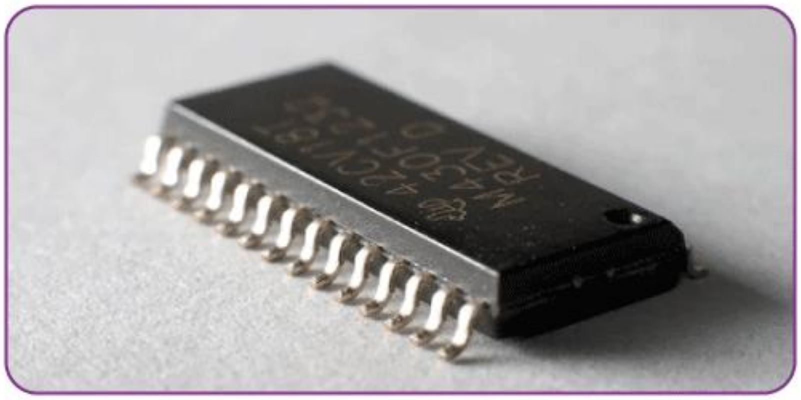
Definition of Integrated Chip

Integrated circuits are made up of several components such as R, C, L, diodes and transistors. They are built on a small single block or chip of a semiconductor known as an integrated circuit (IC). All of them work together to perform a particular task. The IC is easily breakable, so to be attached to a circuit board, it is often housed in a plastic package with metal pins.

INTEGRATED CIRCUIT



Integrated circuits can function as an oscillator, amplifiers, microprocessors or even as computer memory.

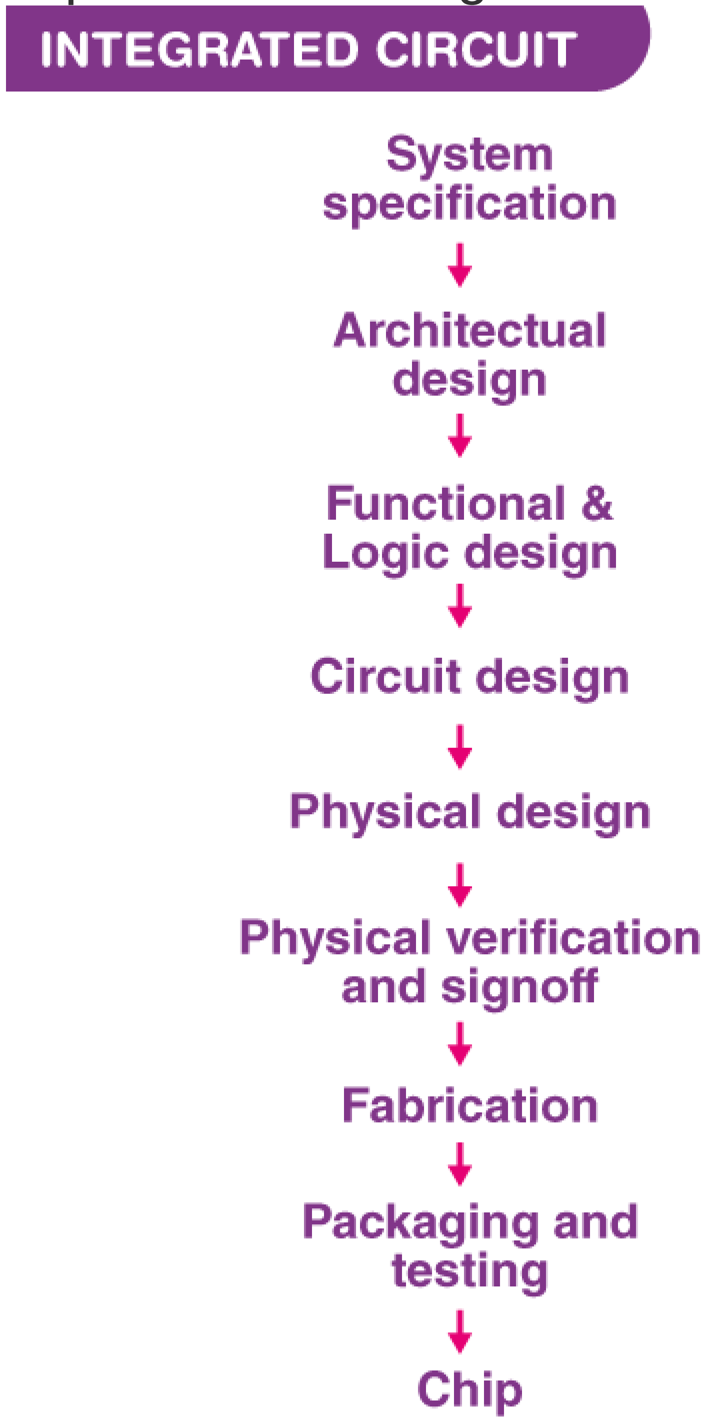
Integrated Circuit Design

An integrated circuit is created using certain logic methods and circuit layouts. The two categories of IC design are as follows:

- **Analog Design**
- **Digital Design**
- **Mixed Design**

Digital Design

The digital design approach is used to create integrated circuits (ICs), which are utilised as computer memories (such as RAM and ROM) and microprocessors. With this approach to design, the circuit density and overall efficiency are both maximised. The ICs created with this technique operate with binary input data like 0 and 1. The process for designing digital integrated circuits is depicted in the diagram below.



Analog Design

IC chip is created by using the analog design process when:

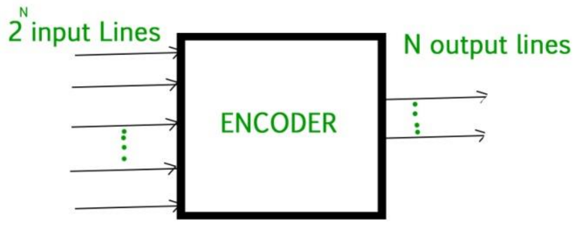
- ICs are utilised as regulators, filters and oscillators.
- Optimal power dissipation, gain and resistance are required.

Mixed Design

The analog and digital design ideas are used in mixed designs. The mixed ICs perform either Analog to Digital or Digital to Analog conversions.

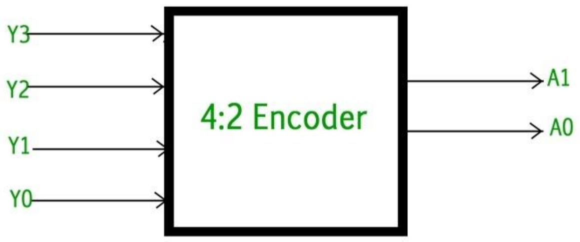
Encoder in Digital Logic

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

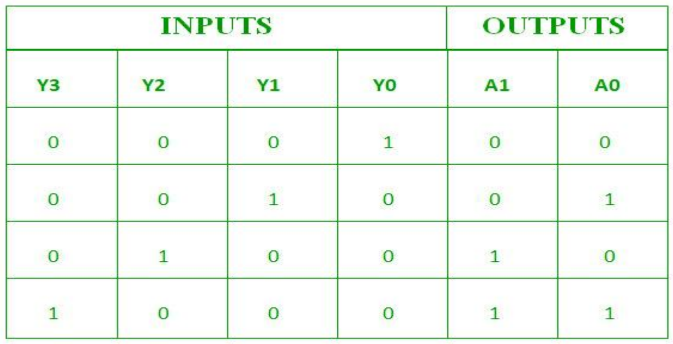


4 : 2 Encoder –

The 4 to 2 Encoder consists of **four inputs Y3, Y2, Y1 & Y0 and two outputs A1 & A0**. At any time, only one of these 4 inputs can be ‘1’ in order to get the respective binary code at the output. The figure below shows the logic symbol of 4 to 2 encoder :



The Truth table of 4 to 2 encoder is as follows :

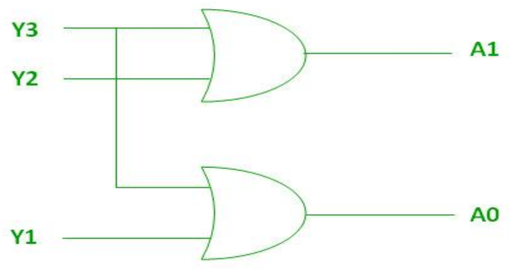


INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Logical expression for A1 and A0 :

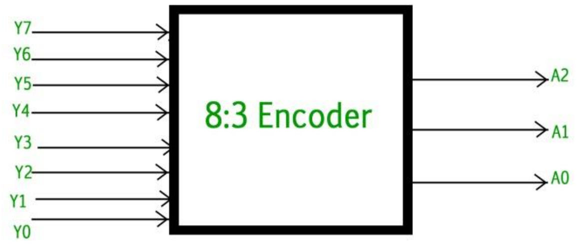
```
A1 = Y3 + Y2

A0 = Y3 + Y1
```

The above two Boolean functions A1 and A0 can be implemented using two input OR gates :

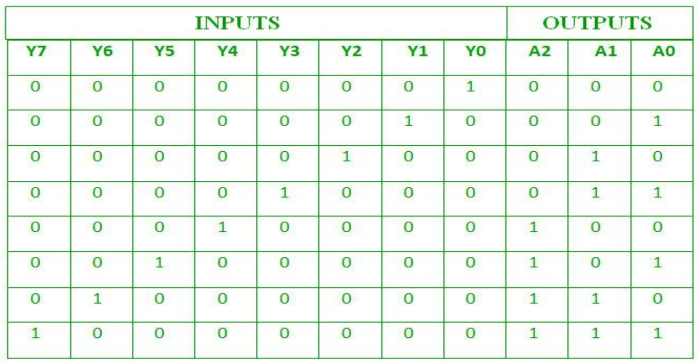


8 : 3 Encoder (Octal to Binary) –

The 8 to 3 Encoder or octal to Binary encoder consists of **8 inputs** : Y7 to Y0 and **3 outputs** : A2, A1 & A0. Each input line corresponds to each octal digit and three outputs generate corresponding binary code.
The figure below shows the logic symbol of octal to binary encoder:



The truth table for 8 to 3 encoder is as follows :

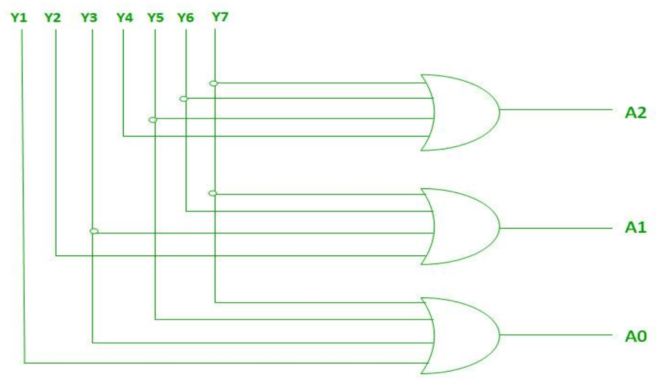


INPUTS								OUTPUTS		
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Logical expression for A2, A1 and A0 :

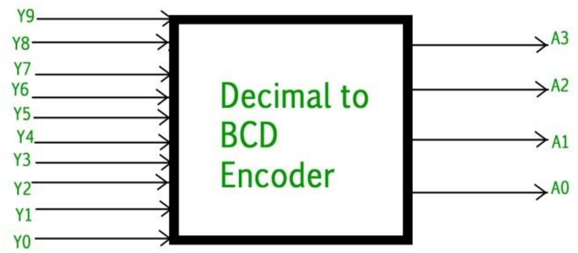
```
A2 = Y7 + Y6 + Y5 + Y4

A1 = Y7 + Y6 + Y3 + Y2

A0 = Y7 + Y5 + Y3 + Y1
```

The above two Boolean functions A2, A1 and A0 can be implemented using four input OR gates :

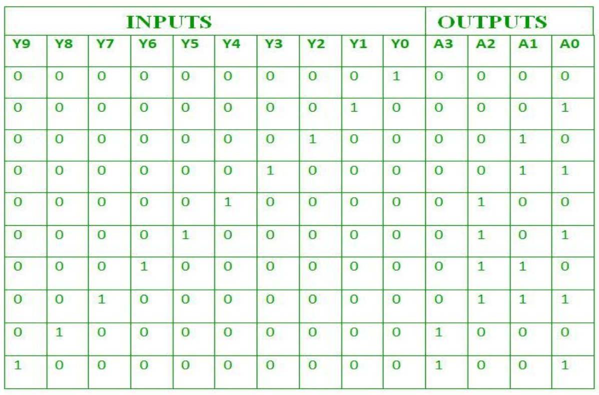


Decimal to BCD Encoder –

The decimal to binary encoder usually consists of **10 input lines** and **4 output lines**. Each input line corresponds to the each decimal digit and 4 outputs correspond to the BCD code. This encoder accepts the decoded decimal data as an input and encodes it to the BCD output which is available on the output lines. The figure below shows the logic symbol of decimal to BCD encoder :



The truth table for decimal to BCD encoder is as follows:



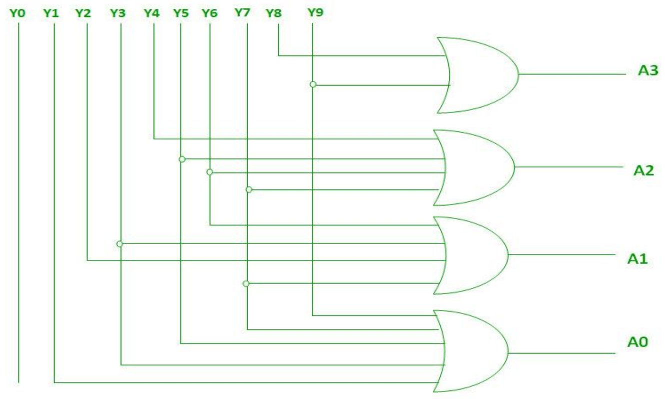
INPUTS										OUTPUTS			
Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Logical expression for A3, A2, A1 and A0 :

```
A3 = Y9 + Y8
```

```
A2 = Y7 + Y6 + Y5 +Y4

A1 = Y7 + Y6 + Y3 +Y2

A0 = Y9 + Y7 +Y5 +Y3 + Y1
```

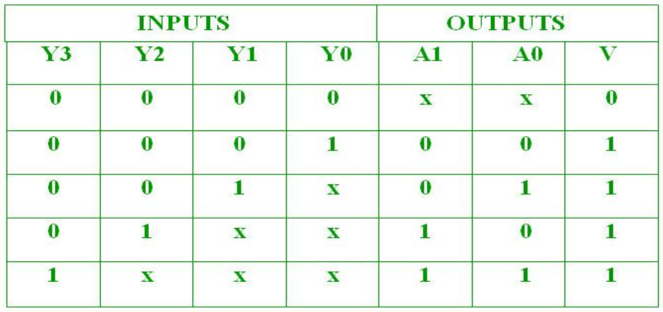
The above two Boolean functions can be implemented using OR gates :



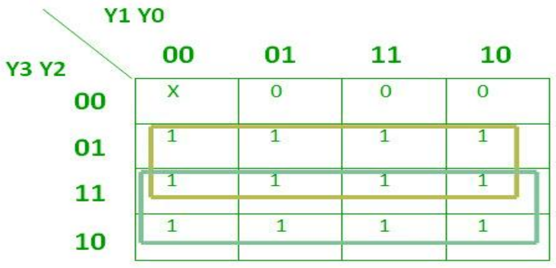
Priority Encoder –

A 4 to 2 priority encoder has **4 inputs** : Y3, Y2, Y1 & Y0 and **2 outputs** : A1 & A0. Here, the input, Y3 has the **highest priority**, whereas the input, Y0 has the **lowest priority**. In this case, even if more than one input is ‘1’ at the same time, the output will be the (binary) code

corresponding to the input, which is having **higher priority**.
The truth table for priority encoder is as follows :

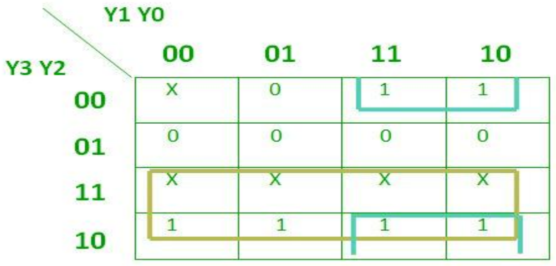


INPUTS				OUTPUTS		
Y3	Y2	Y1	Y0	A1	A0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



Y3 Y2 \ Y1 Y0	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

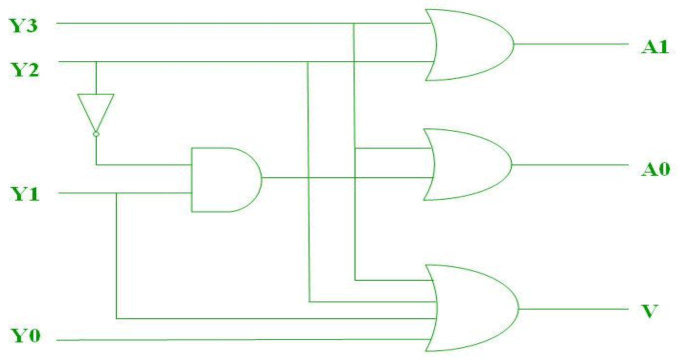
A1=Y3 + Y2



Y3 Y2 \ Y1 Y0	00	01	11	10
00	X	0	1	1
01	0	0	0	0
11	X	X	X	X
10	1	1	1	1

A0=Y3 + Y2’ Y1

The above two Boolean functions can be implemented as :



Drawbacks of Normal Encoders –

1. There is an ambiguity, when all outputs of encoder are equal to zero.
2. If more than one input is active High, then the encoder produces an output, which may not be the correct code.

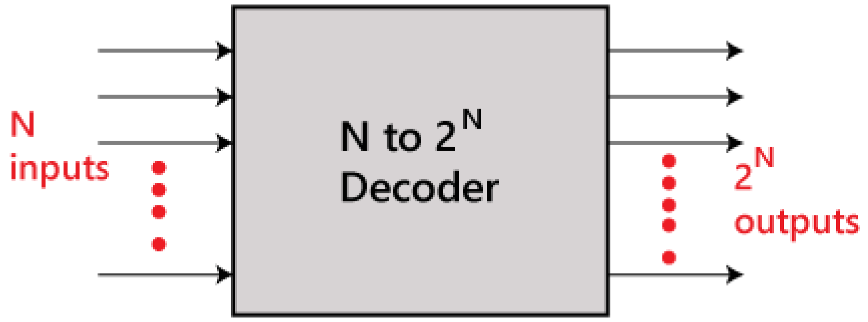
So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the (code corresponding to the active High inputs, which has higher priority.

Uses of Encoders –

1. Encoders are very common electronic circuits used in all digital systems.
2. Encoders are used to translate the decimal values to the binary in order to perform the binary functions such as addition, subtraction, multiplication, etc.
3. Other applications especially for Priority Encoders may include detecting interrupts in microprocessor applications.

Decoder

The combinational circuit that change the binary information into 2^N output lines is known as **Decoders.** The binary information is passed in the form of N input lines. The output lines define the 2^N-bit code for the binary information. In simple words, the **Decoder** performs the reverse operation of the **Encoder**. At a time, only one input line is activated for simplicity. The produced 2^N-bit output code is equivalent to the binary information.

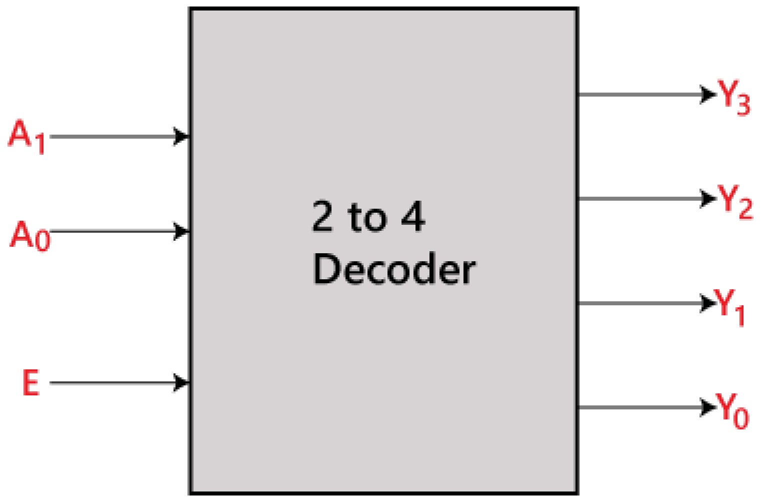


There are various types of decoders which are as follows:

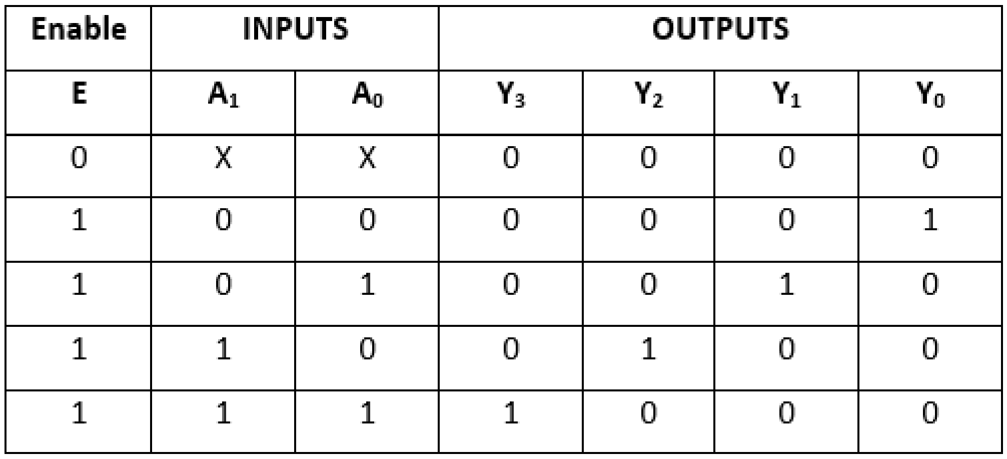
2 to 4 line decoder:

In the 2 to 4 line decoder, there is a total of three inputs, i.e., A_0, and A_1 and E and four outputs, i.e., Y_0, Y_1, Y_2, and Y_3. For each combination of inputs, when the enable 'E' is set to 1, one of these four outputs will be 1. The block diagram and the truth table of the 2 to 4 line decoder are given below.

Block Diagram:



Truth Table:



Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

The logical expression of the term Y0, Y0, Y2, and Y3 is as follows:

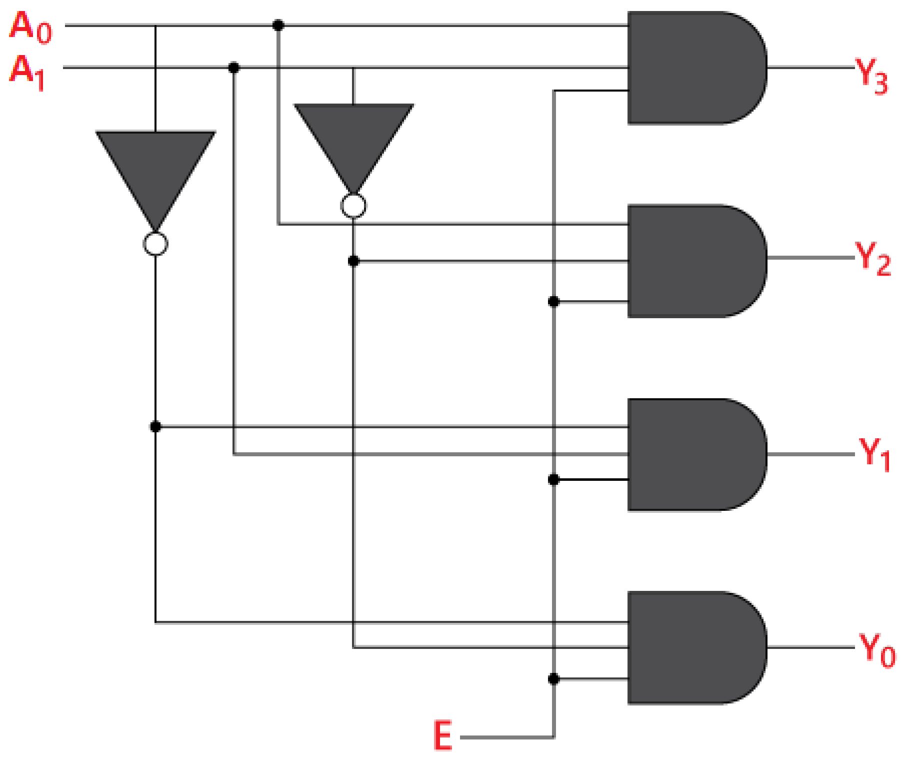
$Y_3=E.A_1.A_0$

$Y_2=E.A_1.A_0'$

$Y_1=E.A_1'.A_0$

$Y0=E.A_1'.A_0'$

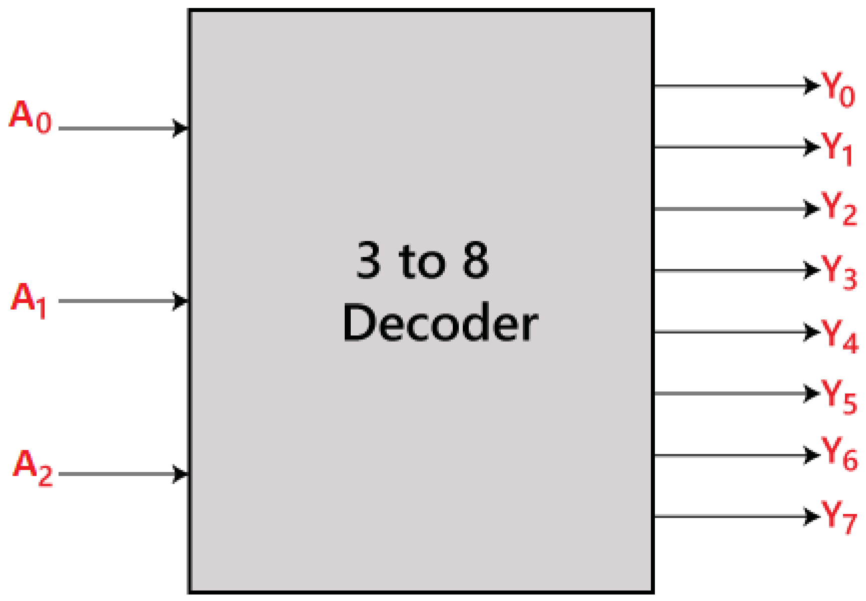
Logical circuit of the above expressions is given below:



3 to 8 line decoder:

The 3 to 8 line decoder is also known as **Binary to Octal Decoder**. In a 3 to 8 line decoder, there is a total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2. This circuit has an enable input 'E'. Just like 2 to 4 line decoder, when enable 'E' is set to 1, one of these four outputs will be 1. The block diagram and the truth table of the 3 to 8 line encoder are given below.

Block Diagram:



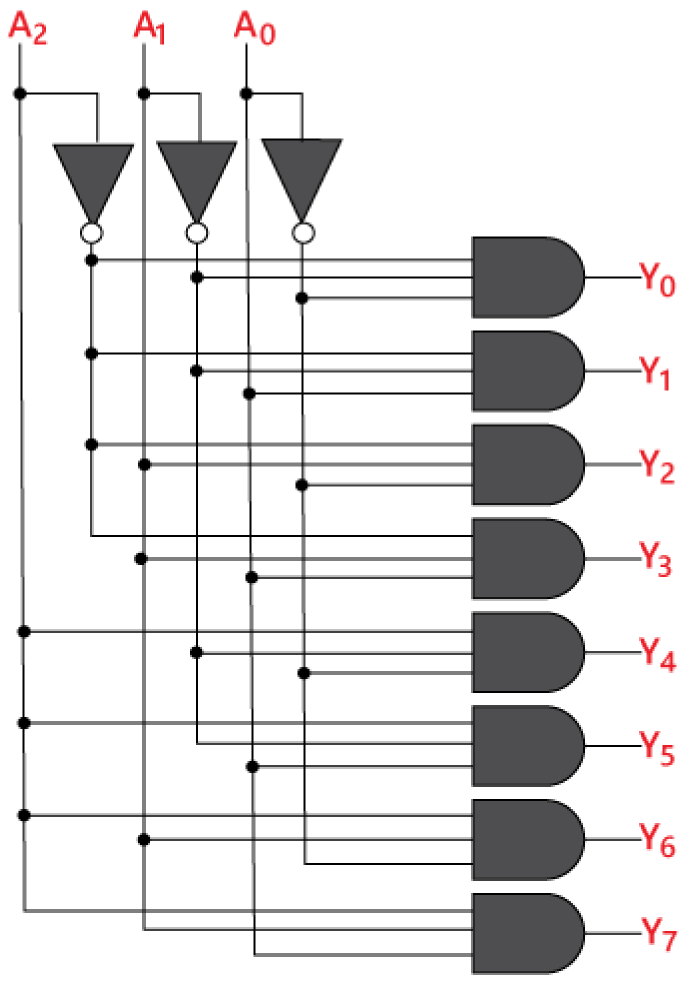
Truth Table:

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

The logical expression of the term Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 is as follows:

$Y_0=A_0'.A_1'.A_2'$
$Y_1=A_0.A_1'.A_2'$
$Y_2=A_0'.A_1.A_2'$
$Y_3=A_0.A_1.A_2'$
$Y_4=A_0'.A_1'.A_2$
$Y_5=A_0.A_1'.A_2$
$Y_6=A_0'.A_1.A_2$
$Y_7=A_0.A_1.A_2$

Logical circuit of the above expressions is given below:



4 to 16 line Decoder

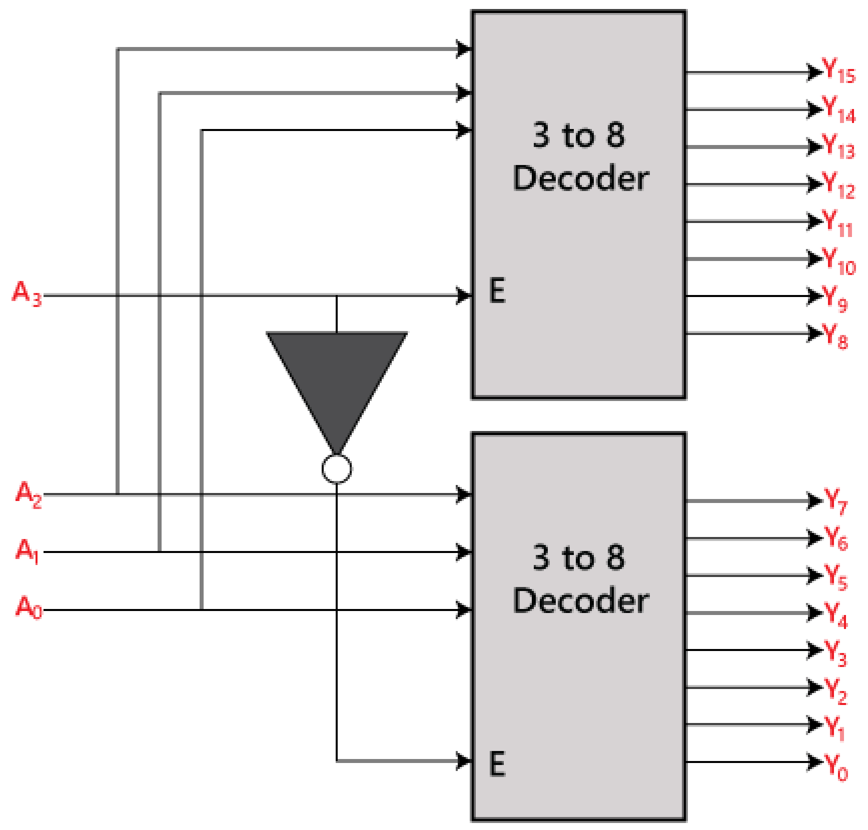
In the 4 to 16 line decoder, there is a total of 16 outputs, i.e., Y_0, Y_1, Y_2,……, Y_{16} and four inputs, i.e., A_0, A1, A_2, and A_3. The 3 to 16 line decoder can be constructed using either 2 to 4 decoder or 3 to 8 decoder. There is the following formula used to find the required number of lower-order decoders.

Required number of lower order decoders=m_2/m_1

m_1 = 8
m_2 = 16

Required number of 3 to 8 decoders=$\frac{16}{8}$=2

Block Diagram:



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

Truth Table:

INPUTS				OUTPUTS															
A_3	A_2	A_1	A_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	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	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The logical expression of the term A0, A1, A2,..., A15 are as follows:

$Y_0=A_0'.A_1'.A_2'.A_3'$
$Y_1=A_0'.A_1'.A_2'.A_3$
$Y_2=A_0'.A_1'.A_2.A_3'$

$Y_3=A_0'.A_1'.A_2.A_3$

$Y_4=A_0'.A_1.A_2'.A_3'$

$Y_5=A_0'.A_1.A_2'.A_3$

$Y_6=A_0'.A_1.A_2.A_3'$

$Y_7=A_0'.A_1.A_2.A_3$

$Y_8=A_0.A_1'.A_2'.A_3'$

$Y_9=A_0.A_1'.A_2'.A_3$

$Y_{10}=A_0.A_1'.A_2.A_3'$

$Y_{11}=A_0.A_1'.A_2.A_3$

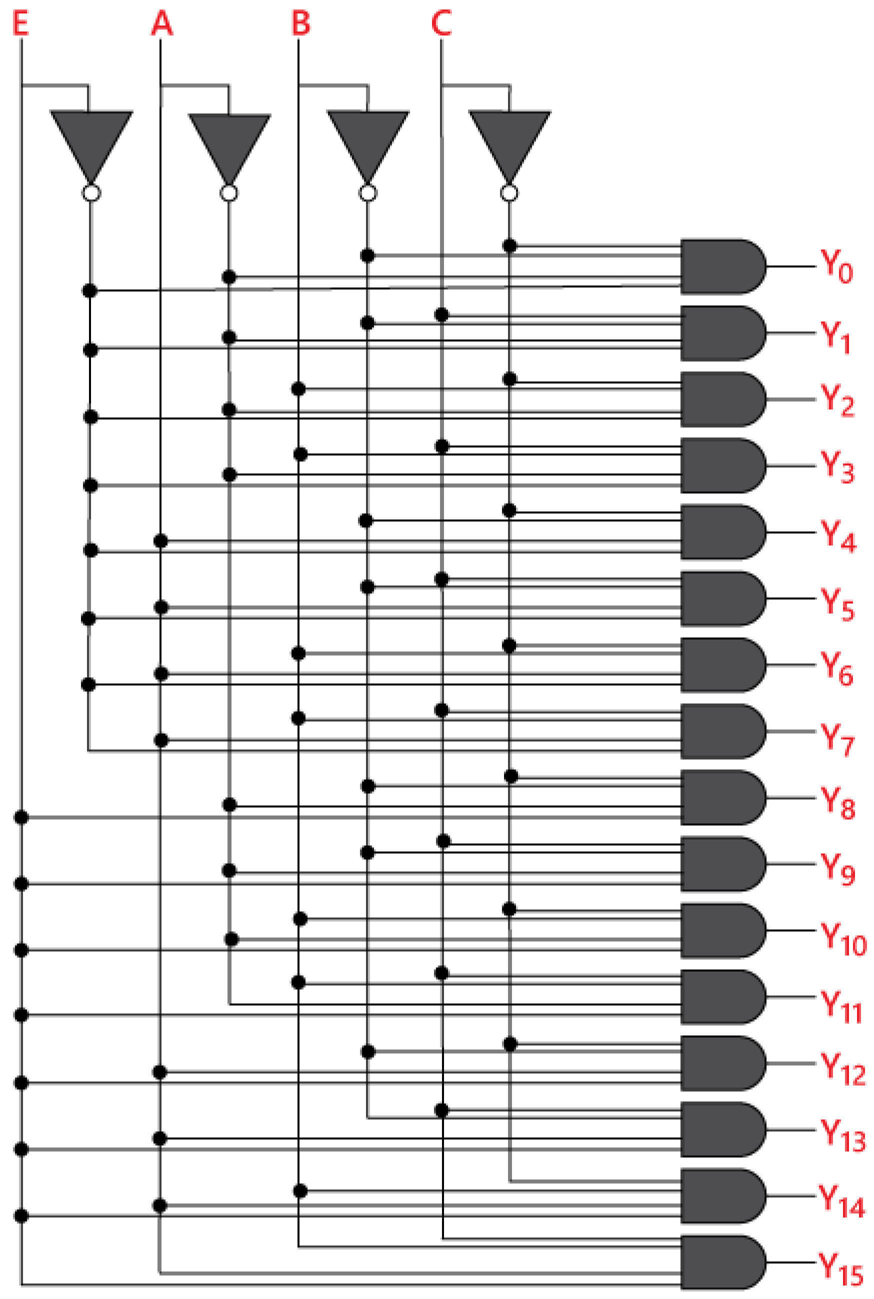
$Y_{12}=A_0.A_1.A_2'.A_3'$

$Y_{13}=A_0.A_1.A_2'.A_3$

$Y_{14}=A_0.A_1.A_2.A_3'$

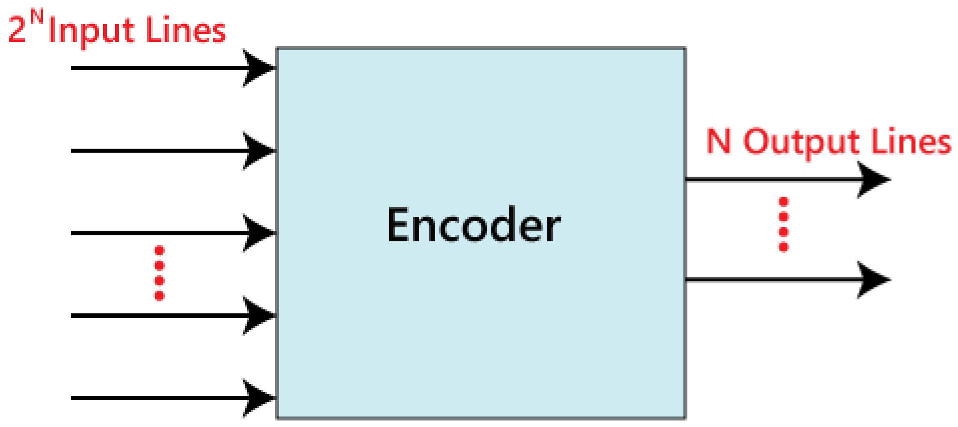
$Y_{15}=A_0.A_1.A_2'.A_3$

Logical circuit of the above expressions is given below:


E
A
B
C
Y_0
Y_1
Y_2
Y_3
Y_4
Y_5
Y_6
Y_7
Y_8
Y_9
Y_{10}
Y_{11}
Y_{12}
Y_{13}
Y_{14}
Y_{15}

Encoders

The combinational circuits that change the binary information into N output lines are known as **Encoders**. The binary information is passed in the form of 2^N input lines. The output lines define the N-bit code for the binary information. In simple words, the **Encoder** performs the reverse operation of the **Decoder**. At a time, only one input line is activated for simplicity. The produced N-bit output code is equivalent to the binary information.

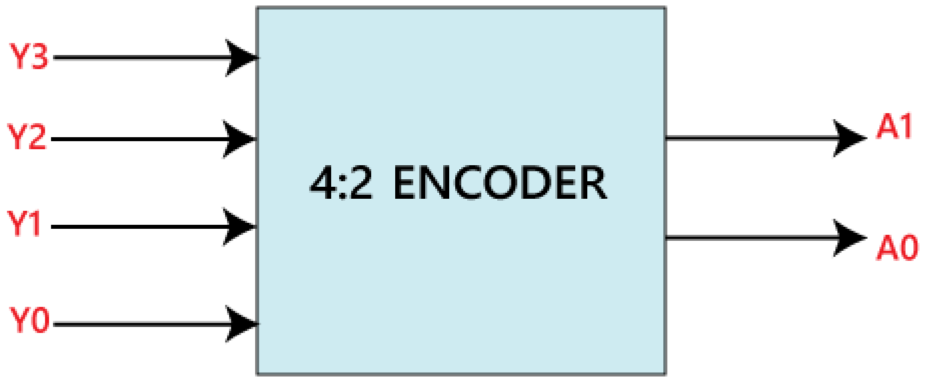


There are various types of encoders which are as follows:

4 to 2 line Encoder:

In 4 to 2 line encoder, there are total of four inputs, i.e., Y_0, Y_1, Y_2, and Y_3, and two outputs, i.e., A_0 and A_1. In 4-input lines, one input-line is set to true at a time to get the respective binary code in the output side. Below are the block diagram and the truth table of the 4 to 2 line encoder.

Block Diagram:



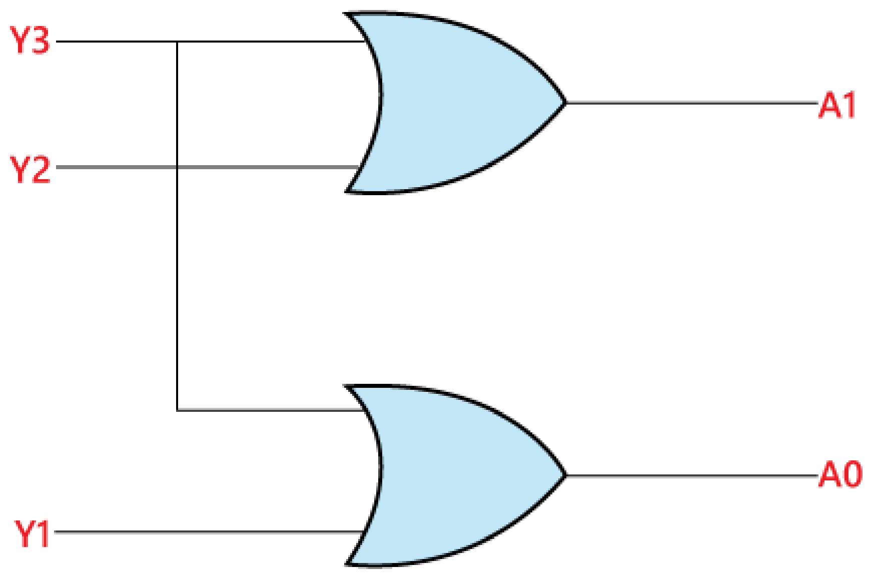
Truth Table:

INPUTS				OUTPUTS	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

The logical expression of the term A0 and A1 is as follows:

$A_1=Y_3+Y_2$
$A_0=Y_3+Y_1$

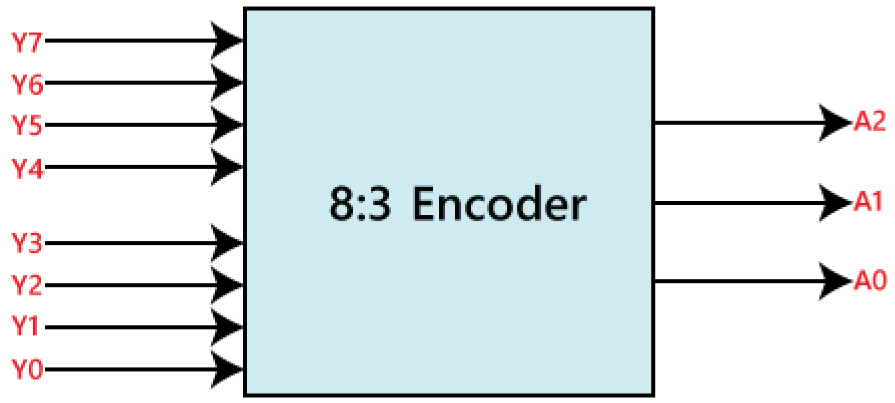
Logical circuit of the above expressions is given below:



8 to 3 line Encoder:

The 8 to 3 line Encoder is also known as **Octal to Binary Encoder**. In 8 to 3 line encoder, there is a total of eight inputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2. In 8-input lines, one input-line is set to true at a time to get the respective binary code in the output side. Below are the block diagram and the truth table of the 8 to 3 line encoder.

Block Diagram:



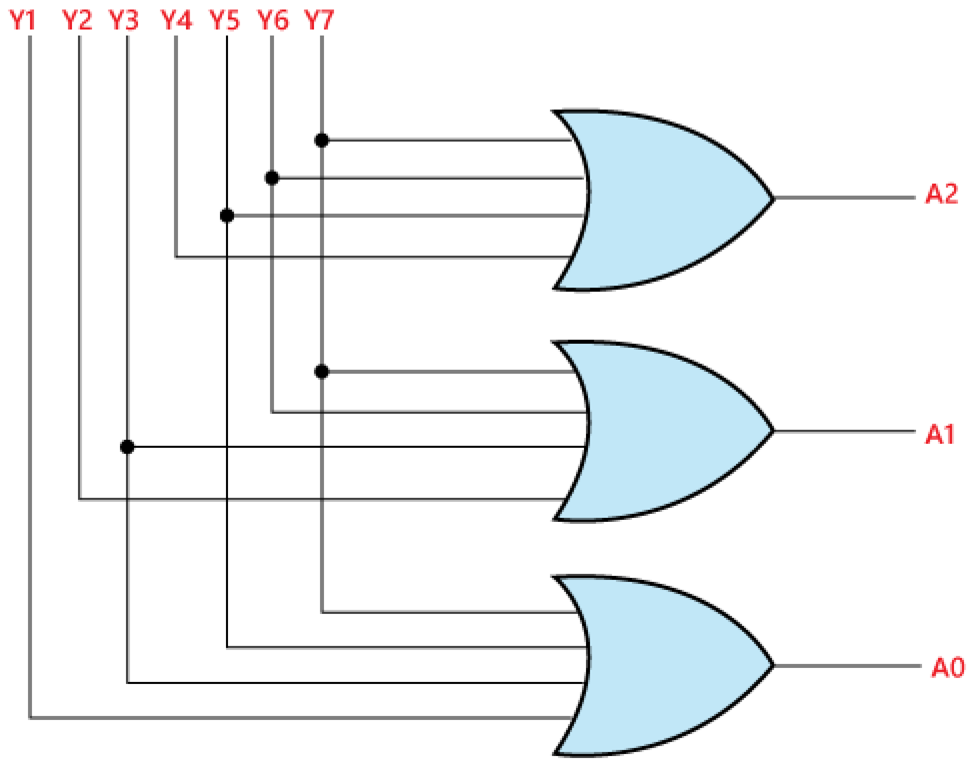
Truth Table:

INPUTS								OUTPUTS		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

The logical expression of the term A0, A1, and A2 are as follows:

$A_2=Y_4+Y_5+Y_6+Y_7$
$A_1=Y_2+Y_3+Y_6+Y_7$
$A_0=Y_7+Y_5+Y_3+Y_1$

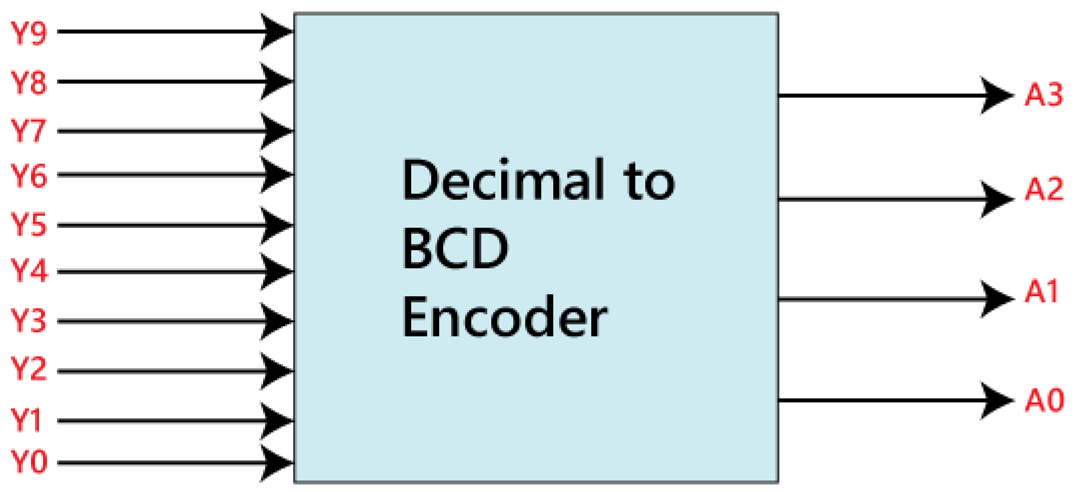
Logical circuit of the above expressions is given below:



Decimal to BCD Encoder

The Octal to Binary Encoder is also known as **10 to 4 line Encoder**. In 10 to 4 line encoder, there are total of ten inputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8, and Y_9 and four outputs, i.e., A_0, A1, A_2, and A_3. In 10-input lines, one input-line is set to true at a time to get the respective **BCD code** in the output side. The block diagram and the truth table of the decimal to BCD encoder are given below.

Block Diagram:



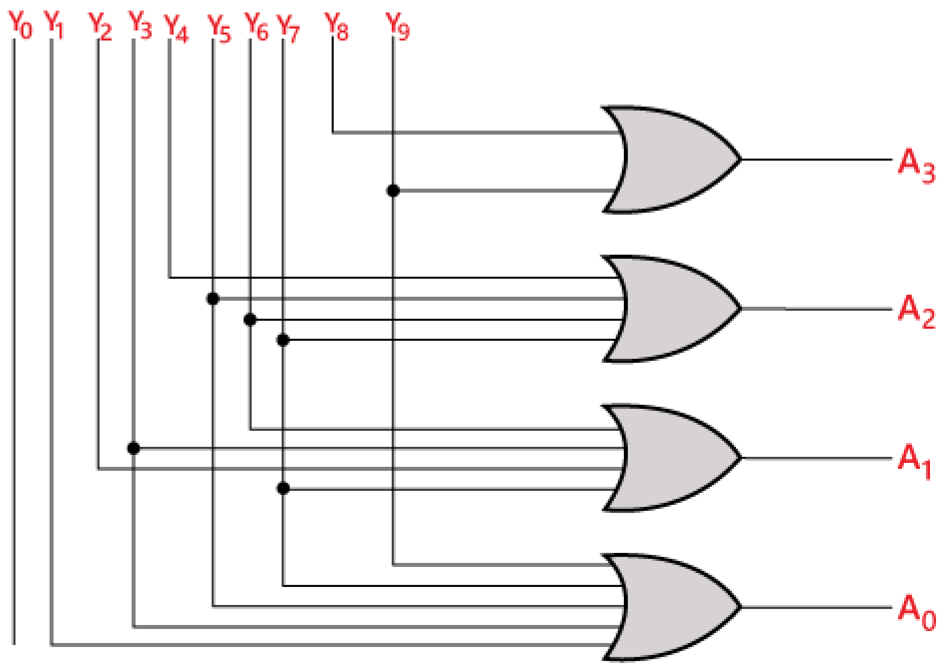
Truth Table:

INPUTS										OUTPUTS			
Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_3	A_2	A_1	A_0
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 logical expression of the term A_0, A_1, A_2, and A_3 is as follows:

$A3 = Y9 + Y8$
$A2 = Y7 + Y6 + Y5 + Y4$
$A1 = Y7 + Y6 + Y3 + Y2$
$A0 = Y9 + Y7 + Y5 + Y3 + Y1$

Logical circuit of the above expressions is given below:



Priority Encoder:

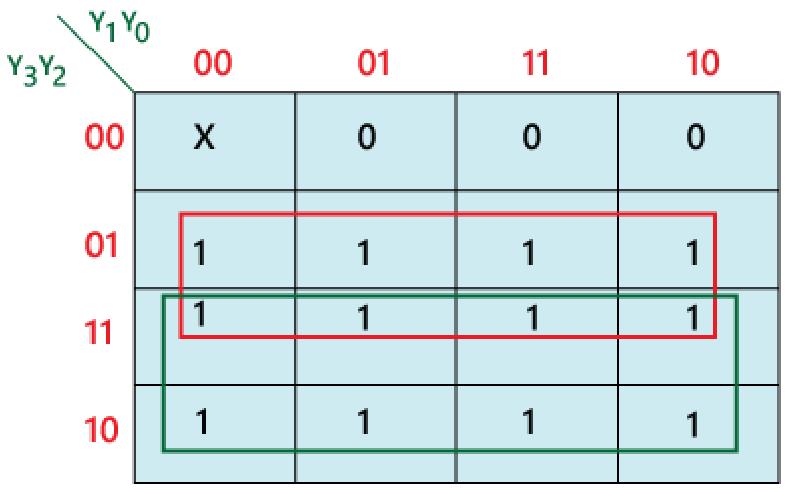
4 to 2 line Priority Encoder:

In this priority encoder, there are total of 4 inputs, i.e., Y_0, Y_1, Y_2, and Y_3, and two outputs, i.e., A_0 and A_1. The Y_3 has high and Y_0 has low priority inputs. When more than one input is '1' at the same time, the output will be the (binary) code corresponding to the higher priority input. Below is the truth table of the 4 to 2 line priority encoder.

Truth Table:

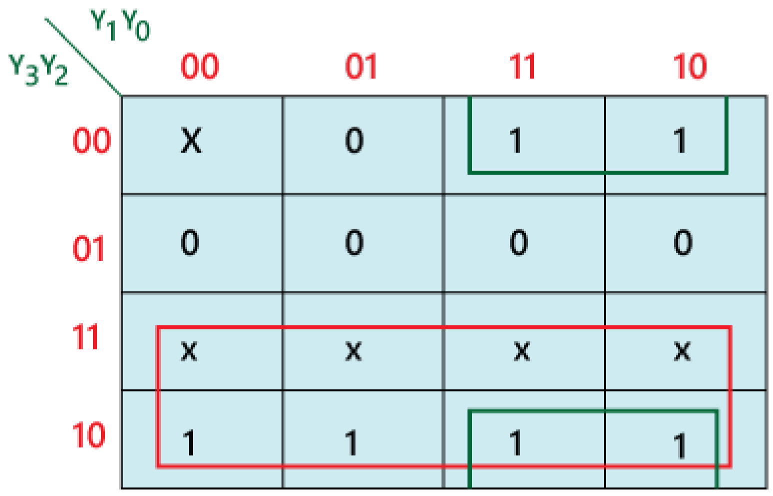
INPUTS				OUTPUTS		
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	X	x	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	x	X	1	0	1
1	X	x	X	1	1	1

The logical expression of the term A_0 and A_1 can be found using **K-map** as:



$Y_3Y_2 \backslash Y_1Y_0$	00	01	11	10
00	X	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$A_1=Y_3+Y_2$



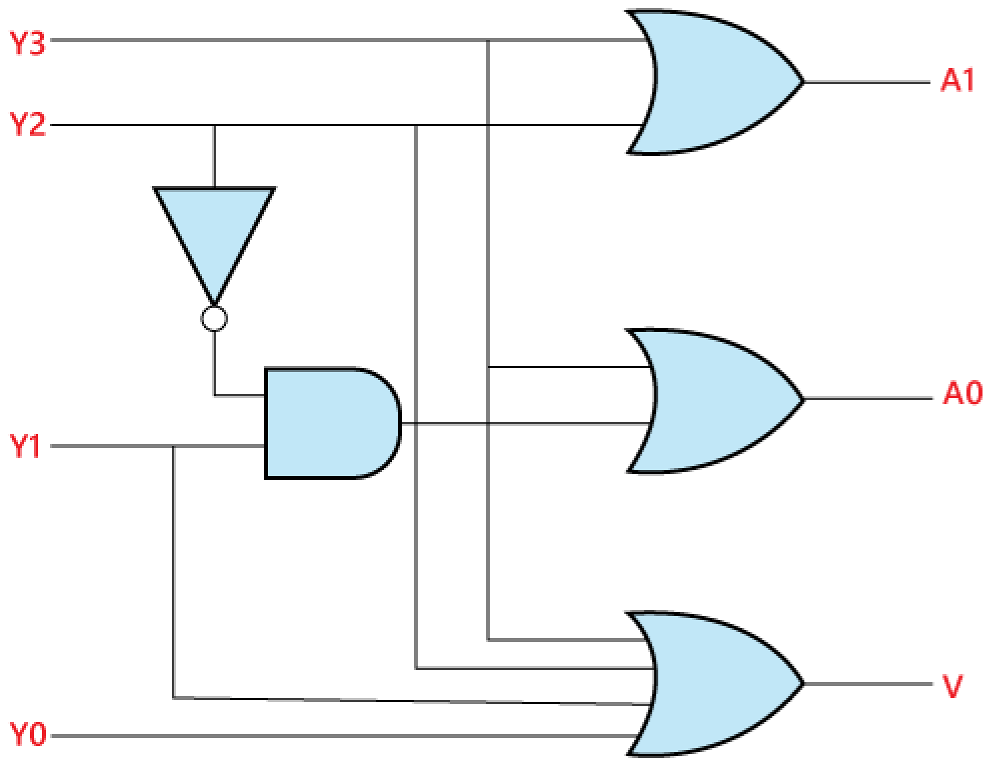
$Y_3Y_2 \backslash Y_1Y_0$	00	01	11	10
00	X	0	1	1
01	0	0	0	0
11	x	x	x	x
10	1	1	1	1

$A_0=Y_3+Y_2'Y_1$

$A_1=Y_3+Y_2$

$A_0=Y_3+Y_2'.Y_1$

Logical circuit of the above expressions is given below:



Uses of Encoders:

1. These systems are very easy to use in all digital systems.
2. Encoders are used to convert a decimal number into the binary number. The objective is to perform a binary operation such as addition, subtraction, multiplication, etc.

Multiplexer

A multiplexer is a combinational circuit that has 2^n input lines and a single output line. Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output.

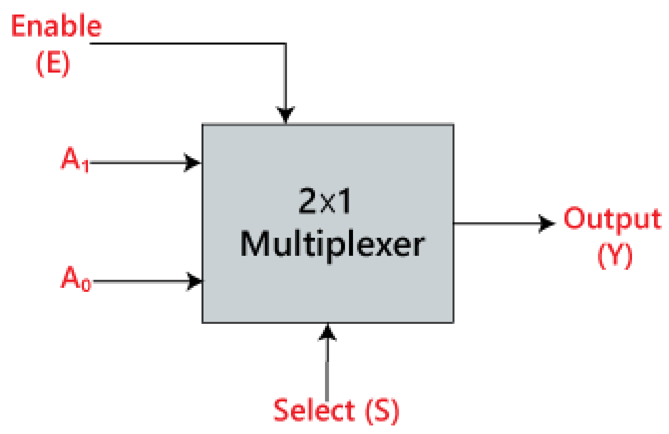
Unlike encoder and decoder, there are n selection lines and 2^n input lines. So, there is a total of 2^N possible combinations of inputs. A multiplexer is also treated as **Mux**.

There are various types of the multiplexer which are as follows:

2×1 Multiplexer:

In 2×1 multiplexer, there are only two inputs, i.e., A_0 and A_1, 1 selection line, i.e., S_0 and single outputs, i.e., Y. On the basis of the combination of inputs which are present at the selection line S^0, one of these 2 inputs will be connected to the output. The block diagram and the truth table of the 2×1 multiplexer are given below.

Block Diagram:



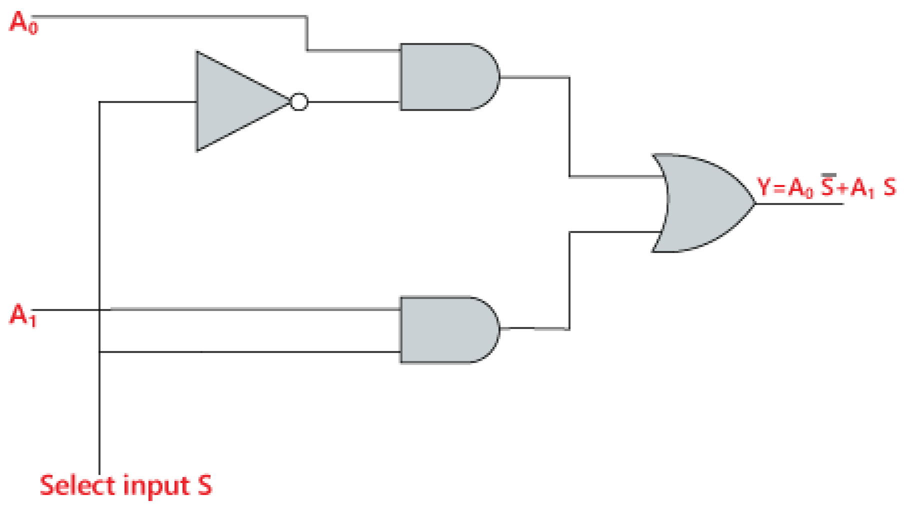
Truth Table:

INPUTS	Output
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

$Y=S_0'.A_0+S_0.A_1$

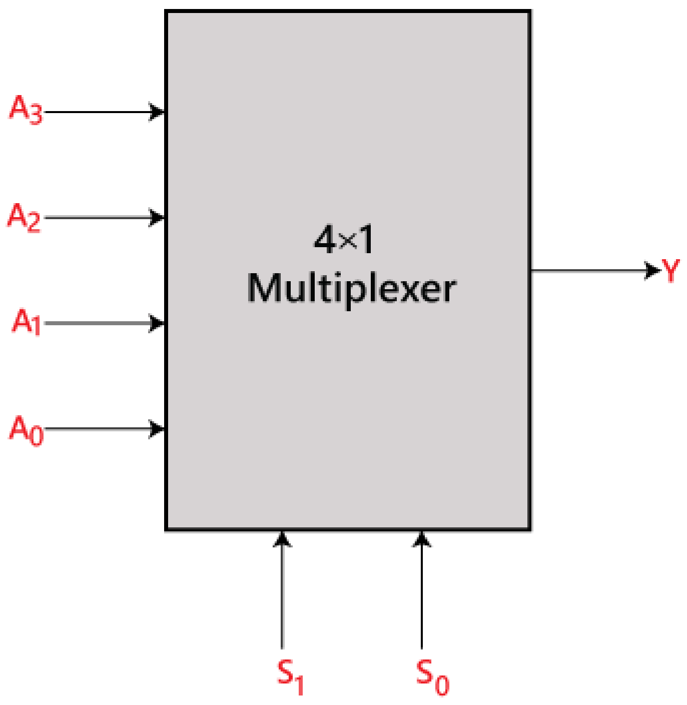
Logical circuit of the above expression is given below:



4×1 Multiplexer:

In the 4×1 multiplexer, there is a total of four inputs, i.e., A_0, A_1, A_2, and A_3, 2 selection lines, i.e., S_0 and S_1 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S^0 and S_1, one of these 4 inputs are connected to the output. The block diagram and the truth table of the 4×1 multiplexer are given below.

Block Diagram:



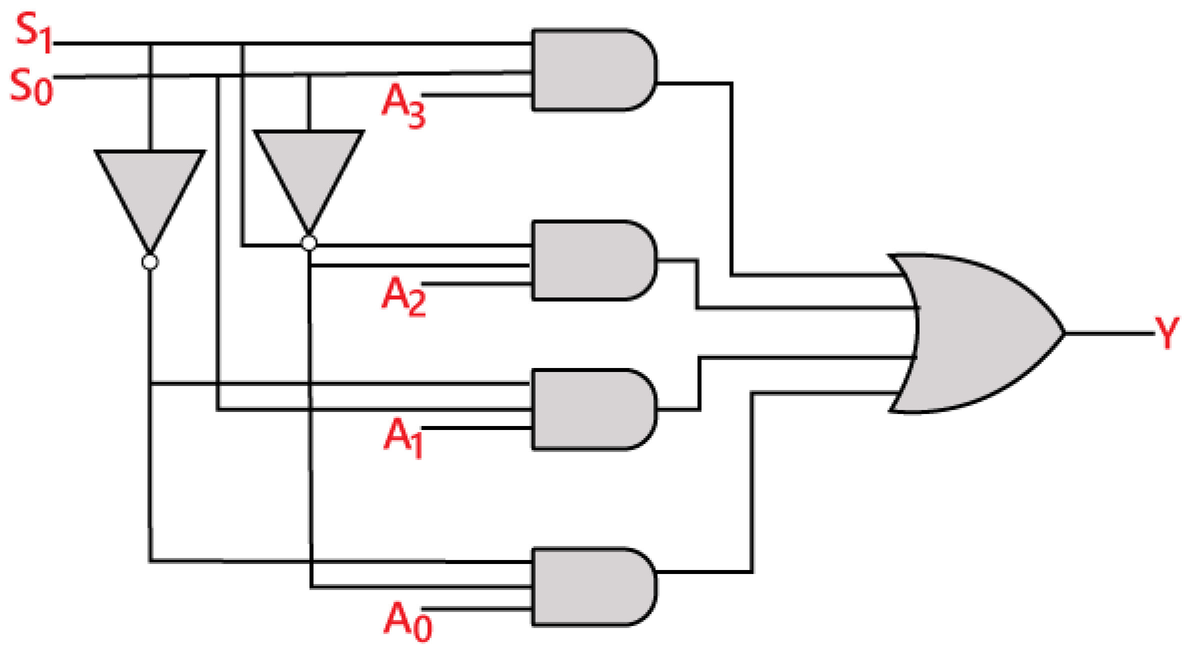
Truth Table:

INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

The logical expression of the term Y is as follows:

$Y=S_1' S_0' A_0+S_1' S_0 A_1+S_1 S_0' A_2+S_1 S_0 A_3$

Logical circuit of the above expression is given below:

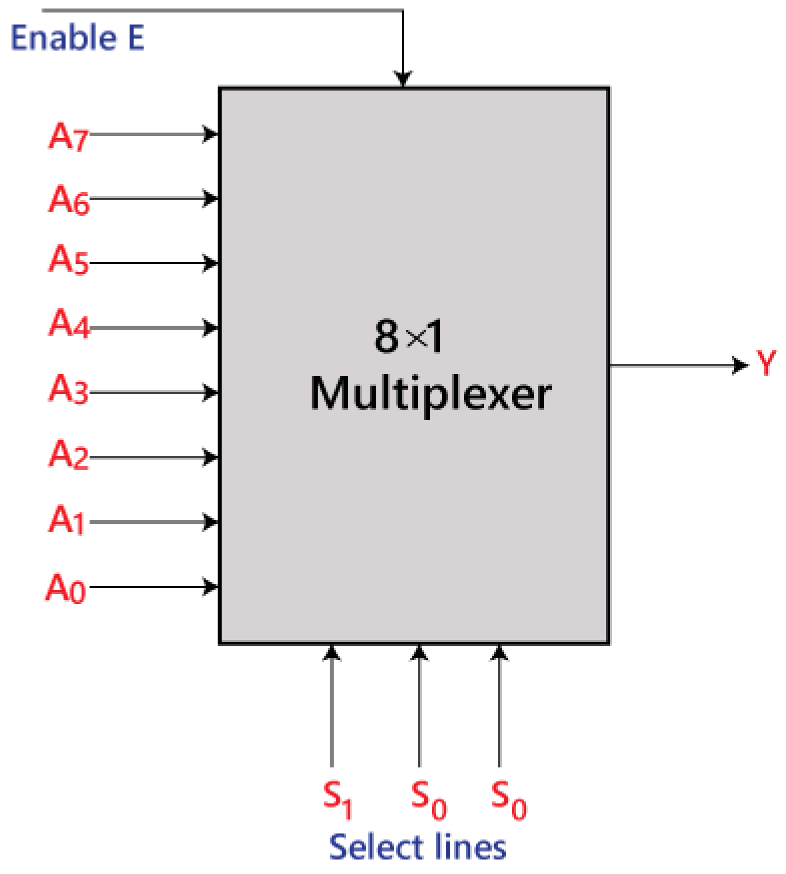


8 to 1 Multiplexer

In the 8 to 1 multiplexer, there are total eight inputs, i.e., A_0, A_1, A_2, A_3, A_4, A_5, A_6, and A_7, 3 selection lines, i.e., S_0, $S_1$and S_2 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S^0, $S^{1,}$ and S_2, one of these 8 inputs are connected to the output.

The block diagram and the truth table of the 8×1 multiplexer are given below.

Block Diagram:



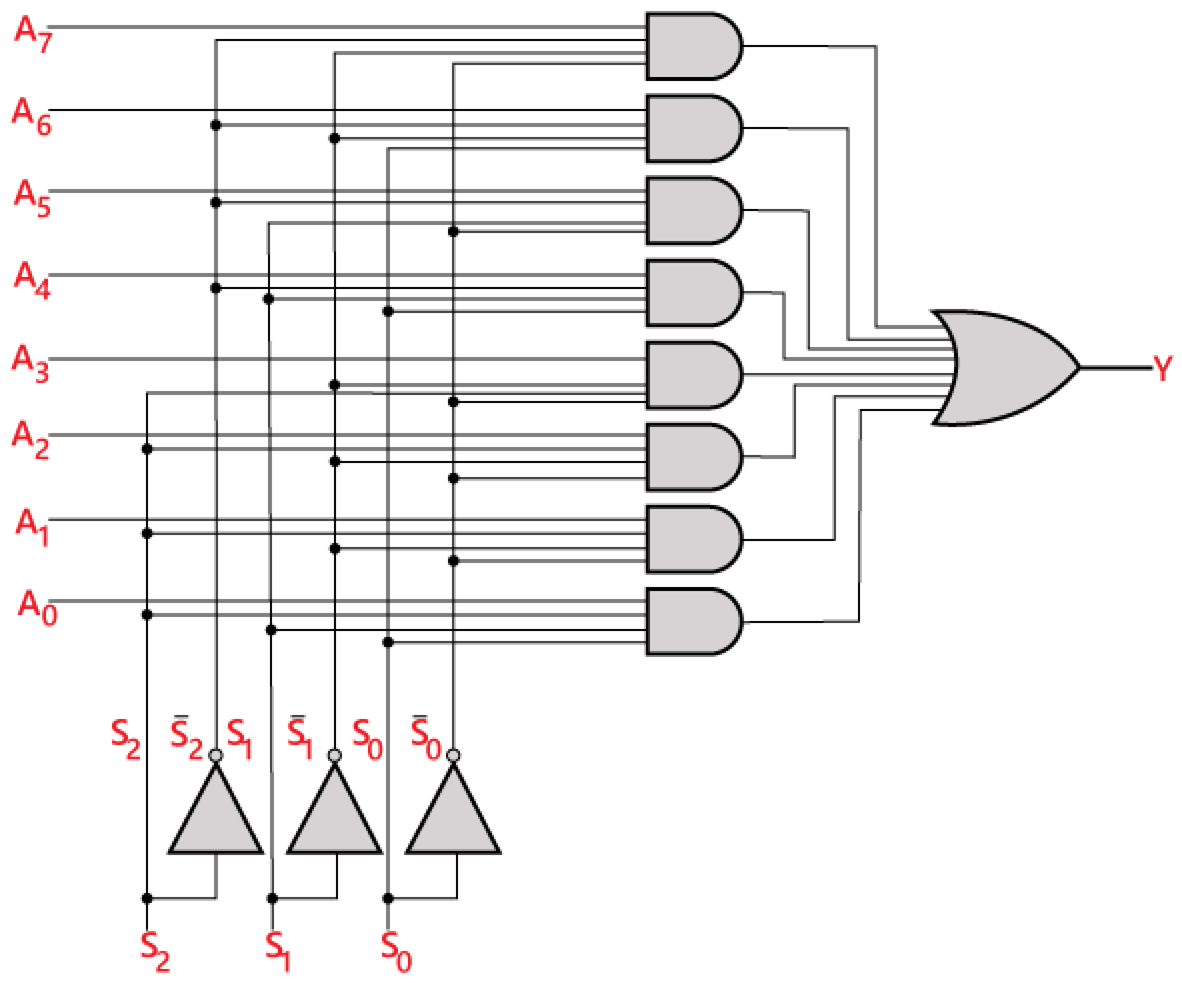
Truth Table:

INPUTS			Output
S_2	S_1	S_0	Y
0	0	0	A_0
0	0	1	A_1
0	1	0	A_2
0	1	1	A_3
1	0	0	A_4
1	0	1	A_5
1	1	0	A_6
1	1	1	A_7

The logical expression of the term Y is as follows:

$Y=S_0'.S_1'.S_2'.A_0+S_0.S_1'.S_2'.A_1+S_0'.S_1.S_2'.A_2+S_0.S_1.S_2'.A_3+S_0'.S_1'.S_2\ A_4+S_0.S_1'.S_2\ A_5+S_0'.S_1.S_2\ .A_6+S_0.S_1.S_3.A_7$

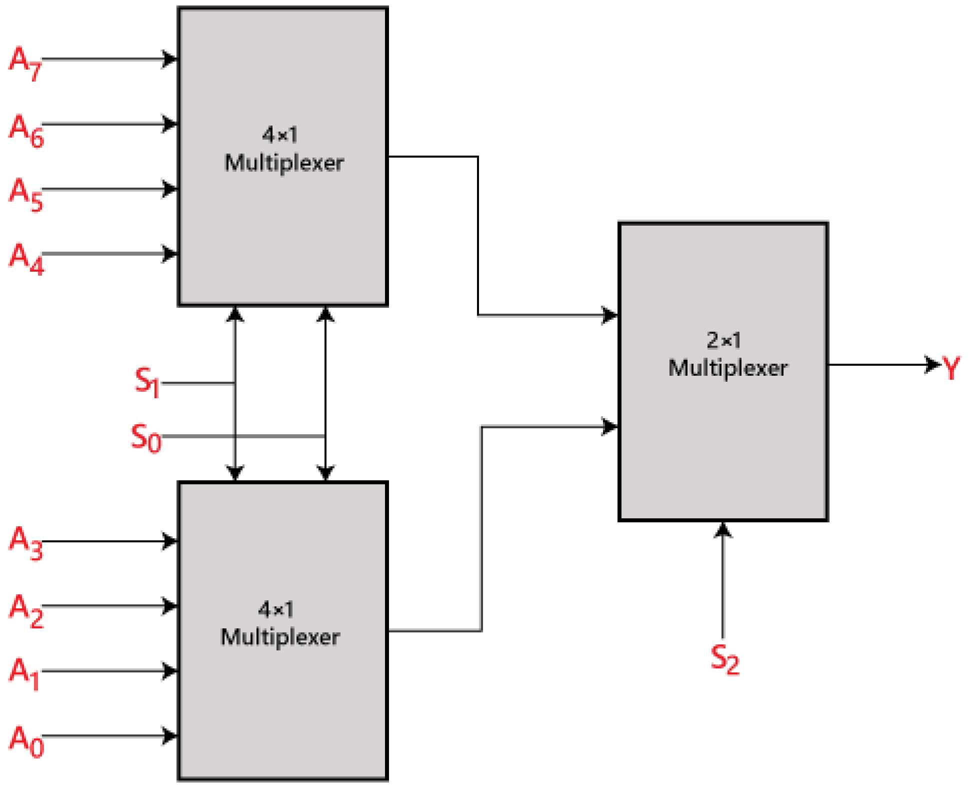
Logical circuit of the above expression is given below:



8 ×1 multiplexer using 4×1 and 2×1 multiplexer

We can implement the 8×1 multiplexer using a lower order multiplexer. To implement the 8×1 multiplexer, we need two 4×1 multiplexers and one 2×1 multiplexer. The 4×1 multiplexer has 2 selection lines, 4 inputs, and 1 output. The 2×1 multiplexer has only 1 selection line.

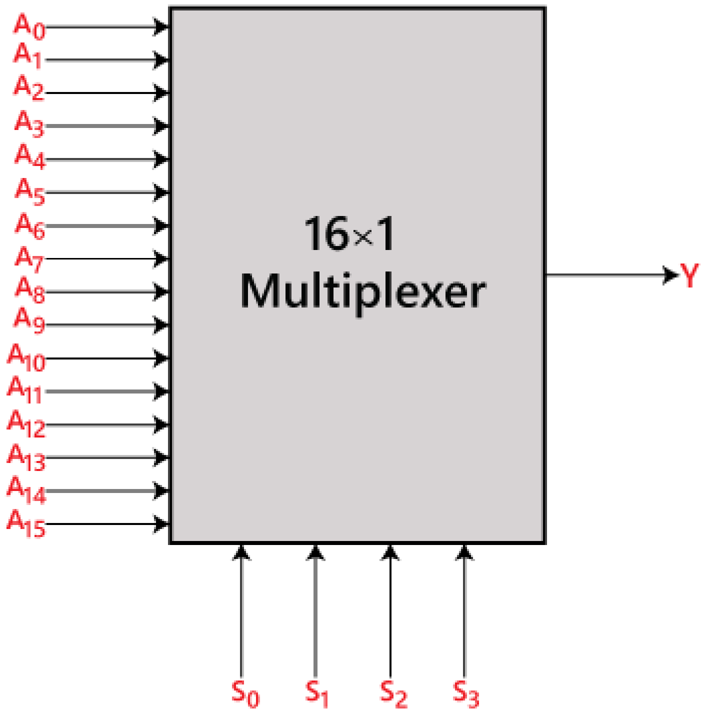
For getting 8 data inputs, we need two 4×1 multiplexers. The 4×1 multiplexer produces one output. So, in order to get the final output, we need a 2×1 multiplexer. The block diagram of 8×1 multiplexer using 4×1 and 2×1 multiplexer is given below.



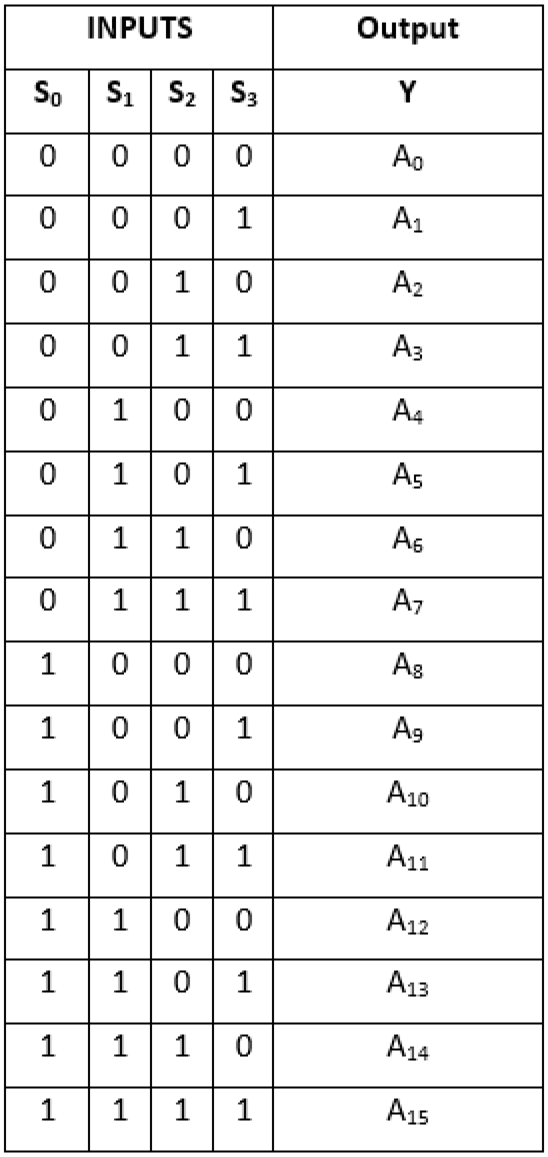
16 to 1 Multiplexer

In the 16 to 1 multiplexer, there are total of 16 inputs, i.e., A_0, A_1, ..., A_{16}, 4 selection lines, i.e., S_0, S_1, S_2, and S_3 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S^0, S^1, and S_2, one of these 16 inputs will be connected to the output. The block diagram and the truth table of the 16×1

Block Diagram:


A_0
A_1
A_2
A_3
A_4
A_5
A_6
A_7
A_8
A_9
A_{10}
A_{11}
A_{12}
A_{13}
A_{14}
A_{15}
16×1 Multiplexer
Y
S_0
S_1
S_2
S_3

Truth Table:



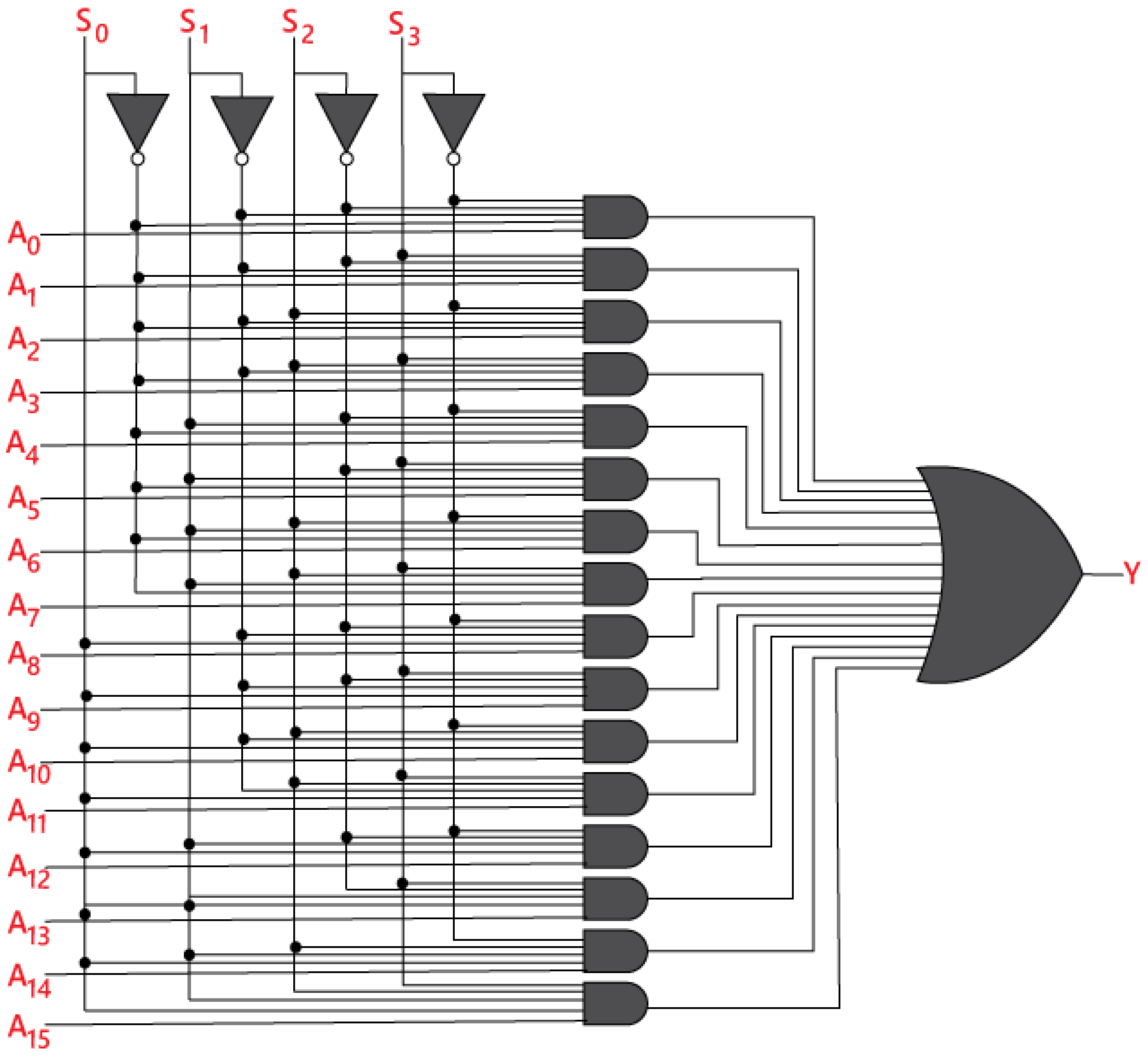
INPUTS				Output
S_0	S_1	S_2	S_3	Y
0	0	0	0	A_0
0	0	0	1	A_1
0	0	1	0	A_2
0	0	1	1	A_3
0	1	0	0	A_4
0	1	0	1	A_5
0	1	1	0	A_6
0	1	1	1	A_7
1	0	0	0	A_8
1	0	0	1	A_9
1	0	1	0	A_{10}
1	0	1	1	A_{11}
1	1	0	0	A_{12}
1	1	0	1	A_{13}
1	1	1	0	A_{14}
1	1	1	1	A_{15}

The logical expression of the term Y is as follows:

$Y=A_0.S_0'.S_1'.S_2'.S_3'+A_1.S_0'.S_1'.S_2'.S_3+A_2.S_0'.S_1'.S_2.S_3'+A_3.S_0'.S_1'.S_2.S_3+A_4.S_0'.S_1.S_2'.S_3'+A_5.S_0'.S_1.S_2'.S_3+A_6.S_1.S_2.S_3'+A_7.S_0'.S_1.S_2.S_3+A_8.S_0.S_1$

$'.S_2'.S_3'+A_9 .S_0.S_1'.S_2'.S_3+Y_10.S_0.S_1'.S_2.S_3 '+A_11$
$.S_0.S_1'.S_2.S_3+A_12$
$S_0.S_1.S_2 '.S_3'+A_13.S_0.S_1.S_2'.S_3+A_14.S_0.S_1 .S_2.S_3'$
$+A_15.S_0.S_1.S_2'.S_3$

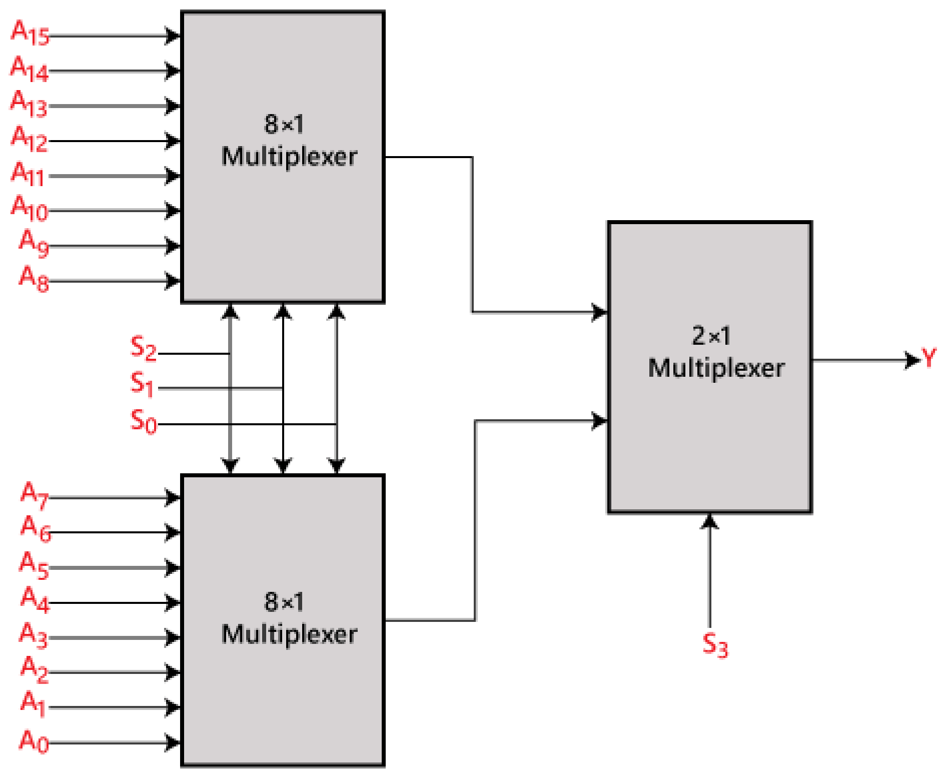
Logical circuit of the above expression is given below:



16×1 multiplexer using 8×1 and 2×1 multiplexer

We can implement the 16×1 multiplexer using a lower order multiplexer. To implement the 8×1 multiplexer, we need two 8×1 multiplexers and one 2×1 multiplexer. The 8×1 multiplexer has 3 selection lines, 4 inputs, and 1 output. The 2×1 multiplexer has only 1 selection line.

For getting 16 data inputs, we need two 8 ×1 multiplexers. The 8×1 multiplexer produces one output. So, in order to get the final output, we need a 2×1 multiplexer. The block diagram of 16×1 multiplexer using 8×1 and 2×1 multiplexer is given below.



De-multiplexer

A De-multiplexer is a combinational circuit that has only 1 input line and 2^N output lines. Simply, the multiplexer is a single-input and multi-output combinational circuit. The information is received from the single input lines and directed to the output line. On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

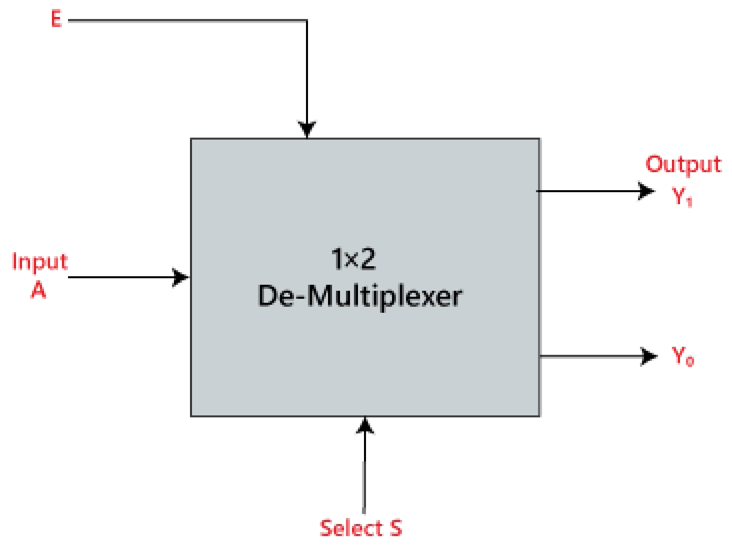
Unlike encoder and decoder, there are n selection lines and 2^n outputs. So, there is a total of 2^n possible combinations of inputs. De-multiplexer is also treated as **De-mux**.

There are various types of De-multiplexer which are as follows:

1×2 De-multiplexer:

In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0, and Y_1, 1 selection lines, i.e., S_0, and single input, i.e., A. On the basis of the selection value, the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 multiplexer are given below.

Block Diagram:



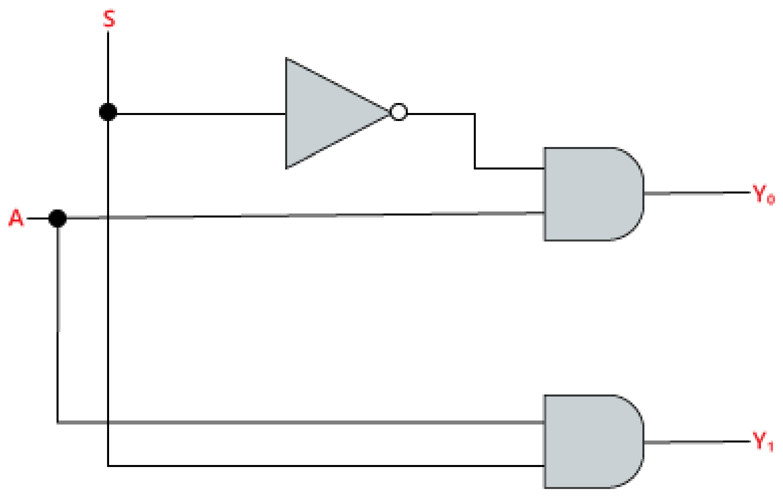
Truth Table:

INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

The logical expression of the term Y is as follows:

$Y_0=S_0'.A$
$Y_1=S_0.A$

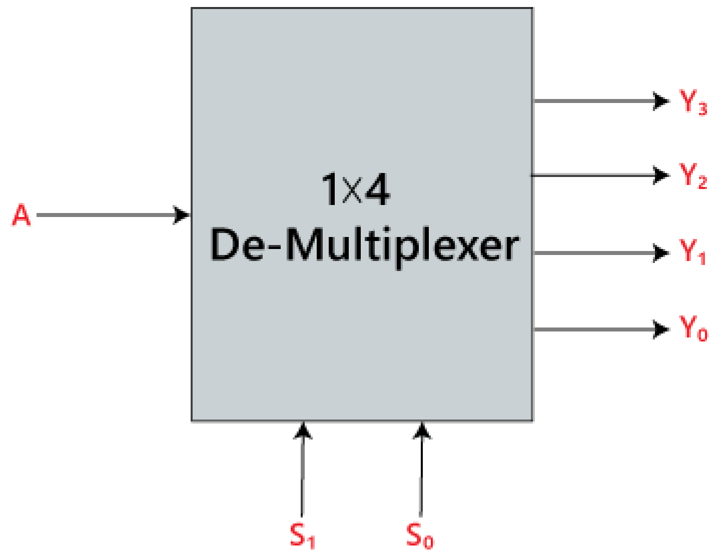
Logical circuit of the above expressions is given below:



1×4 De-multiplexer:

In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y_0, Y_1, Y_2, and Y_3, 2 selection lines, i.e., S_0 and S_1 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S_0 and S_1, the input be connected to one of the outputs. The block diagram and the truth table of the 1×4 multiplexer are given below.

Block Diagram:



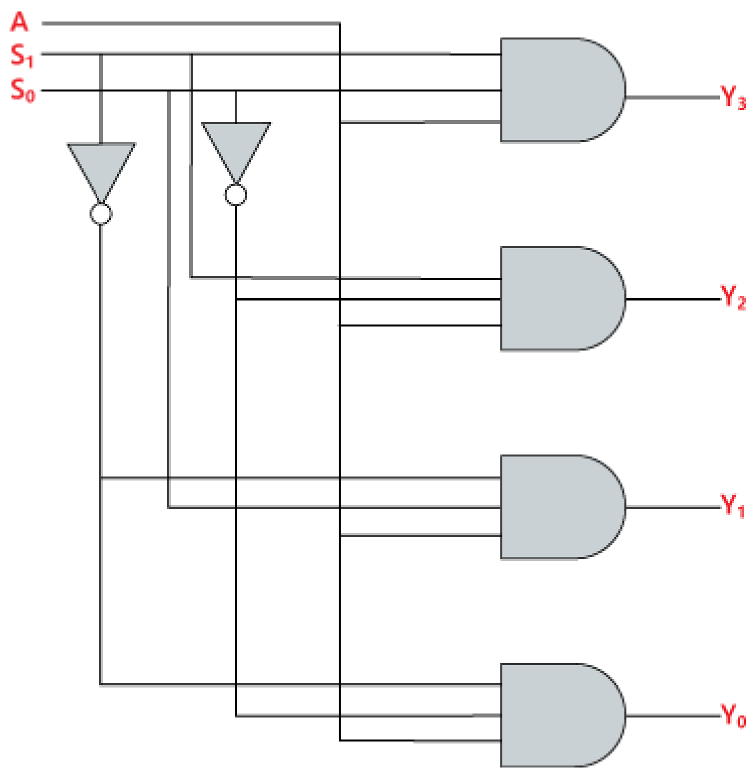
Truth Table:

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

$Y_0=S_1'$ S_0' A
$y_1=S_1'$ S_0 A
$y_2=S_1$ S_0' A
$y_3=S_1$ S_0 A

Logical circuit of the above expressions is given below:

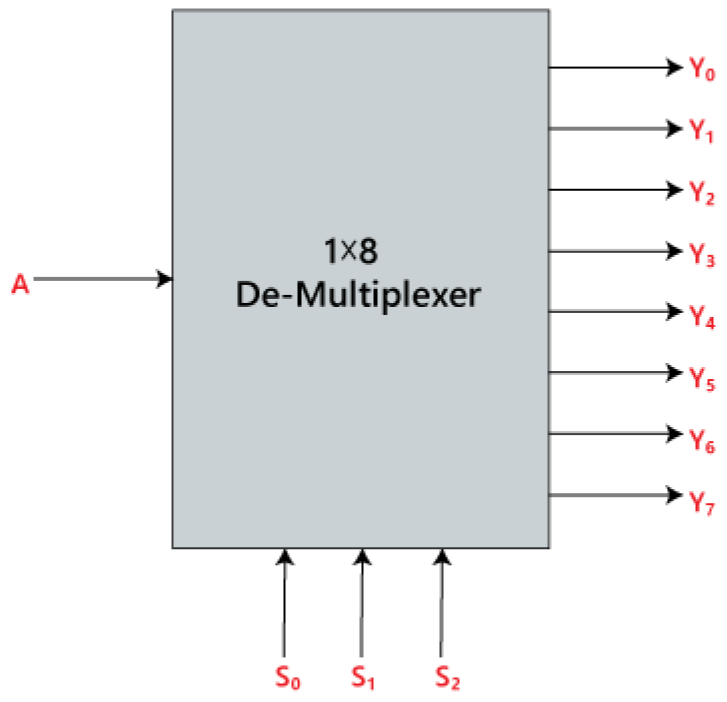


1×8 De-multiplexer

In 1 to 8 De-multiplexer, there are total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7, 3 selection lines, i.e., S_0, $S_1$and S_2 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S^0, S^1 and S_2, the input will be connected to one of these outputs.

The block diagram and the truth table of the 1×8 de-multiplexer are given below.

Block Diagram:



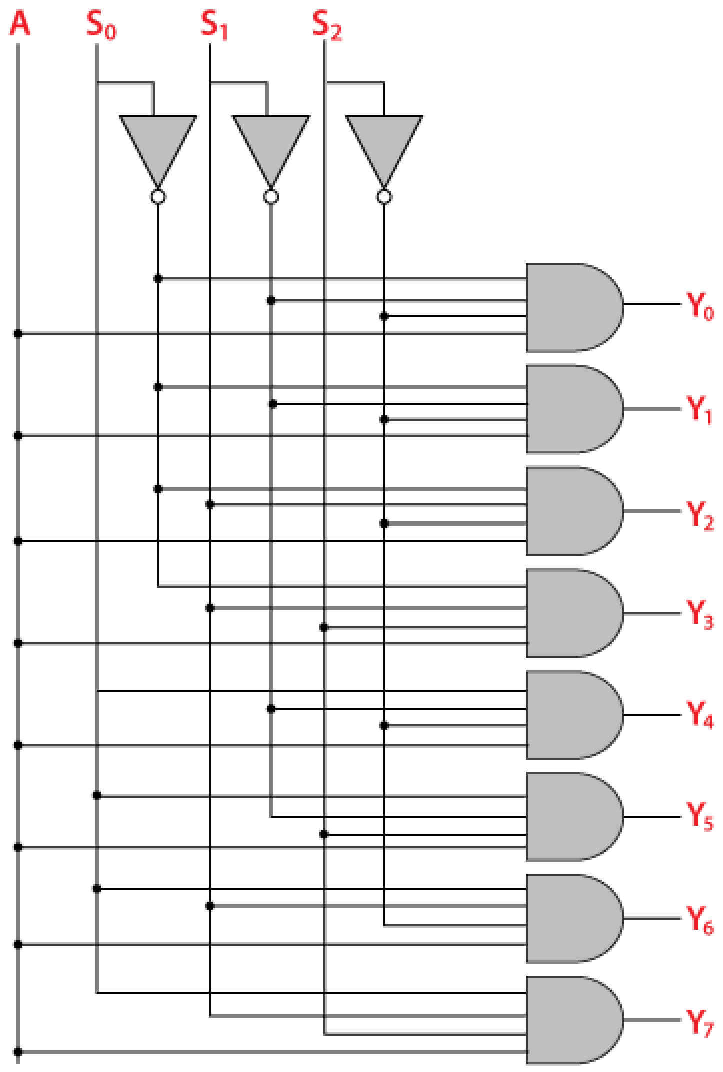
Truth Table:

INPUTS			Output							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

The logical expression of the term Y is as follows:

$Y_0=S_0'.S_1'.S_2'.A$
$Y_1=S_0.S_1'.S_2'.A$
$Y_2=S_0'.S_1.S_2'.A$
$Y_3=S_0.S_1.S_2'.A$
$Y_4=S_0'.S_1'.S_2\ A$
$Y_5=S_0.S_1'.S_2\ A$
$Y_6=S_0'.S_1.S_2\ A$
$Y_7=S_0.S_1.S_3.A$

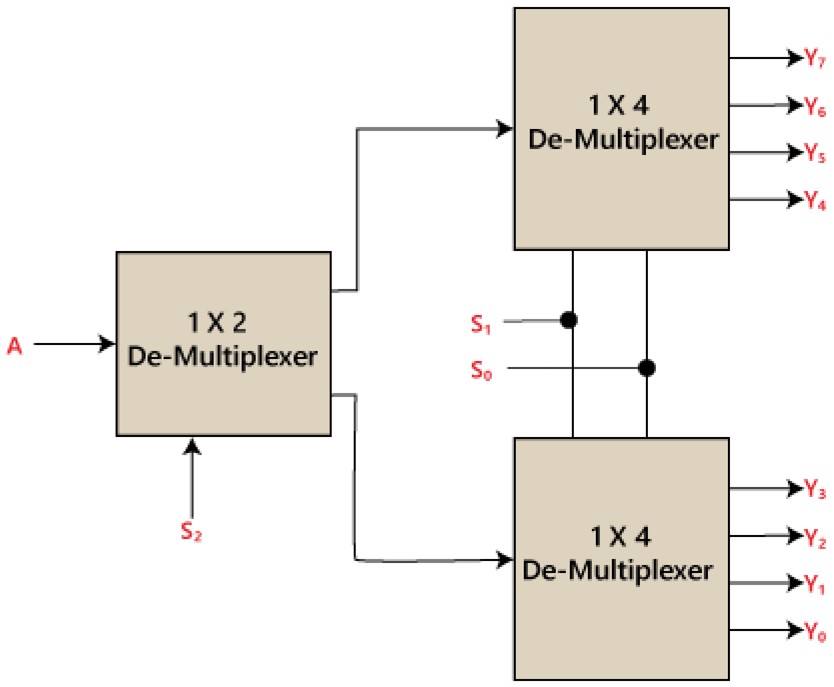
Logical circuit of the above expressions is given below:



1×8 De-multiplexer using 1×4 and 1×2 de-multiplexer

We can implement the 1×8 de-multiplexer using a lower order de-multiplexer. To implement the 1×8 de-multiplexer, we need two 1×4 de-multiplexer and one 1×2 de-multiplexer. The 1×4 multiplexer has 2 selection lines, 4 outputs, and 1 input. The 1×2 de-multiplexer has only 1 selection line.

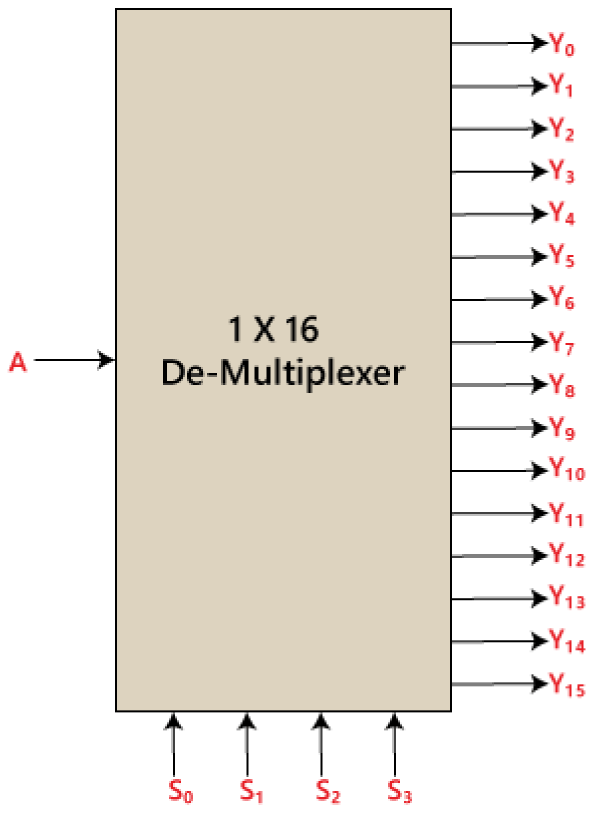
For getting 8 data outputs, we need two 1×4 de-multiplexer. The 1×2 de-multiplexer produces two outputs. So, in order to get the final output, we have to pass the outputs of 1×2 de-multiplexer as an input of both the 1×4 de-multiplexer. The block diagram of 1×8 de-multiplexer using 1×4 and 1×2 de-multiplexer is given below.



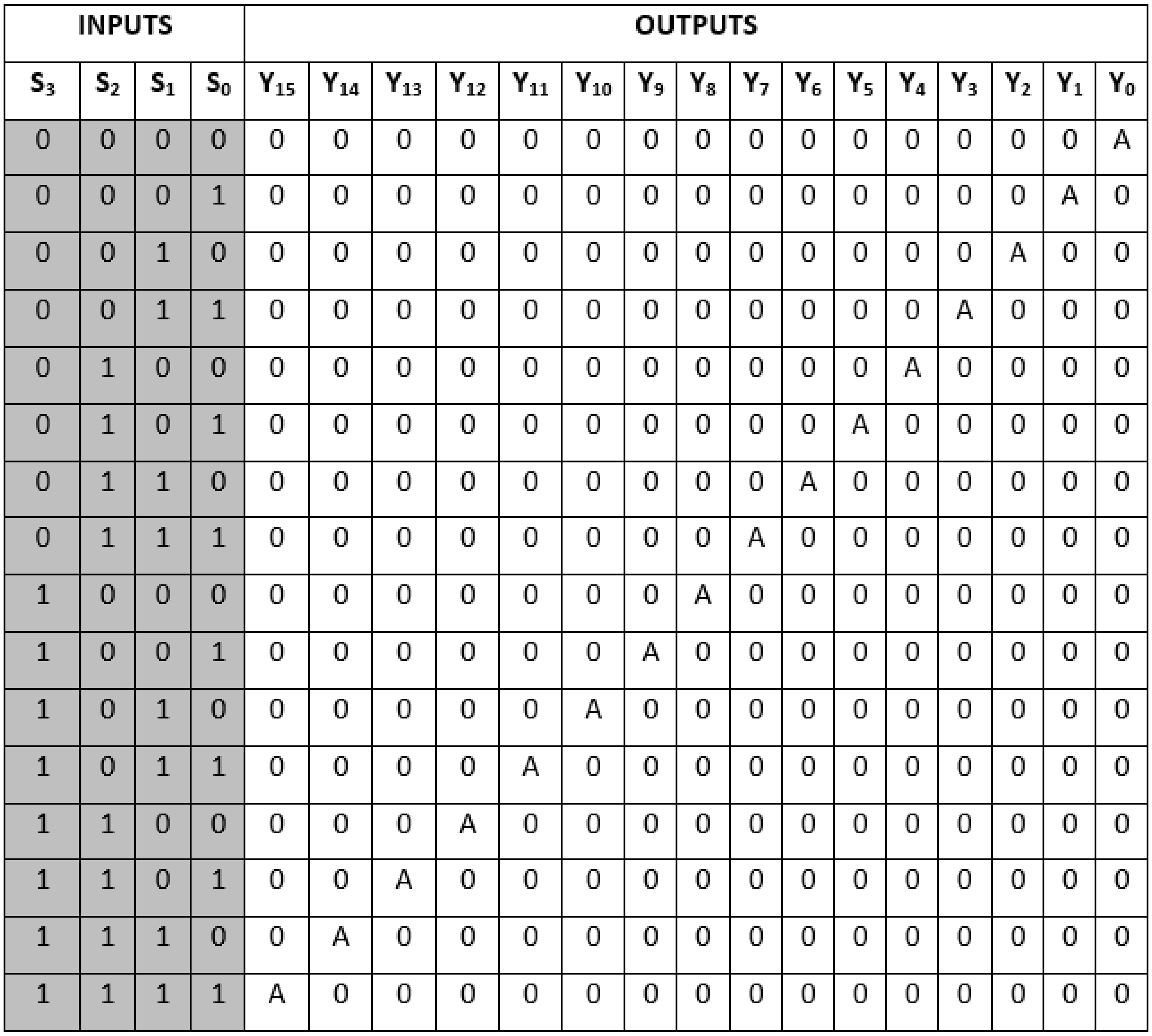
1 x 16 De-multiplexer

In 1×16 de-multiplexer, there are total of 16 outputs, i.e., Y_0, Y_1, ..., Y_{16}, 4 selection lines, i.e., S_0, S_1, S_2, and S_3 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S^0, S^1, and S_2, the input will be connected to one of these outputs. The block diagram and the truth table of the 1×16 de-multiplexer are given below.

Block Diagram:



Truth Table:



INPUTS				OUTPUTS															
S_3	S_2	S_1	S_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The logical expression of the term Y is as follows:

Y_0=A.S_0'.S_1'.S_2'.S_3'
Y_1=A.S_0'.S_1'.S_2'.S_3
Y_2=A.S_0'.S_1'.S_2.S_3'

$Y_3=A.S_0'.S_1'.S_2.S_3$

$Y_4=A.S_0'.S_1.S_2'.S_3'$

$Y_5=A.S_0'.S_1.S_2'.S_3$

$Y_6=A.S_0'.S_1.S_2.S_3'$

$Y_7=A.S_0'.S_1.S_2.S_3$

$Y_8=A.S_0.S_1'.S_2'.S_3'$

$Y_9=A.S_0.S_1'.S_2'.S_3$

$Y_{10}=A.S_0.S_1'.S_2.S_3'$

$Y_{11}=A.S_0.S_1'.S_2.S_3$

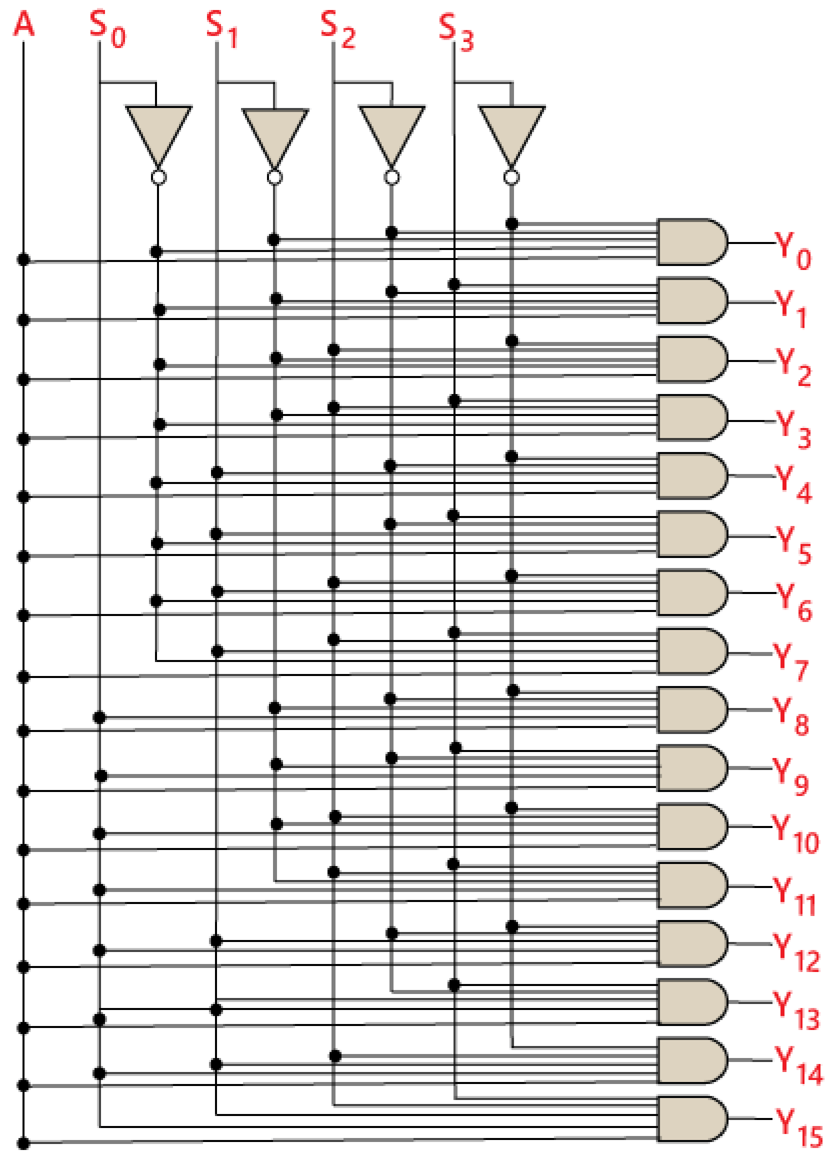
$Y_{12}=A.S_0.S_1.S_2'.S_3'$

$Y_{13}=A.S_0.S_1.S_2'.S_3$

$Y_{14}=A.S_0.S_1.S_2.S_3'$

$Y_{15}=A.S_0.S_1.S_2'.S_3$

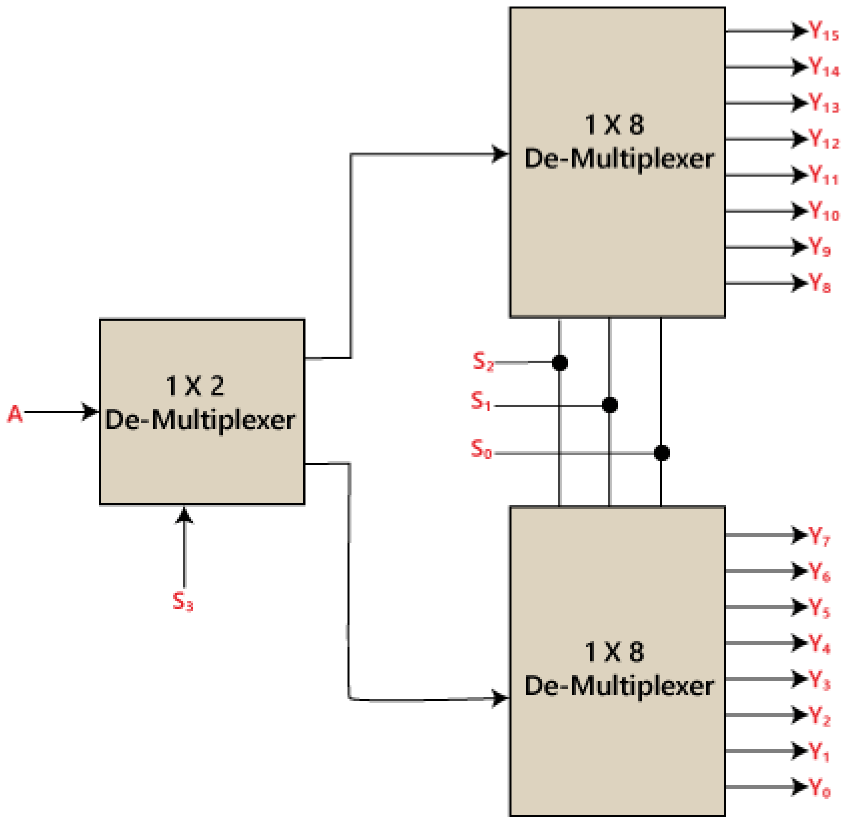
Logical circuit of the above expressions is given below:



1×16 de-multiplexer using 1×8 and 1×2 de-multiplexer

We can implement the 1×16 de-multiplexer using a lower order de-multiplexer. To implement the 1×16 de-multiplexer, we need two 1×8 de-multiplexer and one 1×2 de-multiplexer. The 1×8 multiplexer has 3 selection lines, 1 input, and 8 outputs. The 1×2 de-multiplexer has only 1 selection line.

For getting 16 data outputs, we need two 1×8 de-multiplexer. The 1×8 de-multiplexer produces eight outputs. So, in order to get the final output, we need a 1×2 de-multiplexer to produce two outputs from a single input. Then we pass these outputs into both the de-multiplexer as an input. The block diagram of 1×16 de-multiplexer using 1×8 and 1×2 de-multiplexer is given below.



Registers

A **Register** is a collection of flip flops. A flip flop is used to store single bit digital data. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops. If we want to store an n-bit word, we have to use an n-bit register containing n number of flip flops.

The register is used to perform different types of operations. For performing the operations, the CPU use these registers. The faded inputs to the system will store into the registers. The result returned by the system will store in the registers. There are the following operations which are performed by the registers:

Fetch:

It is used

- To take the instructions given by the users.
- To fetch the instruction stored into the main memory.

Decode:

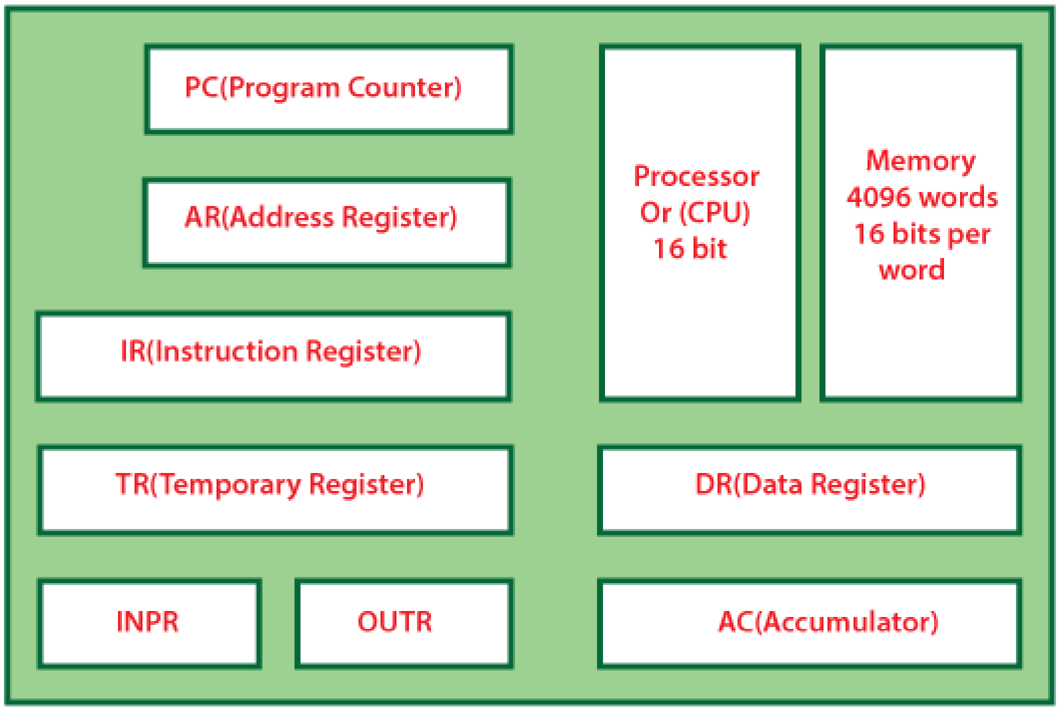
The decode operation is used to interpret the instructions. In decode, the operation performed on the instructions is identified by the CPU. In simple words, the decode operation is used to decode the instructions.

Execute:

The execution operation is used to store the result produced by the CPU into the memory. After storing this result, it is displayed on the user screen.

Types of Registers

There are various types of registers which are as follows:



MAR or Memory Address Register

The MAR is a special type of register that contains the memory address of the data and instruction. The main task of the MAR is to access instruction and data from memory in the execution phase. The MAR stores the address of the memory location where the data is to be read or to be stored by the CPU.

Program Counter

The program counter is also called an instruction address register or instruction pointer. The next memory address of the instruction, which is going to be executed after completing the execution of current instruction is contained in the program counter. In simple words, the program counter contains the memory address of the location of the next instruction.

Accumulator Register

The CPU mostly uses an accumulator register. The accumulator register is used to store the system result. All the results will be stored in the accumulator register when the CPU produces some results after processing.

MDR or Memory Data Register

Memory Data Register is a part of the computer's control unit. It contains the data that we want to store in the computer storage or the data fetched from the computer storage. The MDR works as a buffer that contains anything for which the processor is ready to use it. The MDR contains the copied data of the memory for the processor. Firstly the MDR holds the information, and then it goes to the decoder.

The data which is to be read out or written into the address location is contained in the **Memory Data Register**.

The data is written in one direction when it is fetched from memory and placed into the MDR. In write instruction, the data place into the MDR from another CPU register. This CPU register writes the data into the memory. Half of the minimal interface between the computer storage and the microprogram is the memory data address register, and the other half is the memory data register.

Index Register

The **Index Register** is the hardware element that holds the number. The number adds to the computer instruction's address to create an effective address. In CPU, the index register is a processor register used to modify the operand address during the running program.

Memory Buffer Register

Memory Buffer Register is mostly called MBR. The MBR contains the Metadata of the data and instruction written in or read from memory. In simple words, it adds is used to store the upcoming data/instruction from the memory and going to memory.

Data Register

The data register is used to temporarily store the data. This data transmits to or from a peripheral device.

Shift Register

A group of flip flops which is used to store multiple bits of data and the data is moved from one flip flop to another is known as **Shift Register**. The bits stored in registers shifted when the clock pulse is applied within and inside or outside the registers. To form an n-bit shift register, we have to connect n number of flip flops. So, the number of bits of the binary number is directly proportional to the number of flip flops. The flip flops are connected in such a way that the first flip flop's output becomes the input of the other flip flop.

A **Shift Register** can shift the bits either to the left or to the right. A **Shift Register**, which shifts the bit to the left, is known as **"Shift left register"**, and it shifts the bit to the right, known as **"Right left register"**.

The shift register is classified into the following types:

- Serial In Serial Out
- Serial In Parallel Out
- Parallel In Serial Out
- Parallel In Parallel Out
- Bi-directional Shift Register

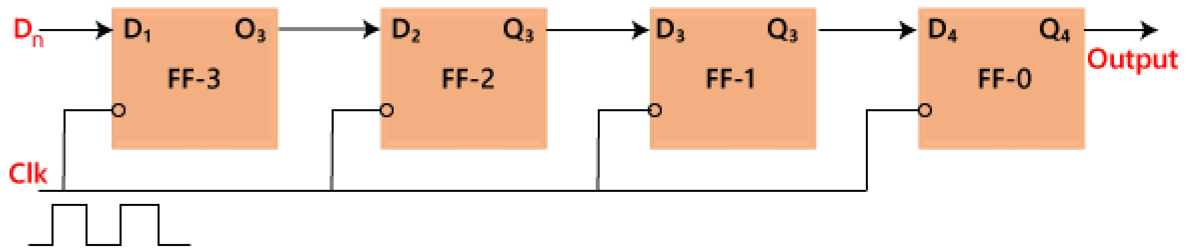
- Universal Shift Register

Serial IN Serial OUT

In "Serial Input Serial Output", the data is shifted "IN" or "OUT" serially. In SISO, a single bit is shifted at a time in either right or left direction under clock control.

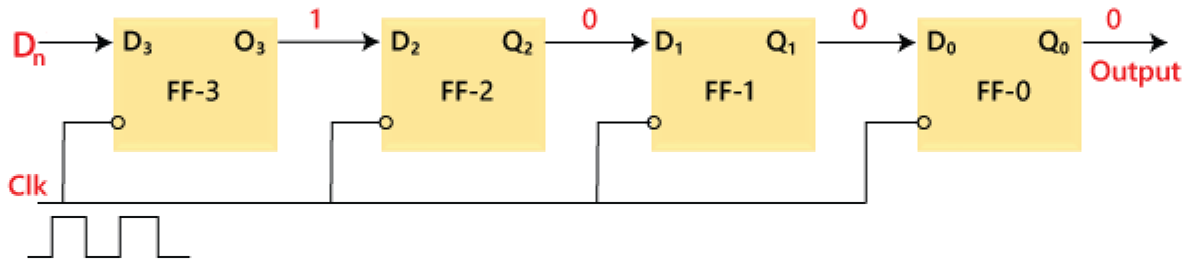
Initially, all the flip-flops are set in "reset" condition i.e. $Y_3 = Y_2 = Y_1 = Y_0 = 0$. If we pass the binary number 1111, the LSB bit of the number is applied first to the Din bit. The D3 input of the third flip flop, i.e., FF-3, is directly connected to the serial data input D3. The output Y_3 is passed to the data input d_2 of the next flip flop. This process remains the same for the remaining flip flops. The block diagram of the **"Serial IN Serial OUT"** is given below.

Block Diagram:

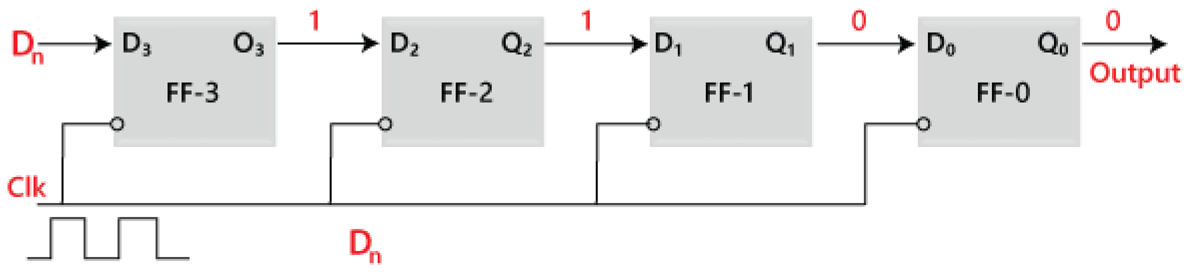


Operation

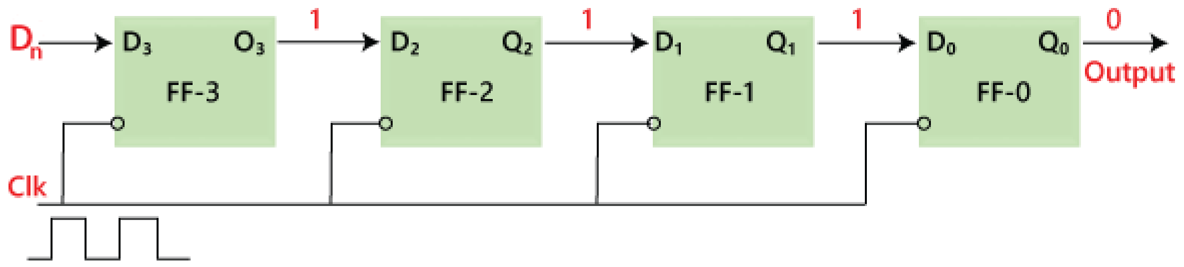
When the clock signal application is disabled, the outputs Y_3 Y_2 Y_1 Y_0 = 0000. The LSB bit of the number is passed to the data input D_{in}, i.e., D_3. We will apply the clock, and this time the value of D_3 is 1. The first flip flop, i.e., FF-3, is set, and the word is stored in the register at the first falling edge of the clock. Now, the stored word is 1000.



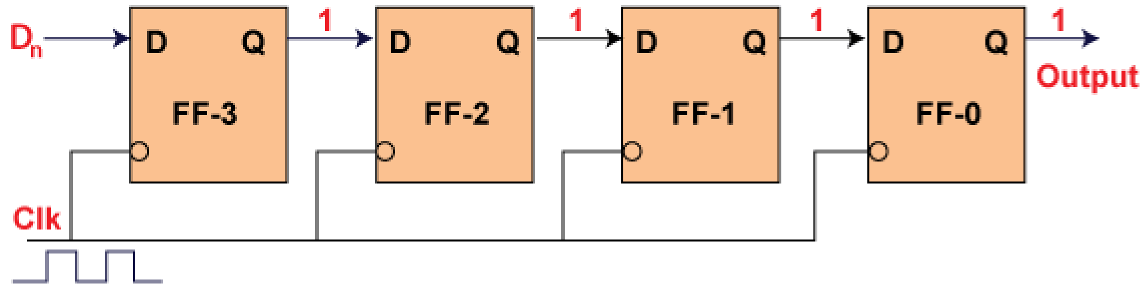
The next bit of the binary number, i.e., 1, is passed to the data input D_2. The second flip flop, i.e., FF-2, is set, and the word is stored when the next negative edge of the clock hits. The stored word is changed to 1100.



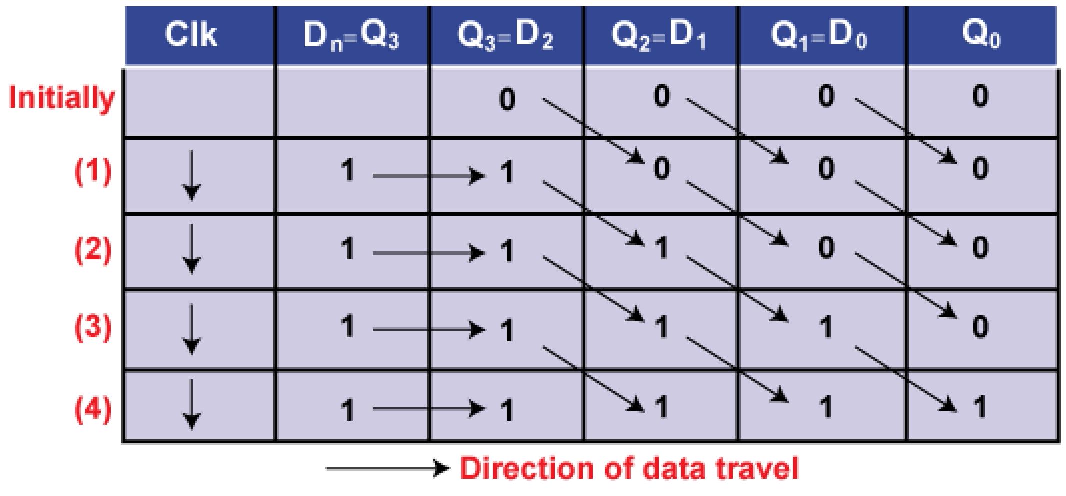
The next bit of the binary number, i.e., 1, is passed to the data input D_1, and the clock is applied. The third flip flop, i.e., FF-1, is set, and the word is stored when the negative edge of the clock hits again. The stored word is changed to 1110.



Similarly, the last bit of the binary number, i.e., 1, is passed to the data input D0, and the clock is applied. The last flip flop, i.e., FF-0, is set, and the word is stored when the clock's negative edge arrives. The stored word is changed to 1111.



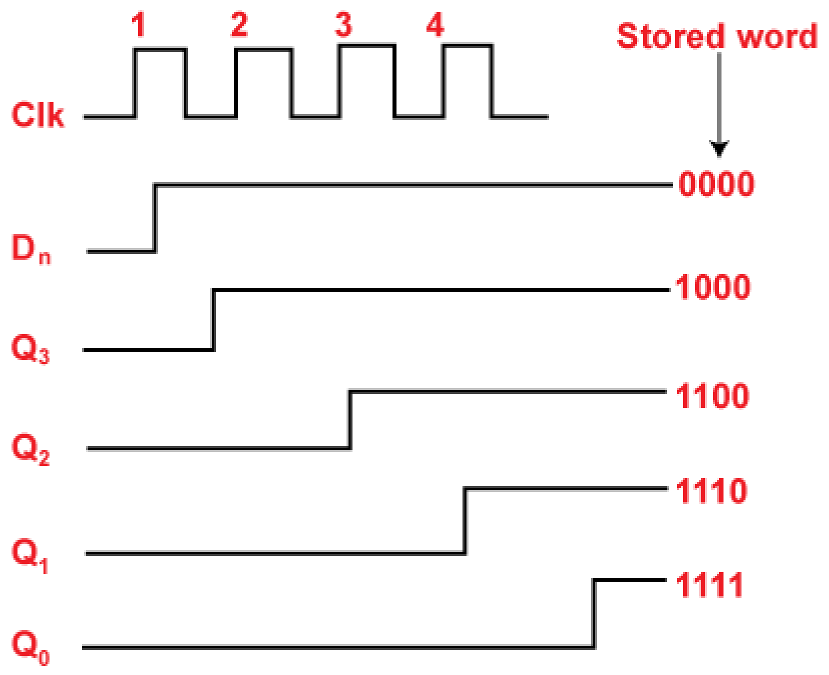
Truth Table



	Clk	$D_n = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(1)	↓	1	1	0	0	0
(2)	↓	1	1	1	0	0
(3)	↓	1	1	1	1	0
(4)	↓	1	1	1	1	1

$\longrightarrow$ Direction of data travel

Waveforms

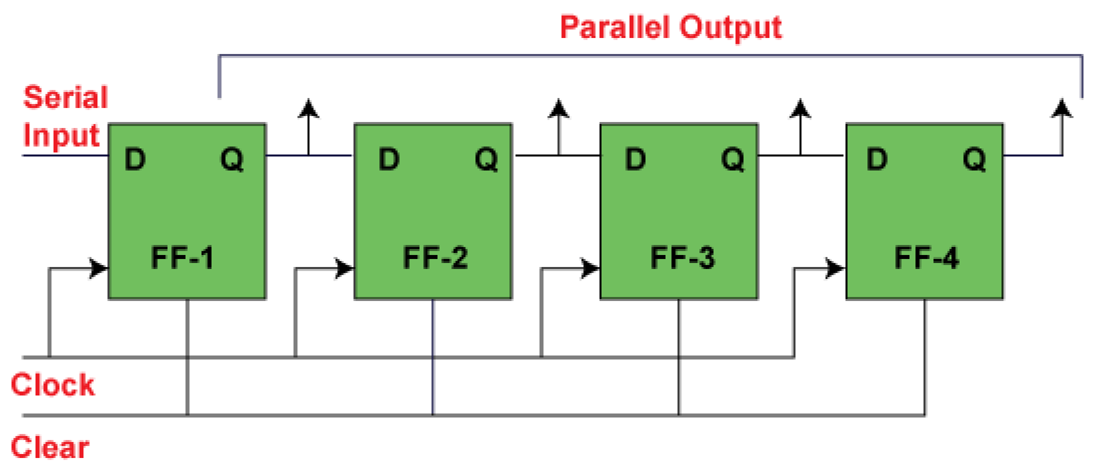


Serial IN Parallel OUT

In the **"Serial IN Parallel OUT"** shift register, the data is passed serially to the flip flop, and outputs are fetched in a parallel way. The data is passed bit by bit in the register, and the output remains disabled until the data is not passed to the data input. When the data is passed to the register, the outputs are enabled, and the flip flops contain their return value

Below is the block diagram of the 4-bit **serial in the parallel-out** shift register. The circuit having four D flip-flops contains a clear and clock signal to reset these four flip flops. In **SIPO**, the input of the second flip flop is the output of the first flip flop, and so on. The same clock signal is applied to each flip flop since the flip flops synchronize each other. The parallel outputs are used for communication.

Block Diagram



Parallel IN Serial OUT

In the **"Parallel IN Serial OUT"** register, the data is entered in a parallel way, and the outcome comes serially. A four-bit **"Parallel IN Serial OUT"** register is designed below. The input of the flip flop is the output of the previous Flip Flop. The input and outputs are connected through the combinational circuit. Through this combinational circuit, the binary input B_0, B_1, B_2, B_3 are passed. The **shift mode** and the **load mode** are the two modes in which the **"PISO"** circuit works.

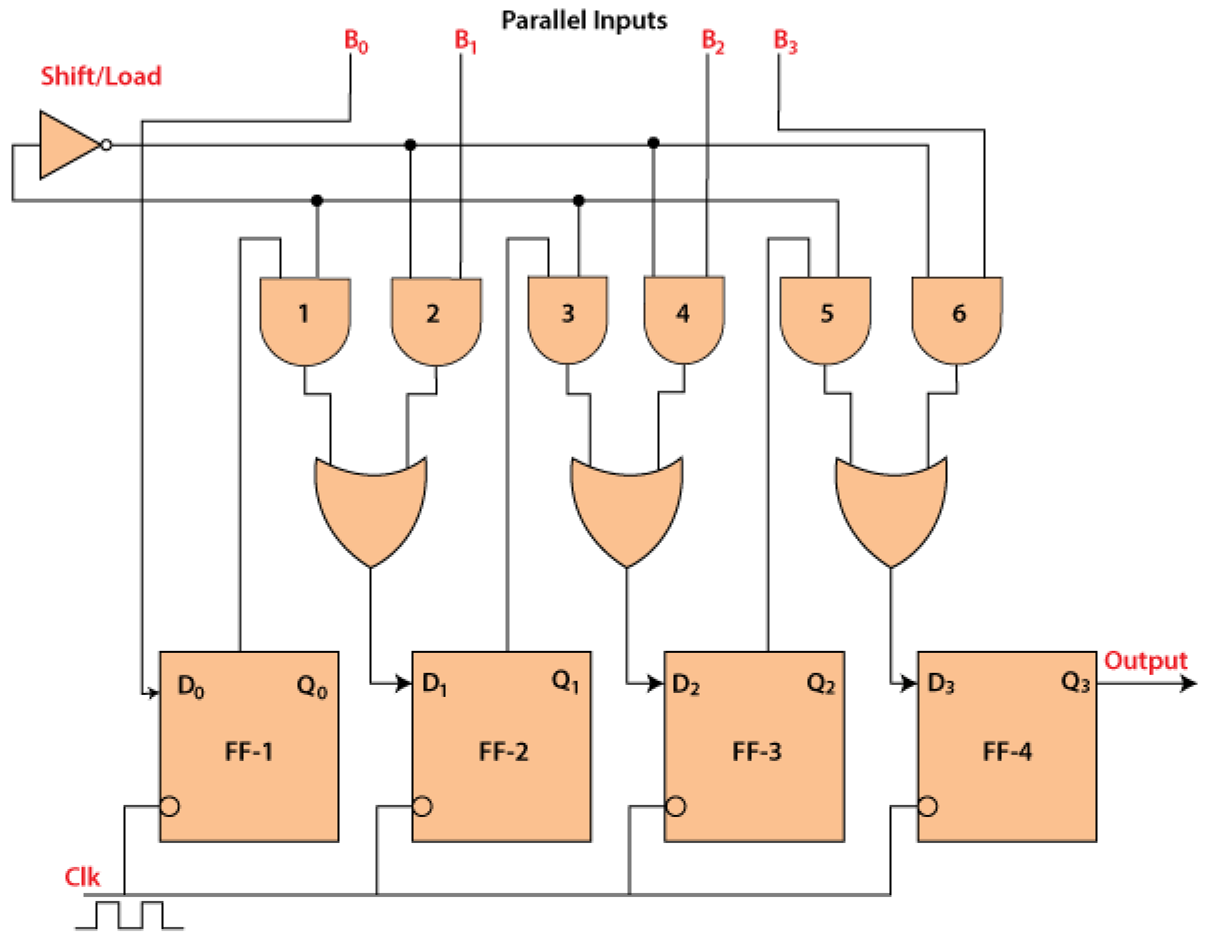
Load mode

The bits B_0, B_1, B_2, and B_3 are passed to the corresponding flip flops when the second, fourth, and sixth "AND" gates are active. These gates are active when the shift or load bar line set to 0. The binary inputs B0, B1, B2, and B3 will be loaded into the respective flip-flops when the edge of the clock is low. Thus, parallel loading occurs.

Shift mode

The second, fourth, and sixth gates are inactive when the load and shift line set to 0. So, we are not able to load data in a parallel way. At this time, the first, third, and fifth gates will be activated, and the shifting of the data will be left to the right bit. In this way, the **"Parallel IN Serial OUT"** operation occurs.

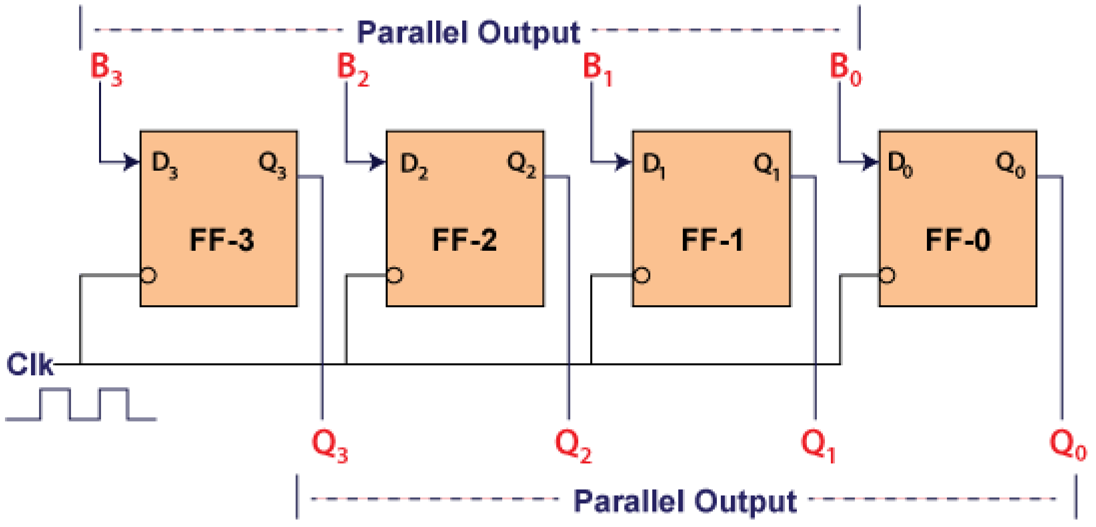
Block Diagram



Parallel IN Parallel OUT

In **"Parallel IN Parallel OUT"**, the inputs and the outputs come in a parallel way in the register. The inputs A_0, A_1, A_2, and A_3, are directly passed to the data inputs D_0, D_1, D_2, and D_3 of the respective flip flop. The bits of the binary input is loaded to the flip flops when the negative clock edge is applied. The clock pulse is required for loading all the bits. At the output side, the loaded bits appear.

Block Diagram



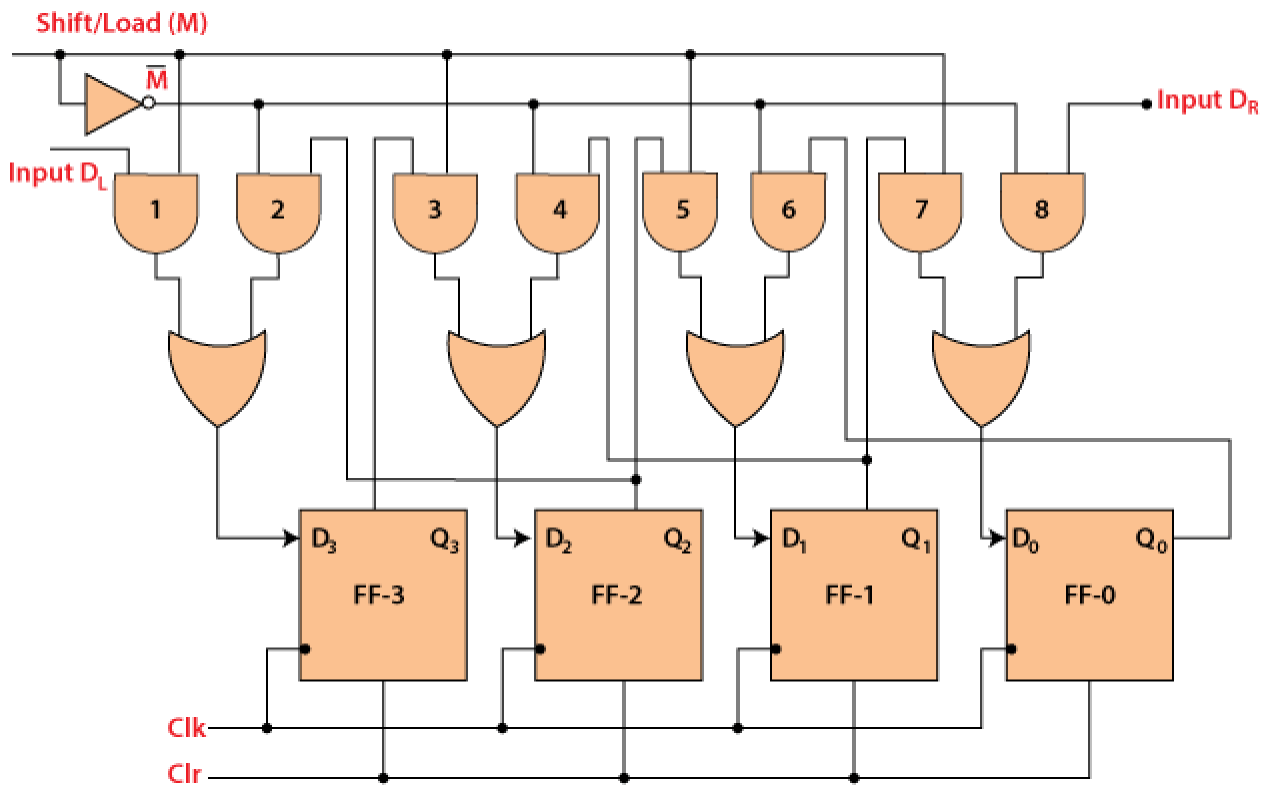
Bidirectional Shift Register

The binary number after shifting each bit of the number to the left by one position will be equivalent to the number produced by multiplying the original number by 2. In the same way, the binary number after shifting each bit of the number to the right by one position will be equivalent to the number produced by dividing the original number by 2.

For performing the multiplication and division operation using the shift register, it is required that the data should be moved in both the direction, i.e., left or right in the register. Such registers are called the **"Bidirectional"** shift register.

Below is the diagram of 4-bit **"bidirectional"** shift register where **D_R** is the **"serial right shift data input"**, **D_L** is the **"left shift data input"**, and M is the **"mode select input"**.

Block Diagram



Operations

1) Shift right operation(M=1)

- The first, third, fifth, and seventh AND gates will be enabled, but the second, fourth, sixth, and eighth AND gates will be disabled.

- The data present on the data input **DR** is shifted bit by bit from the fourth flip flop to the first flip flop when the clock pulse is applied. In this way, the shift right operation occurs.

2) Shift left operation(M=0)

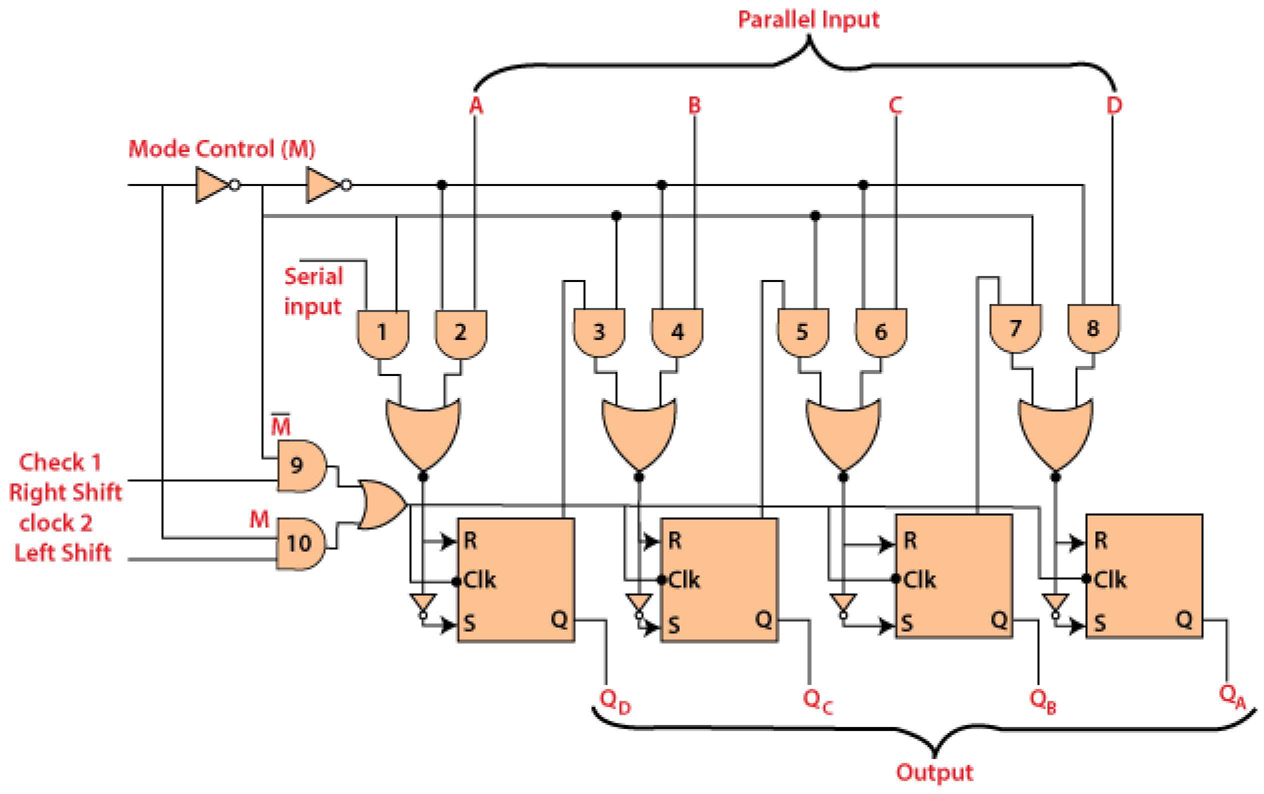
- The second, fourth, sixth and eighth AND gates will be enabled, but the AND gates first, third, fifth, and seventh will be disabled.
- The data present on the data input DR is shifted bit by bit from the first flip flop to the fourth flip flop when the clock pulse is applied. In this way, the shift right operation occurs.

Universal Shift Register

A register where the data is shifted in one direction is known as the **"uni-directional"** shift register. A register in which the data is shifted in both the direction is known as **"bi-directional"** shift register. A **"Universal"** shift register is a special type of register that can load the data in a parallel way and shift that data in both directions, i.e., right and left.

The input M, i.e., the mode control input, is set to 1 to perform the parallel loading operation. If this input set to 0, then the serial shifting operation is performed. If we connect the mode control input with the ground, then the circuit will work as a **"bi-directional"** register. The diagram of the universal shift register is given below. When the input is passed to the **serial input**, the register performs the "serial left" operation. When the input is passed to the input **D**, the register performs the serial right operation.

Block Diagram

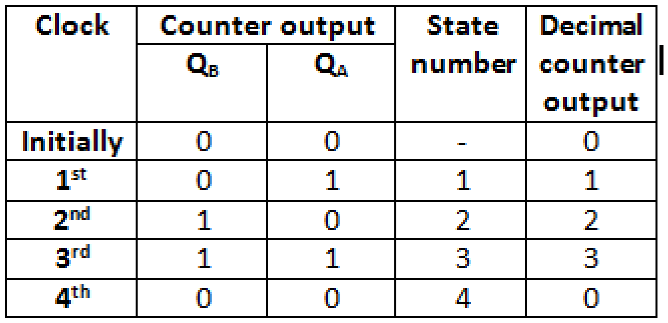


Counters

A special type of sequential circuit used to count the pulse is known as a counter, or a collection of flip flops where the clock signal is applied is known as counters.

The counter is one of the widest applications of the flip flop. Based on the clock pulse, the output of the counter contains a predefined state. The number of the pulse can be counted using the output of the counter.

Truth Table



Clock	Counter output		State number	Decimal counter output
	Q_B	Q_A		
Initially	0	0	-	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

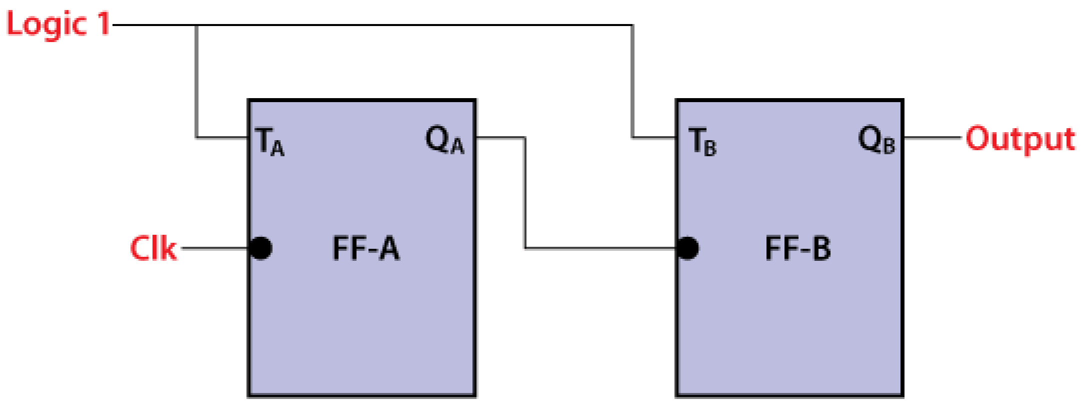
There are the following types of counters:

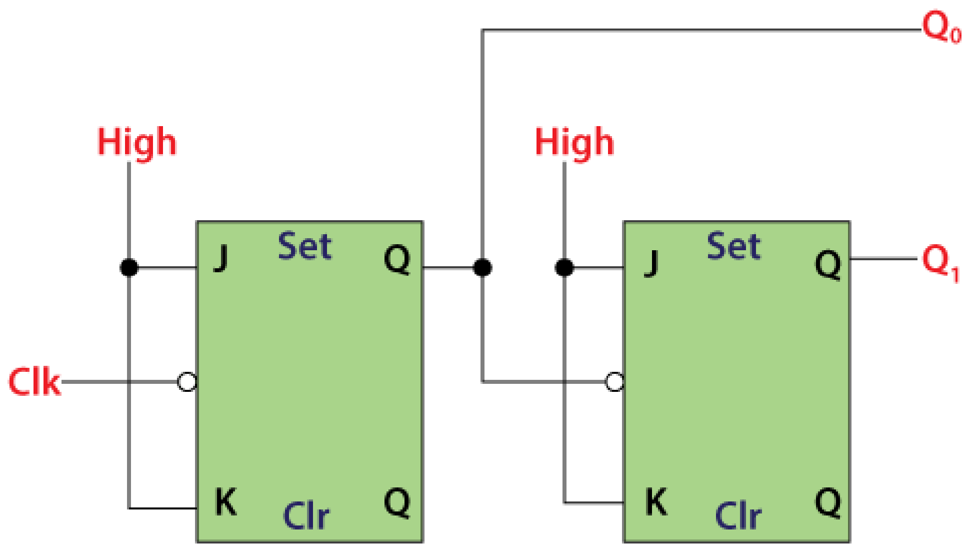
- Asynchronous Counters
- Synchronous Counters

Asynchronous or ripple counters

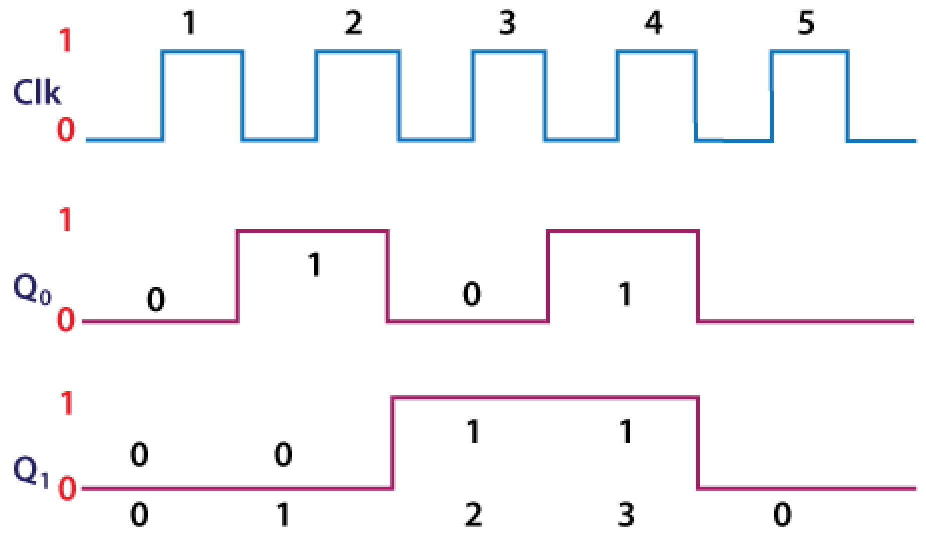
The **Asynchronous counter** is also known as the **ripple counter**. Below is a diagram of the 2-bit **Asynchronous counter** in which we used two T flip-flops. Apart from the T flip flop, we can also use the JK flip flop by setting both of the inputs to 1 permanently. The external clock pass to the clock input of the first flip flop, i.e., FF-A and its output, i.e., is passed to clock input of the next flip flop, i.e., FF-B.

Block Diagram





Signal Diagram



Operation

1. **Condition 1:** When both the flip flops are in reset condition. **Operation:** The outputs of both flip flops, i.e., Q_A Q_B, will be 0.
2. **Condition 2:** When the first negative clock edge passes. **Operation:** The first flip flop will toggle, and the output of this flip flop will change from 0 to 1. The output of this

flip flop will be taken by the clock input of the next flip flop. This output will be taken as a positive edge clock by the second flip flop. This input will not change the second flip flop's output state because it is the negative edge triggered flip flop.
So, $Q_A = 1$ and $Q_B = 0$

3. **Condition 3:** When the second negative clock edge is applied.
Operation: The first flip flop will toggle again, and the output of this flip flop will change from 1 to 0. This output will be taken as a negative edge clock by the second flip flop. This input will change the second flip flop's output state because it is the negative edge triggered

flip flop.
So, Q_A = 0 and Q_B = 1.

4. **Condition 4:** When the third negative clock edge is applied.
   Operation: The first flip flop will toggle again, and the output of this flip flop will change from 0 to 1. This output will be taken as a positive edge clock by the second flip flop. This input will not change the second flip flop's output state because it is the negative edge triggered flip flop.
   So, Q_A = 1 and Q_B = 1
5. **Condition 5:** When the fourth negative clock edge is applied.
   Operation: The first flip flop will toggle

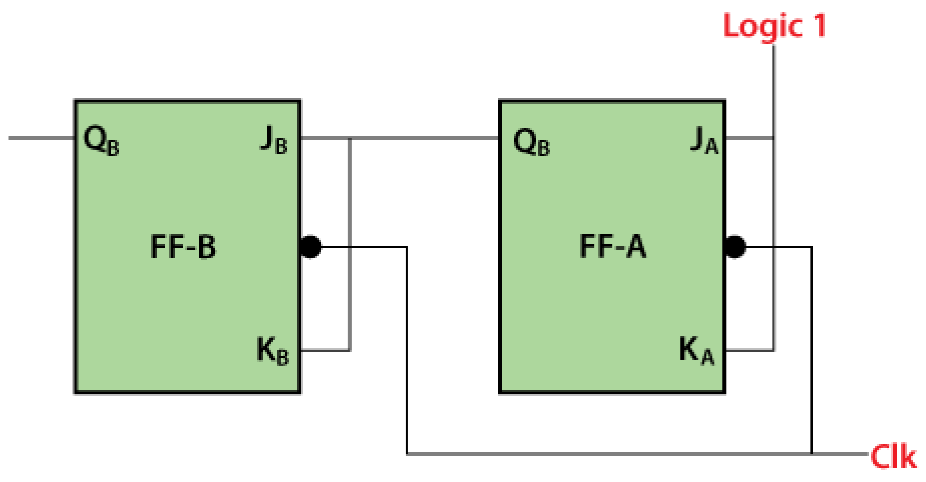
again, and the output of this flip flop will change from 1 to 0. This output will be taken as a negative edge clock by the second flip flop. This input will change the output state of the second flip flop. So, $Q_A = 0$ and $Q_B = 0$

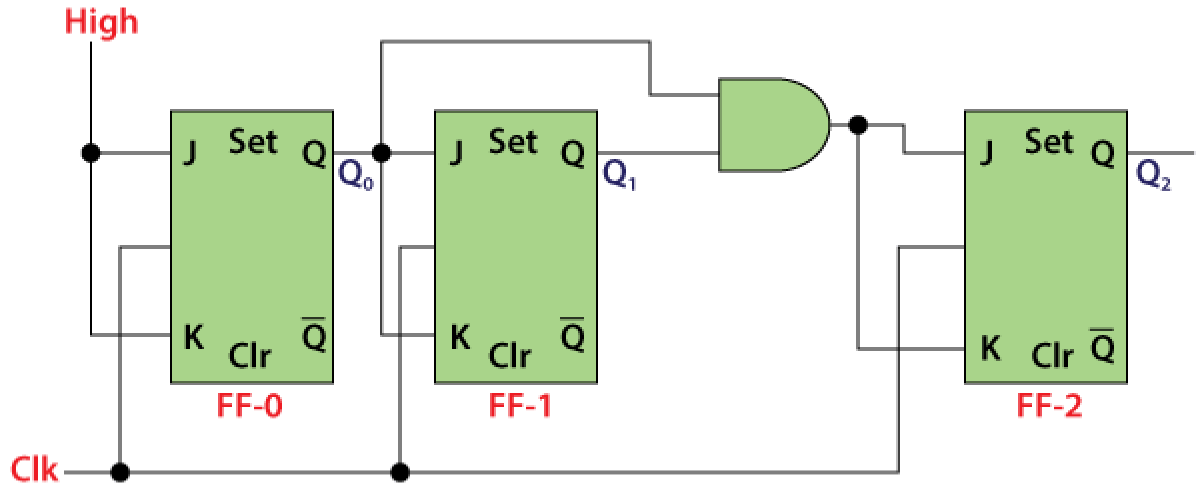
Synchronous counters

In the **Asynchronous counter**, the present counter's output passes to the input of the next counter. So, the counters are connected like a chain. The drawback of this system is that it creates the counting delay, and the propagation delay also occurs during the counting stage. The **synchronous counter** is designed to remove this drawback.

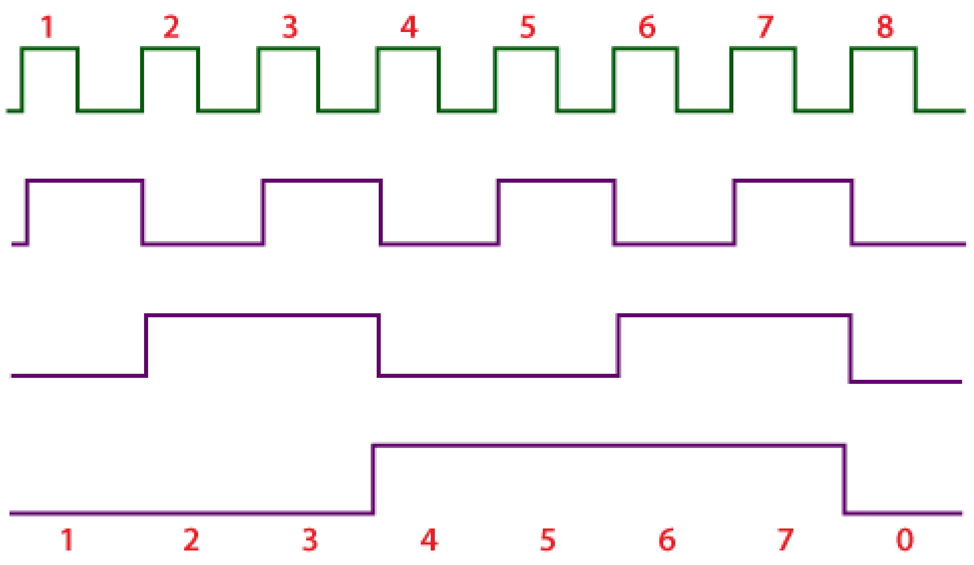
In the **synchronous counter**, the same clock pulse is passed to the clock input of all the flip flops. The clock signals produced by all the flip flops are the same as each other. Below is the diagram of a 2-bit synchronous counter in which the inputs of the first flip flop, i.e., FF-A, are set to 1. So, the first flip flop will work as a toggle flip-flop. The output of the first flip flop is passed to both the inputs of the next JK flip flop.

Logical Diagram





Signal Diagram



Operation

1. **Condition 1:** When both the flip flops are in reset condition. **Operation:** The outputs of both flip flops, i.e., Q_A Q_B, will be 0. So, $\mathbf{Q_A = 0}$ and $\mathbf{Q_B = 0}$

2. **Condition 2:** When the first negative clock edge passes. **Operation:** The first flip flop will be toggled, and the output of this flip flop

will be changed from 0 to 1. When the first negative clock edge is passed, the output of the first flip flop will be 0. The clock input of the first flip flop and both of its inputs will set to 0. In this way, the state of the second flip flop will remain the same.
So, **Q_A = 1** and **Q_B = 0**

3. **Condition 2:** When the second negative clock edge is passed.
Operation: The first flip flop will be toggled again, and the output of this flip flop will be changed from 1 to 0. When the second negative clock edge is passed, the output of the first flip flop will be 1. The clock input of the first flip flop and both of its inputs will set to 1.

In this way, the state of the second flip flop will change from 0 to 1.
So, **Q_A = 0** and **Q_B = 1**

4. **Condition 2:** When the third negative clock edge passes.
   Operation: The first flip flop will toggle from 0 to 1, but at this instance, both the inputs and the clock input set to 0. Hence, the outputs will remain the same as before.
   So, **Q_A = 1** and **Q_B = 1**
5. **Condition 2:** When the fourth negative clock edge passes.
   Operation: The first flip flop will toggle from 1 to 0. At this instance, the inputs and the clock input of the second flip

flop set to 1. Hence, the outputs will change from 1 to 0. So, $\mathbf{Q_A = 0}$ and $\mathbf{Q_B = 0}$

Memory Unit

The memory unit is a component of a computer system. It is used to store data, instructions and information. It is also known as a principal/primary/internal memory. There are two types of memory: –

Read-only memory (ROM):- ROM is a part of the memory unit. This is read only memory, and it can not be used to write.

Random-access memory (RAM):- RAM is also part of the memory unit, and it is used for the temporary storage of program data.

When other computer units require information, this unit provides it. Internal storage unit, main memory, primary storage, and Random Access Memory are used to describe it (RAM). Its size has an impact on its speed, power, and capabilities. In a computer, there are two sorts of memories: primary memory and secondary memory.

The memory unit’s functions

It holds all of the necessary data and instructions for processing.

It keeps track of the processing’s interim results.

Before the final results of processing are sent to an output device, it saves them.

The main memory receives and transmits all inputs and outputs

Memory units are used to measure and represent data. Some of the commonly used memory units are:

1) **Bit:** The computer memory units start from bit. A bit is the smallest memory unit to measure data stored in main memory and storage devices. A bit can have only one binary value out of 0 and 1.

2) **Byte:** It is the fundamental unit to measure data. It contains 8 bits or is equal to 8 bits. Thus a byte can represent 2*8 or 256 values.

3) **Kilobyte:** A kilobyte contains 1024 bytes.

4) **Megabyte:** A megabyte contains 1024 kilobytes.

5) **Gigabyte:** A gigabyte contains 1024 megabyte.

6) **Terabyte:** A terabyte contains 1024 gigabytes.

RAM

RAM (Random Access Memory) is the internal memory of the CPU for storing data, program, and program result. It is a read/write memory which stores data until the machine is working. As soon as the machine is switched off, data is erased.

Access time in RAM is independent of the address, that is, each storage location inside the memory is as easy to reach as other locations and takes the same amount of time. Data in the RAM can be accessed randomly but it is very expensive.

RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup Uninterruptible Power System (UPS) is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.

RAM is of two types −

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Static RAM (SRAM)

The word **static** indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis.

There is extra space in the matrix, hence SRAM uses more chips than DRAM for the same amount of storage space, making the manufacturing costs higher. SRAM is thus used as cache memory and has very fast access.

Characteristic of Static RAM

- Long life
- No need to refresh
- Faster
- Used as cache memory
- Large size
- Expensive
- High power consumption

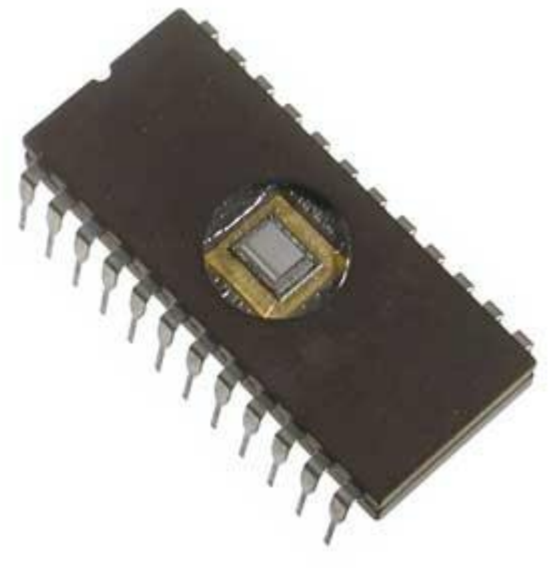
Dynamic RAM (DRAM)

DRAM, unlike SRAM, must be continually **refreshed** in order to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory as it is cheap and small. All DRAMs are made up of memory cells, which are composed of one capacitor and one transistor.

Characteristics of Dynamic RAM

- Short data lifetime
- Needs to be refreshed continuously
- Slower as compared to SRAM
- Used as RAM
- Smaller in size
- Less expensive
- Less power consumption

ROM stands for **Read Only Memory**. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as **bootstrap**. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.



Let us now discuss the various types of ROMs and their characteristics.

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

Advantages of ROM

The advantages of ROM are as follows −

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing

- Contents are always known and can be verified