F = A$	$F = A$ MINUS 1	$F = A$

Parity Generator & Checkers



- **Design an even parity generator and checker for 3-bit codes**
- **Solution: Use 3-bit odd function to generate even parity bit**
- **Use 4-bit odd function to check for errors in even parity codes**
- **Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0**
- **If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error**





PARITY BIT

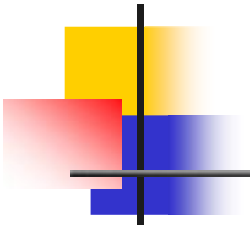
What Parity Bit is?

- The parity generating technique is one of the most widely used error detection techniques for the data transmission.
- **Parity bit** is added to the word containing data in order to make number of 1s either even or odd.Thus it is used to detect errors , during the transmission of binary data

PARITY GENERATOR AND CHECKER

- A **parity generator** is a combinational logic circuit that **generates the parity bit** in the transmitter. On the other hand, a circuit that **checks the parity** in the receiver is called **parity checker**.

- The sum of the data bits and parity bits can be even or odd . In **even parity**, the added parity bit will make the total **number of 1s an even amount** whereas in **odd parity** the added parity bit will make the total **number of 1s odd amount.**

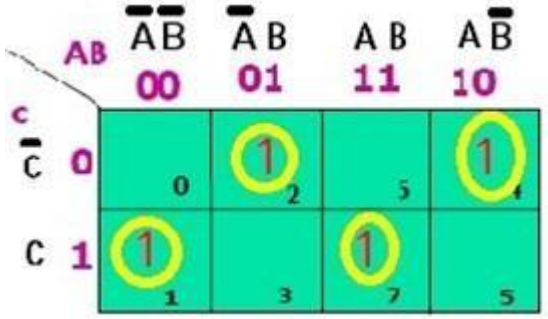


EVEN PARITY GENERATOR

In **even parity** bit scheme, the parity bit is '**0**' if there are **even number of 1s** in the data stream and the parity bit is '**1**' if there are **odd number of 1s** in the data stream.

Even Parity Generator

3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

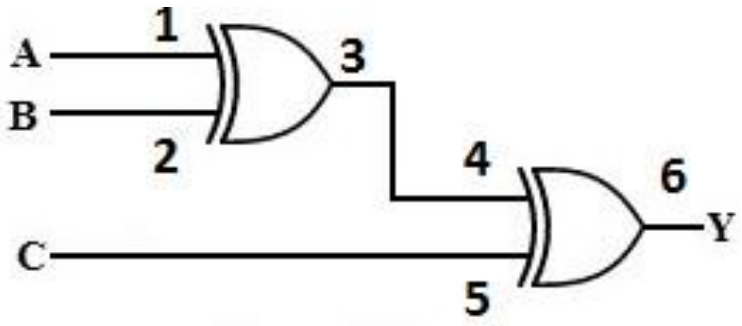


$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

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

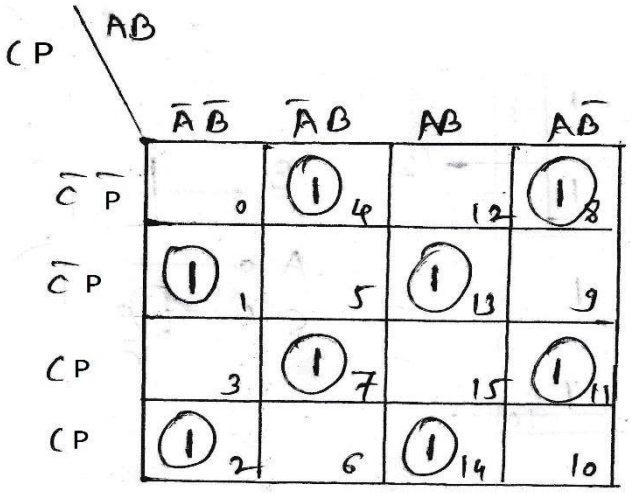
$$= \bar{A}\,(B \oplus C) + A\,(\overline{B \oplus C})$$

$$P = A \oplus B \oplus C$$



Even Parity Detector (Checker)

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



$$F = \bar{A}\bar{B}(\bar{C}P + C\bar{P}) + \bar{A}B(\bar{C}\bar{P} + CP) + AB(\bar{C}P + C\bar{P}) + A\bar{B}(\bar{C}\bar{P} + CP)$$

$$= \bar{A}\bar{B}(C \oplus P) + \bar{A}B(\overline{C \oplus P}) + AB(C \oplus P) + A\bar{B}(\overline{C \oplus P})$$

$$= (C \oplus P)(\bar{A}\bar{B} + AB) + (\overline{C \oplus P})(\bar{A}B + A\bar{B})$$

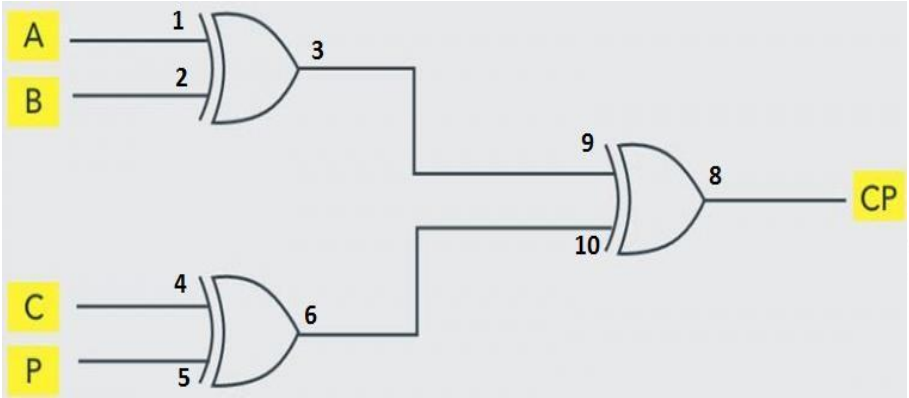
$$= (C \oplus P)(\overline{A \oplus B}) + (C\overline{\oplus P})(A \oplus B)$$

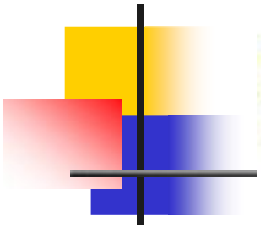
$$= E\bar{F} + \bar{E}F \quad (\text{Assume } E = C \oplus P \text{ and } F = A \oplus B)$$

$$= E \oplus F$$

$$= C \oplus P \oplus A \oplus B$$

$$F = A \oplus B \oplus C \oplus P$$



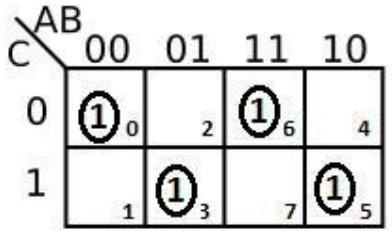


ODD PARITY GENERATOR

In **odd parity** bit scheme, the parity bit is '**1**' if there are **even number of 1**s in the data stream and the parity bit is '**0**' if there are **odd number of 1s** in the data stream.

Odd Parity Generator

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
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	0

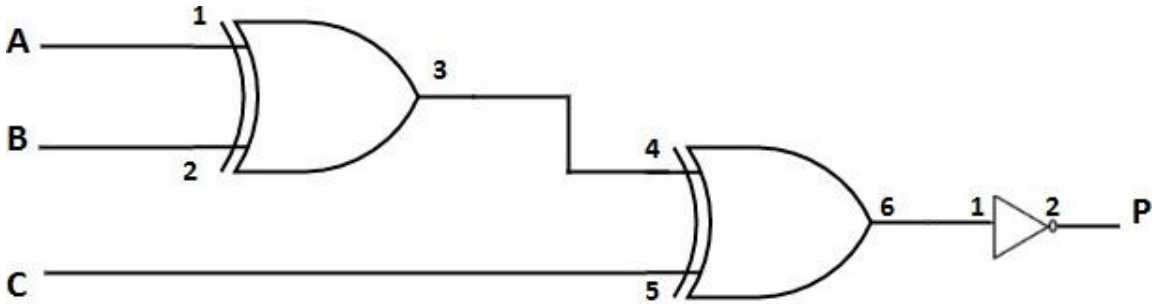


$$P = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,B\,C + A\,B\,\bar{C} + \bar{A}\,B\,C$$

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

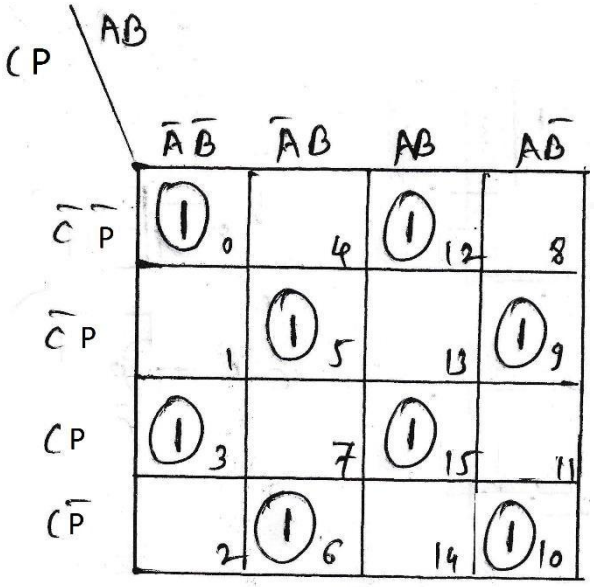
$$= \bar{A}\,\overline{(B \oplus C)} + A\,(B \oplus C)$$

$$P = \overline{A \oplus B \oplus C}$$



Odd Parity Detector (Checker)

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



$$F = \bar{A}\bar{B}(\bar{C}\bar{P}+CP)+\bar{A}B(\bar{C}P+C\bar{P})+AB(\bar{C}\bar{P}+CP)+A\bar{B}(\bar{C}P+C\bar{P})$$

$$= \bar{A}\bar{B}(\overline{C\oplus P})+\bar{A}B(C\oplus P)+AB(\overline{C\oplus P})+A\bar{B}(C\oplus P)$$

$$= (\overline{C\oplus P})(\bar{A}\bar{B}+AB)+(C\oplus P)(\bar{A}B+A\bar{B})$$

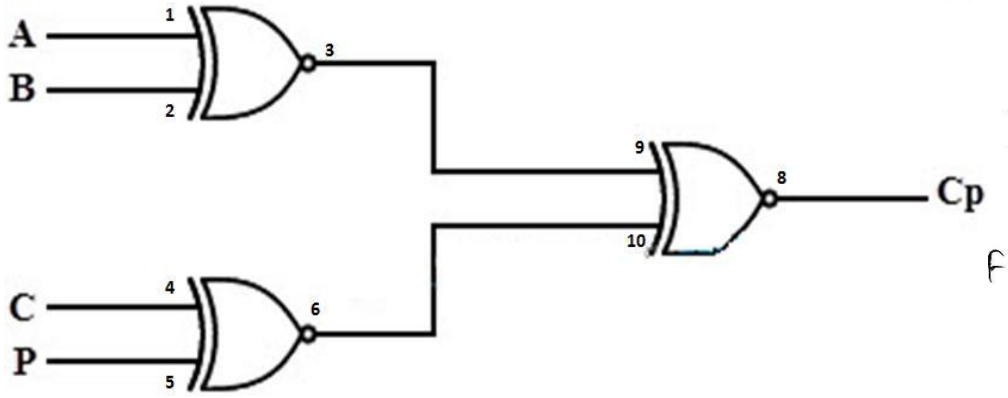
$$= (\overline{C\oplus P})(\overline{A\oplus B})+(C\oplus P)(A\oplus B)$$

$$= \bar{E}\bar{F}+EF \quad (\text{Assume } E = C\oplus P \text{ and } F = A\oplus B)$$

$$= \overline{E\oplus F}$$

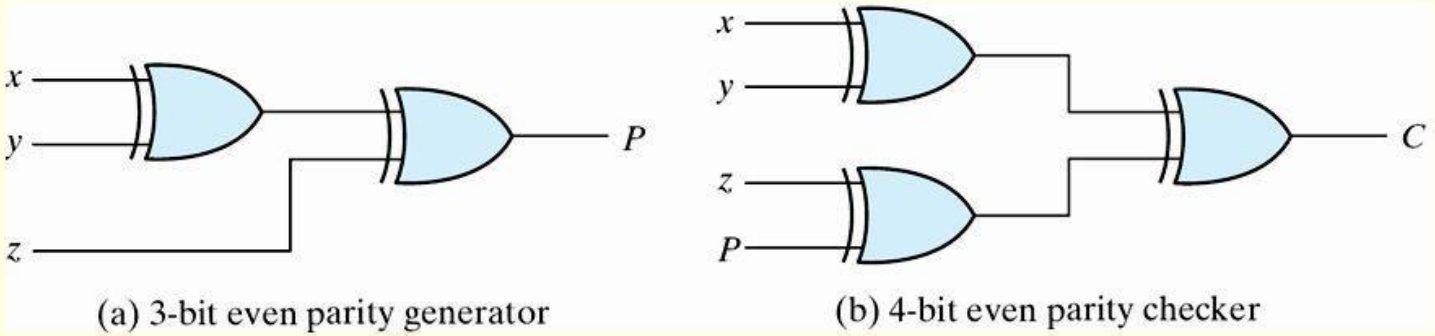
$$= \overline{C\oplus P\oplus A\oplus B}$$

$$F = \overline{A\oplus B\oplus C\oplus P}$$



Parity Generation and Checking

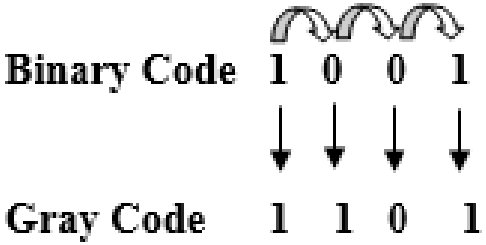
- Parity Generation and Checking
  - A parity bit: P = $x \oplus y \oplus z$
  - Parity check: C = $x \oplus y \oplus z \oplus P$
    - C=1: one bit error or an odd number of data bit error
    - C=0: correct or an even # of data bit error



(a) 3-bit even parity generator (b) 4-bit even parity checker

Figure 3.36 Logic Diagram of a Parity Generator and Checker

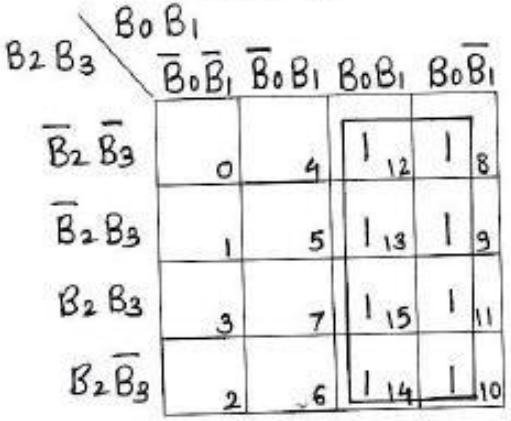
Binary to Gray Code Converter



Dec No.	B_0	B_1	B_2	B_3	G_0	G_1	G_2	G_3	Dec No.
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1
2	0	0	1	0	0	0	1	1	3
3	0	0	1	1	0	0	1	0	2
4	0	1	0	0	0	1	1	0	6
5	0	1	0	1	0	1	1	1	7
6	0	1	1	0	0	1	0	1	5
7	0	1	1	1	0	1	0	0	4
8	1	0	0	0	1	1	0	0	12
9	1	0	0	1	1	1	0	1	13
10	1	0	1	0	1	1	1	1	15
11	1	0	1	1	1	1	1	0	14
12	1	1	0	0	1	0	1	0	10
13	1	1	0	1	1	0	1	1	11
14	1	1	1	0	1	0	0	1	9
15	1	1	1	1	1	0	0	0	8

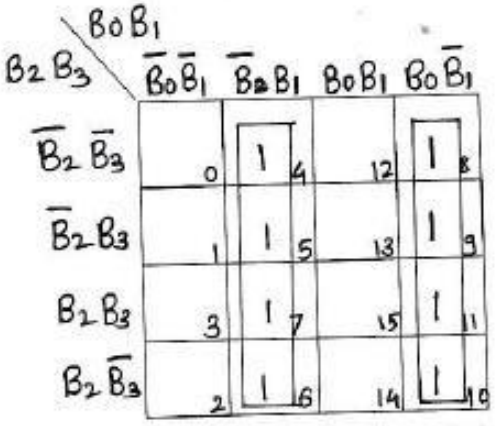
Binary to Gray Code Converter

Kmap for G_0:



$G_0 = B_0$

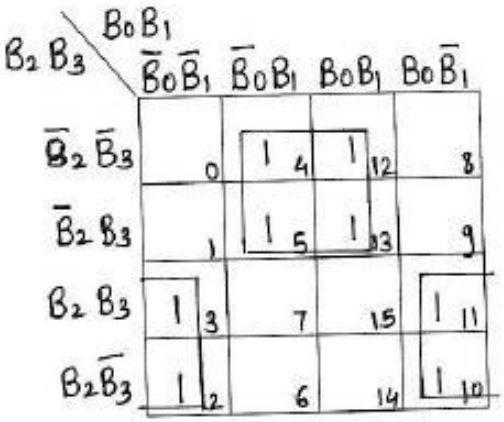
Kmap for G_1:



$G_1 = \bar{B}_0 B_1 + B_0 \bar{B}_1$

$G_1 = B_0 \oplus B_1$

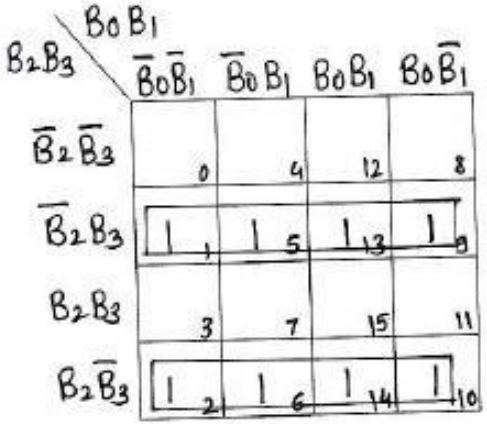
Kmap for G_2:



$G_2 = B_1 \bar{B}_2 + \bar{B}_1 B_2$

$G_2 = B_1 \oplus B_2$

Kmap for G_3:

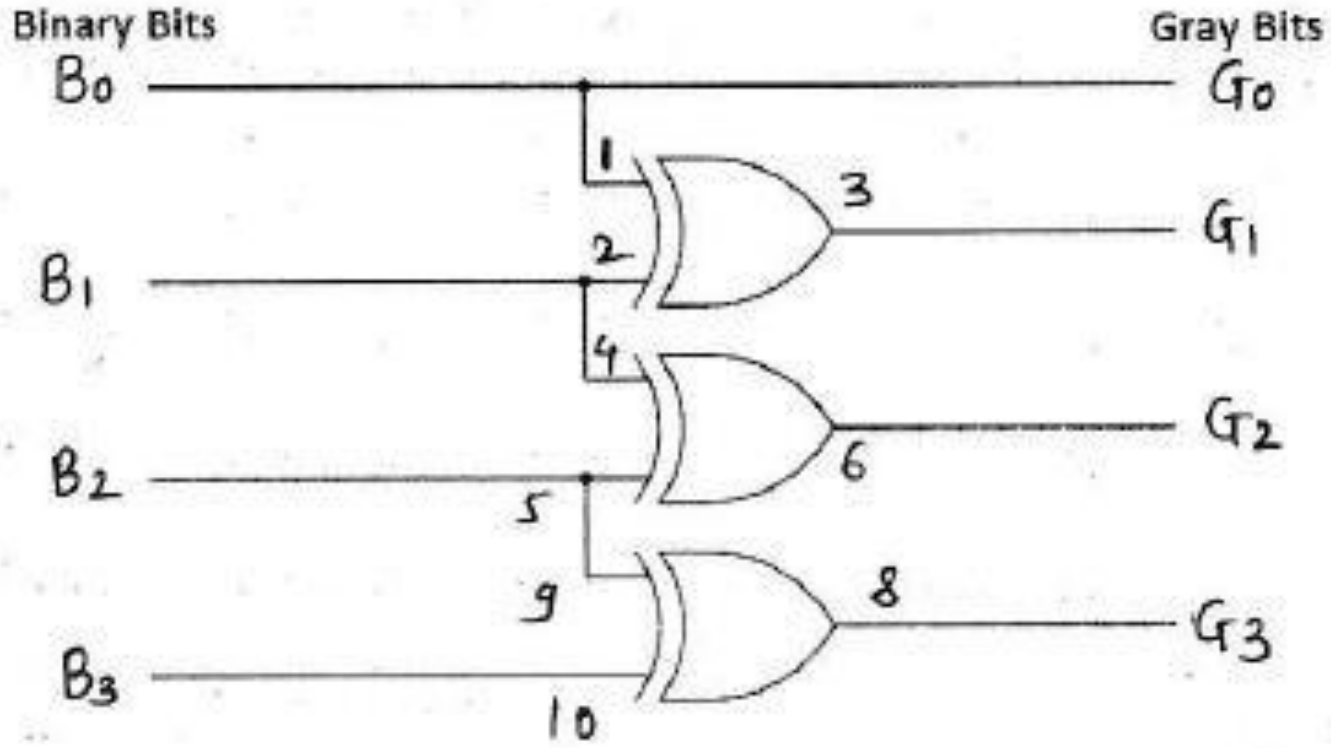


$G_3 = \bar{B}_2 B_3 + B_2 \bar{B}_3$

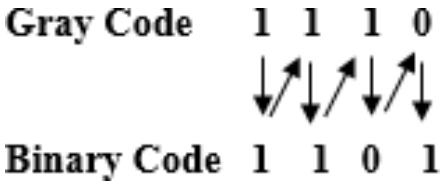
$G_3 = B_2 \oplus B_3$

Binary to Gray Code Converter

Circuit Diagram:



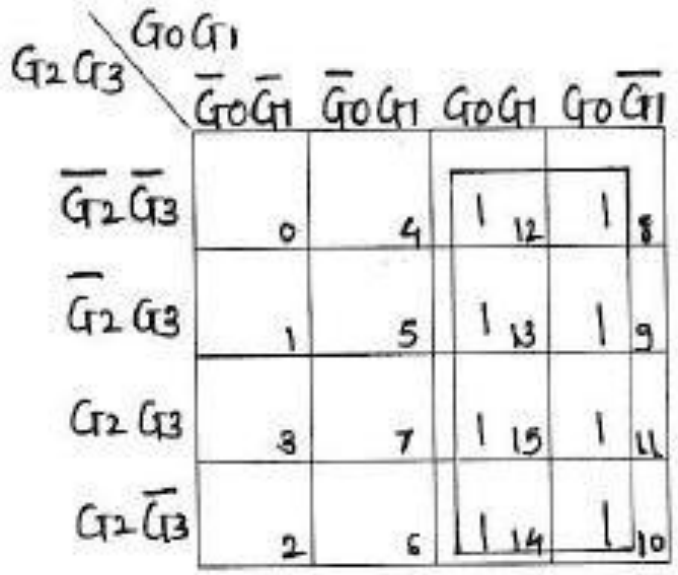
Gray to Binary Code Converter



Dec No.	G_0	G_1	G_2	G_3	B_0	B_1	B_2	B_3
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	0	0
7	0	1	1	1	0	1	0	1
8	1	0	0	0	1	1	1	1
9	1	0	0	1	1	1	1	0
10	1	0	1	0	1	1	0	0
11	1	0	1	1	1	1	0	1
12	1	1	0	0	1	0	0	0
13	1	1	0	1	1	0	0	1
14	1	1	1	0	1	0	1	1
15	1	1	1	1	1	0	1	0

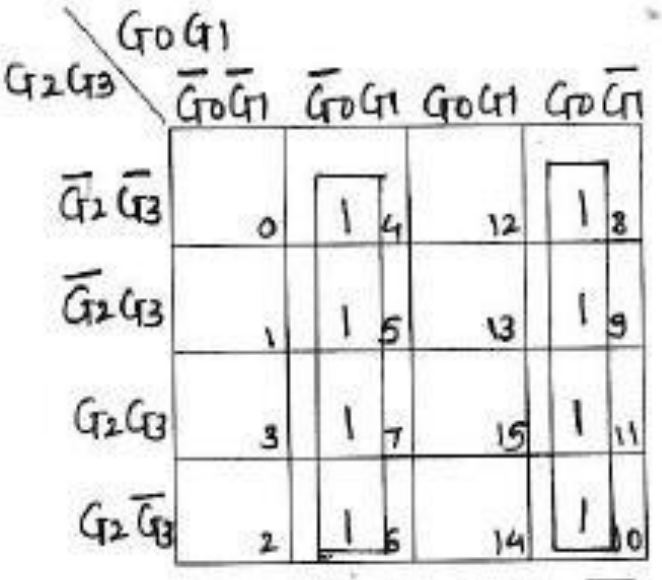
Gray to Binary Code Converter

Kmap for B_0



$B_0 = G_0$

Kmap for B_1

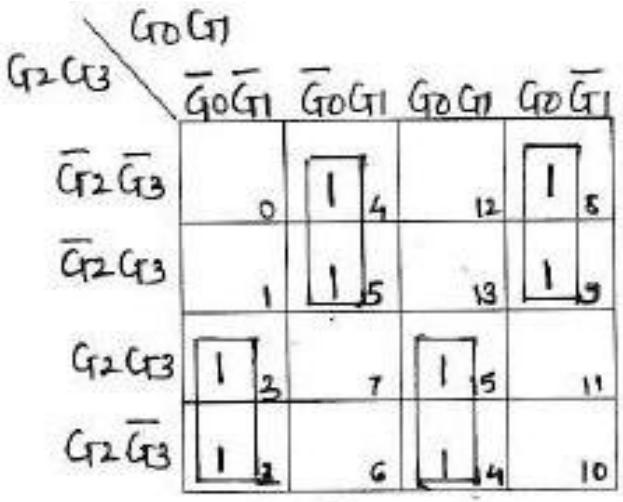


$B_1 = \bar{G}_0 G_1 + G_0 \bar{G}_1$

$B_1 = G_0 \oplus G_1$

Gray to Binary Code Converter

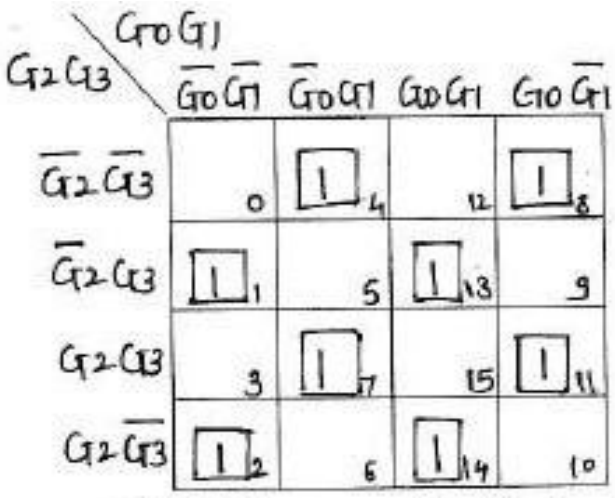
Kmap for B_2



$$B_2 = \bar{G}_0\bar{G}_1G_2 + \bar{G}_0G_1\bar{G}_2 + G_0G_1G_2 + G_0\bar{G}_1\bar{G}_2$$

$$\boxed{B_2 = G_0 \oplus G_1 \oplus G_2}$$

Kmap for B_3



$$B_3 = \bar{G}_0\bar{G}_1\bar{G}_2G_3 + \bar{G}_0\bar{G}_1G_2\bar{G}_3 + \bar{G}_0G_1\bar{G}_2\bar{G}_3 + \bar{G}_0G_1G_2G_3 + G_0G_1\bar{G}_2G_3 + G_0G_1G_2\bar{G}_3 + G_0\bar{G}_1\bar{G}_2\bar{G}_3 + G_0\bar{G}_1G_2G_3$$

$$\boxed{B_3 = G_0 \oplus G_1 \oplus G_2 \oplus G_3}$$

Gray to Binary Code Converter

Circuit Diagram: