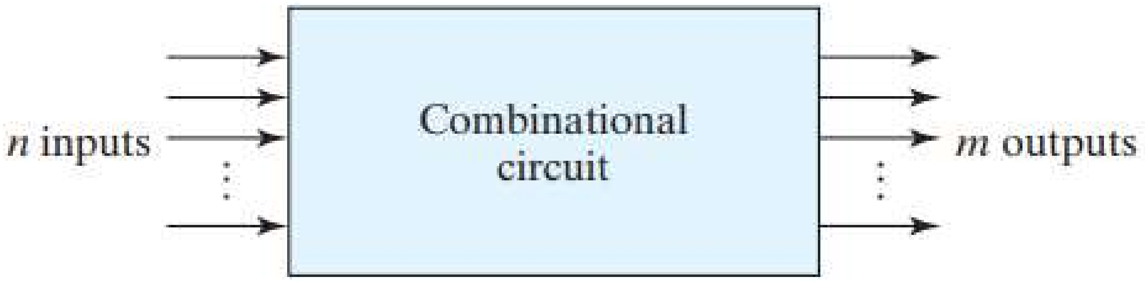
UNIT-II

Combinational Logic Circuits

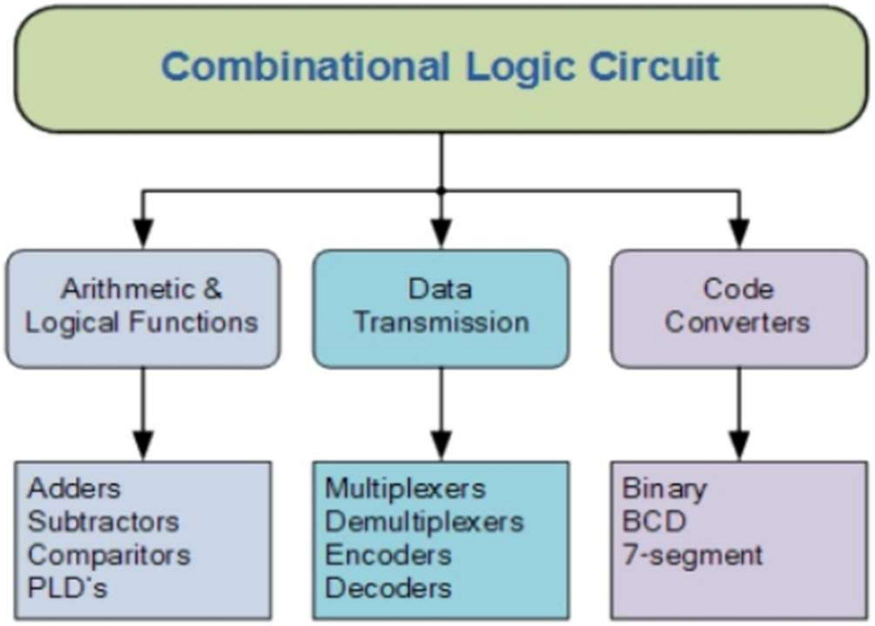
Combinational Circuit

- Interconnection of logic gates
- n-inputs are obtained by external source
- *m* output variables are produced by the internal combinational logic circuit and go to an external destination
- The diagram of a combinational circuit has logic gates with no feedback paths or memory elements



Classification

Classification of Combinational Logic



Implementation of Combinational circuits

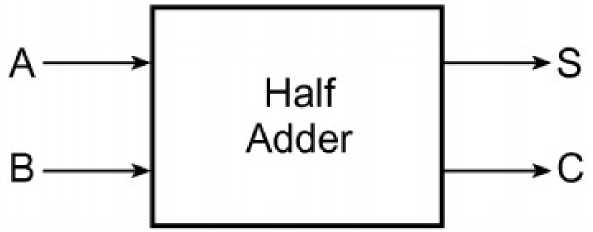
The different steps involved in the design of a combinational logic circuit are as follows:

1. Statement of the problem.
2. Identification of input and output variables.
3. Expressing the relationship between the input and output variables.
4. Construction of a truth table to meet input–output requirements.
5. Writing Boolean expressions for various output variables in terms of input variables.
6. Minimization of Boolean expressions.
7. Implementation of minimized Boolean expressions.

Design of a Half adder

A *half-adder* is an arithmetic circuit block that can be used to add two bits

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



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

CARRY $C = A.B$

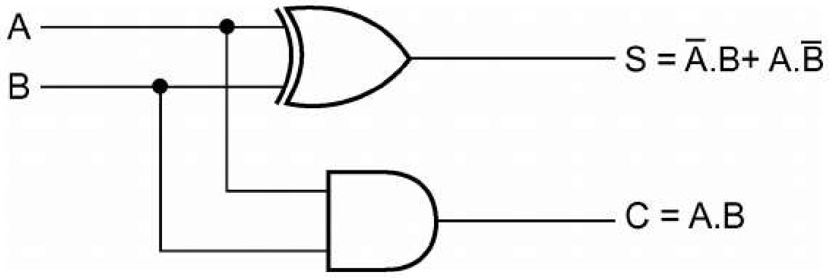


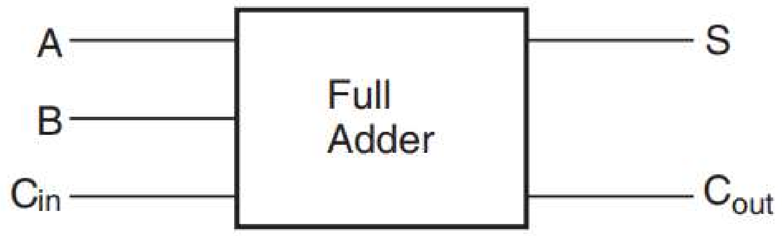
Figure 7.5 Logic implementation of a half-adder.

Design of a full adder

A *full-adder* is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY output

Addition of n-bit binary numbers require the use of a full adder

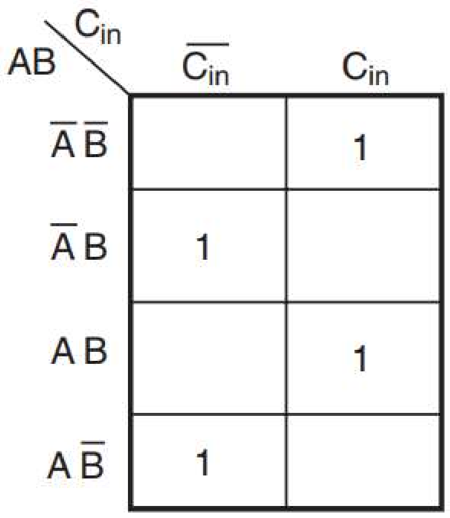
The process of addition proceeds bit by bit basis, right to left, from LSB to MSB



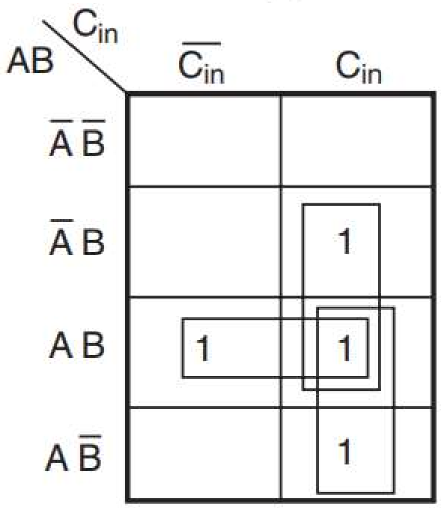
A	B	C_{in}	SUM (S)	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table of a full adder.

Design of a full adder



(a)



Karnaugh maps for the sum and carry-out of a full adder.

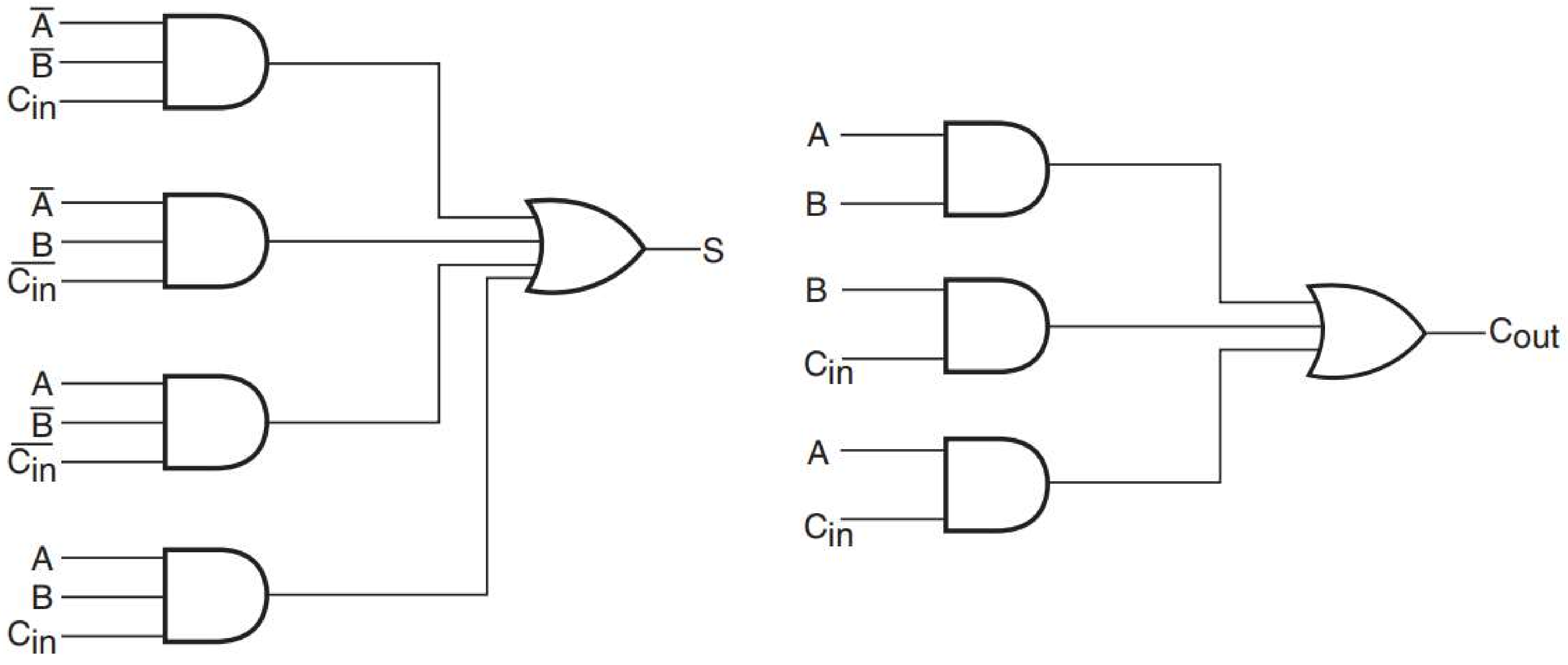
A	B	C_{in}	SUM (S)	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table of a full adder.

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$

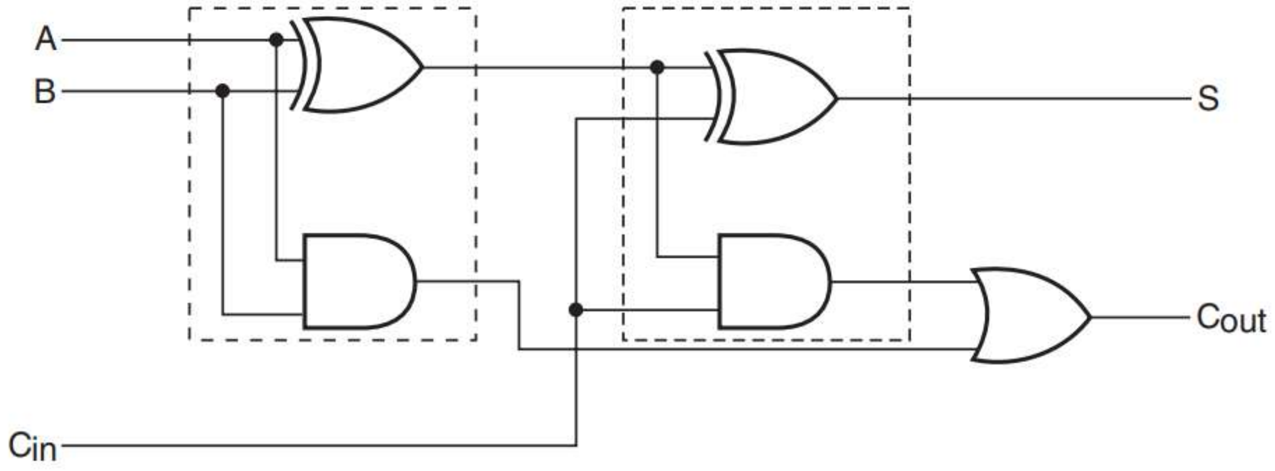
$$C_{out} = B.C_{in} + A.B + A.C_{in}$$

Design of a full adder with basic gates

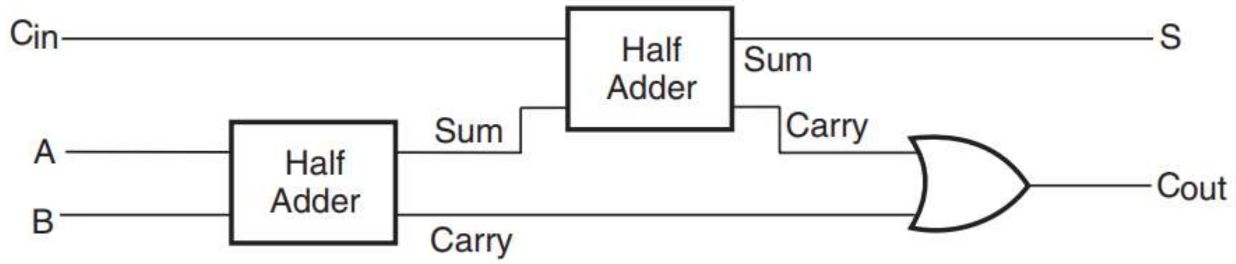


Logic circuit diagram of a full adder.

Design of full adder with two half adders

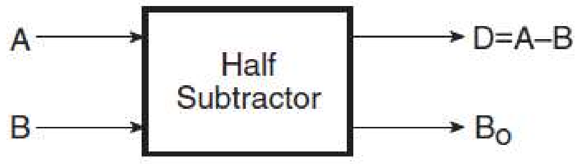


(a)



(b)

Design of half subtractor



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

Figure 7.12 Half-subtractor.

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

$$B_o = \overline{A}.B$$

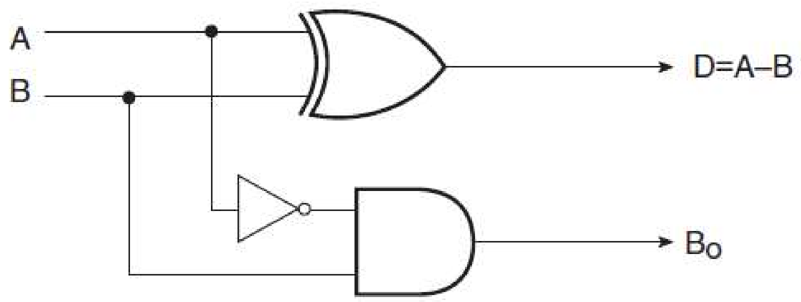
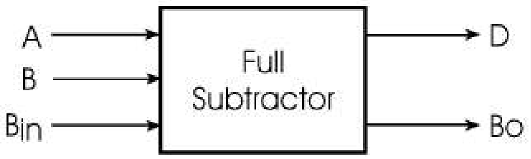


Figure 7.13 Logic diagram of a half-subtractor.

Design of full subtractor

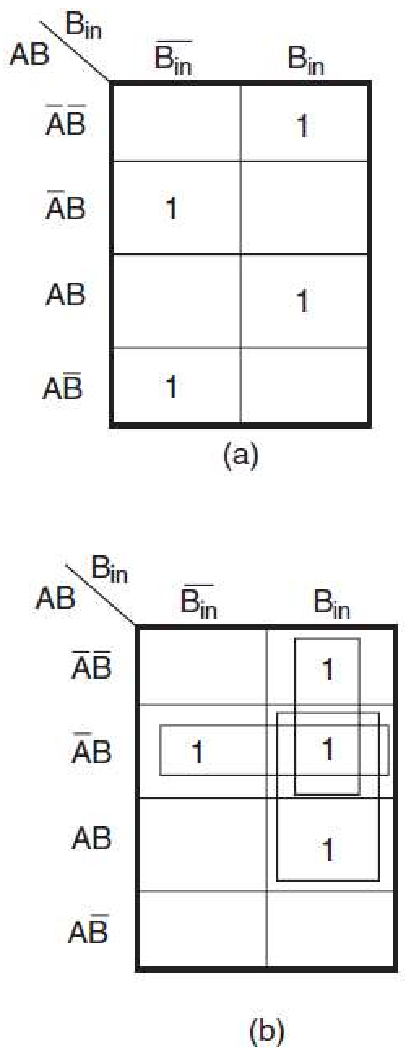


Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figure 7.14 Truth table of a full subtractor.

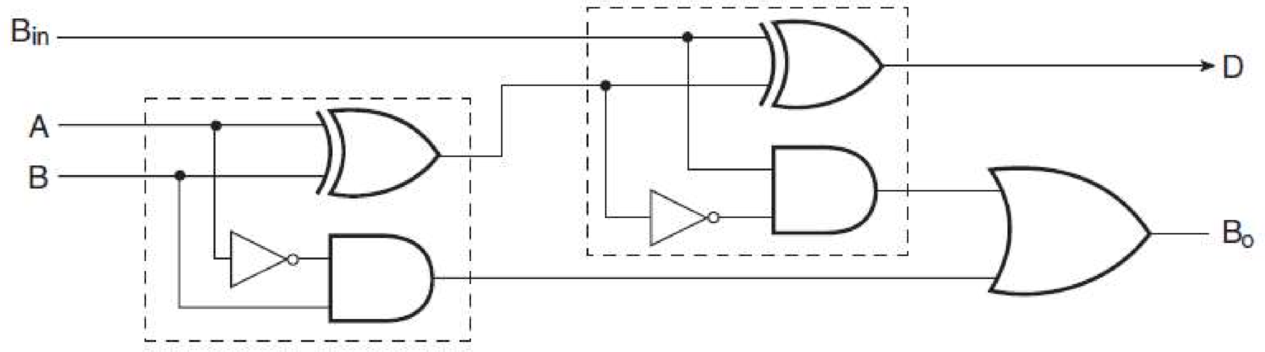
$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in}$$

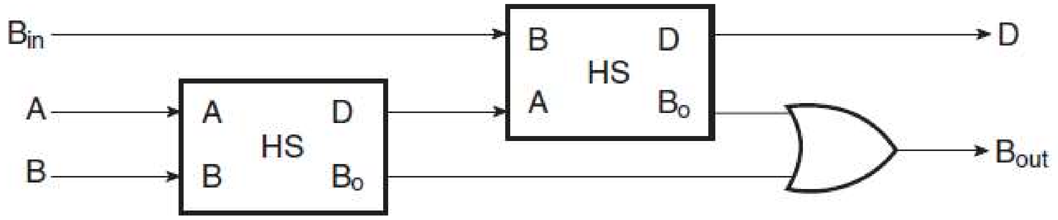


Karnaugh maps for difference and borrow outputs.

Design of full subtractor with two half subtractors



(a)



(b)

Figure 7.16 Logic implementation of a full subtractor with half-subtractors.

Binary adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.

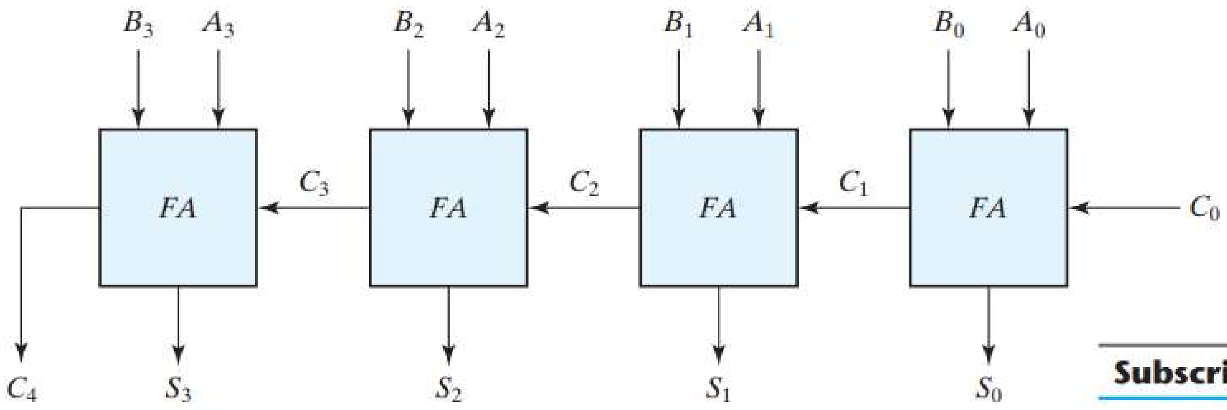


FIGURE 4.9
Four-bit adder

Subscript *i*:	**3**	**2**	**1**	**0**	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Ripple adder

- Inputs $A3$ and $B3$ are available as soon as input signals are applied to the adder. However, input carry $C3$ does not settle to its final value until $C2$ is available from the previous stage. Similarly, $C2$ has to wait for $C1$ and so on down to $C0$. Thus, only after the carry propagates and ripples through all stages will the last output $S3$ and carry $C4$ settle to their final correct value.

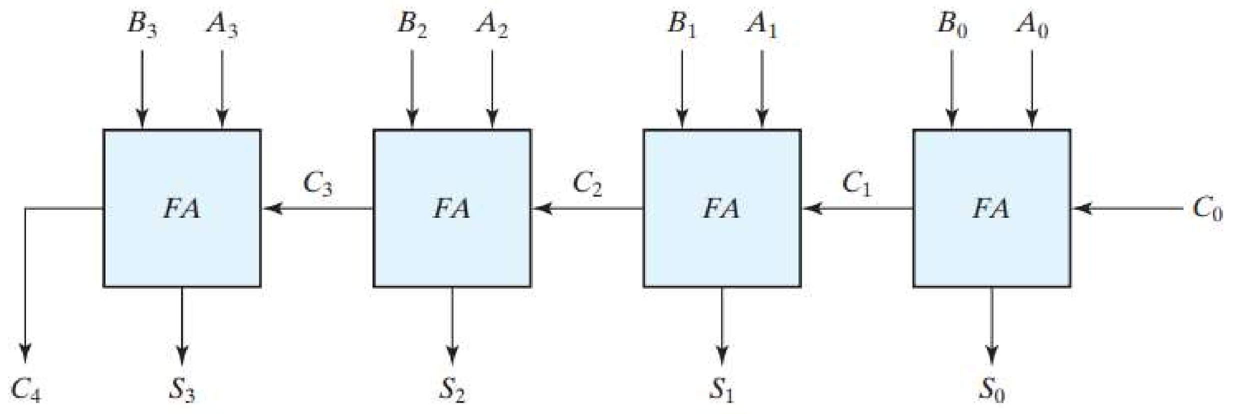
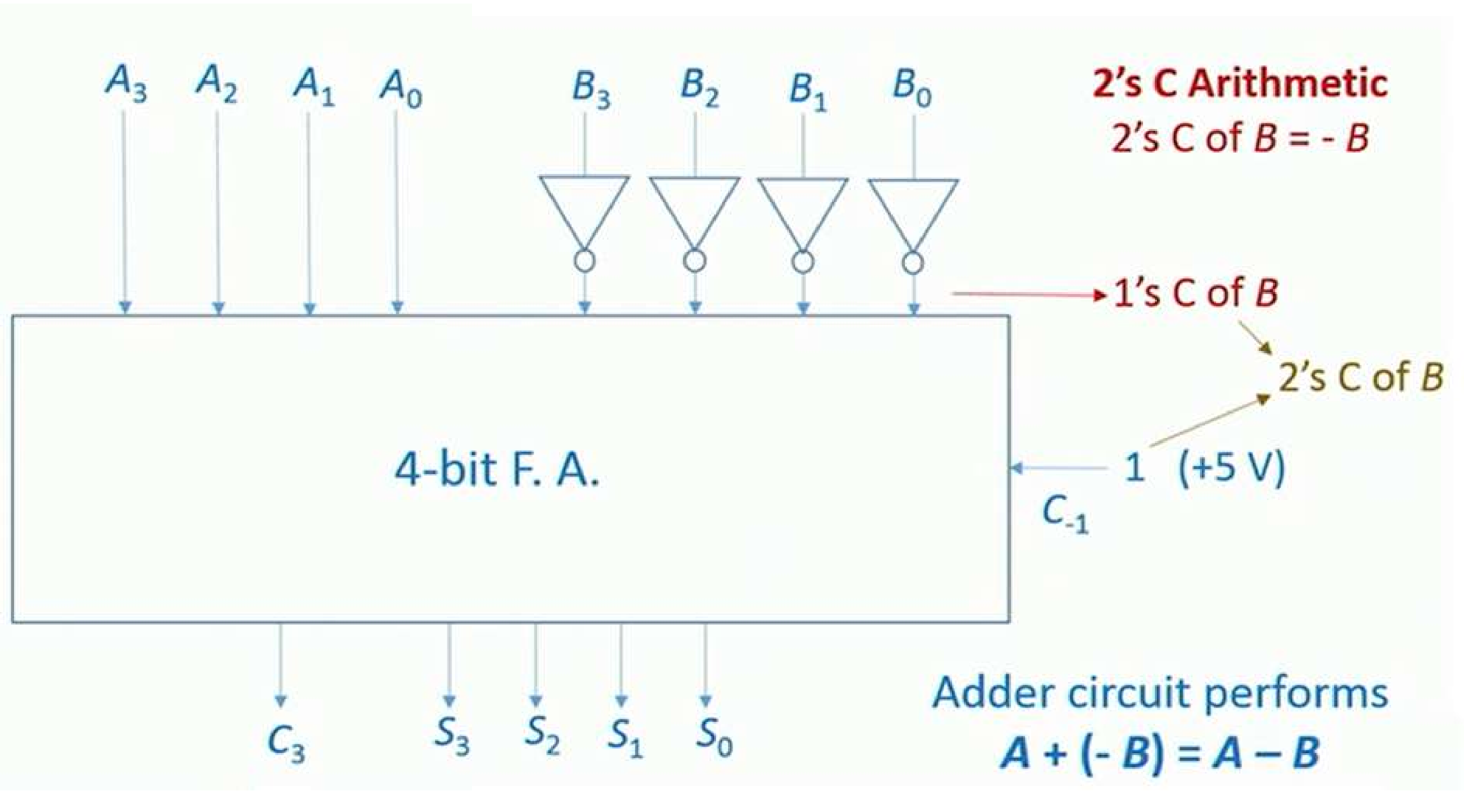
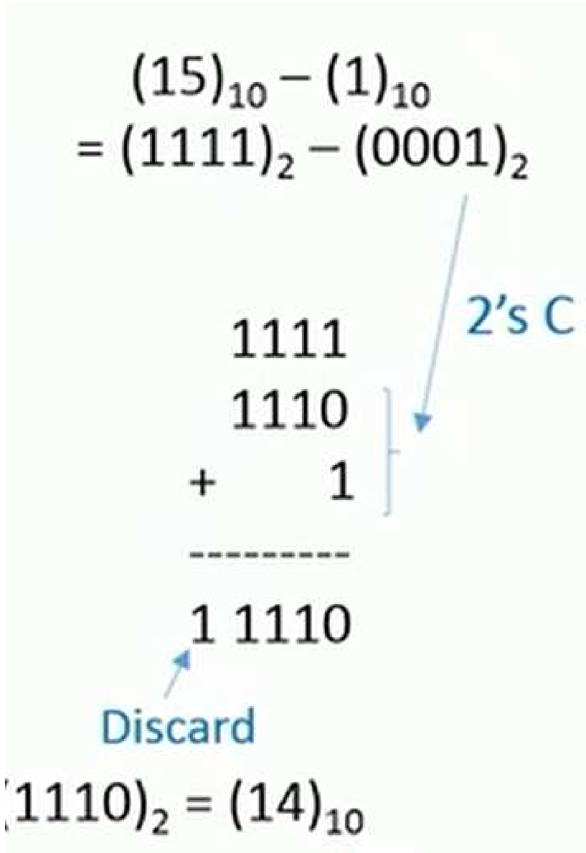


FIGURE 4.9
Four-bit adder

Drawback: The time required to add long data words may be prohibitive, because the carry has to propagate from the least significant bit to the most significant bit.

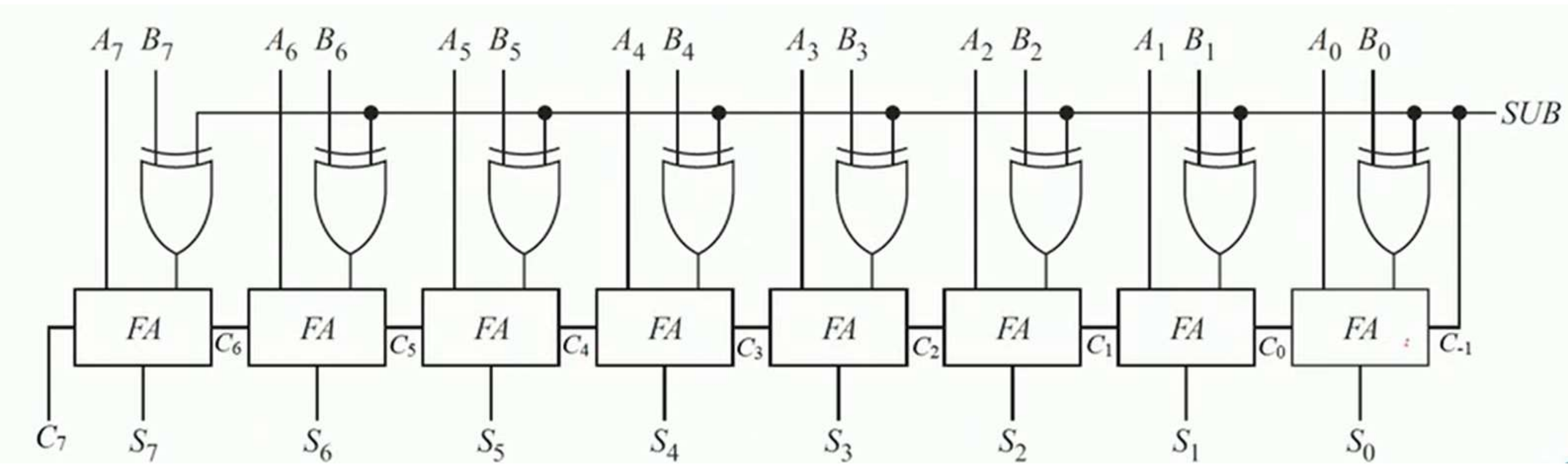
Binary subtraction using Adder circuit





- Subtraction can be done by full-adder circuit by 2's compliment method

Binary subtraction using Adder circuit



If SUB = 0, then $C_{-1} = 0$

Ex-OR gate output = $B_i \oplus 0 = B_i$

F.A. output = $A + B$

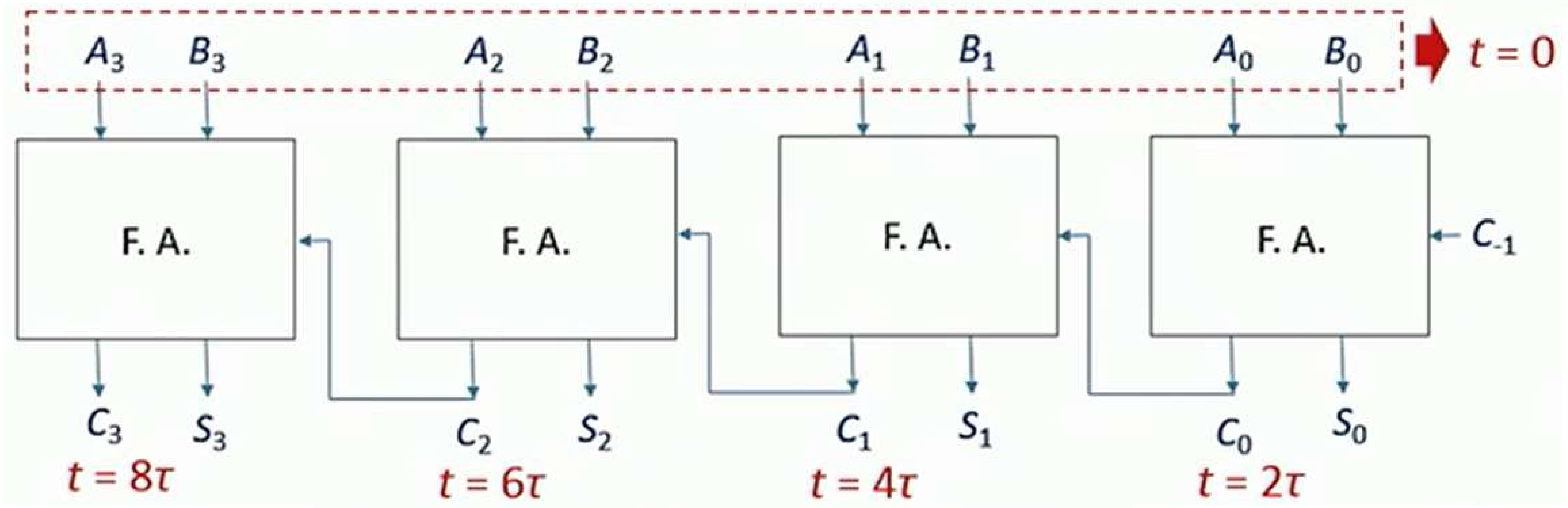
If SUB = 1, then $C_{-1} = 1$

Ex-OR gate output = $B_i \oplus 1 = B_i'$

F.A. output = $A + (-B) = A - B$

Binary subtraction using Adder circuit

Propagation delay is cumulative.
For n-bit addition, $2n\tau$ delay.



Each basic gate delay = τ
(AND, OR)

Each Ex-OR gate delay = 2τ

C_3 available after 8τ delay.

For S_3, $A_3 \oplus B_3$ result after 2τ.
This is Ex-Ored with C_2.
C_2 available after 6τ.
S_3 available after 8τ.

Carry look ahead adder

- The carry propagation time is an important attribute of the adder because it limits the speed with which two numbers are added.
- *carry look ahead logic* is a technique for reducing the carry propagation time in a parallel adder
- Pi called CARRY PROPAGATE and Gi called CARRY GENERATE.

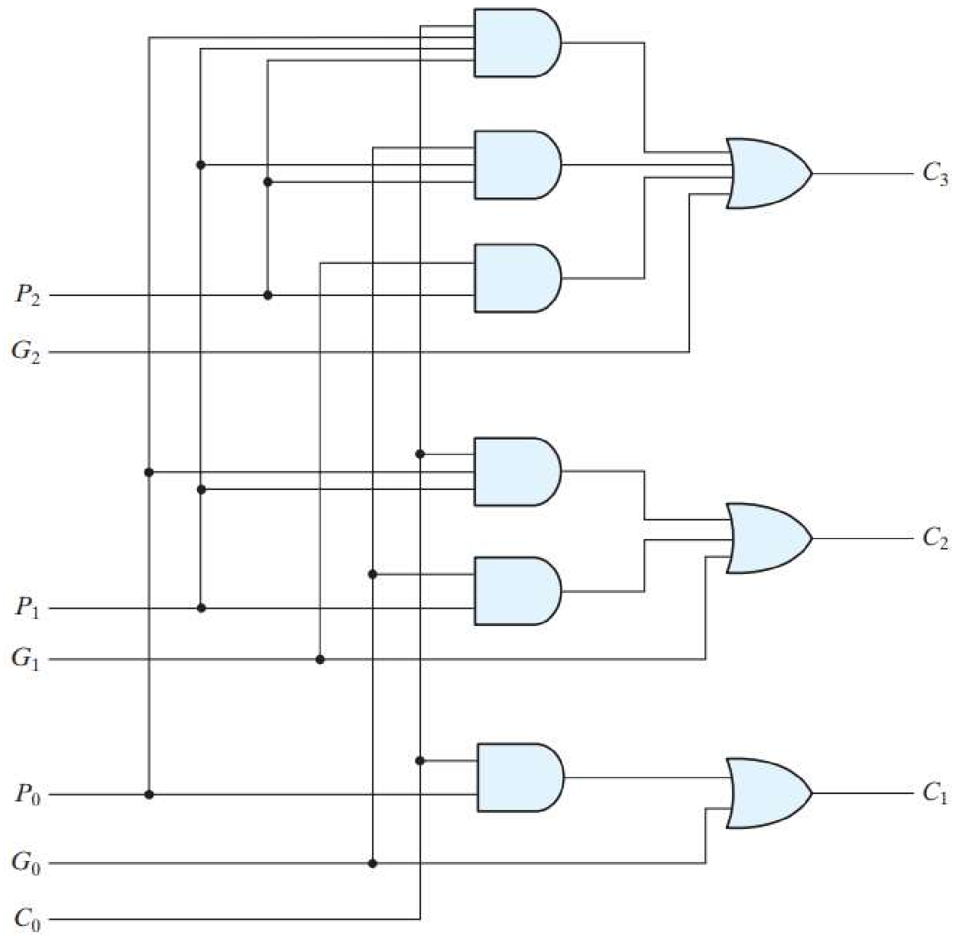
$$P_i = A_i \oplus B_i \qquad S_i = P_i \oplus C_i$$
$$G_i = A_i B_i \qquad C_{i+1} = G_i + P_i C_i$$

$C_0 =$ input carry

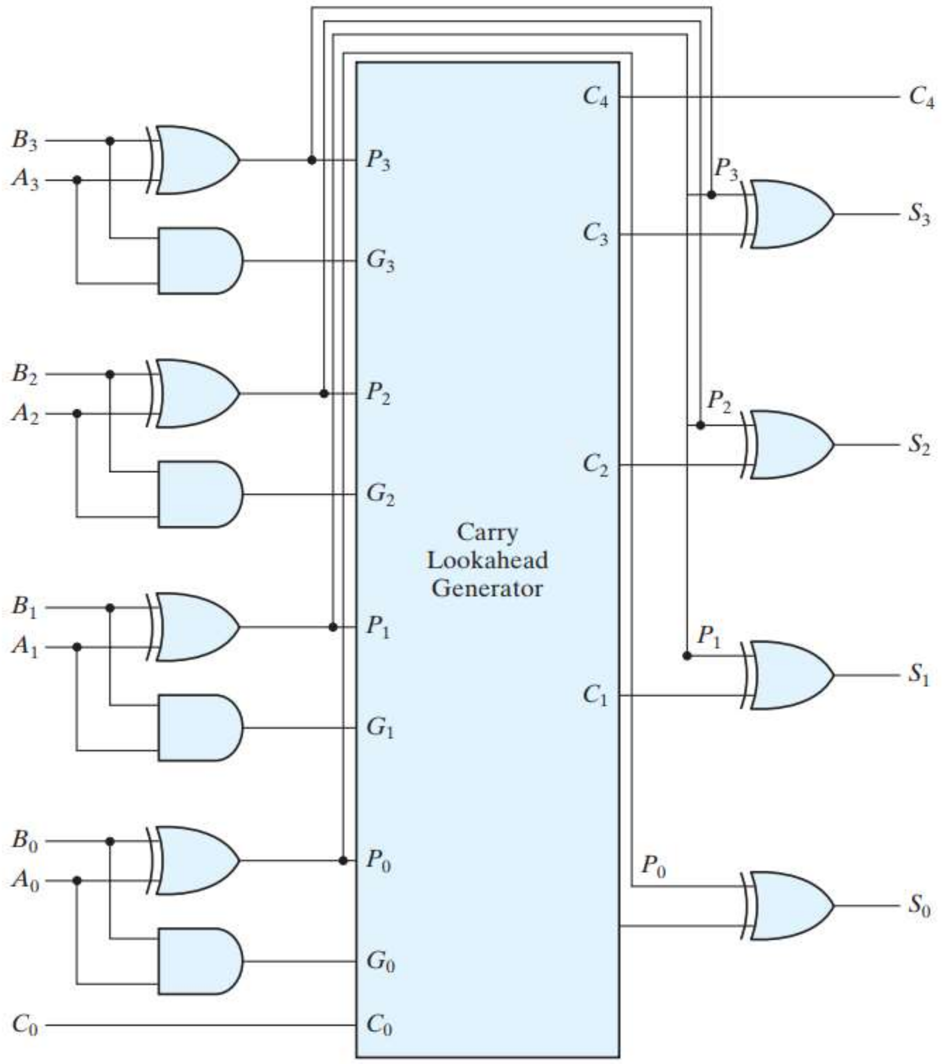
$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 = P_2 P_1 P_0 C_0$



Carry look ahead adder

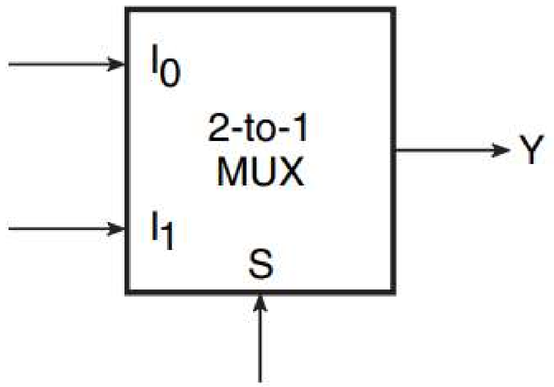


Multiplexer

- A multiplexer steers one of the many inputs to an output based on the select inputs.
- Mux is also called data selector or many to one circuit or the universal logic circuit or parallel to serial converter
- Used to design more complex combinational circuits

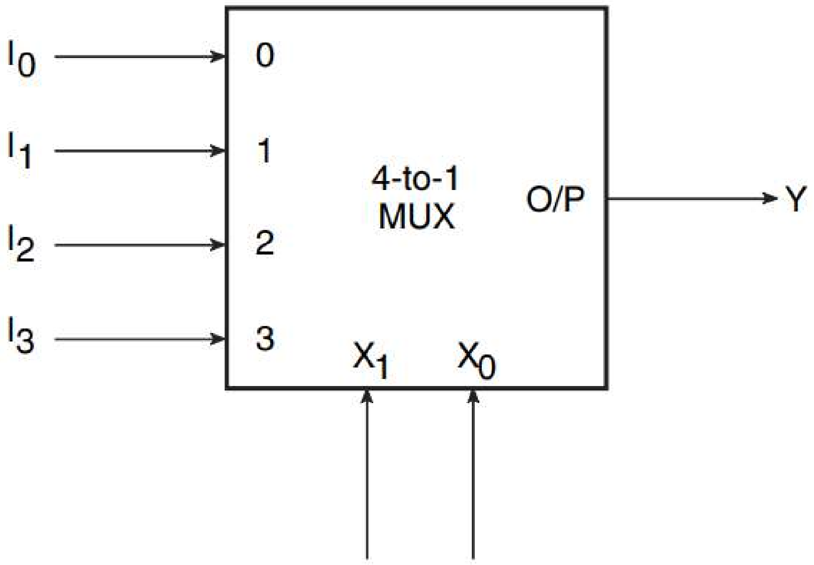
  A multiplexer selects binary information present on any one of the input lines, depending upon the logic status of the selection inputs, and routes it to the output line.
- If there are n selection lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n-to-1 multiplexer or $2^n \times 1$ multiplexer.
- Mux contains AND gate followed by OR gate

Multiplexer or Data Selector



S	Y
0	I_0
1	I_1

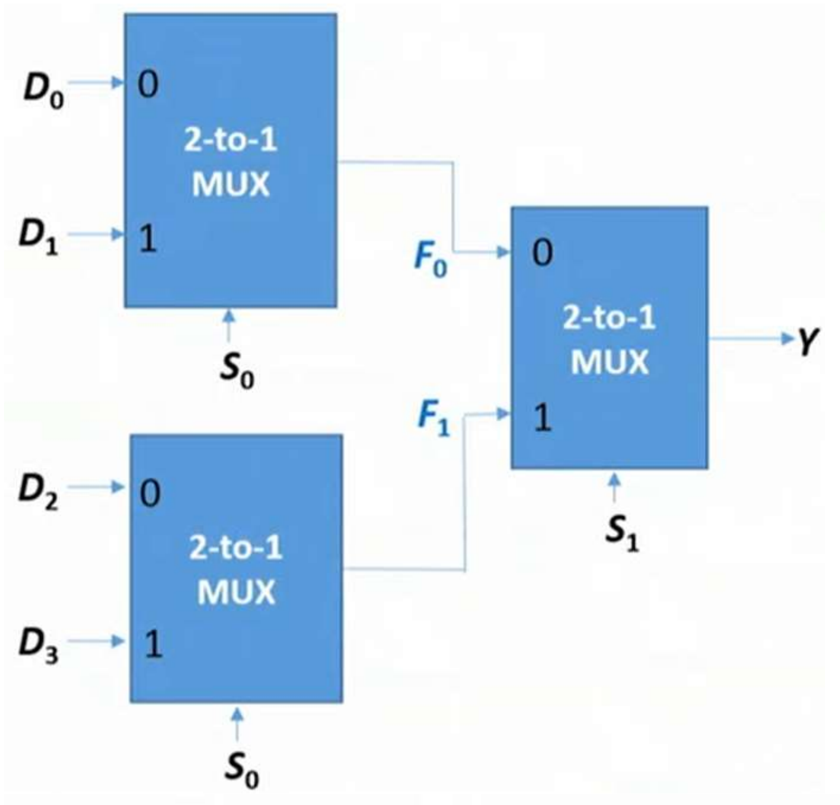
$$Y = S_0' I_0 + S_1 I_1$$



X_1	X_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = S_0' S_1' I_0 + S_0' S_1 I_1 + S_0 S_1' I_2 + S_0 S_1 I_3$$

4-to-1 Multiplexer from 2-to-1 Mux

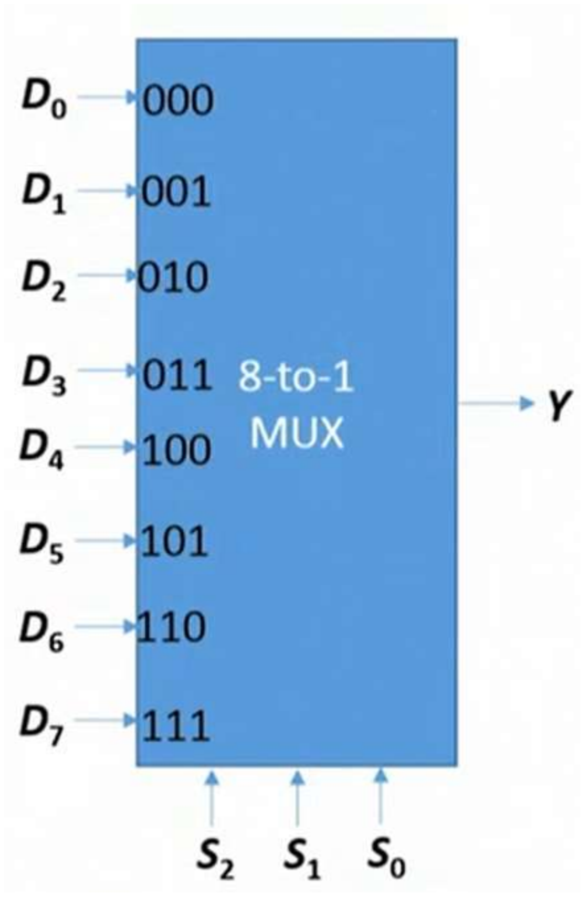


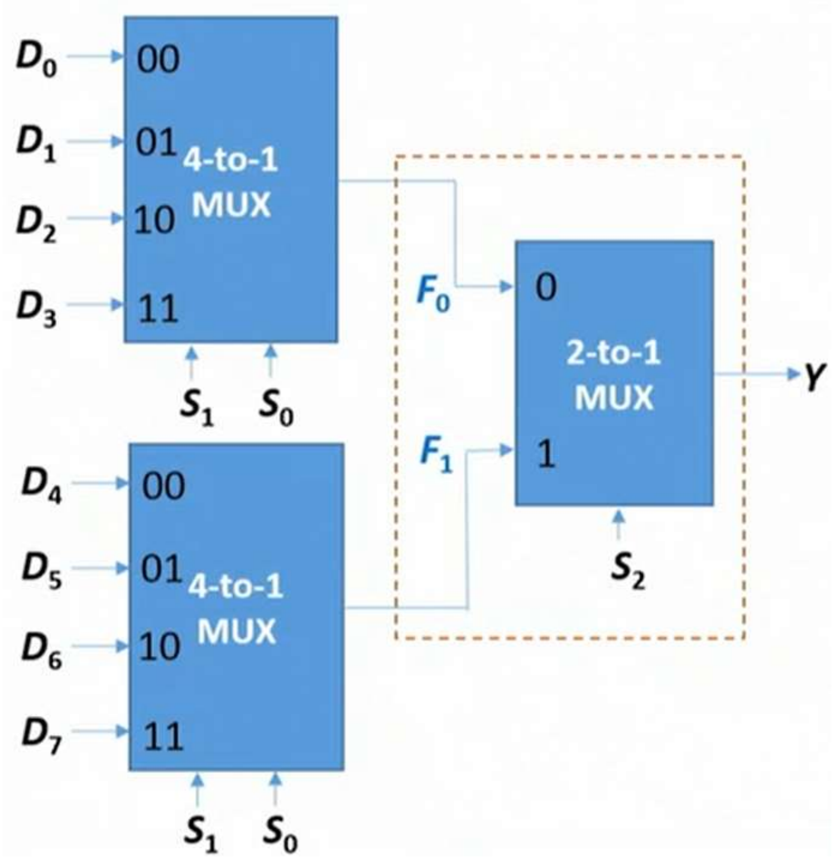
$$Y = S_1'S_0'D_0 + S_1'S_0D_1 + S_1S_0'D_2 + S_1S_0D_3$$
$$= S_1'.(S_0'D_0 + S_0D_1) + S_1.(S_0'D_2 + S_0D_3)$$
$$= S_1'.F_0 + S_1.F_1$$

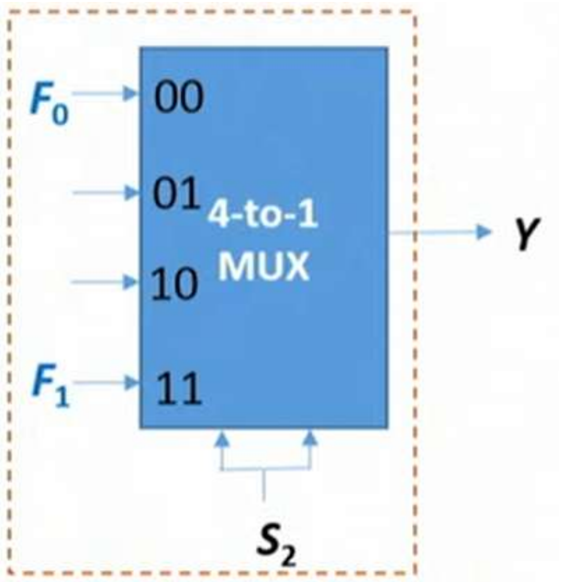
- A multiple number of devices (Mux) of a given size can be used to construct multiplexers that can handle a larger number of input channels.
- For instance, 8-to-1 multiplexers can be used to construct 16-to-1 or 32-to-1 or even larger multiplexer circuits.

Higher order Mux from a lower order Mux

- Higher order multiplexer can be obtained from lower order multiplexers by cascading







Multiplexer for implementing Boolean function

- Multiplexer as a universal logic circuit
  - efficient method for implementing a Boolean function of *n* variables with a multiplexer that has *n* - 1 selection inputs
  - first n - 1 variables of the function are connected to the selection inputs of the multiplexer
  - Remaining single variable of the function is used for the data inputs.

Implement 16x1 Mux with lower order multiplexers

Multiplexer for implementing Boolean function

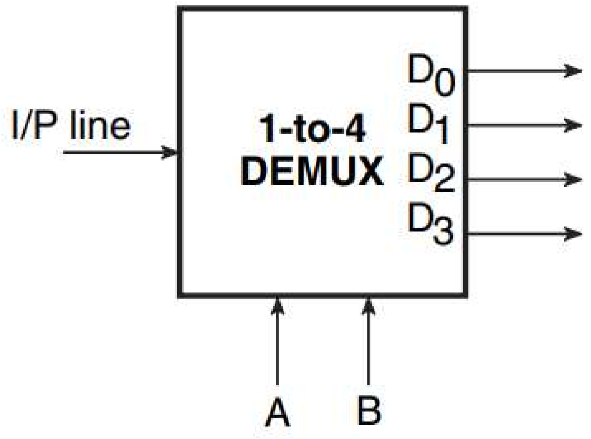
$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$

Multiplexer for implementing Boolean function

$$Y = F(A, B, C, D) = \Sigma m(0, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15)$$

Demultiplexer

- A *demultiplexer* is a combinational logic circuit with one input line, 2^n output lines and n select lines



I/P	Select		O/P			
	A	B	D_0	D_1	D_2	D_3
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

Implementation of Full Adder using Demultiplexer

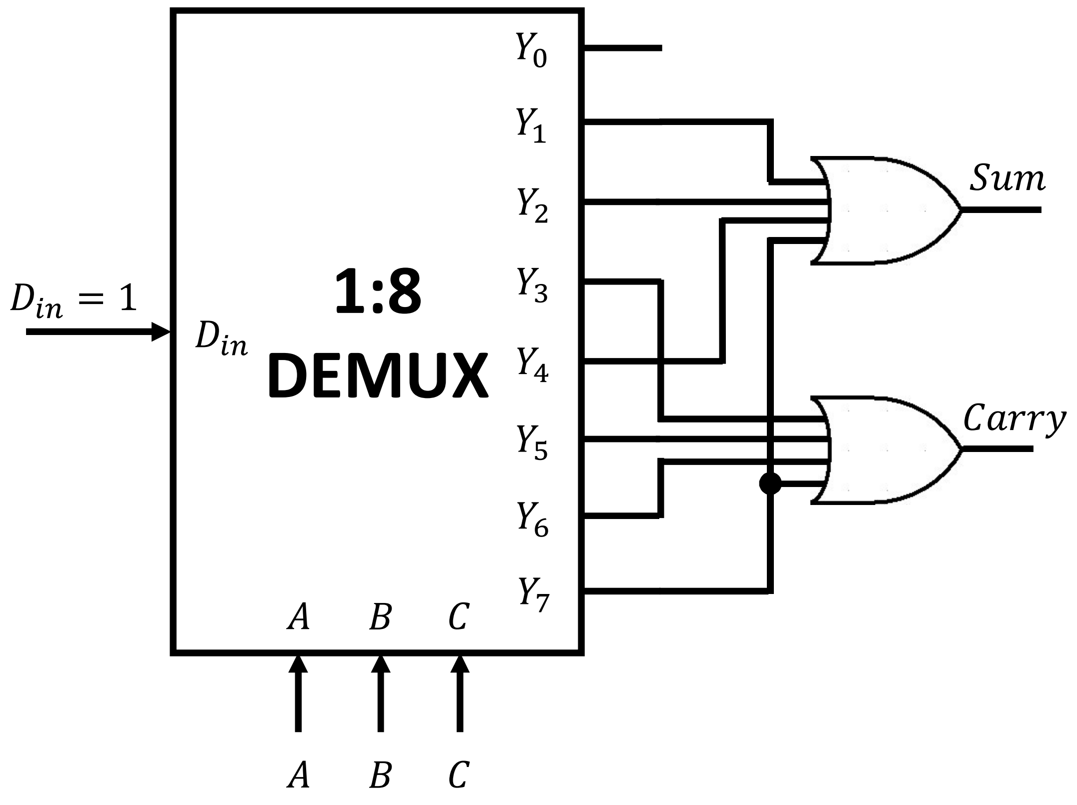
Truth Table

Decimal	Inputs			Outputs	
	A	B	C	Sum	Carry
0	0	0	0	0	0
1	0	0	1	$\textcircled{1}$	0
2	0	1	0	$\textcircled{1}$	0
3	0	1	1	0	$\textcircled{1}$
4	1	0	0	$\textcircled{1}$	0
5	1	0	1	0	$\textcircled{1}$
6	1	1	0	0	$\textcircled{1}$
7	1	1	1	$\textcircled{1}$	$\textcircled{1}$

Output Expressions are,

Sum = $\sum_m(1,2,4,7)$

Carry = $\sum_m(3,5,6,7)$



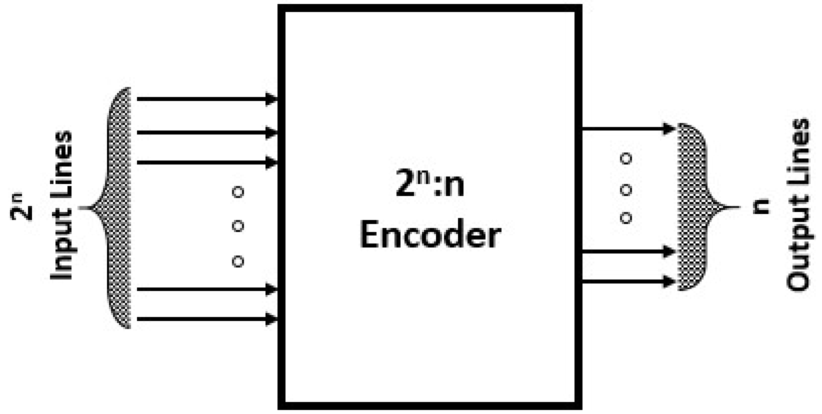
Encoder

- It is a combinational logic function that has 2^n input lines and n output lines
- The n output lines generate the binary code for the possible 2^n input lines

  Input – 2^n

  Output – n

  Encoder - 2^n:n
- An encoder accepts an active level on one of its inputs, representing digit, such as a decimal or octal digits, and converts it to a coded output such as BCD or binary.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- An encoder has a number of input lines, only one of which input is activated at a given time and produces an N-bit output code, depending on which input is activated.



Four to Two Line Binary Encoder

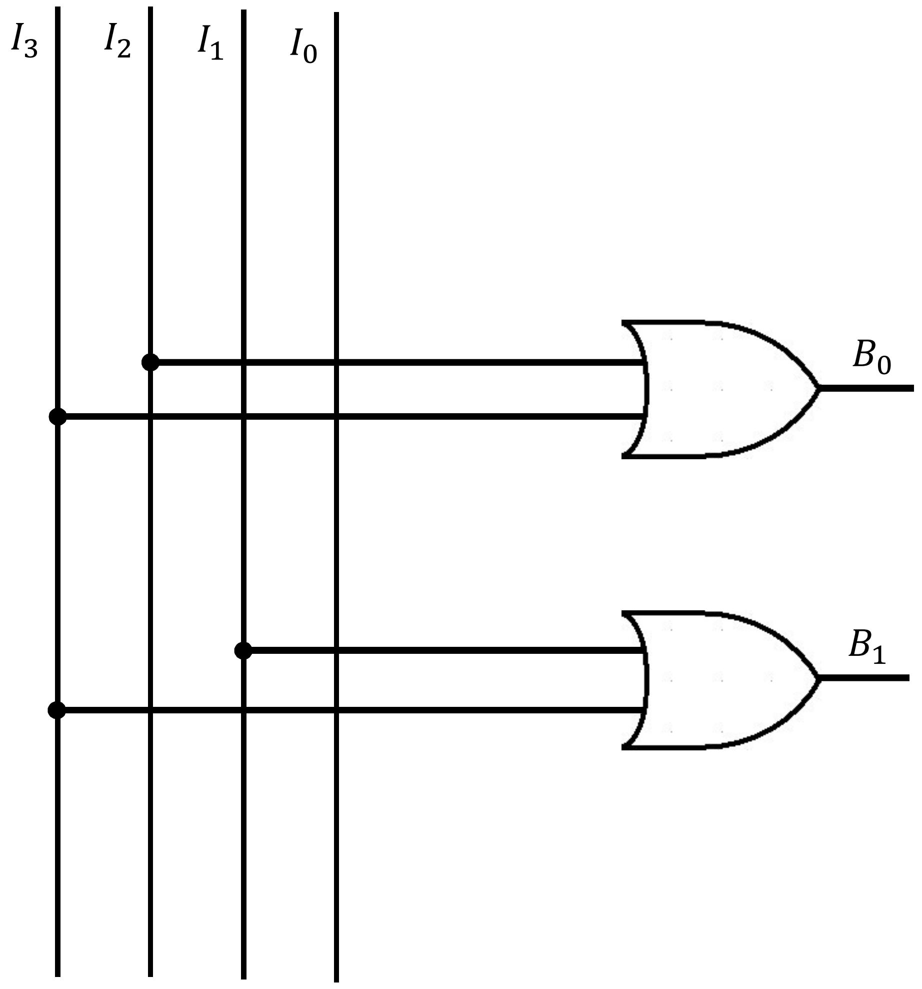
Truth Table

Decimal Value	Input				Output	
	I_3	I_2	I_1	I_0	B_1	B_0
0	0	0	0	1	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	0
3	1	0	0	0	1	1

- Input – 4
- Output – 2
- Outputs are,

$$B_0 = I_2 + I_3$$

$$B_1 = I_1 + I_3$$



Eight to Three Line Binary Encoder (Octal to Binary Encoder

Truth Table

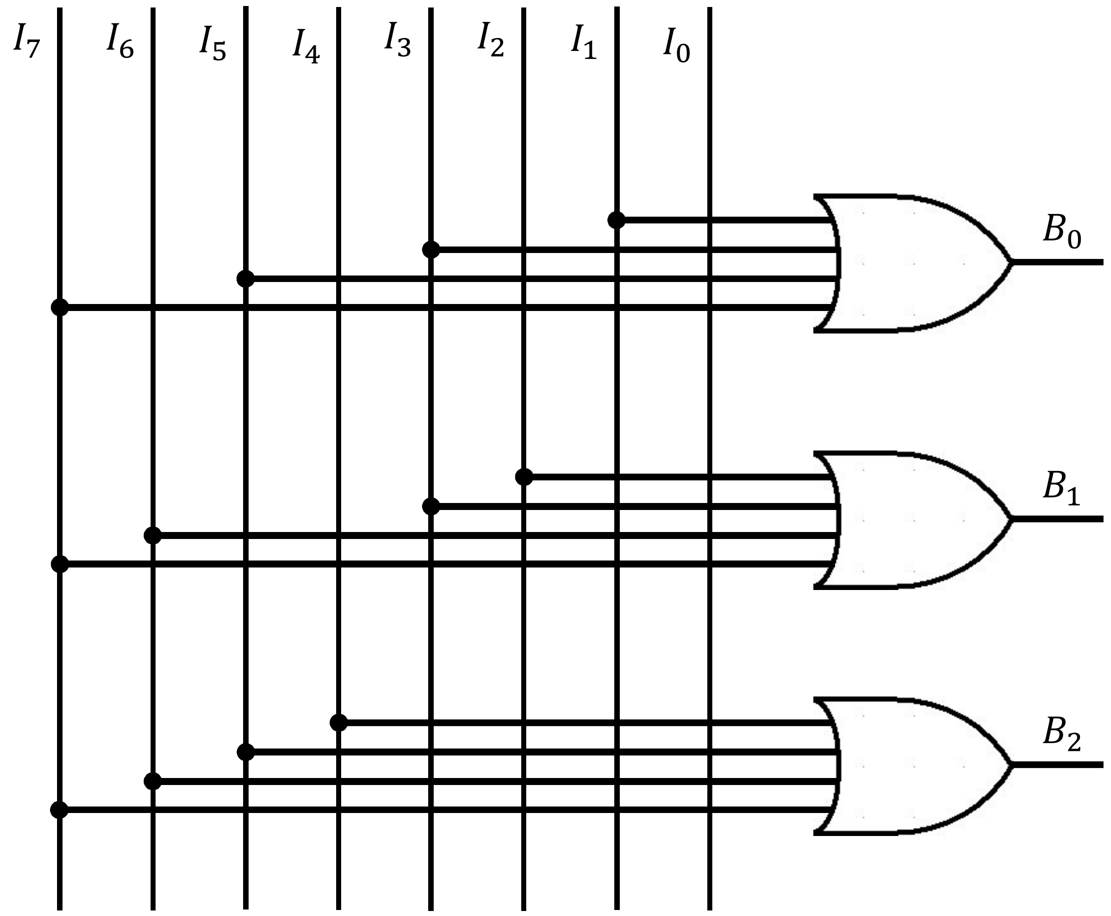
Decimal Value	Input								Output		
	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	B_2	B_1	B_0
0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	$\bigcirc\!\!\!\!\!1$
2	0	0	0	0	0	1	0	0	0	$\bigcirc\!\!\!\!\!1$	0
3	0	0	0	0	1	0	0	0	0	$\bigcirc\!\!\!\!\!1$	$\bigcirc\!\!\!\!\!1$
4	0	0	0	1	0	0	0	0	$\bigcirc\!\!\!\!\!1$	0	0
5	0	0	1	0	0	0	0	0	$\bigcirc\!\!\!\!\!1$	0	$\bigcirc\!\!\!\!\!1$
6	0	1	0	0	0	0	0	0	$\bigcirc\!\!\!\!\!1$	$\bigcirc\!\!\!\!\!1$	0
7	1	0	0	0	0	0	0	0	$\bigcirc\!\!\!\!\!1$	$\bigcirc\!\!\!\!\!1$	$\bigcirc\!\!\!\!\!1$

- Input – 8
- Output – 3
- Outputs are,

$$B_0 = I_1 + I_3 + I_5 + I_7$$

$$B_1 = I_2 + I_3 + I_6 + I_7$$

$$B_2 = I_4 + I_5 + I_6 + I_7$$



Decimal to BCD Encoder

Truth table of decimal-to-BCD encoder

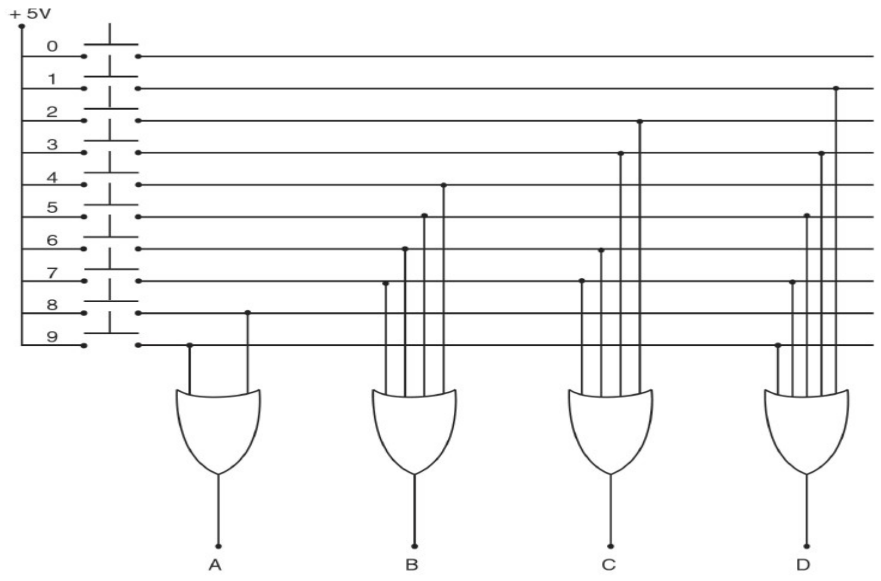
Decimal inputs										BCD outputs			
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

$$A = 8 + 9$$

$$B = 4 + 5 + 6 + 7$$

$$C = 2 + 3 + 6 + 7$$

$$D = 1 + 3 + 5 + 7 + 9$$



Decimal-to-BCD encoder

Priority Encoder

A priority encoder is an encoder that includes the priority function

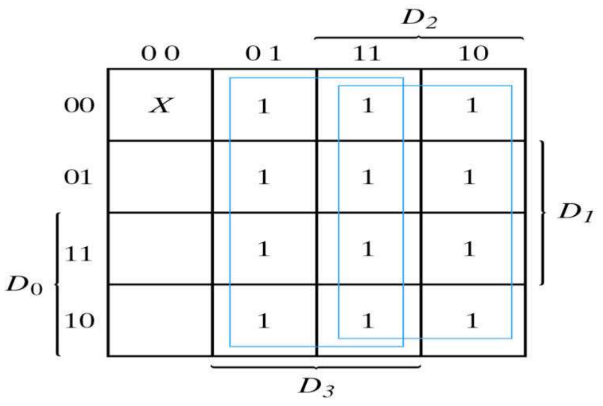
- If two or more inputs are equal to 1 at the same time, the input having the

highest priority will take precedence.

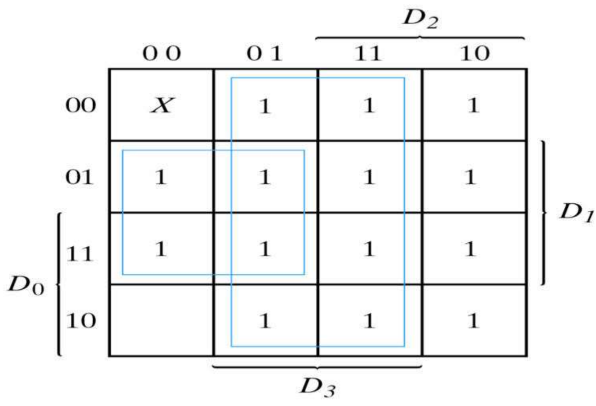
- In addition to two outputs x, and y, the truth table has a third output designated by V, which is a valid bit indicator that is set 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.
- X's in the output column indicate don't care conditions, the X's in the input columns are useful for representing a truth table in condensed form.
- The higher the subscript number, the higher the priority of the input. Input D3 has the highest priority, so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3)

Inputs				**Outputs**		
D_0	D_1	D_2	D_3	x	y	V
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

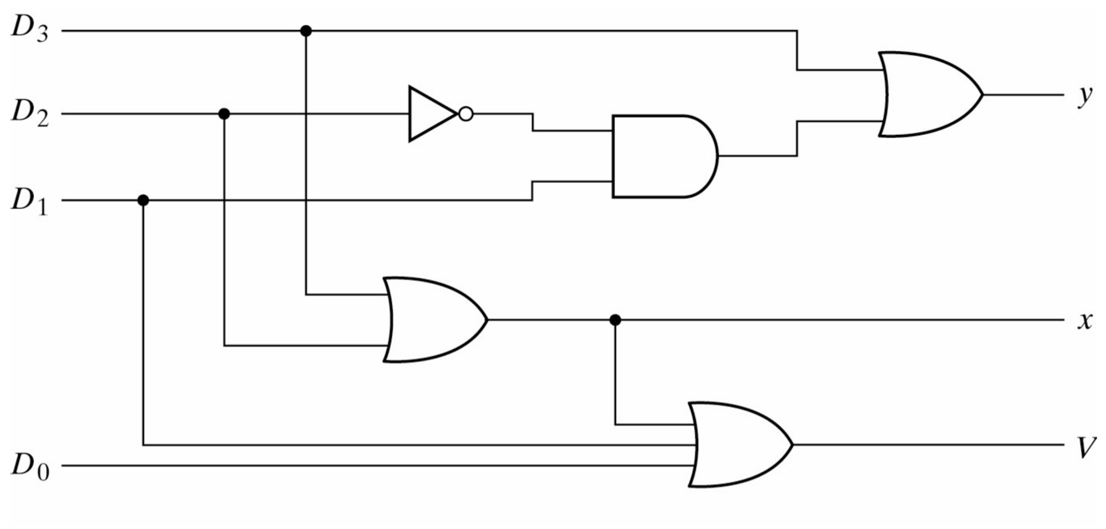
Inputs				**Outputs**		
D_0	D_1	D_2	D_3	x	y	V
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1



$x = D_2 + D_3$



$y = D_3 + D_1D'_2$



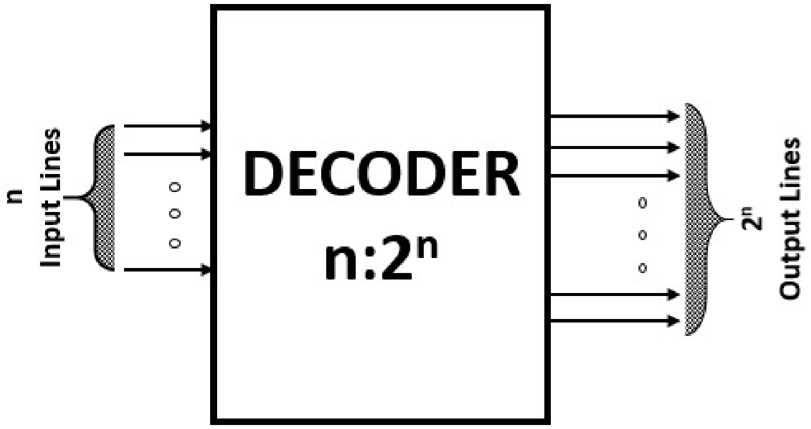
Decoder

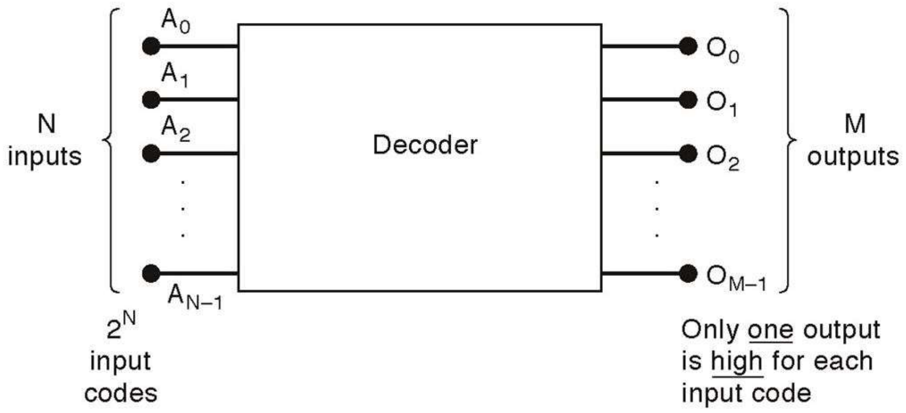
- A decoder, is a combinational circuit that decodes the information on n input lines to a maximum of 2^n unique output lines.
- It has an n-bit binary input code and one activated output out of 2^n output code.
- A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n-bit input value.
- It is similar to demultiplexer, with only one exception that it has no data input.

Input – n

Output – 2^n

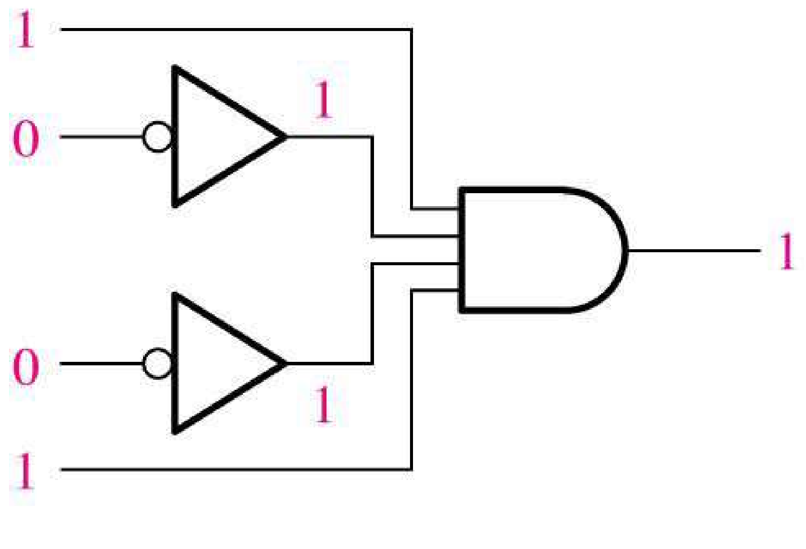
Decoder – n : 2^n



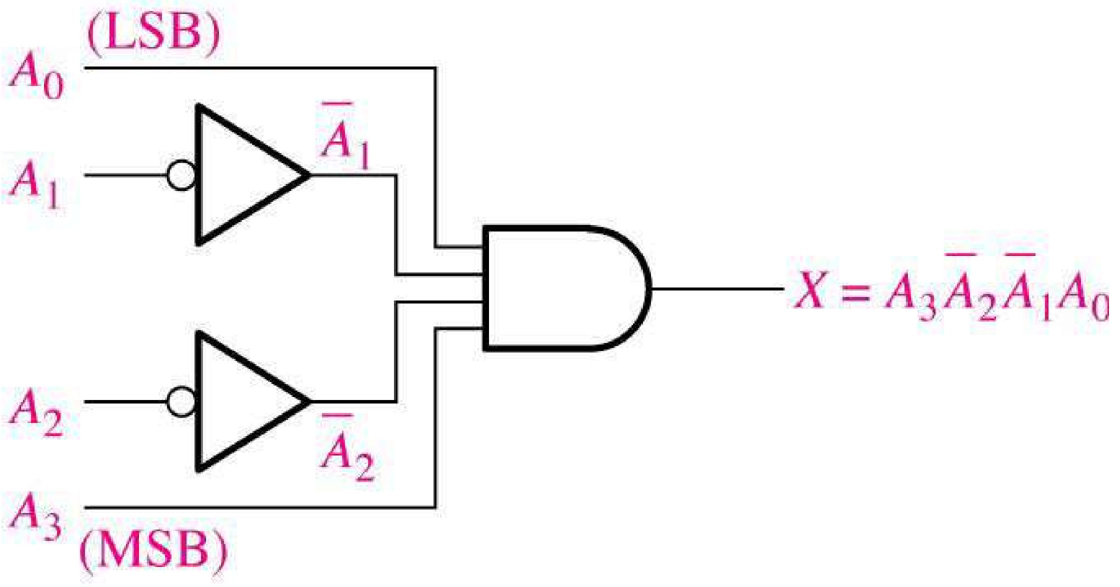


Decoder: Basic Binary function

- An AND gate can be used as the basic decoding element because it produces a HIGH output only when all inputs are HIGH

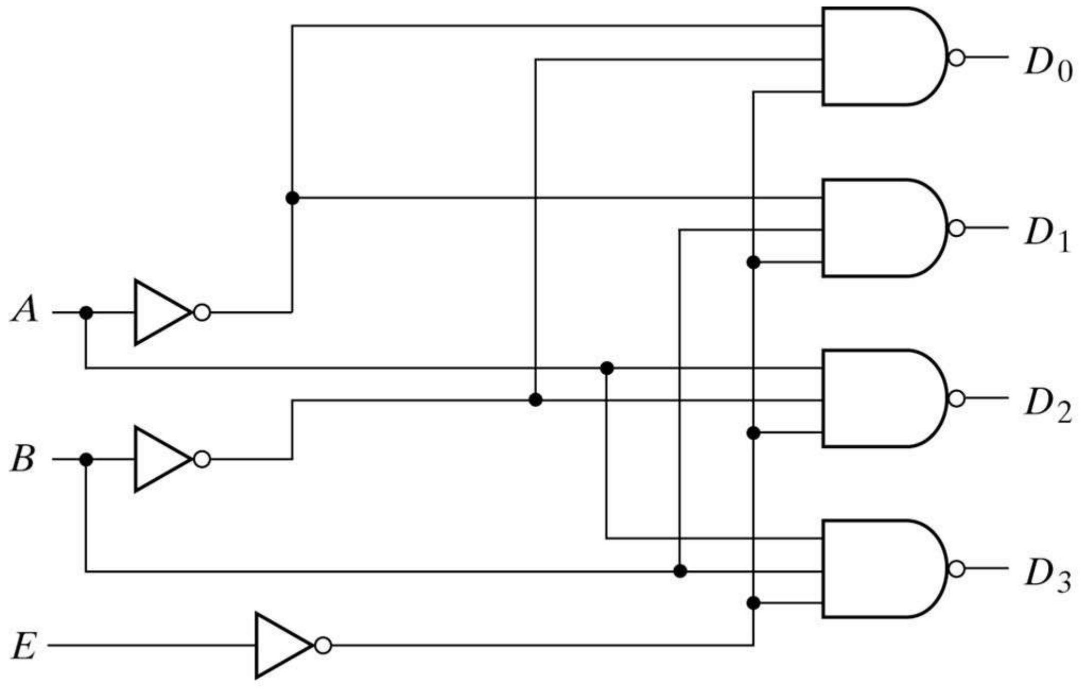


(a)



(b)

2 to 4 Decoder with active low output

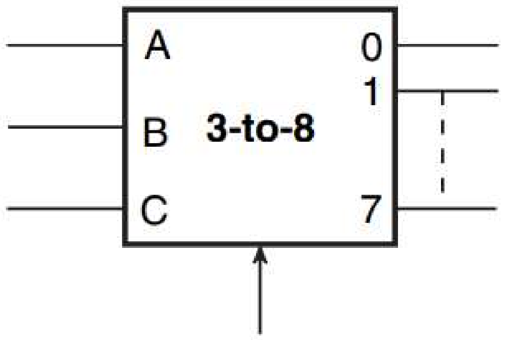


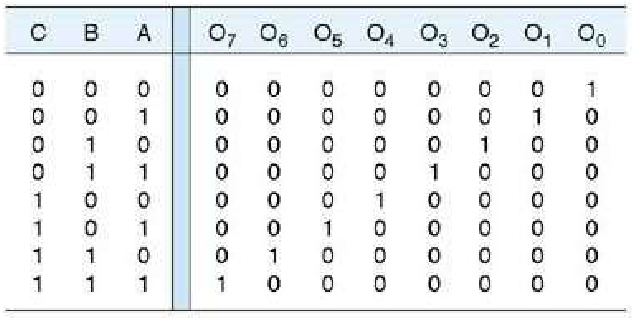
(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

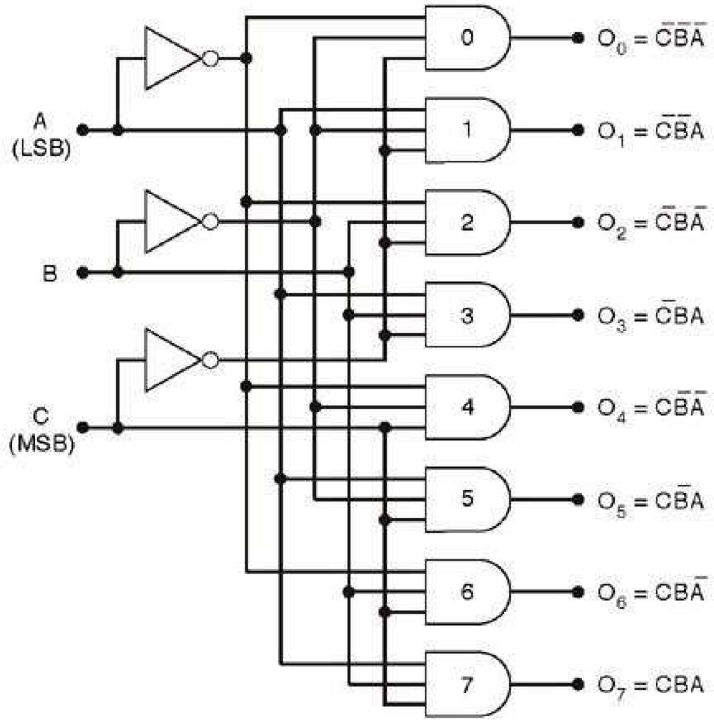
(b) Truth table

3-to-8 Decoder with active high output



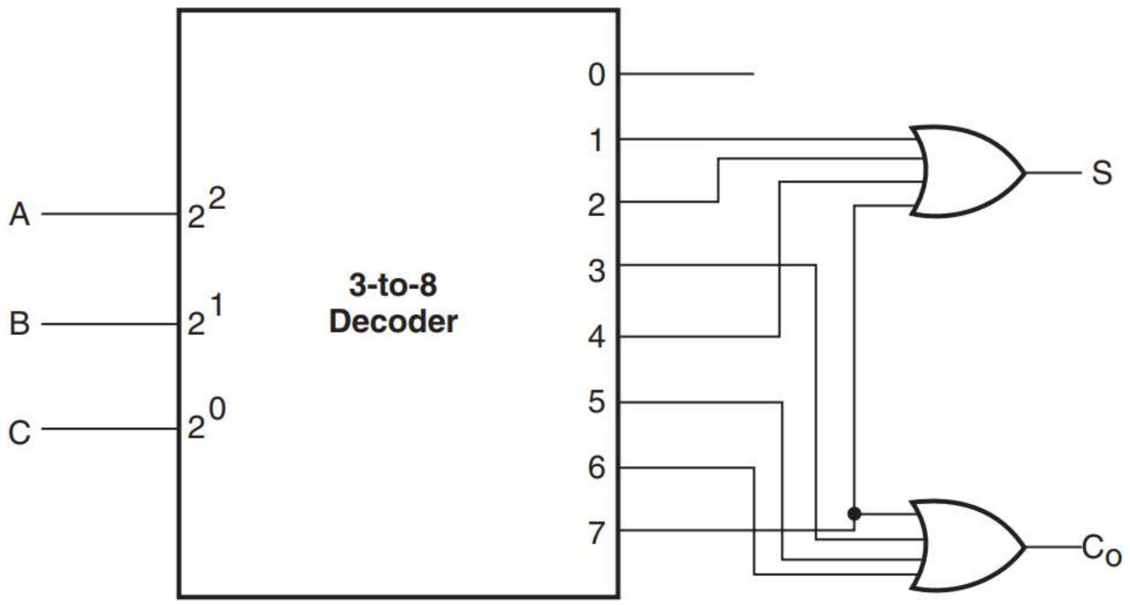


C	B	A	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



Implementing full adder using a decoder

A	B	C	S	C_o
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



Decoder Vs Demultiplexer

- **Decoder**
  - Decoder is a many inputs to many outputs device.
  - There are no selection lines.

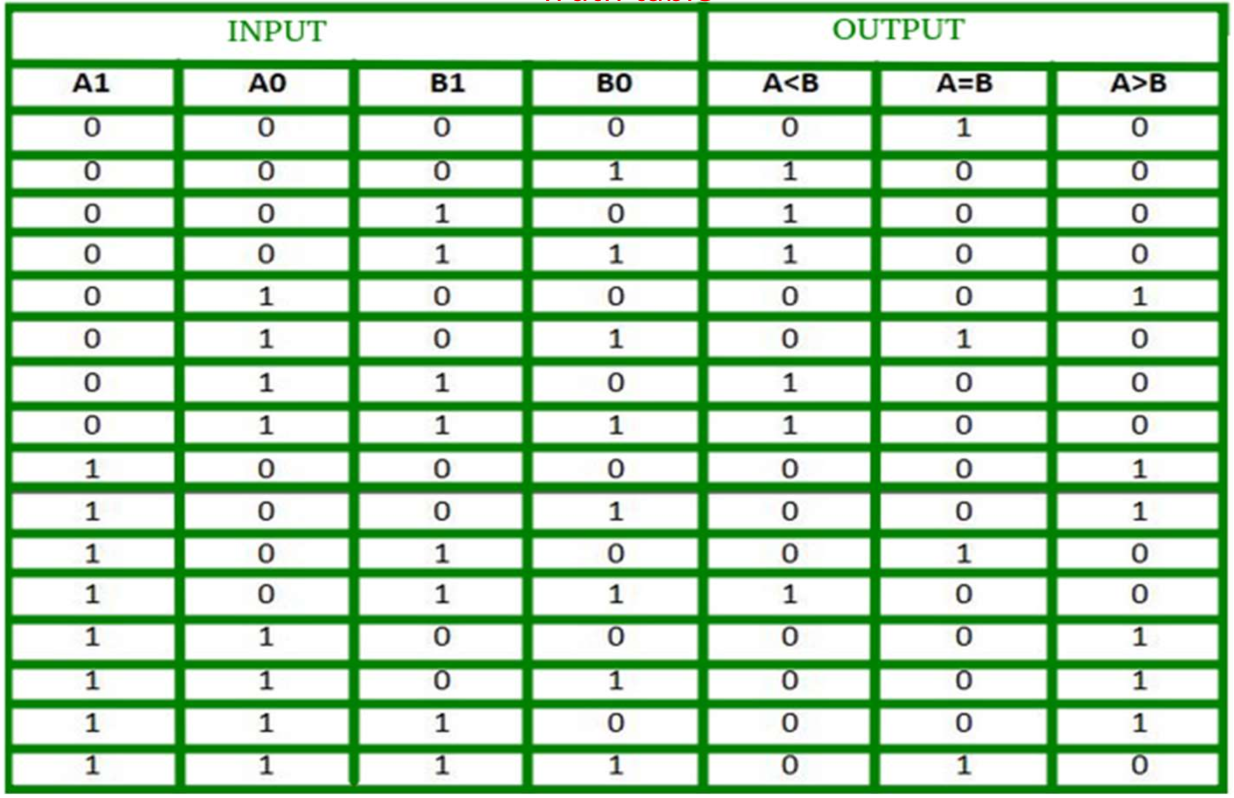
- **Demultiplexer**
  - Demultiplexer is a one input to many outputs device.
  - The selection of specific output line is controlled by the value of selection lines.

Magnitude comparator

2- Bit magnitude comparator

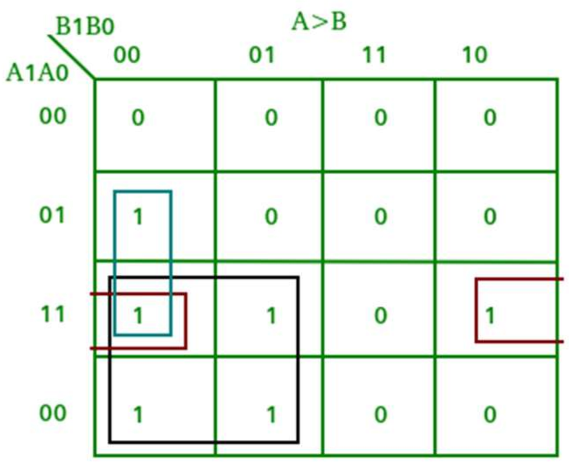
- A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to and greater than between two binary numbers.

Truth table

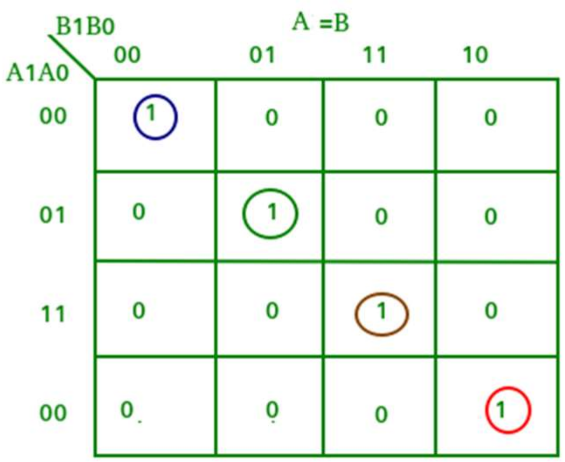


INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

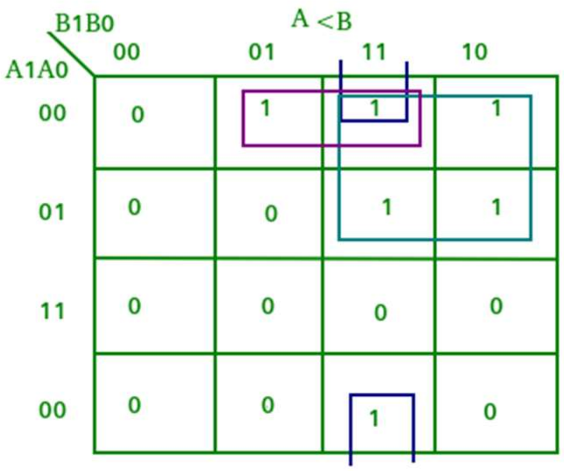
K map for 2- Bit magnitude comparator



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



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



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

Design equations

A>B:A1B1’ + A0B1’B0’ + A1A0B0’

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

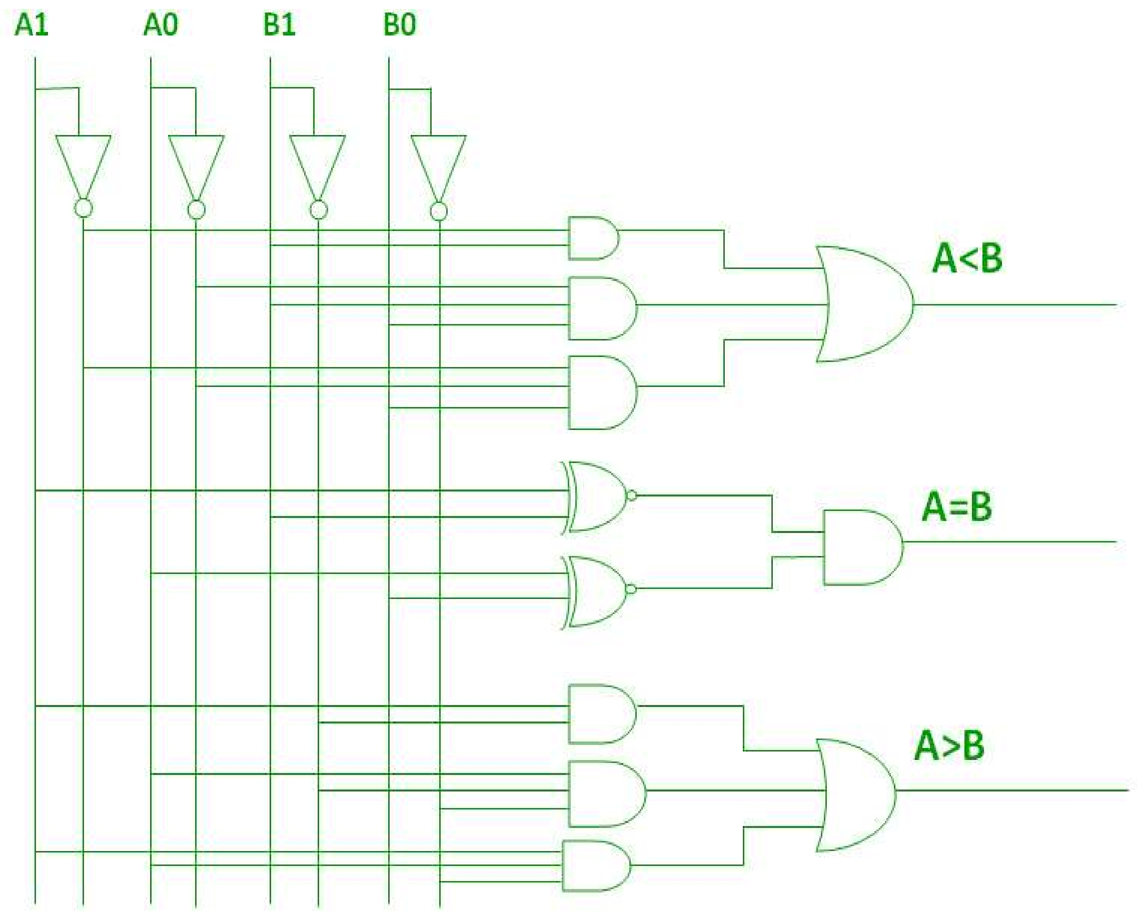
: A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’)

: (A0B0 + A0’B0’) (A1B1 + A1’B1’)

: (A0 Ex-Nor B0) (A1 Ex-Nor B1)

A<B:A1’B1 + A0’B1B0 + A1’A0’B0

Logic design of 2 –bit magnitude comparator



Code Converters

A code converter is a logic circuit that changes data presented in one form of binary code to another type of binary code, such as

- BCD to binary
- BCD to 7-segment
- binary to BCD
- BCD to XS3
- binary to Gray code
- Gray code to binary.

Code Converters

A code converter is a logic circuit that changes data presented in one form of binary code to another type of binary code, such as

- BCD to binary
- BCD to 7-segment
- binary to BCD
- BCD to XS3
- binary to Gray code
- Gray code to binary

What is meant by code?

The digital data is represented, stored and transmitted as group of binary bits. This group is also called as binary code. The binary code is represented by the number as well as alphanumeric letter.

Codes are classified in to 2 types

- Weighted
- Non-weighted

WEIGHTED CODE

- In weighted code, each digit position has a weight or value.
- The sum of all digits multiplied by a weight gives a total amount being represented.
- BCD or 8421 is a type of weighted code where each digit position is assigned a specific weight.

NON WEIGHTED CODE

- In non weighted code there is no positional weight
- i.e., Each position within the binary number is not assigned a prefix value.
- No specific positions are assigned to bit positions in non weighted code.
- The non weighted codes are (1) Gray code
  (2) Excess-3 code.

Code converters

Code conversion is used to change the data present in one type of binary code to another type of binary code. Some of the codes are BCD, Gray, Excess 3, ASCII and so on.

Some of the conversions are

- Binary to Gray code conversion
- Gray to Binary code conversion
- BCD to Excess-3 code conversion
- Excess-3 to BCD code conversion

Binary to Gray code conversion

The Gray code is non-weighted code, as the position of bit does not contain any weight. The Gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit- distance code.

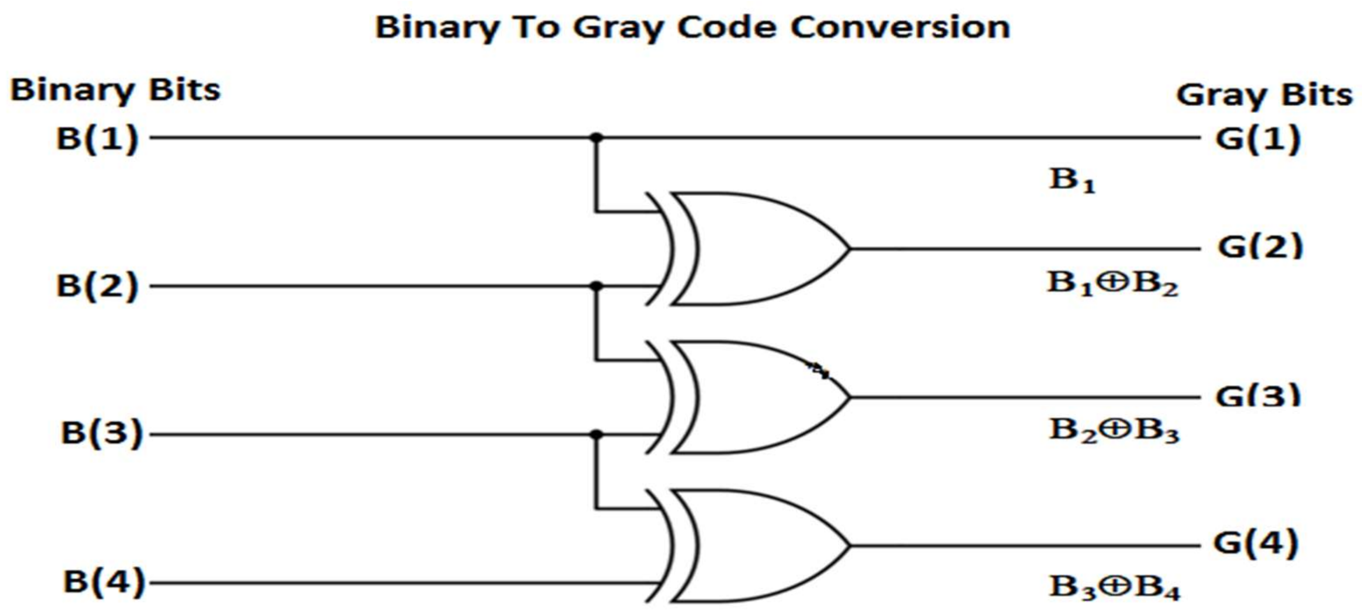
A four-bit binary to Gray code conversion table is as shown below.

Four Bit Binary Number				Four Bit Gray Code			
B_1	B_2	B_3	B_4	G_1	G_2	G_3	G_4
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

The generation of 4-bit Gray code can be calculated by using formula.

$$G_1 = B_1$$
$$G_2 = B_1 \oplus B_2$$
$$G_3 = B_2 \oplus B_3$$
$$G_4 = B_3 \oplus B_4$$

The circuit for Binary to Gray code conversion is



Gray to binary code conversion

In Gray to binary conversion, the input is Gray code and output is its equivalent binary code.

The generation of four-bit binary equivalent code can be calculated by using formula.

$$B_1 = G_1$$

$$B_2 = G_2 \oplus B_1$$

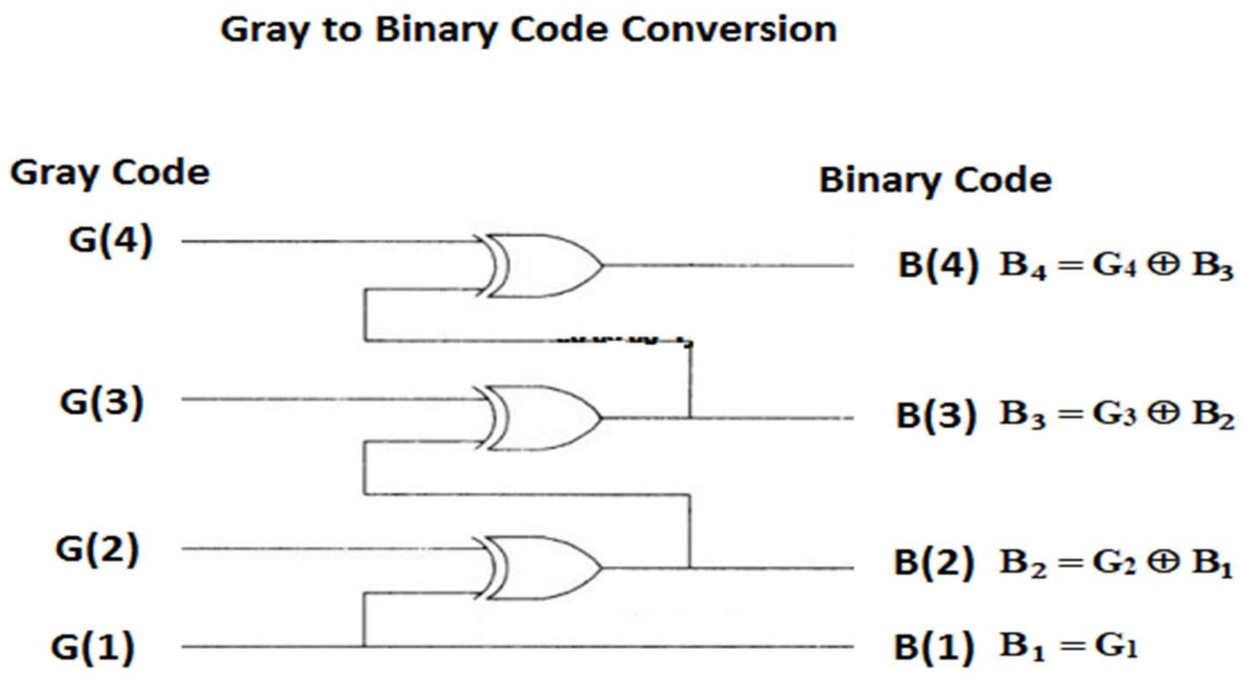
$$B_3 = G_3 \oplus B_2$$

$$B_4 = G_4 \oplus B_3$$

A four-bit Gray to binary code conversion table is as shown below.

Four Bit Gray Code				Four Bit Binary Number			
G_1	G_2	G_3	G_4	B_1	B_2	B_3	B_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

- The Gray to binary conversion method can be done by using xoring logic gate.
- A four-bit Gray to binary code converter is as shown below.


Gray to Binary Code Conversion
Gray Code
Binary Code
G(4)
G(3)
G(2)
G(1)
B(4) $B_4 = G_4 \oplus B_3$
B(3) $B_3 = G_3 \oplus B_2$
B(2) $B_2 = G_2 \oplus B_1$
B(1) $B_1 = G_1$

BCD to Excess 3 code

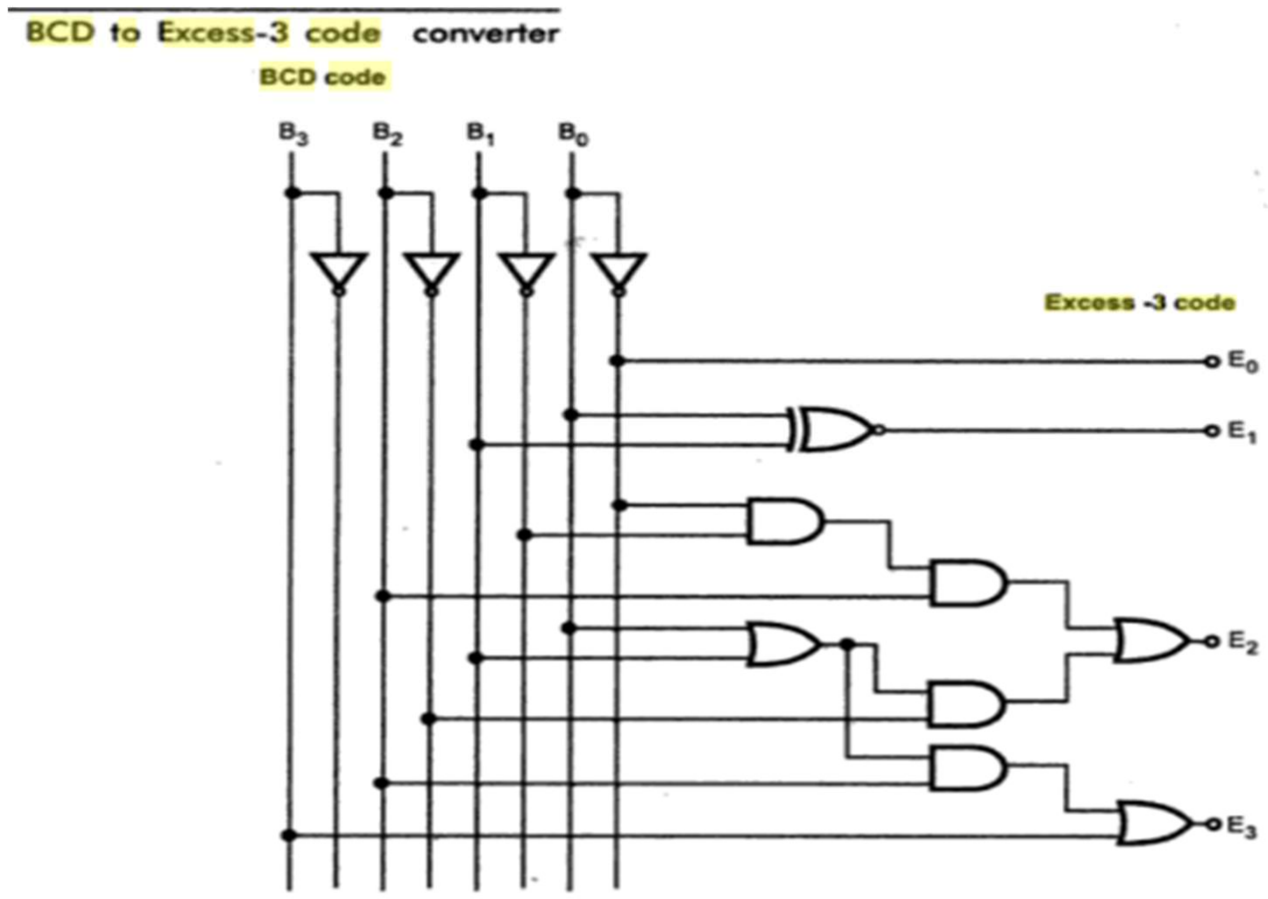
- Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit then it can be represented by using 4-bit binary number for each digit.
- To find the decimal equivalent of the given binary number. Add 0011 to each four-bit group in binary coded decimal number (BCD) to get desired excess-3 equivalent.
- The variables B_0, B_1, B_2, and B_3 represent the bits of the binary numbers.
- The variable 'B_0' represents the LSB, and the variable 'B_3' represents the MSB.
- The variables E_0, E_1, E_2, and E_3 represent the bits of the Excess-3 code.
- The variable 'E_0' represents the LSB, and the variable 'E_3' represents the MSB.

The truth table for BCD to Excess-3 code converter can be determined as shown in table below.

For impossible four bit Excess-3 code we use output as Don’t care conditions. The 'don't care conditions' is defined by the variable 'X'.

	BCD Number				Excess-3 Code Number			
Decimal	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

The circuit diagram for BCD to Excess-3 code converter is shown below



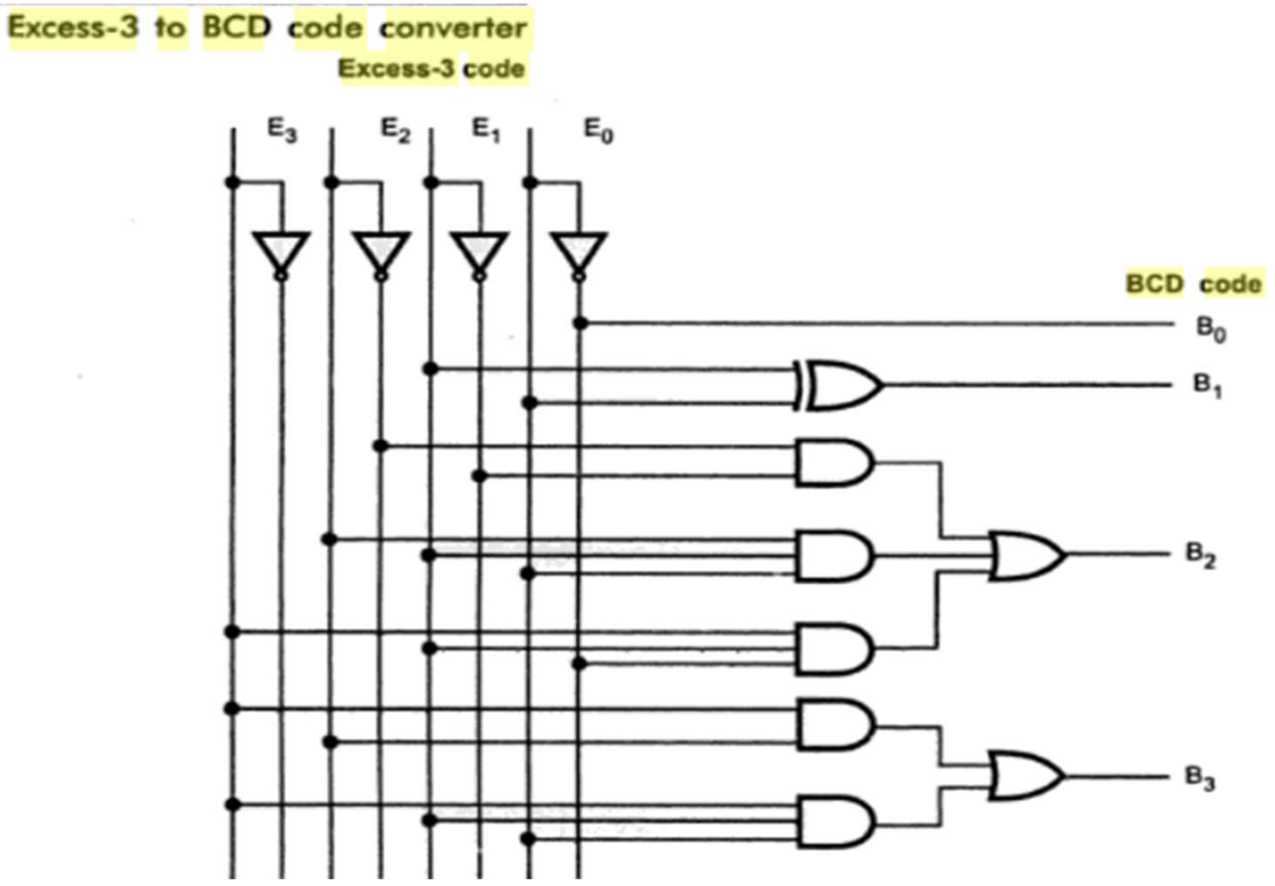
Excess-3 to BCD converter

- The process of converting Excess-3 to BCD is opposite to the process of converting BCD to Excess-3.
- The BCD code can be calculated by subtracting 3, i.e., 0011 from each four-digit Excess-3 code.
- The variables E_0, E_1, E_2, and E_3 represent the bits of the Excess-3 code.
- The variable 'E_0' represents the LSB, and the variable 'E_3' represents the MSB.
- The variables B_0, B_1, B_2, and B_3 represent the bits of the binary numbers.
- The variable 'B_0' represents the LSB, and the variable 'B_3' represents the MSB.
- The 'don't care conditions' is defined by the variable 'X'.

The truth table for BCD to Excess-3 code converter can be determined as shown in table below.

	Excess-3 Code Number				BCD Number			
Decimal	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
0	0	0	1	1	0	0	0	0
1	0	1	0	0	0	0	0	1
2	0	1	0	1	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	1	1	1	0	1	0	0
5	1	0	0	0	0	1	0	1
6	1	0	0	1	0	1	1	0
7	1	0	1	0	0	1	1	1
8	1	0	1	1	1	0	0	0
9	1	1	0	0	1	0	0	1

The circuit diagram for Excess-3 code to BCD converter is shown below


Excess-3 to BCD code converter
Excess-3 code
E_3
E_2
E_1
E_0
BCD code
B_0
B_1
B_2
B_3