Combinational Circuits

Combinational circuit

- The outputs at any instant of time depend upon the inputs present at that instant of time
- The input values explicitly determine the output
- There is no memory in these circuits

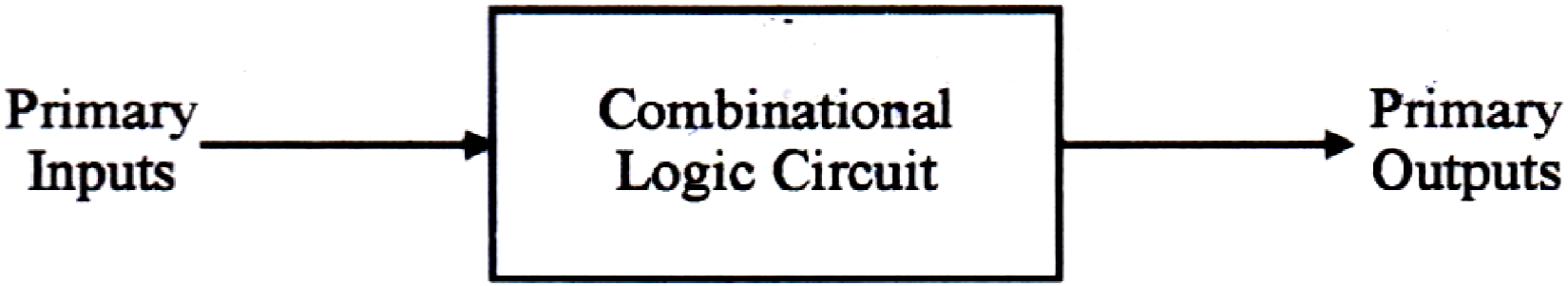


Fig. Block diagram of combinational circuit

Steps to design a combinational digital circuit:

- From the problem statement derive the truth table
- From the truth table derive the un-simplified logic expression
- Simplify the logic expression
- From the simplified expression draw the digital logic circuit

Sum-of-Products Form

- A **Sum-of-products (SOP)** expression will appear as two or more **AND** terms **OR**ed together.

1. $ABC + \overline{A}B\overline{C}$
2. $AB + \overline{A}B\overline{C} + \overline{C}\,\overline{D} + D$
3. $\overline{A}B + C\overline{D} + EF + GK + H\overline{L}$

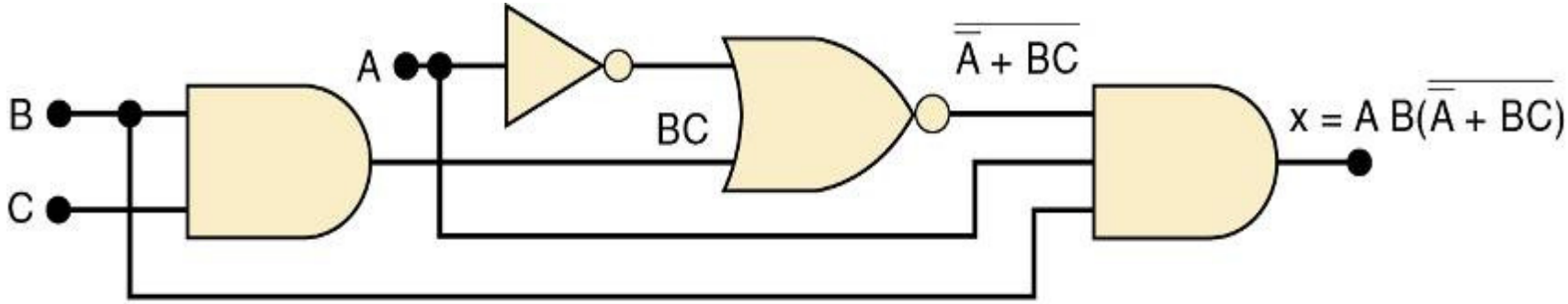
Products -of- Sum Form

- The **product-of-sums (POS)** form consists of two or more **OR** terms (sums) **AND**ed together.

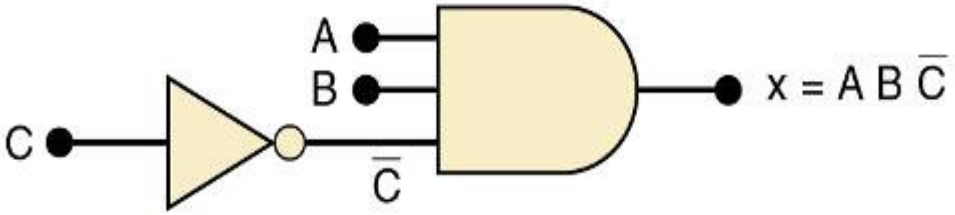
1. $(A + \overline{B} + C)(A + C)$
2. $(A + \overline{B})(\overline{C} + D)F$
3. $(A + C)(B + \overline{D})(\overline{B} + C)(A + \overline{D} + \overline{E})$

Simplifying Logic Circuits

- The circuits shown provide the same output
  - Circuit (b) is clearly less complex.



(a)



(b)

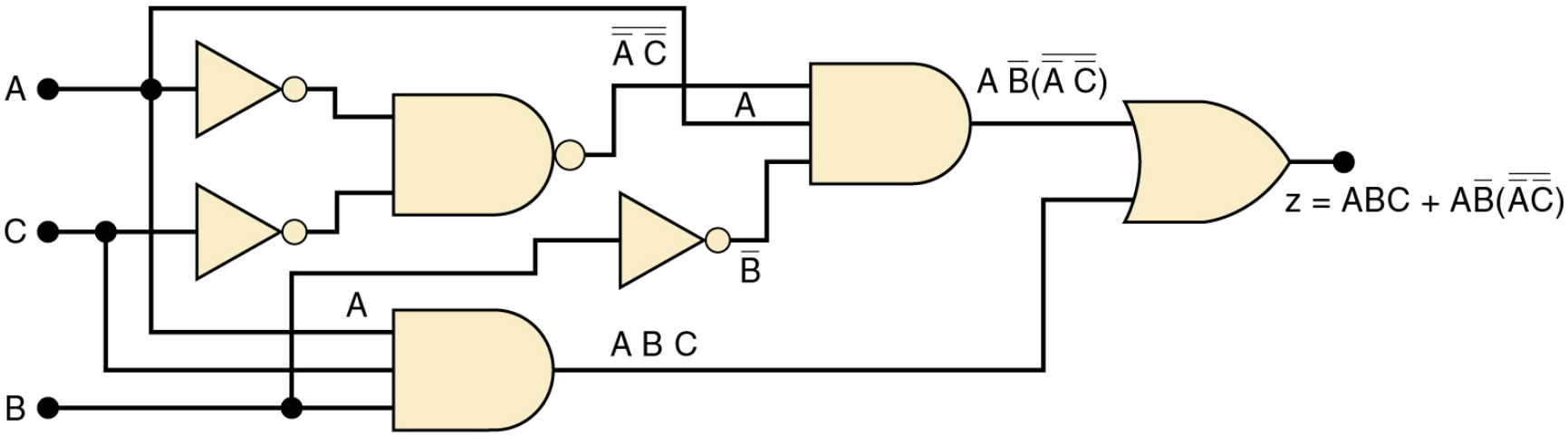
Logic circuits can be simplified using Boolean algebra and Karnaugh mapping.

Steps for Simplification of Combinational Circuits

- Place the expression in SOP form by applying DeMorgan's theorems and multiplying terms.

- **Check the SOP form for common factors.**
  - Factoring where possible should eliminate one or more terms.

Simplify the logic circuit shown.

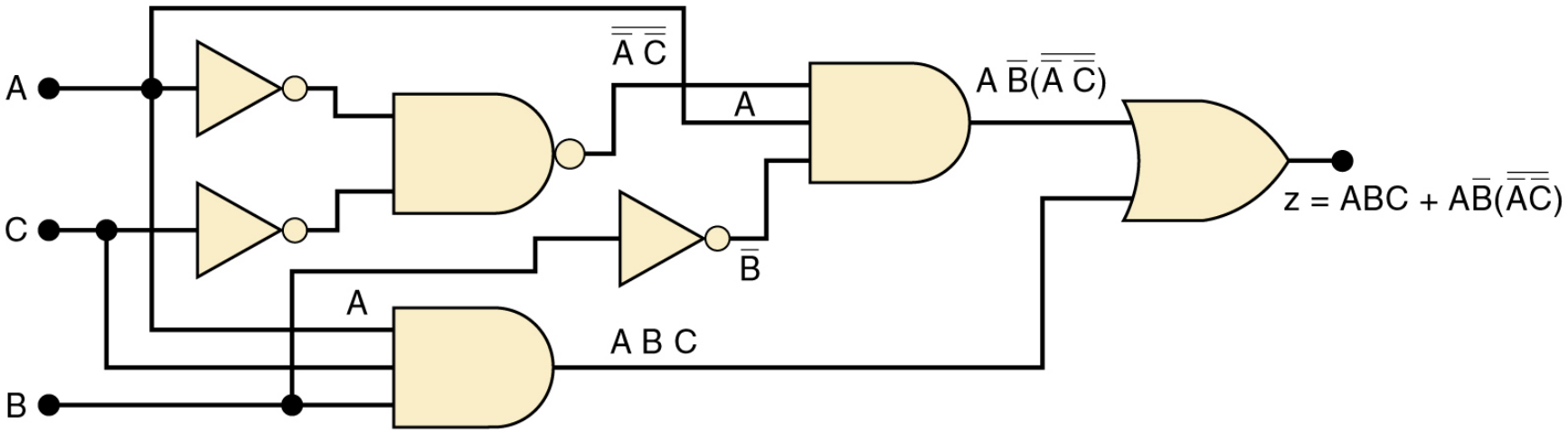


The first step is to determine the expression for the output: $z = ABC + A\overline{B} \cdot (\overline{\overline{A}\,\overline{C}})$

Once the expression is determined, break down large inverter signs by DeMorgan's theorems & multiply out all terms.

$$
\begin{aligned}
z &= ABC + A\overline{B}(\overline{\overline{A}} + \overline{\overline{C}}) && \text{[theorem (17)]} \\
&= ABC + A\overline{B}(A + C) && \text{[cancel double inversions]} \\
&= ABC + A\overline{B}A + A\overline{B}C && \text{[multiply out]} \\
&= ABC + A\overline{B} + A\overline{B}C && [A \cdot A = A]
\end{aligned}
$$

Cont…



Factoring—the first & third terms above have ***AC*** in common, which can be factored out:

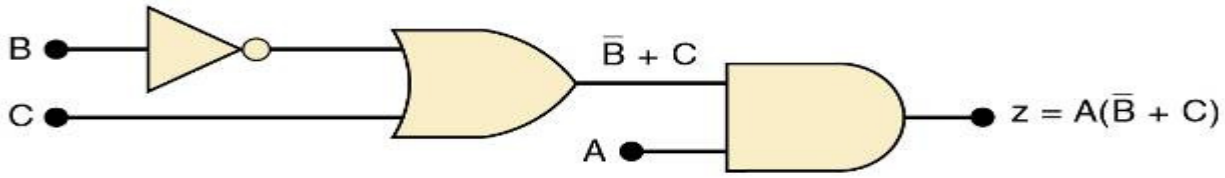
$$z = AC(B + \overline{B}) + A\overline{B}$$

Since ***B*** + ***B*** = 1, then…

$$\begin{aligned} z &= AC(1) + A\overline{B} \\ &= AC + A\overline{B} \end{aligned}$$

Factor out ***A***, which results in…

Simplifed logic circuit.



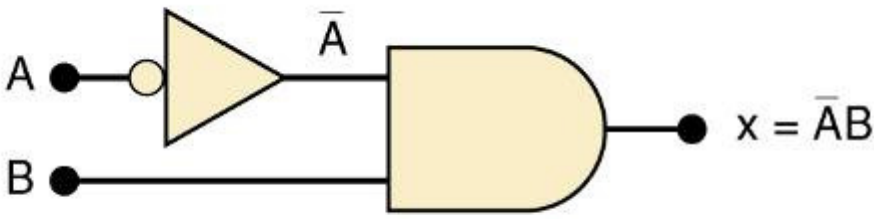
$$z = A(C + \overline{B})$$

Designing Combinational Logic Circuits

- ***To solve any logic design problem:***
  - Interpret the problem and set up its truth table.
  - Write the **AND** (product) term for each case where output = 1.
  - Combine the terms in SOP form.
  - Simplify the output expression if possible.
  - Implement the circuit for the final, simplified expression.

Circuit that produces a 1 output only for the A = 0, B = 1 condition.

A	B	x
0	0	0
0	1	1
1	0	0
1	1	0

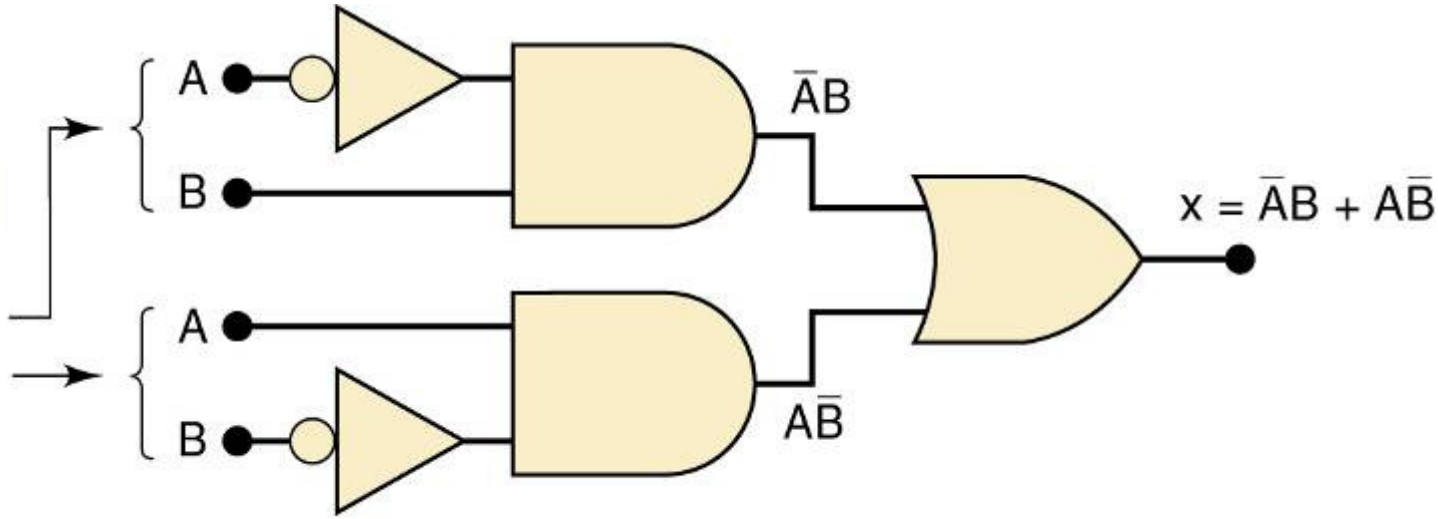


Cont….

•Each set of input conditions that is to produce a 1 output is implemented by a separate **AND** gate.

•The **AND** outputs are **OR**ed to produce the final output.

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0



Karnaugh Map Method

- A graphical method of simplifying logic equations or truth tables—also called a K map.
- Theoretically can be used for any number of input variables—practically limited to 5 or 6 variables.

The truth table values are placed in the K map.
Shown here is a two-variable map.

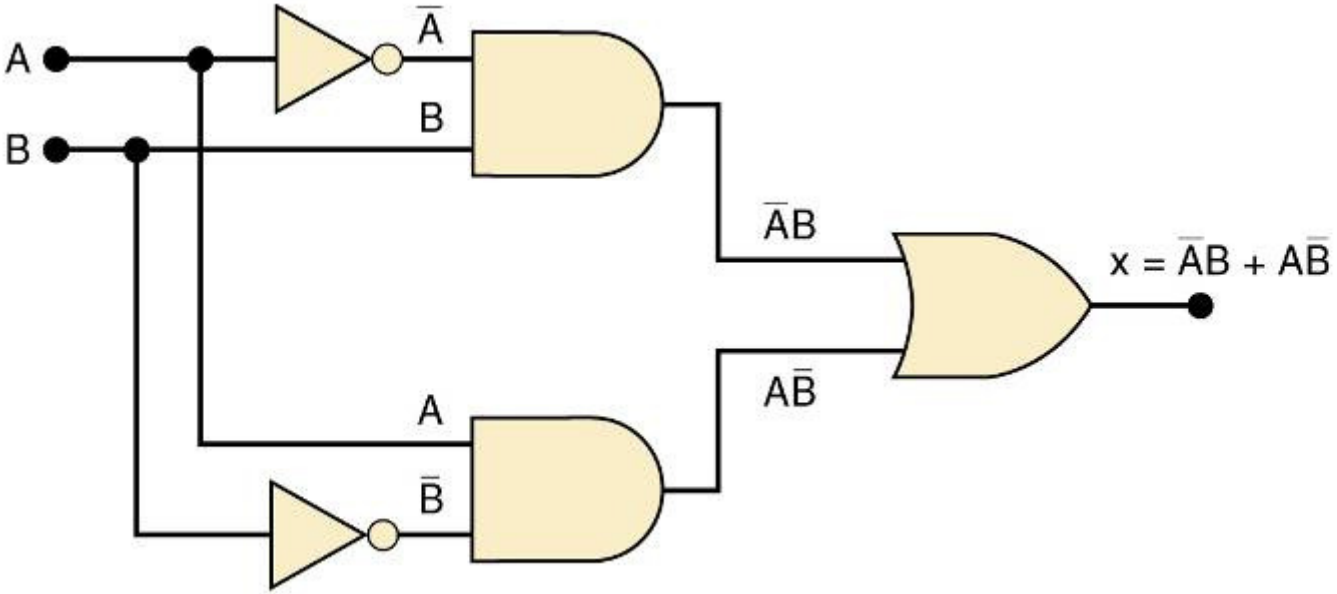
A	B	X
0	0	1 → $\bar{A}\bar{B}$
0	1	0
1	0	0
1	1	1 → AB

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

	$\bar{B}$	B
$\bar{A}$	1	0
A	0	1

Exclusive OR and Exclusive NOR Circuits

Exclusive **OR** circuit and truth table.

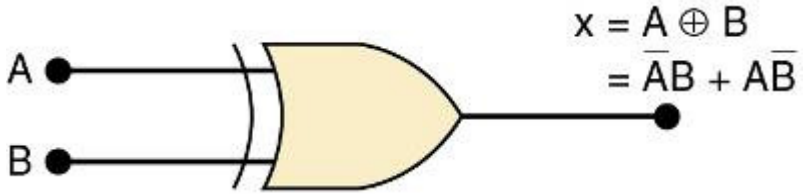


A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

Output expression: $x = \overline{A}B + A\overline{B}$

This circuit produces a HIGH output whenever the two inputs are at opposite levels.

*Traditional **XOR** gate symbol.*



An **XOR** gate has only *two* inputs, combined so that $x = \overline{A}B + A\overline{B}$.

A shorthand way indicate the **XOR** output expression is: $x = A \oplus B$.

…where the symbol ⊕ represents the **XOR** gate operation.

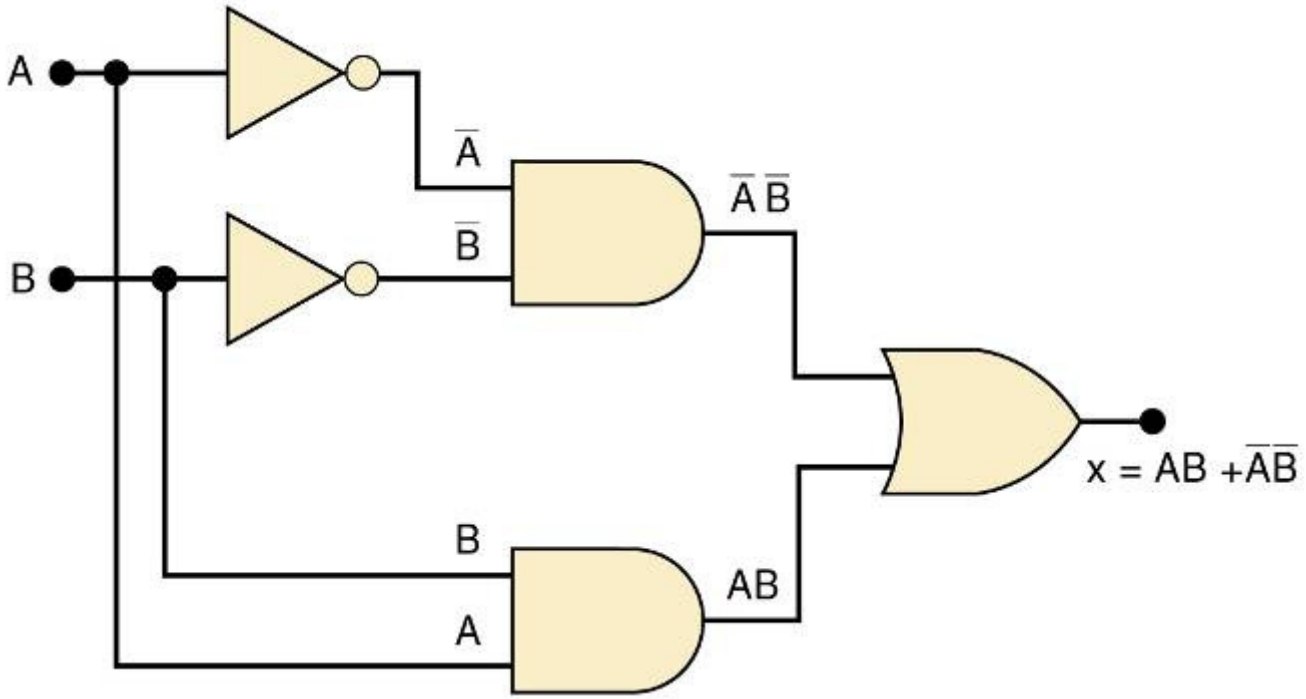
Output is HIGH only when the two inputs are at different levels.

Quad XOR chips containing four XOR gates.

74LS86 Quad **XOR** (TTL family)
74C86 Quad **XOR** (CMOS family)
74HC86 Quad **XOR** (high-speed CMOS)

- The exclusive **NOR** (**XOR**) produces a HIGH output whenever the two inputs are at the *same* level.
  - **XOR** and **XNOR** outputs are opposite.

*Exclusive **NOR** circuit and truth table.*

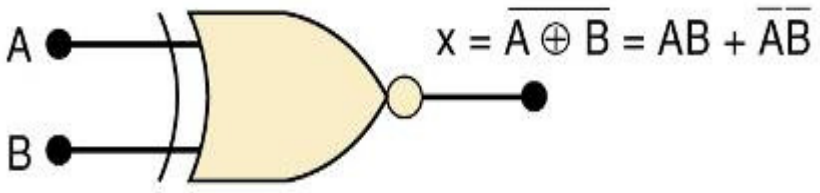


A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

Output expression: $x = AB + \overline{A}\,\overline{B}$

XNOR produces a HIGH output whenever the two inputs are at the same levels.

Traditional **XNOR** gate symbol.



An **XNOR** gate has only *two* inputs, combined so that $x = AB + \overline{A}\,\overline{B}$.

A shorthand way indicate the **XOR** output expression is: $x = \overline{A \oplus B}$.

XNOR represents inverse of the **XOR** operation.

Output is HIGH only when the two inputs are at the same level.

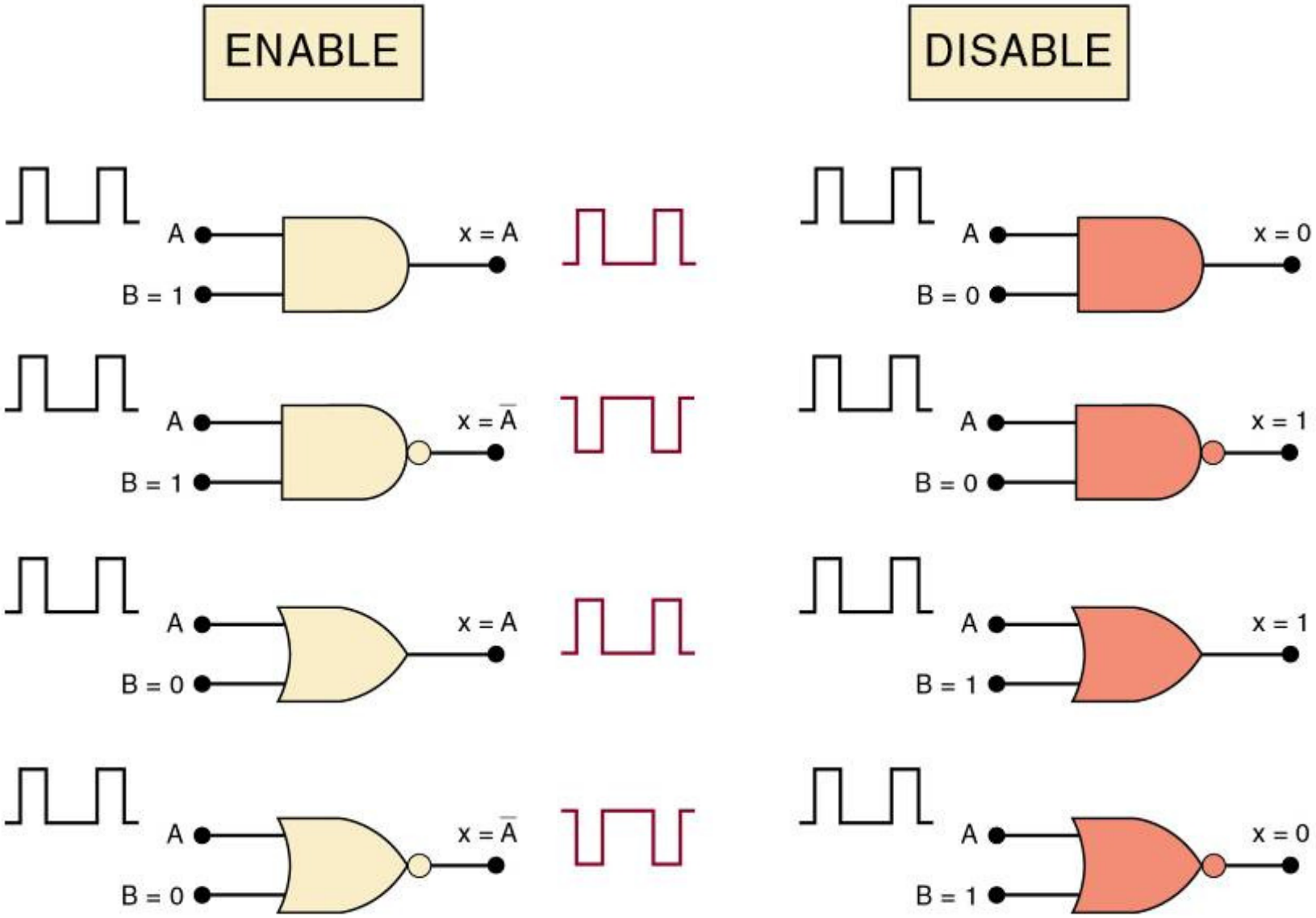
Quad XNOR chips with four XNOR gates.

74LS266 Quad **XNOR** (TTL family)
74C266 Quad **XOR** (CMOS)
74HC266 Quad **XOR** (high-speed CMOS)

Enable/Disable Circuits

- Situations requiring enable/disable circuits occur frequently in digital circuit design.
  - A circuit is *enabled* when it *allows* the passage of an input signal to the output.
  - A circuit is *disabled* when it *prevents* the passage of an input signal to the output.

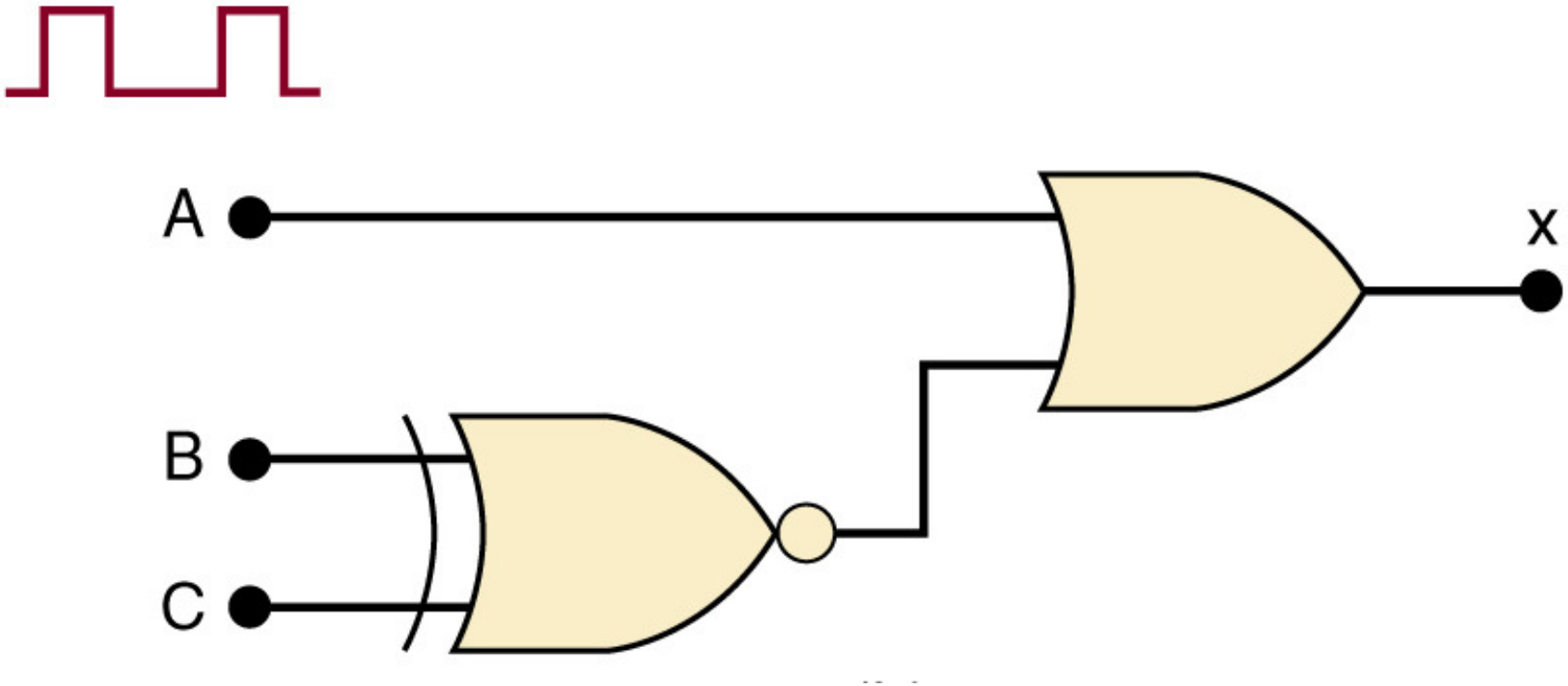
Enable/Disable Circuits



Enable/Disable Circuits

•A logic circuit that will allow a signal to pass to output only when one, but *not* both control inputs are HIGH.

•Otherwise, output will stay HIGH.

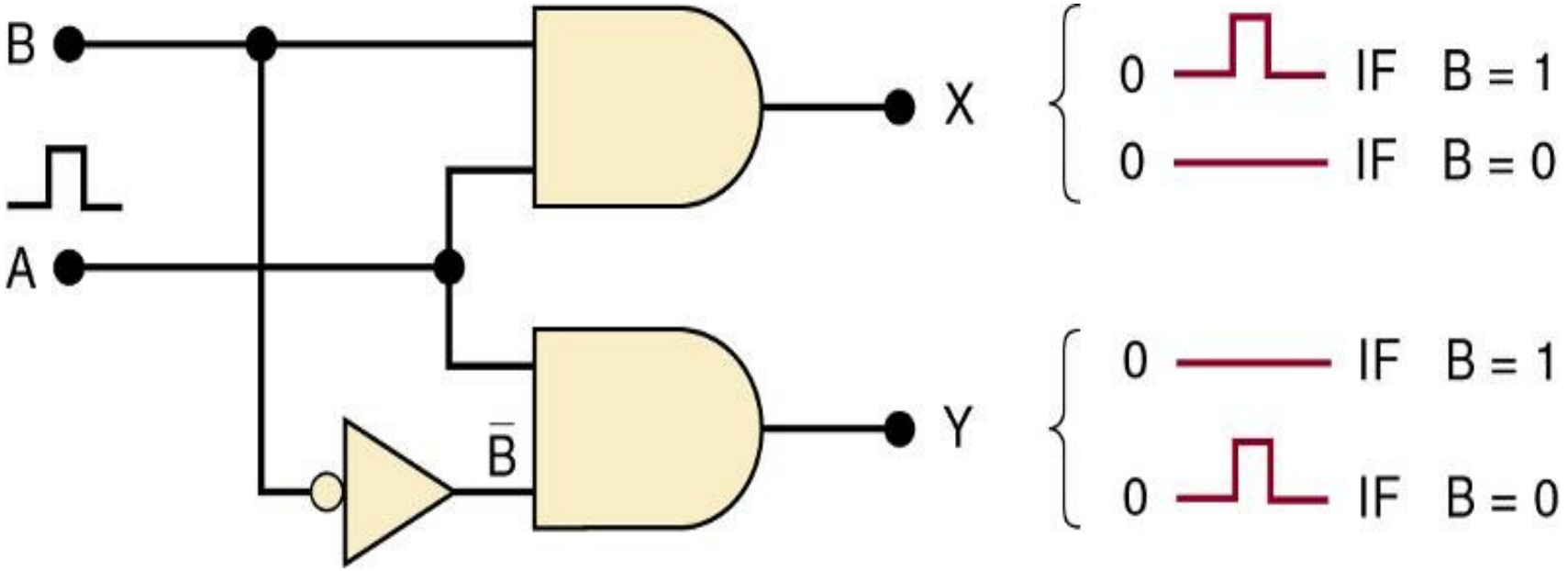


Enable/Disable Circuits

A logic circuit with input signal *A*, control input *B*, and outputs *X* and *Y*, which operates as:

When *B* = 1, output *X* will follow input *A*, and output *Y* will be 0.

When *B* = 0, output *X* will be 0, and output *Y* will follow input *A*.



Half Adder

Half Adder : Half adder is a logic circuit that adds two bits producing a sum and a carry to be used for the next higher position. The logical truth table for a half adder is shown below :

Truth Table
(Half Adder)

Input		Output	
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The equation for sum and carry can be written from the truth table as

$$\text{Sum} = \overline{A}.\ B + A.\ \overline{B}$$

$$\text{Carry} = A.\ B$$

Realizing the equation for sum and carry using AND, OR and NOT gates.

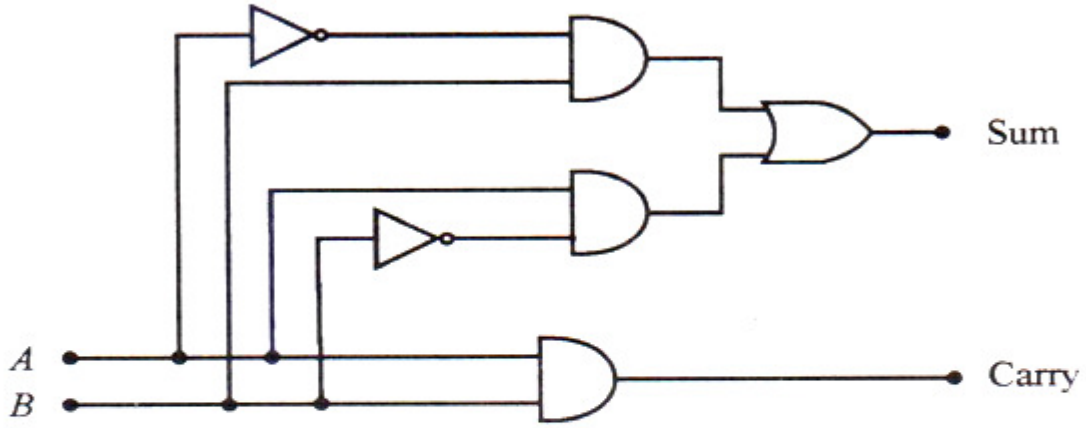


Fig. Logic Circuit using AND,OR and NOT gate

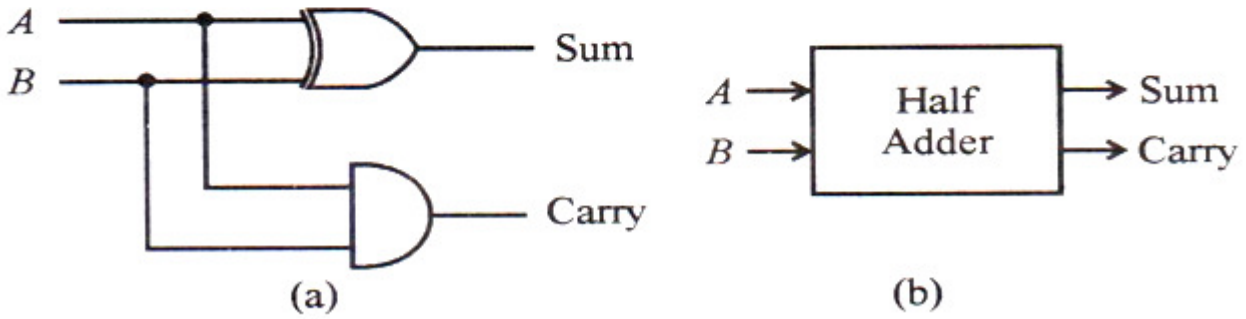


Fig. (a)Logic Circuit using Ex OR and AND gate (b) Logic Symbol

Full Adder

Full Adder : Full adder is a combinational circuit which adds three bits, the third bit is the carry bit (C_i) which is generated from the lower bit addition. The symbol of full adder is shown in Fig. where *A* and B are the data input, C_i is the carry generated by the previous stage. At the output we get the sum and carry output C_o which forms the carry input for the next stage.

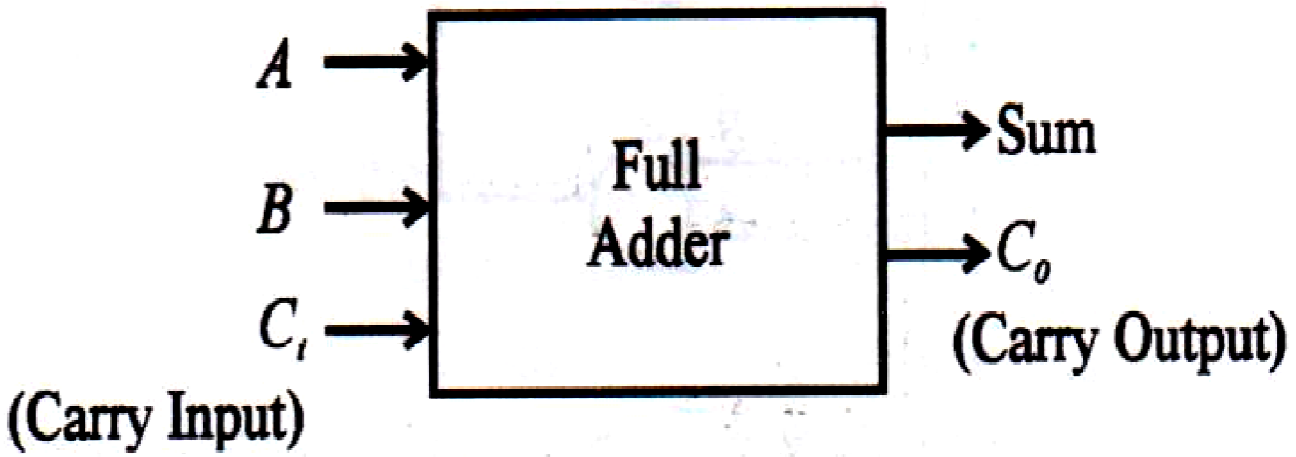


Fig. Symbol of Full Adder Circuit

Truth Table

(Full adder)

Inputs			Outputs	
A	B	C_i	Sum	Carry
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

The equation for sum and carry can be writen from the truth table as :

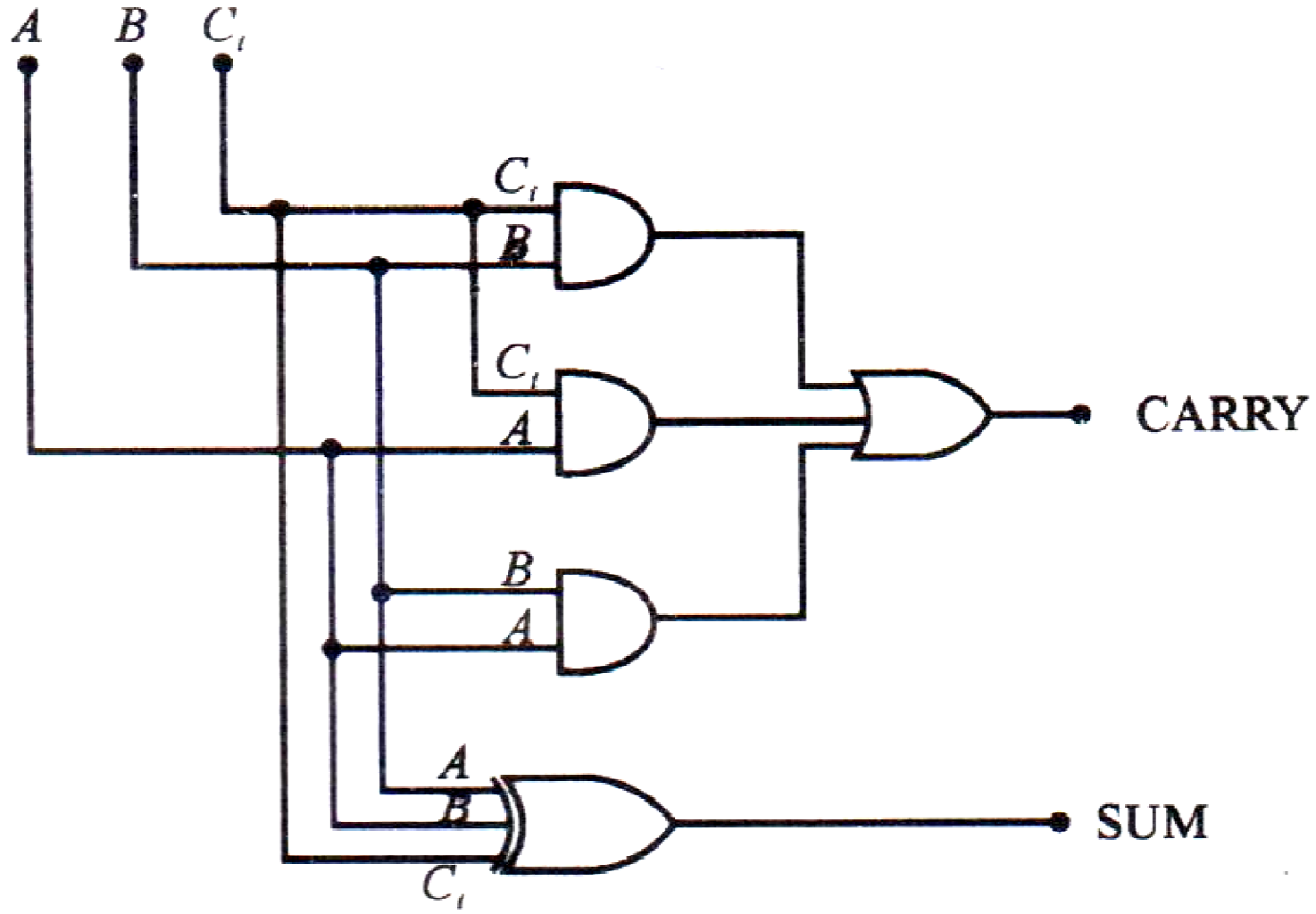
$$Sum = \bar{A}\,\bar{B}\,C_i + \bar{A}\,B\,\bar{C}_i + A\,\bar{B}\,\bar{C}_i + A\,B\,C_i$$

$$Carry = \bar{A}\,B\,C_i + A\,\bar{B}\,C_i + A\,B\,\bar{C}_i + A\,B\,C_i$$

The minimized equation for carry is :

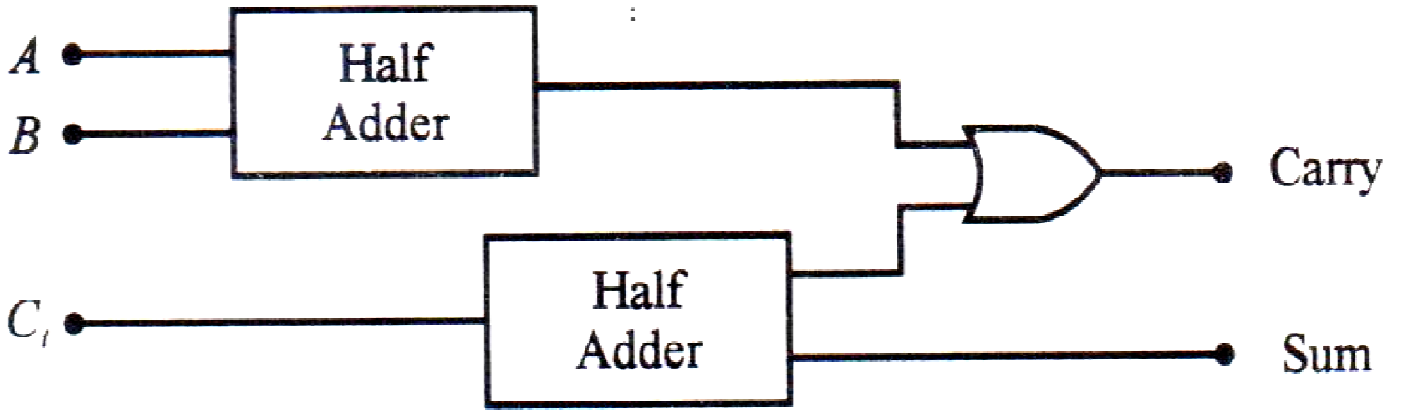
$$Carry = AB + BC_i + AC_i$$

Realizing the equation for sum and carry using AND, OR and Ex-OR gate.

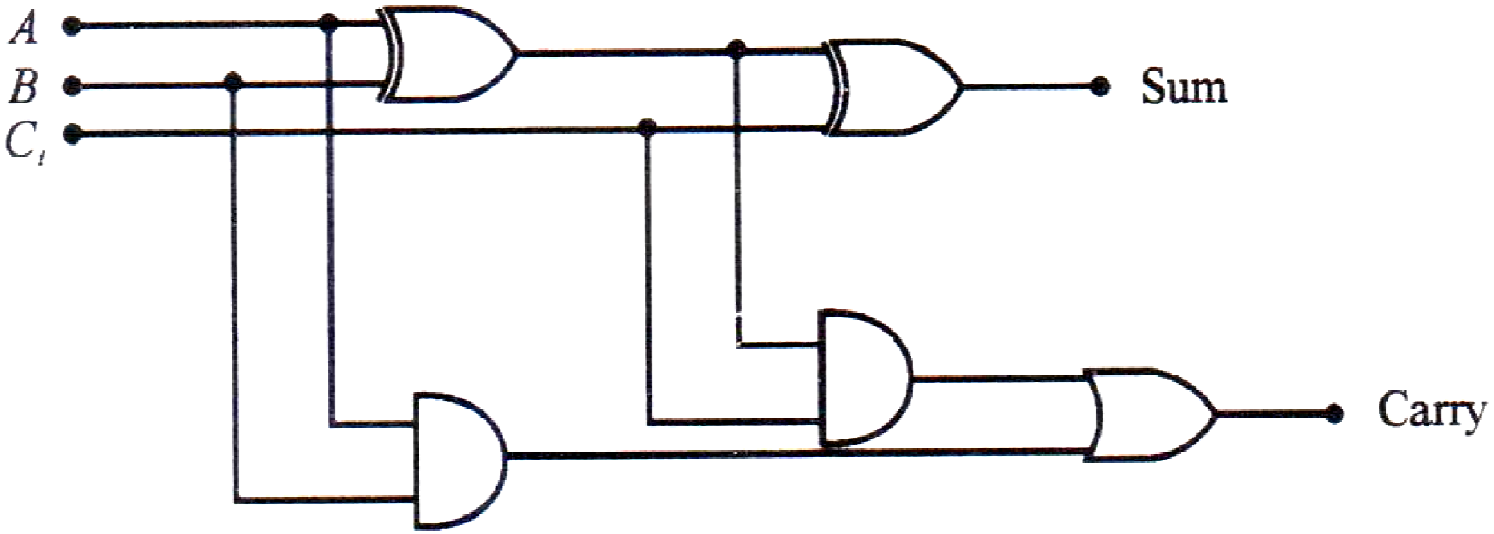


Full Adder Combinational Circuit

Full Adder Circuits realization through two half adder and an OR gate



(a) Block representation



(b) Combinational Circuit

Half Subtractor

Truth Table
(Half Subtractor)

Input		Output	
A	**B**	**Difference**	**Borrow**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The equation for difference and borrow can be written from the truth table as

$$\text{Difference} = \bar{A}B + A\bar{B}$$

$$\text{Borrow} = \bar{A}B$$

The equation can be realized using AND, OR and NOT gate as

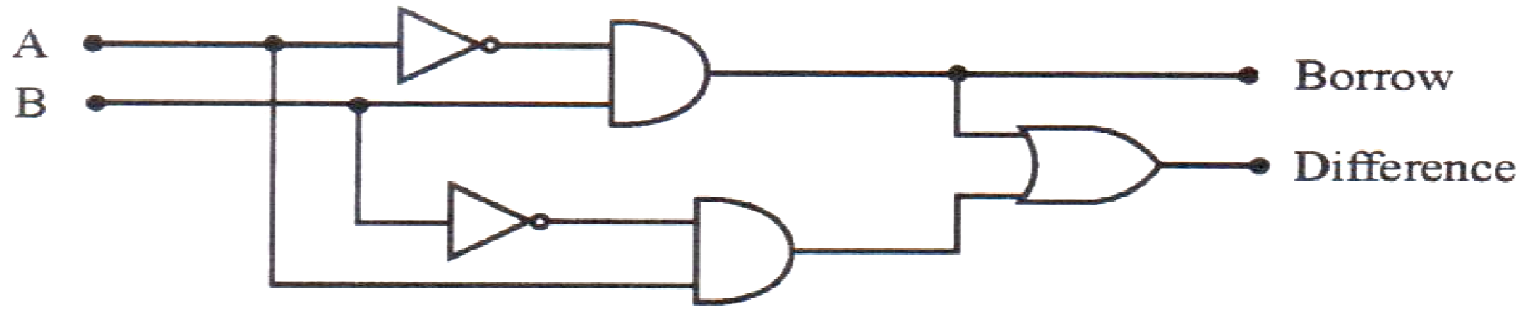


Fig. Half Subtractor using AND,OR and NOT gate

The realization of the equations using Ex - OR and other basic gates is shown

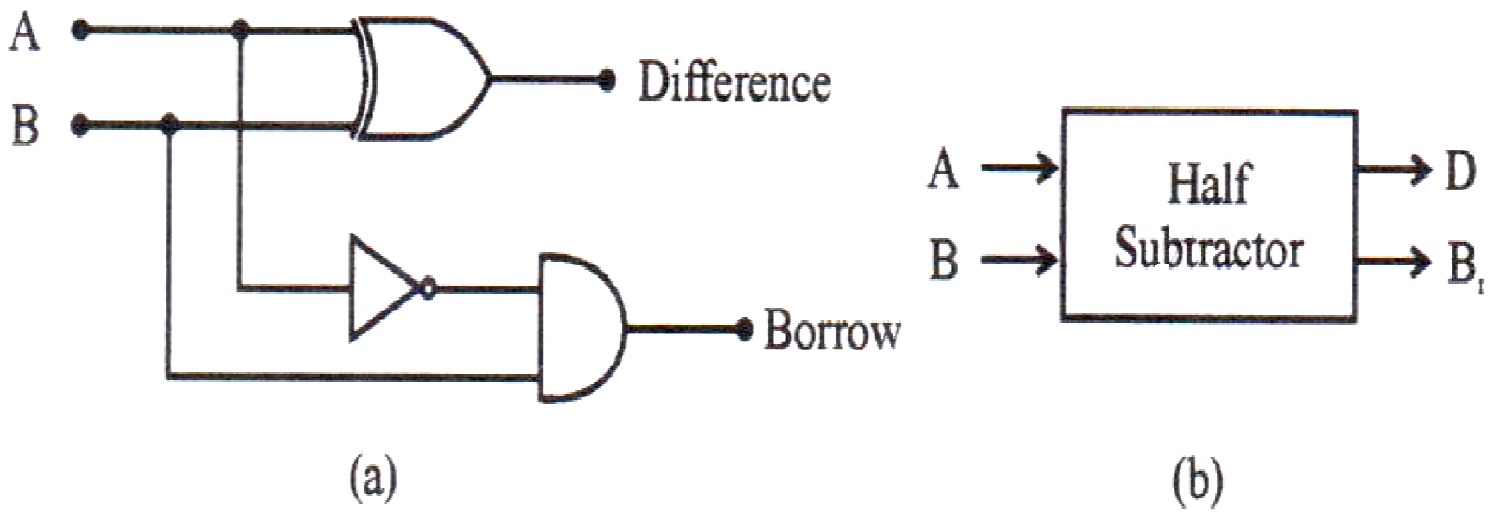


Fig. (a) Half Subtractor using Ex-OR , AND and NOT gate (b)Symbol

Full Subtractor

In the truth table of full subtractor A and B are the bits of minuend and subtraher respectively, B_{in} denotes the borrow input bit. The two outputs are D the difference ar B_r the borrow bits respectively.

Truth Table
(Full Subtractor)

A	B	B_{in}	D	B_r
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

Consider for instance second row(A=0, B-0 and B_{in}=1) , 0 (bit B) has to be subtracted from 0 (bit A) . Now subtract B_{in} from the result derived from the subtractrin of B from A, this gives D as 1 and B, as 1.

Actually this is calculated as :

$(A - B) - B_{in}$ for any row

From the truth table the boolean expression for the difference and borrow output can be written as :

$$D = \bar{A}\ \bar{B}\ B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$$

$$B_r = \bar{A}\ \bar{B}\ B_{in} + \bar{A}\ B\ \bar{B}_{in} + \bar{A}\ B\ B_{in} + A\ B\ B_{in}$$

The boolean expression for D is the minimal and for B, can be reduced to

$$B_r = \bar{A}\ B + \bar{A}\ B_{in} + B\ B_{in}$$

From these boolean expressions for D and B_{in}, the logic circuit for full subtractor can be constructed as shown in Fig

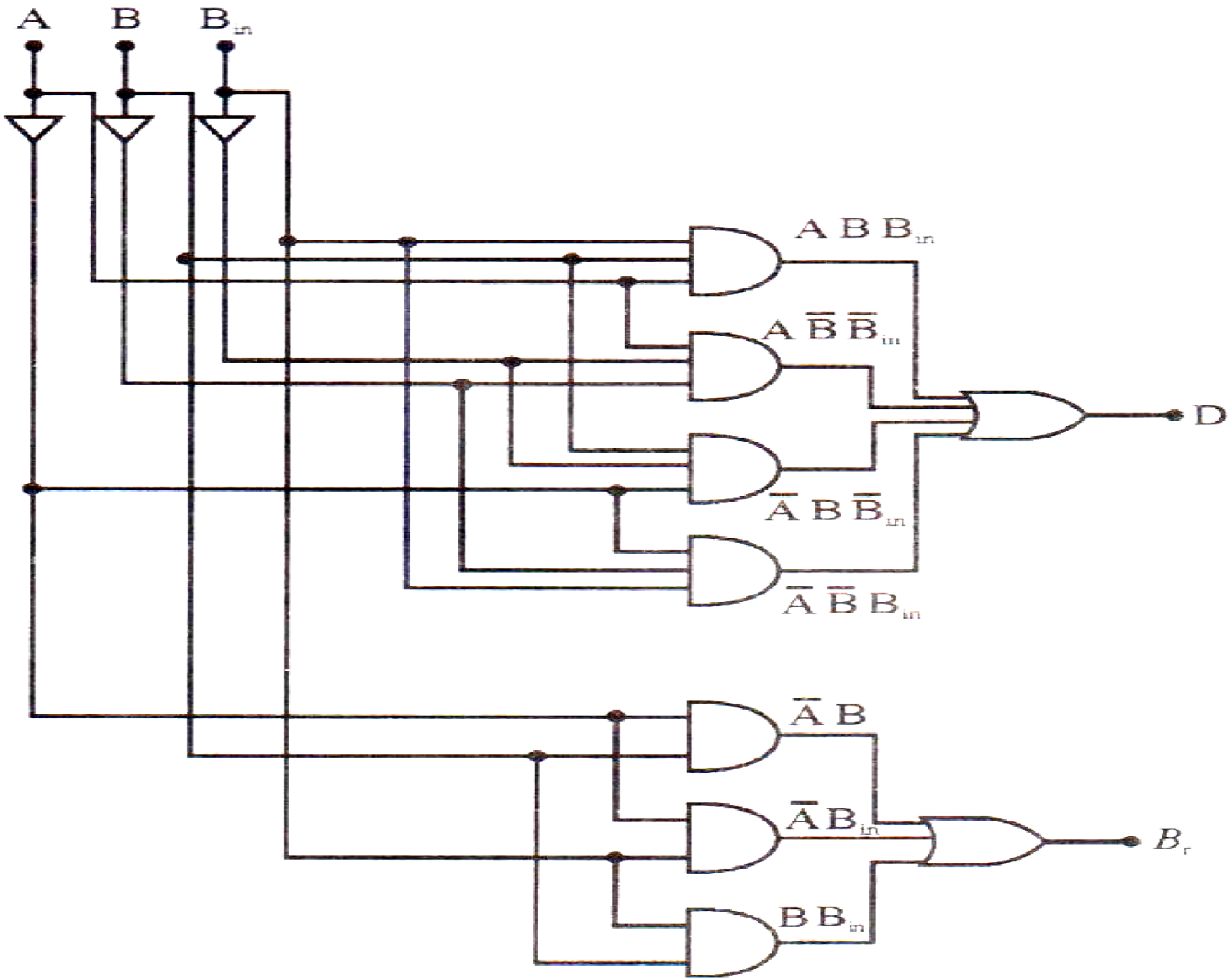


Fig. Full Subtractor

The same function can be performed by the circuit using two half subtractors (see Fig.

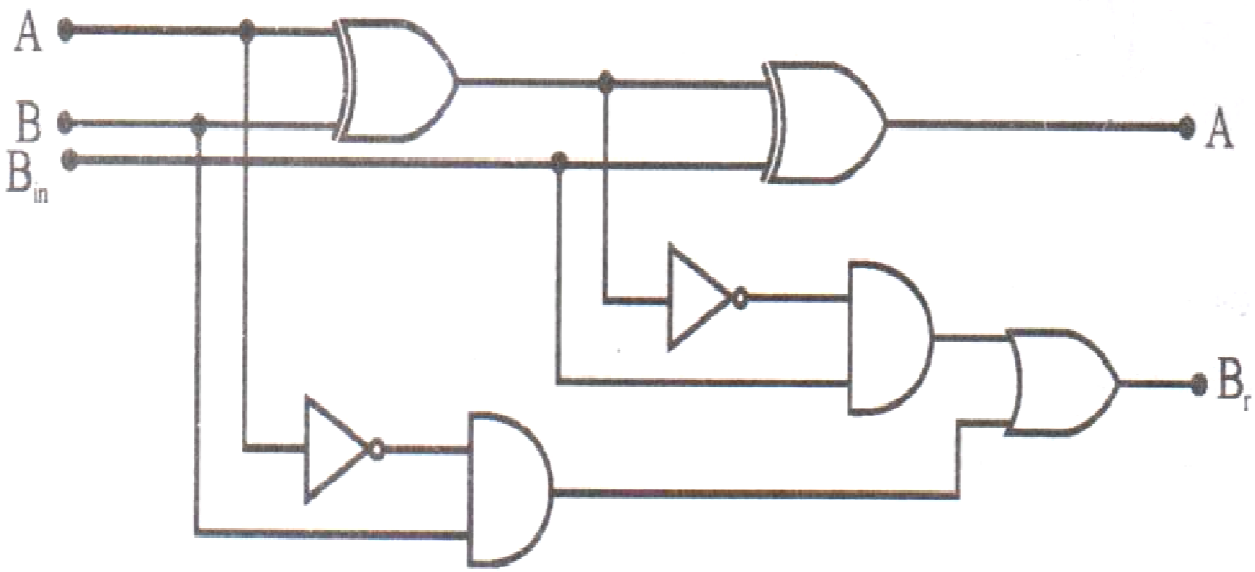
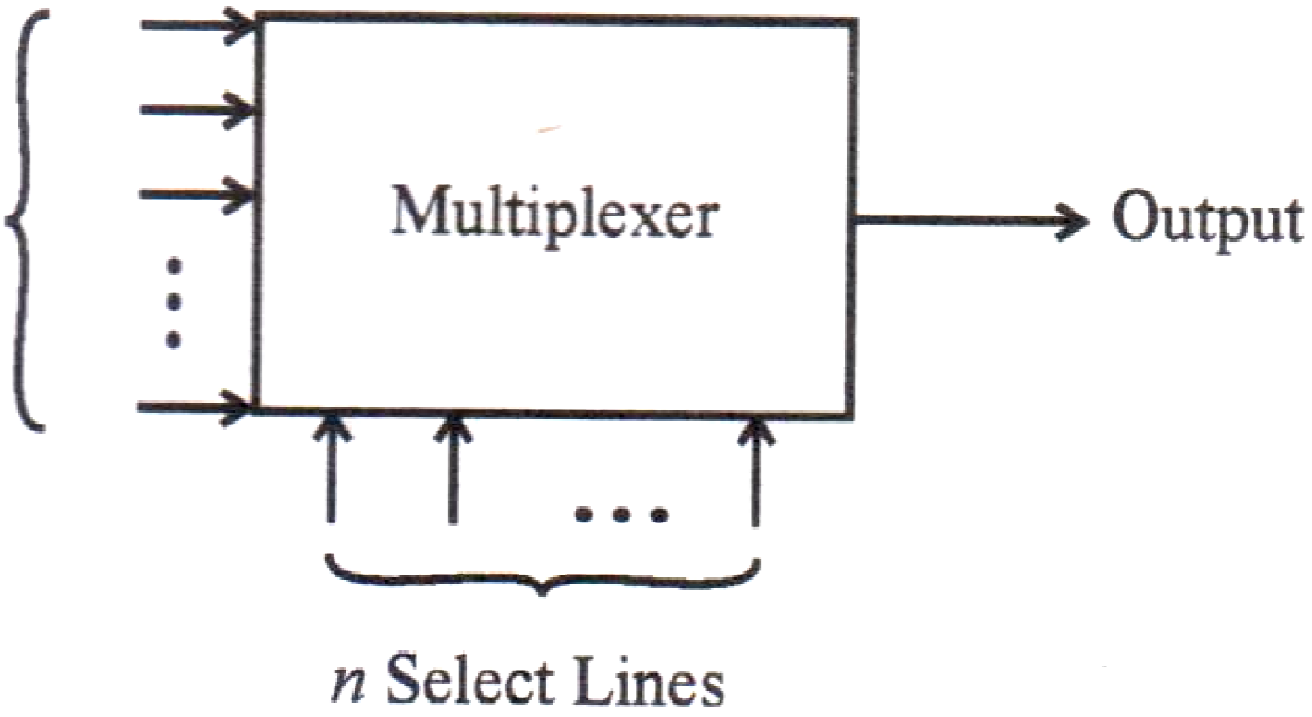


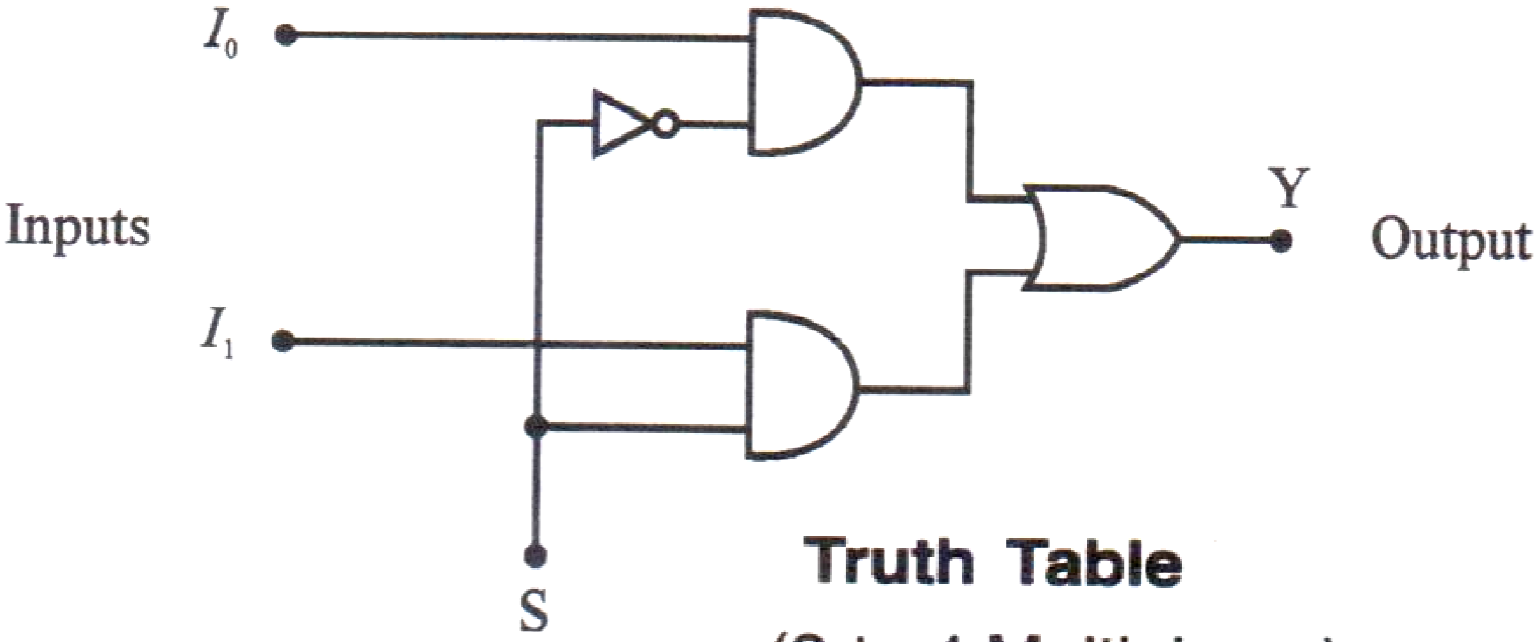
Fig. Full Subtractor using two half subtractor circuits

Multiplexer



Block Diagram of Multiplexer

Two to One Multiplexer

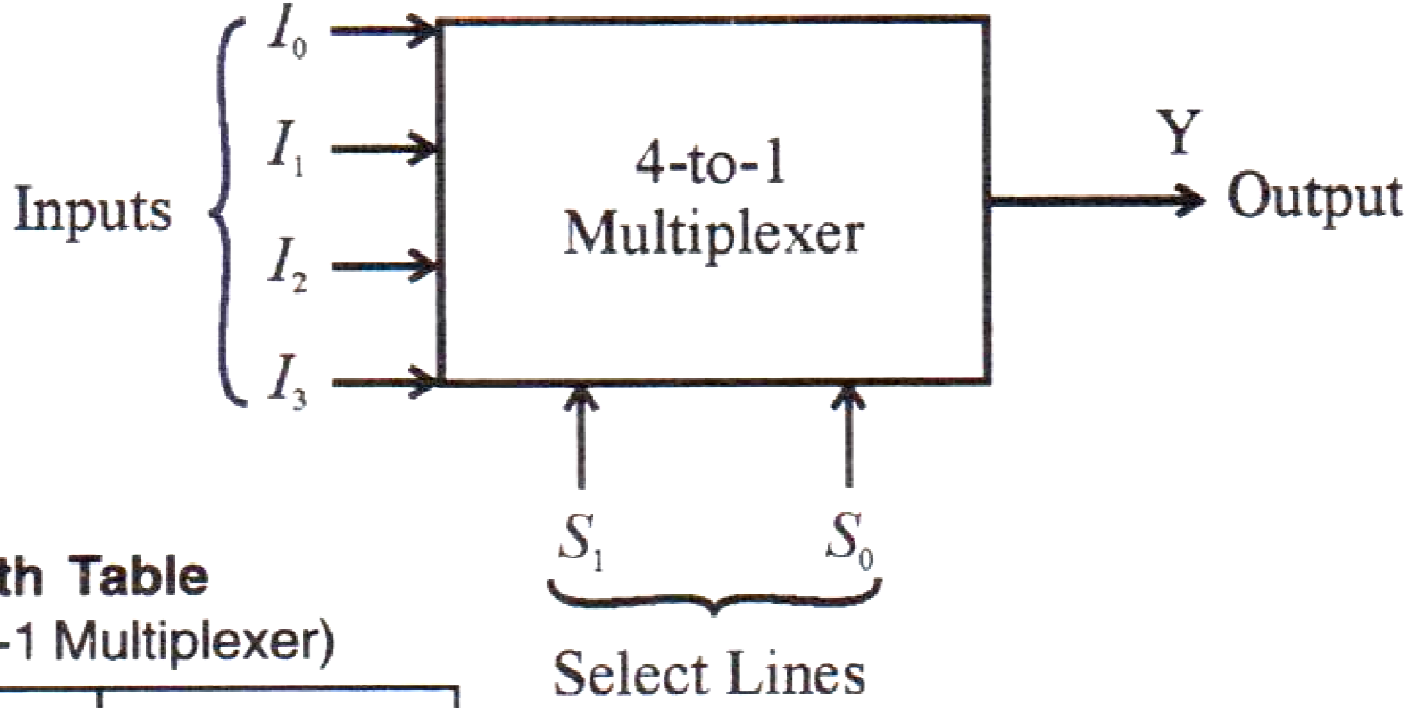


Truth Table

(2-to-1 Multiplexer)

Select Line	Output
S	Y
0	I_0
1	I_1

Four to One Multiplexer



Truth Table
(4-to-1 Multiplexer)

Select Line		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From the truth table we see that when:

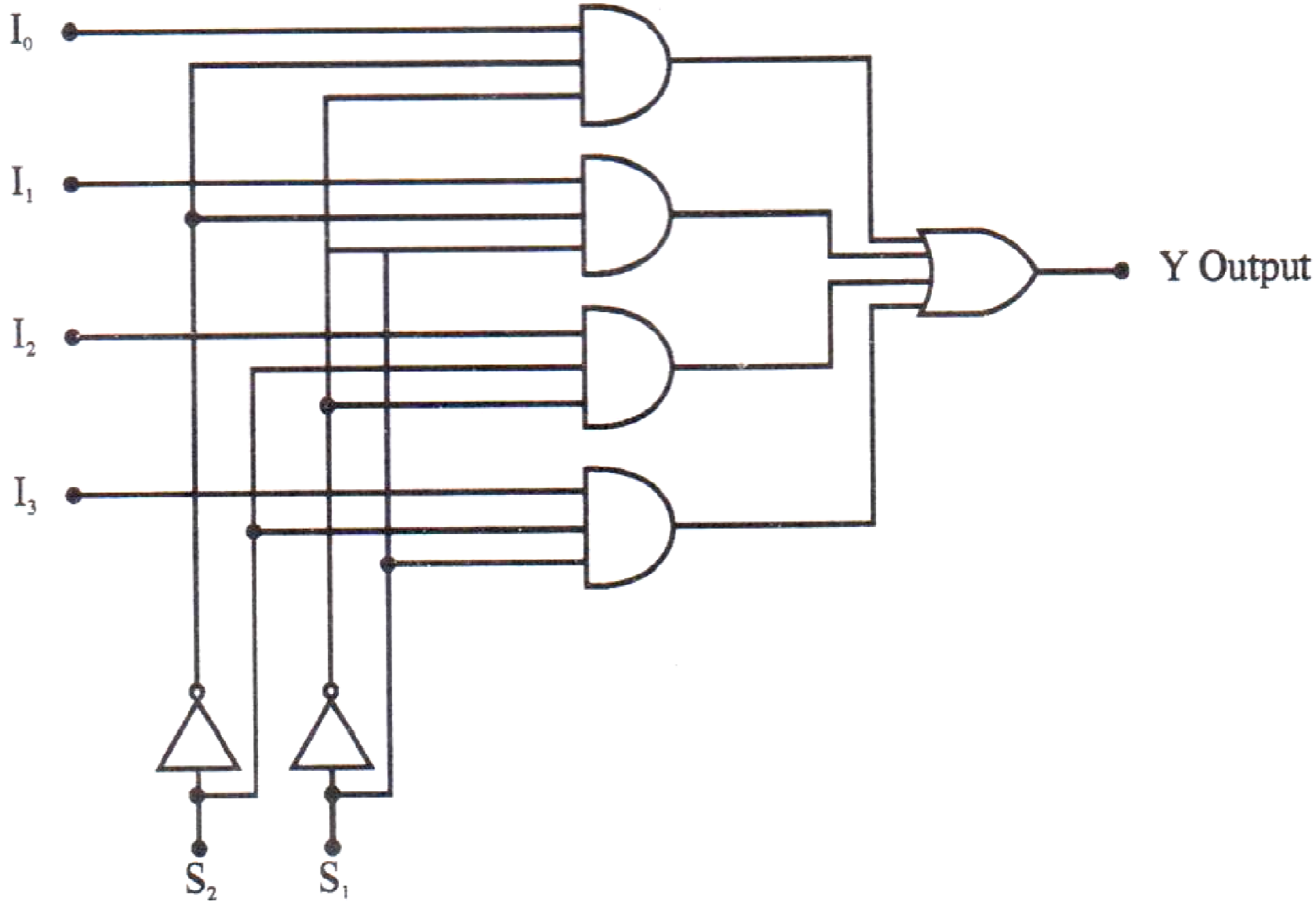
$S_1 = S_0 = 0$, then at the output we get data of I_0

$S_1 = 0$, $S_0 = 1$, then at the output we get data of I_1

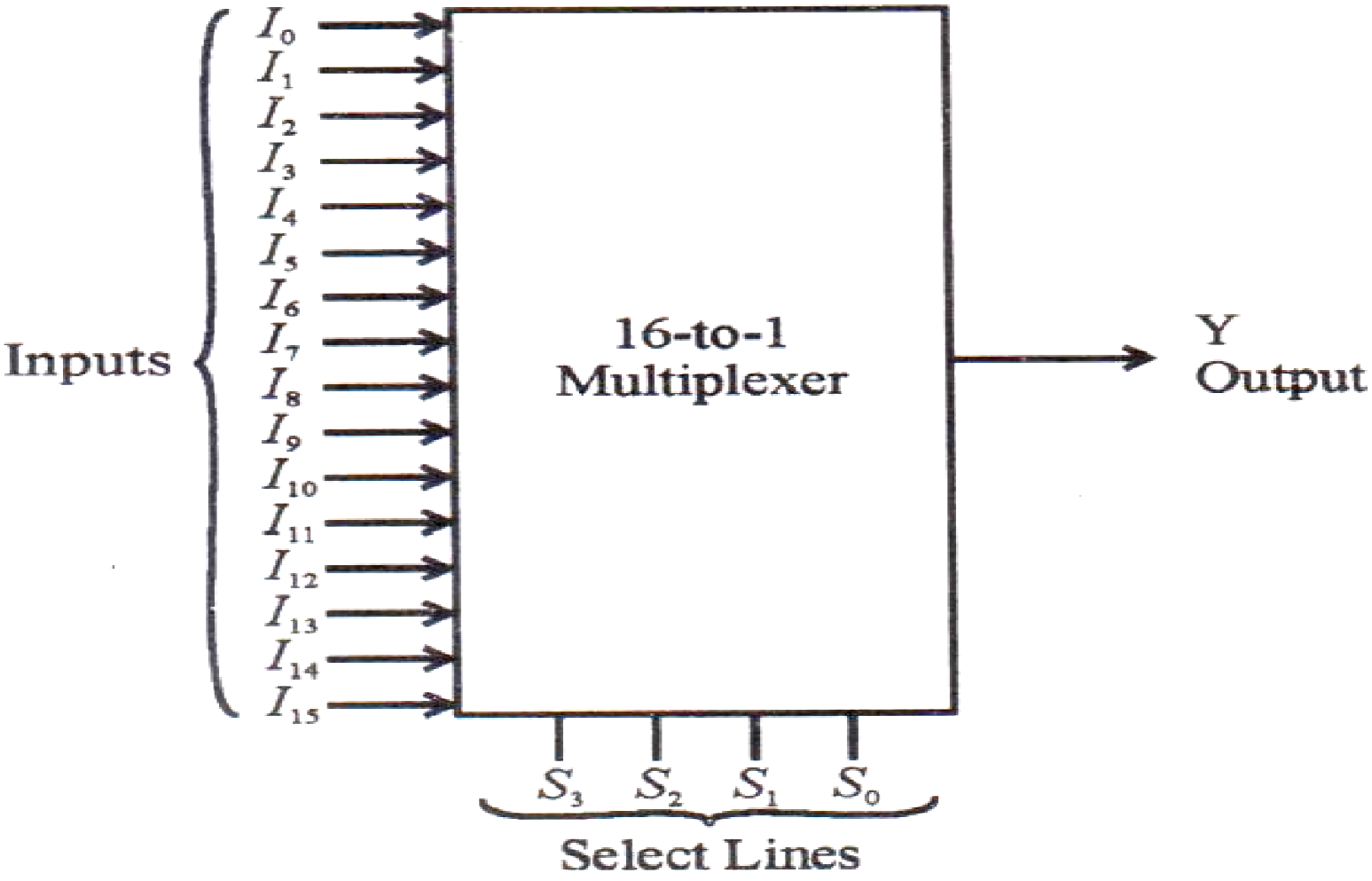
$S_1 = 1$, $S_0 = 0$, then at the output we get data of 1_2.

$S_1 = S_0 = 1$, then at the output we get data of I_3.

Combinational Circuit for Four to One Multiplexer

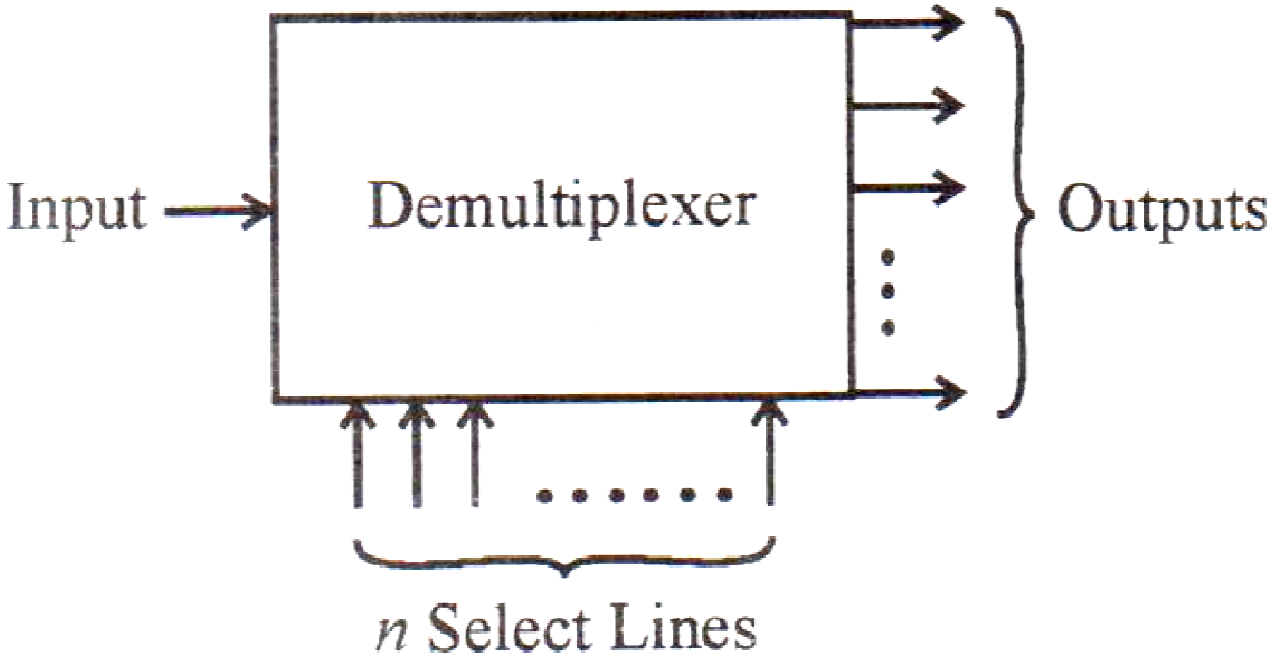


Sixteen to One Multiplexer



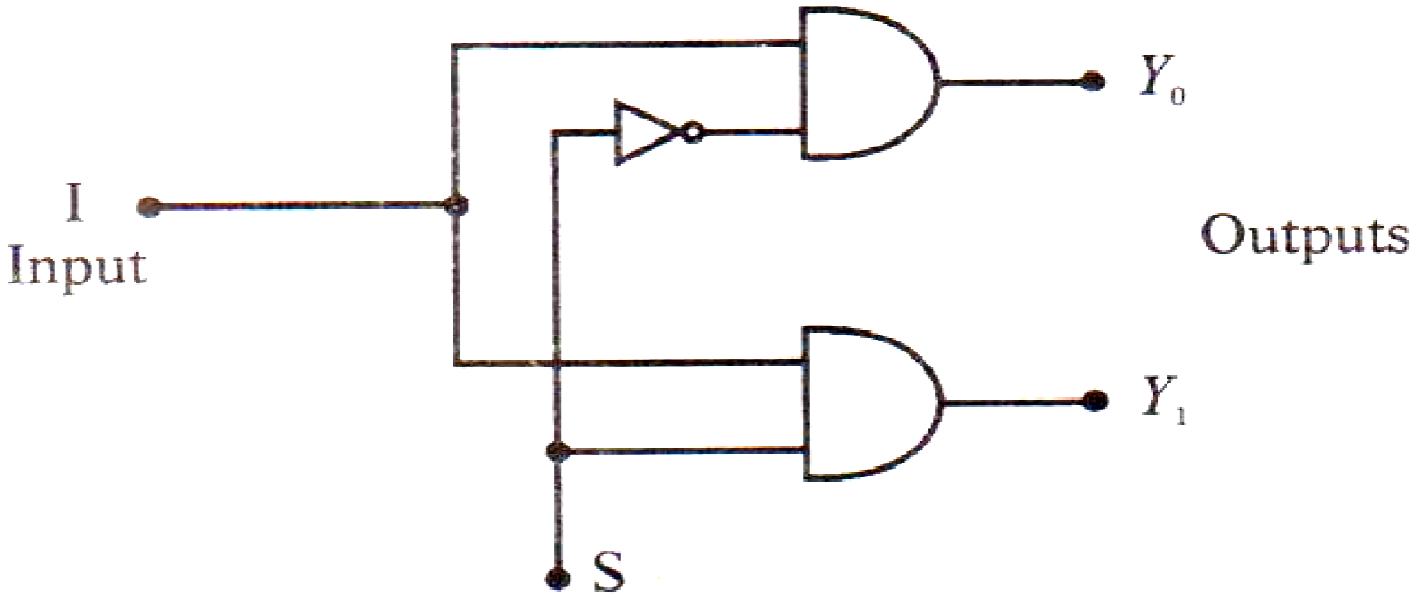
Block Diagram of 16-to-1 Multiplexer

De-Multiplexer



Block Diagram of De-multiplexer

One to two Multiplexer

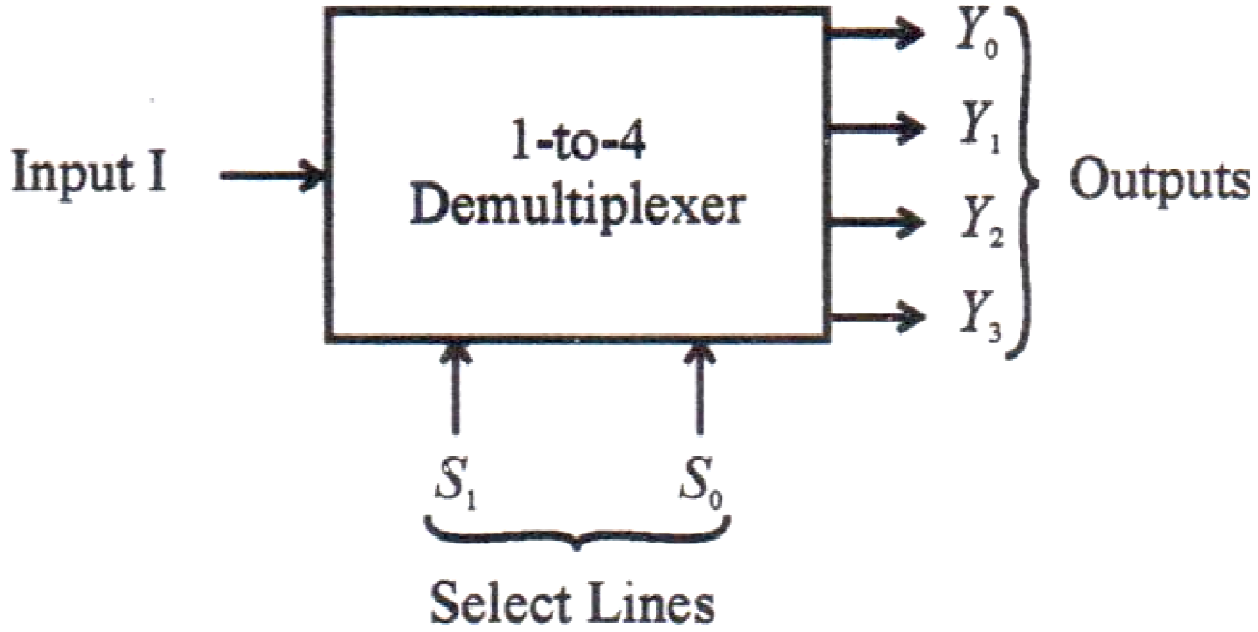


1-to-2 De-multiplexer

Truth Table
(1-to-2 Demultiplexer)

Select Line	Output	
S	Y_1	Y_0
0	0	I
1	I	0

One to Four Demultiplexer



Truth Table
(1-to4 Demultiplexer)

Select Line		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

From the truth table of 1-to-4 de-multiplexer we see that when

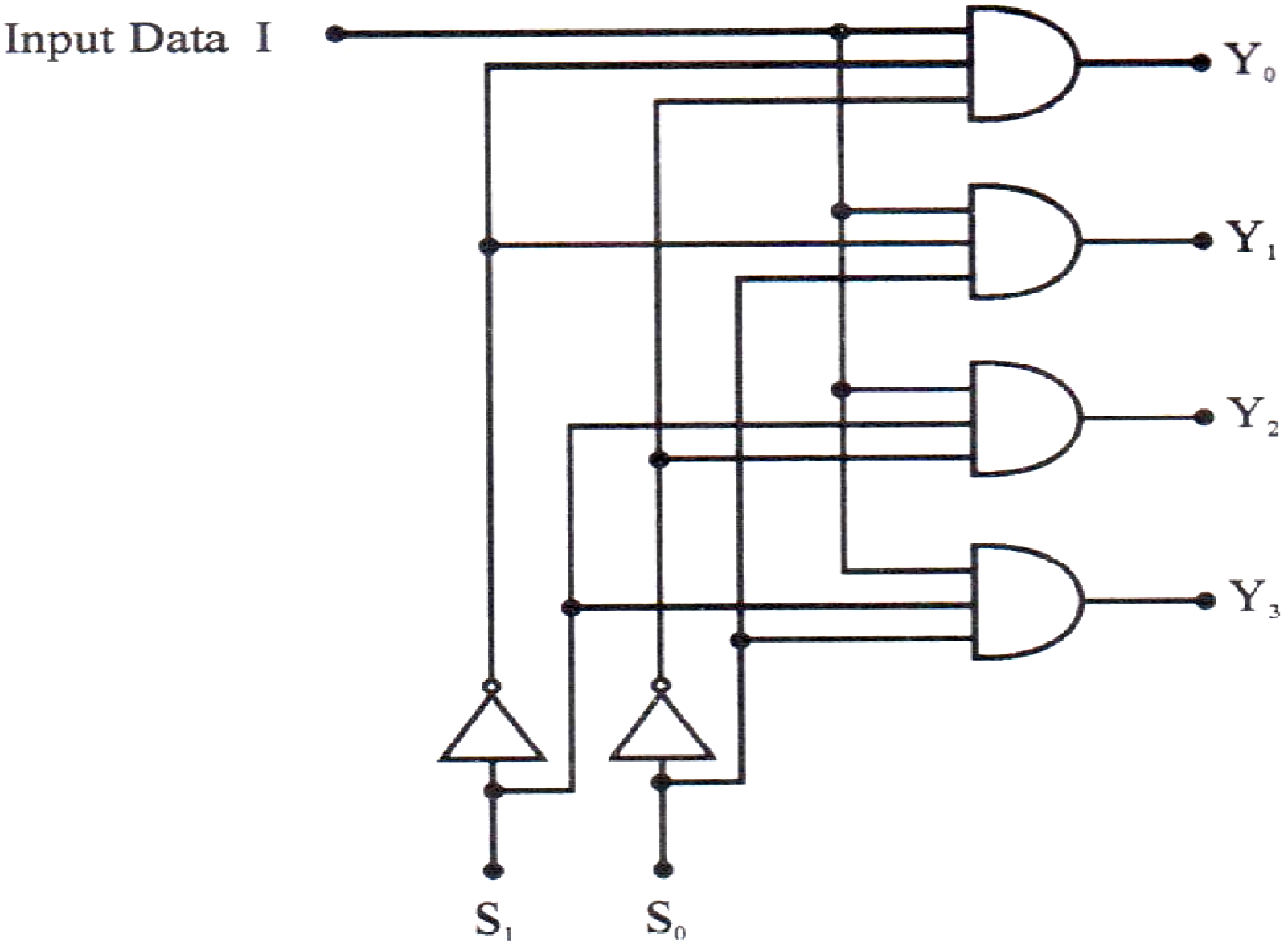
$S_1 = S_0 = 0$, then input data is present on Y_0 output.

$S_1 = 0$, $S_0 = 1$, then input data is present on Y_1 output.

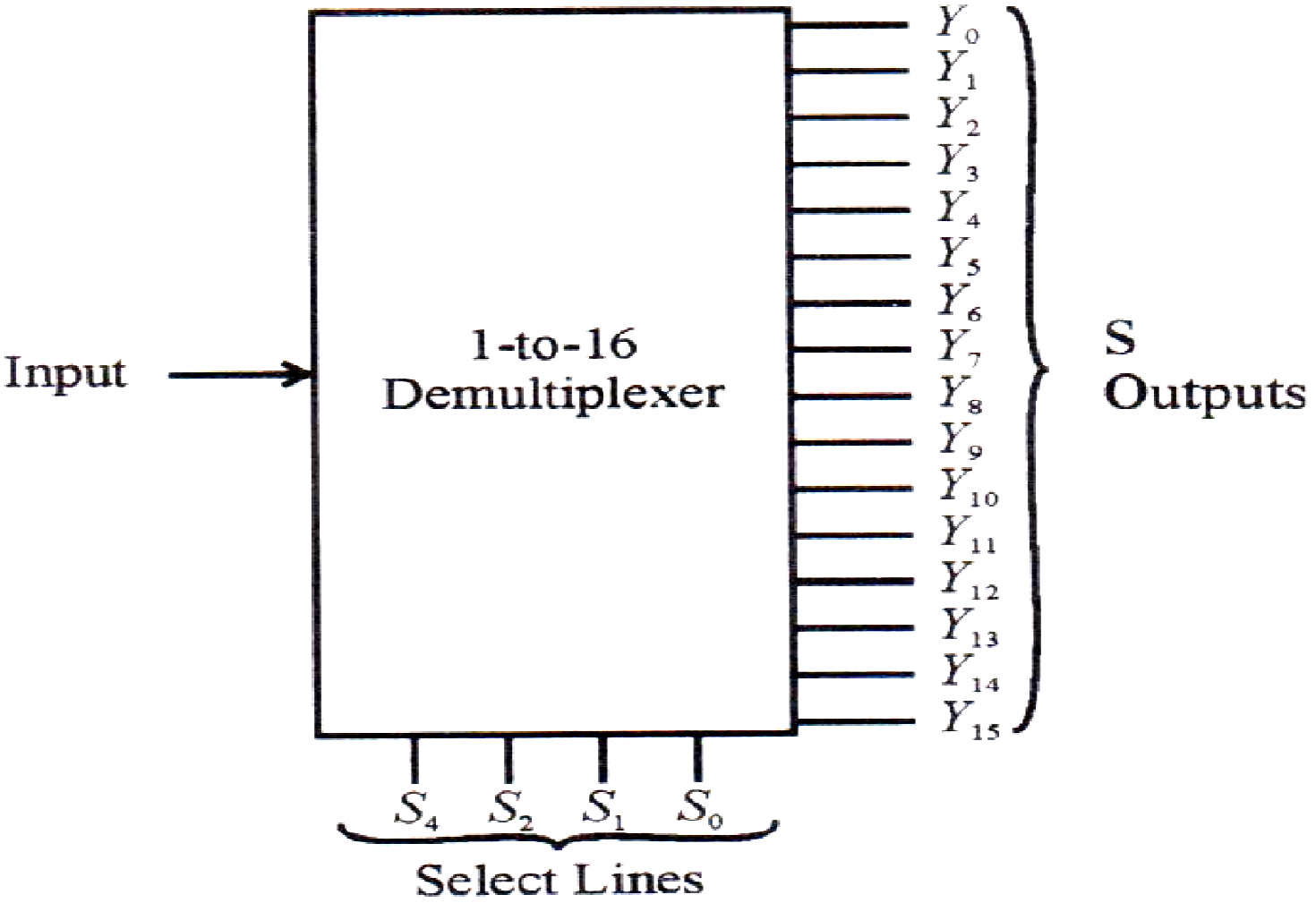
$S_1 = 1$, $S_0 = 0$, then input data is present on Y_2 output.

$S_1 = S_0 = 1$, then input data is present on Y_3 output.

Combination Circuit for One to Four Demultiplexer



Block Diagram of 1-to 16 Demultiplexer



Encoder

Binary encoders

- **Converts one of 2^n inputs to an n-bit output**
- **Useful for compressing data**
- **Can be developed using AND/OR gates**

The encoder is a logic circuit that provides the appropriate code (e.g. binary, as output for each input voltage signal.

The commonly used encoders are :

Decimal to BCD Encoder

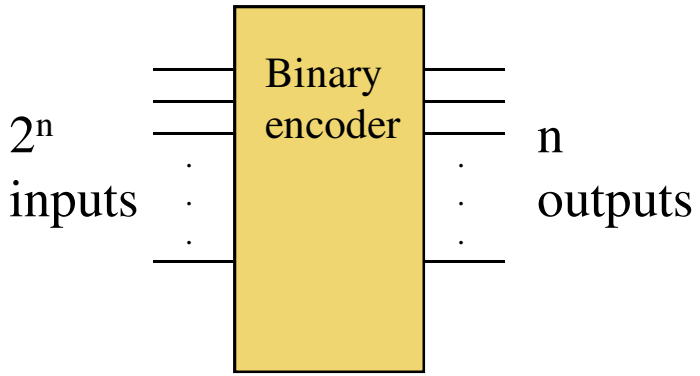
Octal to Binary Encoder.

Cont….Encoders

- If the a decoder's output code has fewer bits than the input code, the device is usually called an encoder.

  e.g. 2^n-to-n

- The simplest encoder is a 2^n-to-n binary encoder
  - One of 2^n inputs = 1
  - Output is an n-bit binary number



Decimal to BCD Encoder

Truth Table

(Decimal to BCD Encoder)

Input										Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D	C	B	A
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

The Boolean expression can be written from the truth table as

$$A = D_1+D_3+D5+D_7+D_9$$

$$B = D_2+D_3+D_6+D_7$$

$$C = D_4+D_5+D_6+D_7$$

$$D = D_8+D_9$$

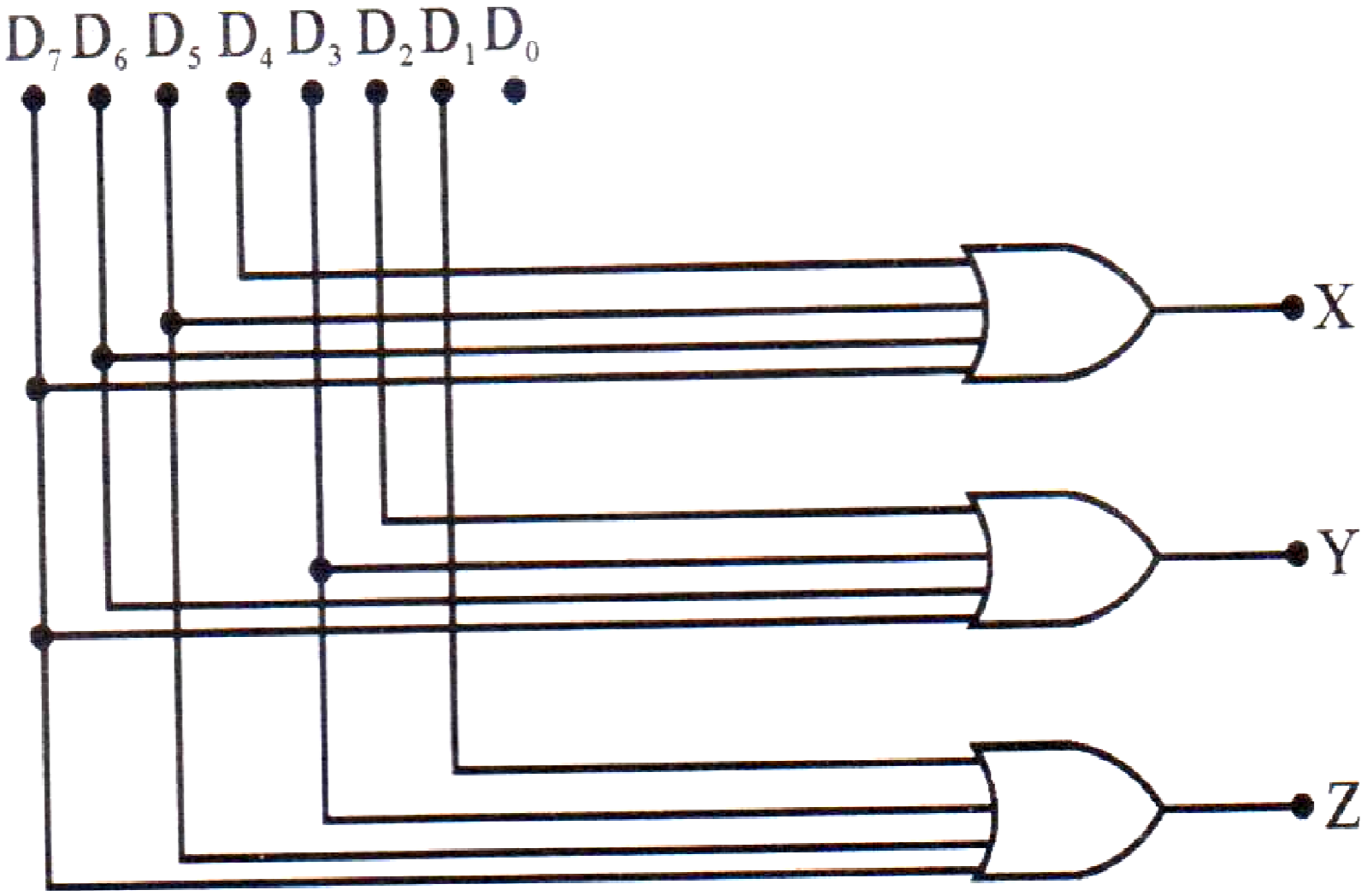
Realizing these equations as shown in the Fig.

$$Y = D_2+D_3+D_6+D_7$$

$$Z = D_1+D_3+D_5+D_7$$

Realizing the outputs using gates

Octal to Binary Encoder Circuit



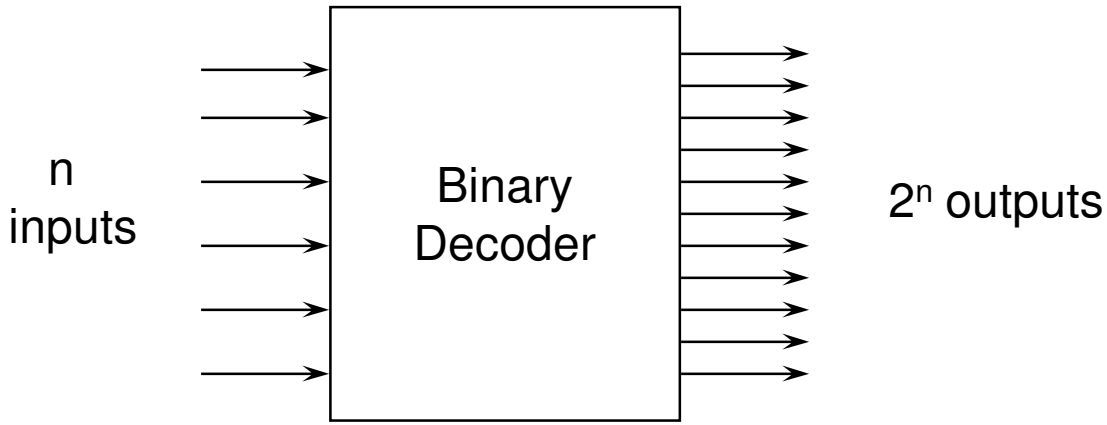
Decoder

Binary decoders

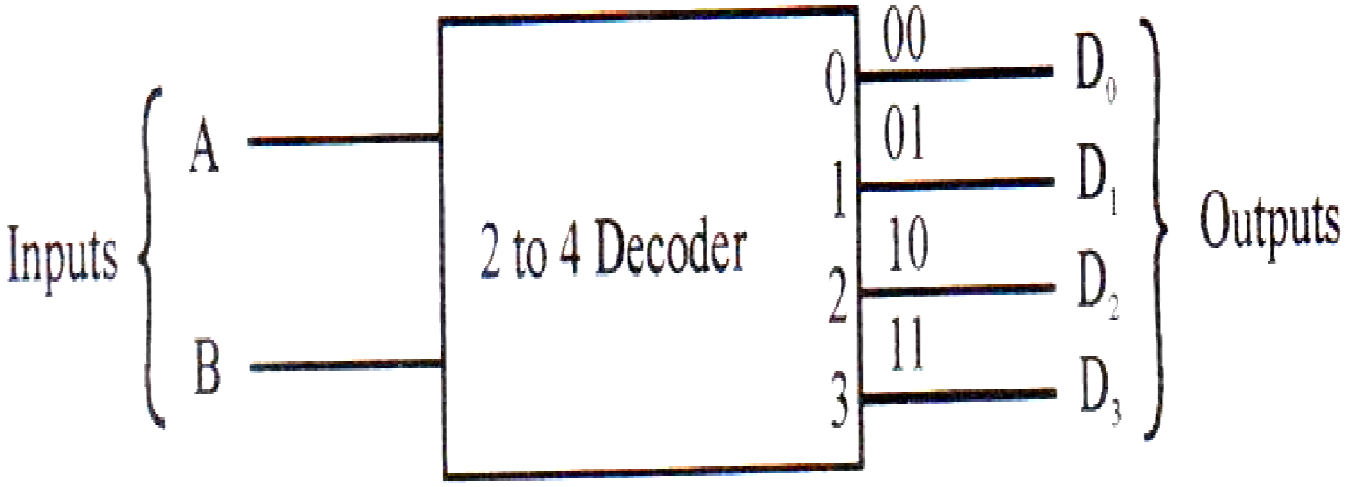
- Converts an n-bit code to a single active output
- Can be developed using AND/OR gates
- Can be used to implement logic circuits.

° **Black box with n input lines and 2^n output lines**

° **Only one output is a 1 for any given input**



Decoder



Block Diagram of 2 to 4 Decoder

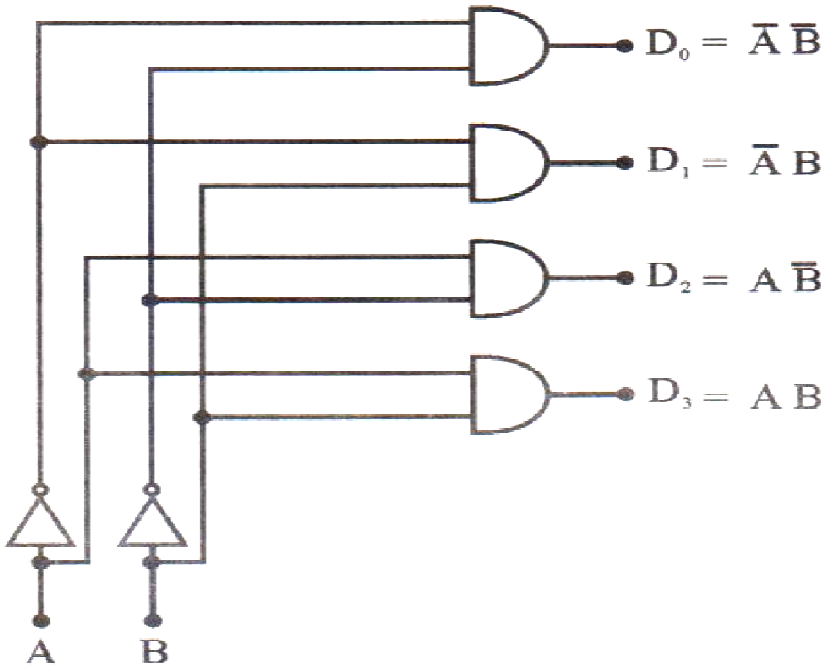
Truth Table
(2 to 4 Decoder)

Input		Output			
A	B	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

From the truth table the equation for D_0, D_1, D_2, D_3 can be written as:

$$D_0 = \overline{A}\,\overline{B}$$
$$D_1 = \overline{A}\,B$$
$$D_2 = A\,\overline{B}$$
$$D_3 = AB$$

Realizing these equations using AND and NOT gates



Combinational circuit of 2 to 4 Decoder