UNIT 1:

Digital Components

Computer Organization and Architecture-

Computer Organization and Architecture is used to design computer systems. Computer Architecture is considered to be those attributes of a system that are visible to the user like addressing techniques, instruction sets, and bits used for data, and have a direct impact on the logic execution of a program, It defines the system in an abstract manner, It deals with What does the system do.

Whereas, Computer Organization is the way in which a system has to structure and It is operational units and the interconnections between them that achieve the architectural specifications, It is the realization of the abstract model, and It deals with How to implement the system

Computer organization and architecture refer to the design and structure of a computer system. It encompasses how hardware components are interconnected and how they work together to execute instructions. Below is an overview of computer organization and architecture, along with a simplified diagram:

1. Von Neumann Architecture: The von Neumann architecture, proposed by John von Neumann in the 1940s, is the foundation of most modern computers. It consists of four main components:

a. **Central Processing Unit (CPU):** The CPU is the brain of the computer. It performs arithmetic and logical operations, controls the execution of instructions, and manages data flow. It includes the control unit (CU) and the arithmetic logic unit (ALU).

b. **Memory:** Memory stores both data and instructions that the CPU needs to process. It is usually divided into primary memory (RAM) for fast access and secondary memory (e.g., hard disk) for long-term storage.

c. **Input/Output (I/O) Devices:** These devices enable communication between the computer and the outside world. Examples include keyboards, mice, monitors, printers, and networking devices.

d. **Control Unit and Bus:** The control unit manages the flow of data and instructions between the CPU, memory, and I/O devices. It utilizes a system of buses (data bus, address bus, and control bus) to facilitate data transfer.

2. Instruction Cycle: The CPU executes instructions in a sequence of steps called the instruction cycle:

a. **Fetch:** The CPU fetches the next instruction from memory, based on the value of the program counter (PC).

b. **Decode:** The fetched instruction is decoded by the control unit to determine the operation to be performed.

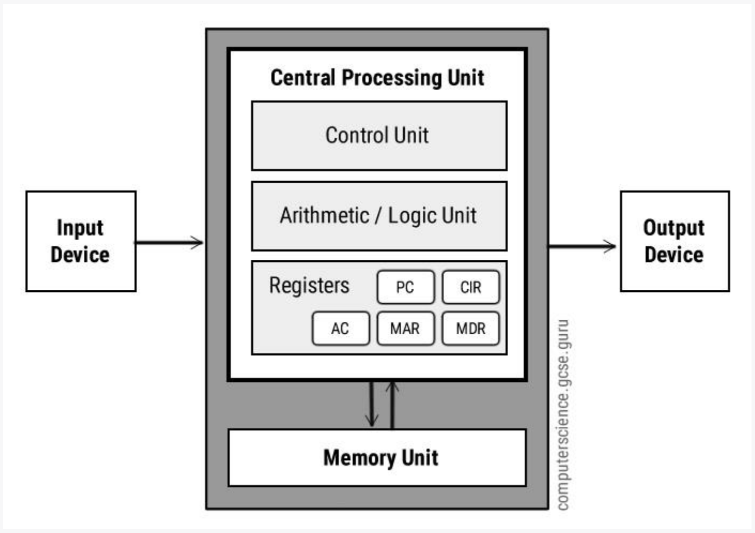
c. **Execute:** The decoded instruction is executed by the ALU or passed on to other units for further processing.

3. Memory Hierarchy: Modern computers use a memory hierarchy, with multiple levels of memory, to optimize performance. The hierarchy consists of registers (fastest but smallest), cache memory, main memory (RAM), and secondary storage (hard disk, SSD).

4. Instruction Set Architecture (ISA): The ISA defines the set of instructions that a CPU can execute, as well as the formats and operation codes of these instructions.

5. Pipelining: Pipelining is a technique that allows the CPU to execute multiple instructions simultaneously by breaking them down into smaller stages and processing them in parallel.

Diagram: A simplified diagram of computer organization and architecture:



In this diagram, the CPU consists of the ALU and the control unit, which communicate with memory and I/O devices through buses. Memory stores data and instructions, and the I/O devices handle input and output operations. The control unit manages the overall flow of data and instructions.

Logic Gates – Definition, Types, Uses

A **semiconductor** material's electrical conductivity is somewhere between that of a conductor, such as metallic copper, and that of an insulator, such as glass. As the temperature rises, its resistivity reduces, whereas metals have the opposite effect. The conductivity of a crystal structure can be modified in a favorable way by introducing impurities (doping) to it. When two separate doped regions in the same crystal exist, a semiconductor junction is generated. The behavior of charge carriers such as electrons, ions, and electron holes at these junctions is the foundation of diodes, transistors, and most modern electronics.
Semiconductors include silicon, germanium, gallium arsenide, and elements on the periodic table's so-called metalloid staircase. After silicon, gallium arsenide is the second most common semiconductor, and it's used in laser diodes, solar cells, microwave-frequency integrated circuits, and other things. Silicon is a critical component in the manufacture of nearly all electrical circuits.

Logic Gates
A logic gate is a simple switching circuit that determines whether an input pulse can pass through to the output in digital circuits.

The building blocks of a digital circuit are logic gates, which execute numerous logical operations that are required by any digital circuit. These can take two or more inputs but only produce one output.

The mix of inputs applied across a logic gate determines its output. Logic gates use Boolean algebra to execute logical processes. Logic gates are found in nearly every digital gadget we use on a regular basis. Logic gates are used in the architecture of our telephones, laptops, tablets, and memory devices.

Boolean Algebra
Boolean algebra is a type of logical algebra in which symbols represent logic levels.

The digits(or symbols) 1 and 0 are related to the logic levels in this algebra; in electrical circuits, logic 1 will represent a closed switch, a high voltage, or a device's "on" state. An open switch, low voltage, or "off" state of the device will be represented by logic 0.

At any one time, a digital device will be in one of these two binary situations. A light bulb can be used to demonstrate the operation of a logic gate. When logic 0 is supplied to the switch, it is turned off, and the bulb does not light up. The switch is in an ON state when logic 1 is applied, and the bulb would light up. In integrated circuits (IC), logic gates are widely employed.

Truth Table: *The outputs for all conceivable combinations of inputs that may be applied to a logic gate or circuit are listed in a truth table. When we enter values into a truth table, we usually express them as 1 or 0, with 1 denoting True logic and 0 denoting False logic.*
Types of Logic Gates
A logic gate is a digital gate that allows data to be transferred. Logic gates, use logic to determine whether or not to pass a signal. Logic gates, on the other hand, govern the flow of information based on a set of rules. The following types of logic gates are commonly used:

1. AND
2. OR
3. NOT
4. NOR
5. NAND
6. XOR
7. XNOR

Basic Logic Gates

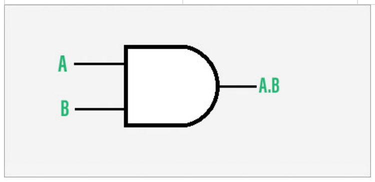
AND Gate

An AND gate has a single output and two or more inputs.

1. When all of the inputs are 1, the output of this gate is 1.
2. The AND gate's Boolean logic is **Y=A.B** if there are two inputs A and B.

An AND gate's symbol and truth table are as follows:

Input		**Output**
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



Symbol of AND gate

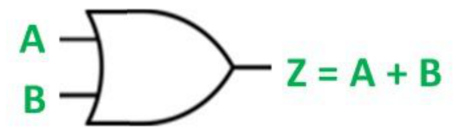
Therefore, in And gate, the output is high when all the inputs are high.

OR Gate

Two or more inputs and one output can be used in an OR gate.

1. The logic of this gate is that if at least one of the inputs is 1, the output will be 1.
2. The OR gate's output will be given by the following mathematical procedure if there are two inputs A and B: Y=A+B

Input		**Output**
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



Symbol of OR gate

Therefore, in the OR gate, the output is high when any of the inputs is high.

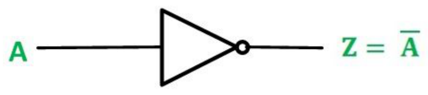
NOT Gate

The NOT gate is a basic one-input, one-output gate.

1. When the input is 1, the output is 0, and vice versa. A NOT gate is sometimes called an inverter because of its feature.
2. If there is only one input A, the output may be calculated using the Boolean equation Y=A'.

Input	**Output**
A	Not A
0	1

1	0



Symbol of NOT gate

A NOT gate, as its truth table shows, reverses the input signal.

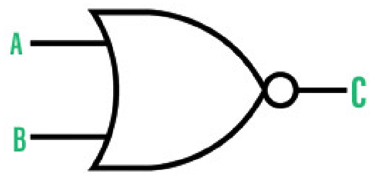
Universal Logic Gates

NOR Gate

A NOR gate, sometimes known as a "NOT-OR" gate, consists of an OR gate followed by a NOT gate.

1. This gate's output is 1 only when all of its inputs are 0. Alternatively, when all of the inputs are low, the output is high.
2. The Boolean statement for the NOR gate is Y=(A+B)' if there are two inputs A and B.

Input		**Output**
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



Symbol of NOR gate

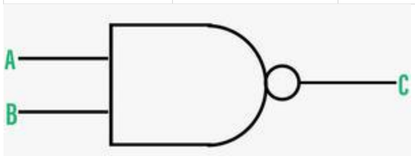
By comparing the truth tables, we can observe that the outputs of the NOR gate are the polar opposite of those of an OR gate. The NOR gate is sometimes known as a universal gate since it may be used to implement the OR, AND, and NOT gates.

NAND Gate

A NAND gate, sometimes known as a 'NOT-AND' gate, is essentially a Not gate followed by an AND gate.

1. This gate's output is 0 only if none of the inputs is 0. Alternatively, when all of the inputs are not high and at least one is low, the output is high.
2. If there are two inputs A and B, the Boolean expression for the NAND gate is Y=(A.B)'

Input		**Output**
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0



Symbol of NAND gate

By comparing their truth tables, we can observe that their outputs are the polar opposite of an AND gate. The NAND gate is known as a universal gate because it may be used to implement the AND, OR, and NOT gates.

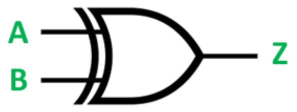
Other Logic Gates

XOR Gate

The Exclusive-OR or ‘Ex-OR’ gate is a digital logic gate that accepts more than two inputs but only outputs one value.

1. If any of the inputs is ‘High,’ the output of the XOR Gate is ‘High.’ If both inputs are ‘High,’ the output is ‘Low.’ If both inputs are ‘Low,’ the output is ‘Low.’
2. The Boolean equation for the XOR gate is Y=A’.B+A.B’ if there are two inputs A and B.

Input		**Output**
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Symbol of XOR gate

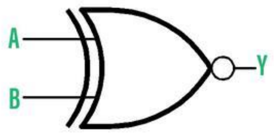
Its outputs are based on OR gate logic, as we can see from the truth table.

XNOR Gate

The Exclusive-NOR or ‘EX-NOR’ gate is a digital logic gate that accepts more than two inputs but only outputs one.

1. If both inputs are ‘High,’ the output of the XNOR Gate is ‘High.’ If both inputs are ‘Low,’ the output is ‘High.’ If one of the inputs is ‘Low,’ the output is ‘Low.’
2. If there are two inputs A and B, then the XNOR gate’s Boolean equation is: Y=A.B+A’B’.

Input		Output
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1



Symbol of XNOR gate

The truth table shows that its outputs are based on NOR gate logic.

Uses of Logic Gates

1. Logic gates are utilized in a variety of technologies. These are components of chips (ICs), which are components of computers, phones, laptops, and other electronic devices.
2. Logic gates may be combined in a variety of ways, and a million of these combinations are necessary to make the newest gadgets, satellites, and even robots.
3. Simple logic gate combinations can also be found in burglar alarms, buzzers, switches, and street lights. Because these gates can make a choice to start or stop based on logic, they are often used in a variety of sectors.
4. Logic gates are also important in data transport, calculation, and data processing. Even transistor-transistor logic and CMOS circuitry make extensive use of logic gates.

What is Adder?

As the name suggests, Adder is used to add binary numbers. Adder circuit is basically a combinational logic circuit. It is a memory less circuit and performs an operation assigned to it logically by a Boolean expression. The output depends upon the present input at any given time.

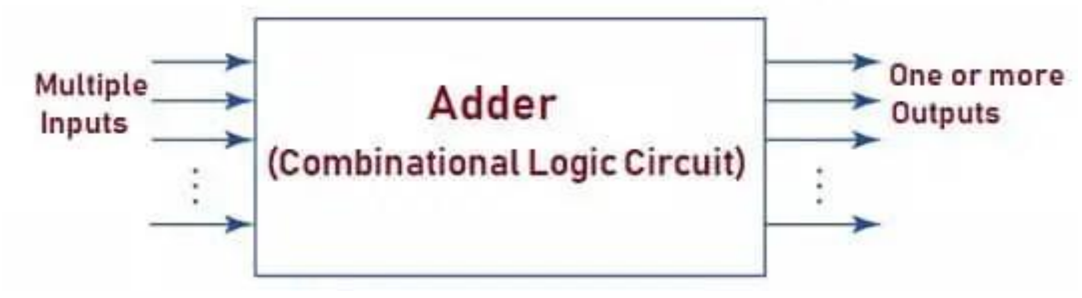


Fig. 1 – Block Diagram of Adder Circuit

Adder is used in the processor to increment and decrement operators, calculate addresses and to perform Arithmetic and logical operation in ALU.

Classification of Adders

Adders are broadly classified into two types. They are:

- Half Adder
- Full Adder
- Multi-bit Adder

Half Adder

Half Adder is a combinational arithmetic circuit that adds two binary numbers and produces sum bit (S) and carry bit (C) as the output. It is used to add 2 single-bit binary numbers.

Full Adder

It is a combinational arithmetic circuit constructed by combining two Half Adder circuits. It is used to add 3 one-bit binary numbers.

Multi-bit Adder

Multi-bit Adders are constructed using Full Adders either Serially or in Parallel known as:

- Serial Adder
- Parallel Adder

Serial Adder

Serial Adder is constructed using Full-Adder. It has three single-bit inputs and two single-bit outputs. It is a circuit that performs binary addition bit by bit for every clock (CLK) pulse. It is a sequential logic circuit.

Parallel Adder

Several Full-Adders are cascaded to perform binary addition faster. This circuit is used to find the sum of 2 binary numbers greater than one bit in length. It is a combinational logic circuit.

For every clock pulse the bits are added simultaneously. There are different types of Parallel Adders. They are:

- Ripple carry Adder
- Carry Look Ahead Adder

- Carry Save Adder
- Carry Increment Adder
- Carry Skip Adder
- Carry Select Adder
- Carry By-pass Adder

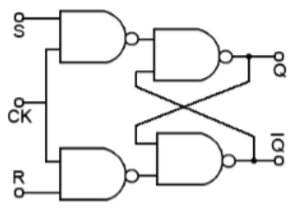
Flip-flop types, their Conversion and Applications

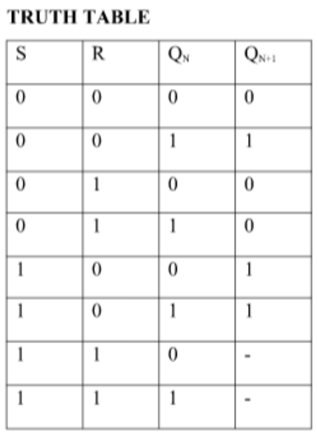
Flip-flop is a circuit that maintains a state until directed by input to change the state. A basic flip-flop can be constructed using four-NAND or four-NOR gates. **Types of flip-flops:**

1. SR Flip Flop
2. JK Flip Flop
3. D Flip Flop
4. T Flip Flop

Logic diagrams and truth tables of the different types of flip-flops are as follows:

S-R Flip Flop :



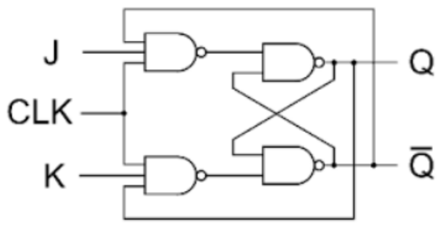


TRUTH TABLE

S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

Characteristics Equation for SR Flip Flop: $Q_{N+1} = Q_N R' + SR'$

J-K Flip Flop:

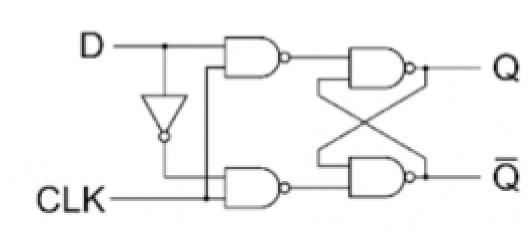


TRUTH TABLE

J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Characteristics Equation for JK Flip Flop: $Q_{N+1} = JQ'_N + K'Q_N$

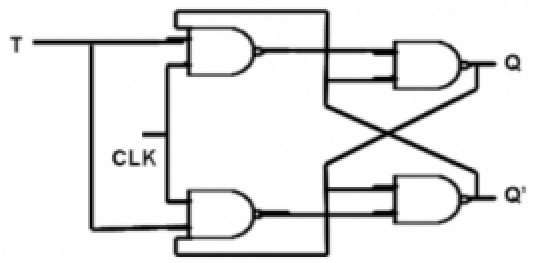
D Flip Flop:

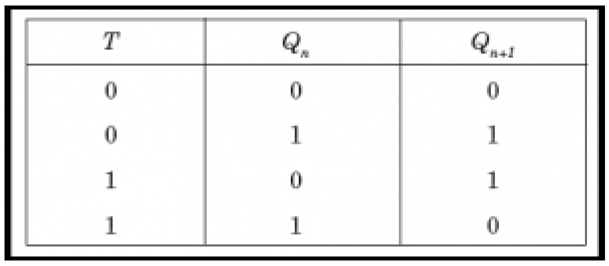


Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Characteristics Equation for D Flip Flop: $Q_{N+1} = D$

T Flip Flop:





T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Characteristics Equation for T Flip Flop: $Q_{N+1} = Q'_N T + Q_N T' = Q_N$ XOR T

Conversion for Flip Flops:

EXCITATION TABLE:

Q_N	Q_{N+1}	S	R	J	K	D	T
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	0	1
1	1	X	0	X	0	1	0

Steps To Convert from One Flip Flop to Other :
Let there be required flipflop to be constructed using sub-flipflop:

1. Draw the truth table of the required flip-flop.
2. Write the corresponding outputs of sub-flipflop to be used from the excitation table.
3. Draw K-Maps using required flipflop inputs and obtain excitation functions for sub-flipflop inputs.
4. Construct a logic diagram according to the functions obtained.

i) Convert SR To JK Flip Flop

J	K	Q_N	Q_{N+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Excitation Functions:

KQ_N

J

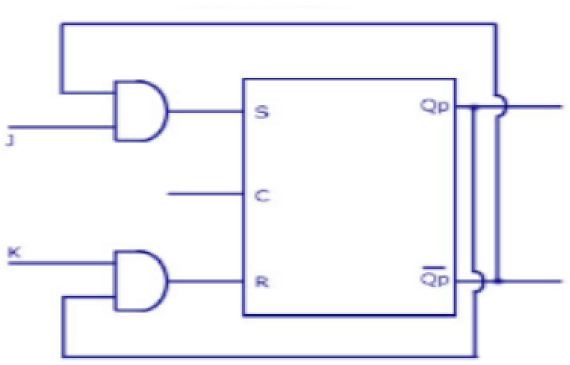
$S = JQ_N'$

0	X	0	0
1	X	0	1

$R = KQ_N$

KQ_N

X	0	1	X
0	0	1	0



ii) Convert SR To D FlipFlop:

D	Q_N	Q_{N+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

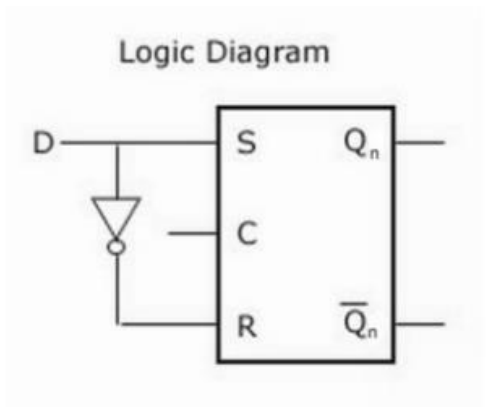
Excitation Functions: S = D, R = D′

S:

D,Q_N		
	0	0
	1	X

R:

D,Q_N		
	X	1
	0	0



Applications of Flip-Flops:

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- Counters
- Frequency Dividers
- Shift Registers
- Storage Registers
- Bounce elimination switch
- Data storage
- Data transfer
- Latch
- Registers
- Memory

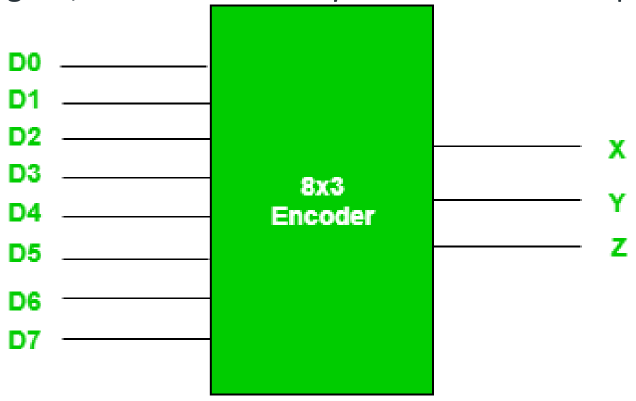
Encoders and Decoders

Binary code of N digits can be used to store 2^N distinct elements of coded information. This is what encoders and decoders are used for. **Encoders** convert 2^N lines of input into a code of N bits and **Decoders** decode the N bits into 2^N lines.

1. Encoders –

An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

As an example, let's consider **Octal to Binary** encoder. As shown in the following figure, an octal-to-binary encoder takes 8 input lines and generates 3 output lines.



Truth Table –

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
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

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
1	0	0	0	0	0	0	0	1	1	1

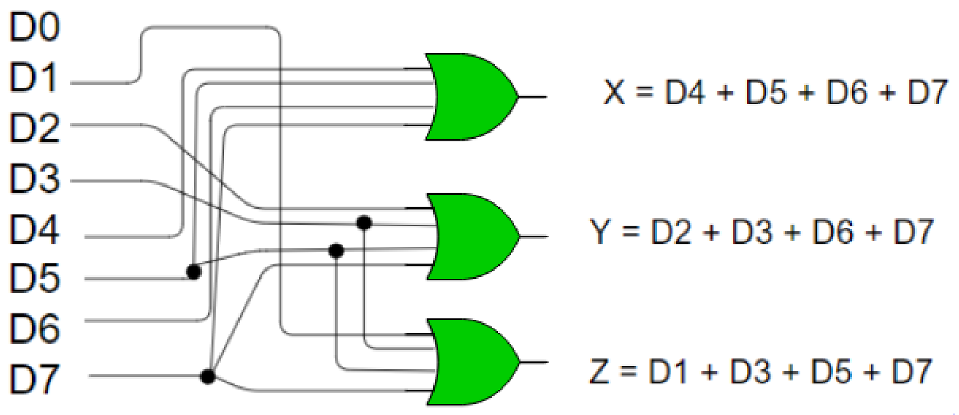
As seen from the truth table, the output is 000 when D0 is active; 001 when D1 is active; 010 when D2 is active and so on.

Implementation –
From the truth table, the output line Z is active when the input octal digit is 1, 3, 5 or 7. Similarly, Y is 1 when input octal digit is 2, 3, 6 or 7 and X is 1 for input octal digits 4, 5, 6 or 7. Hence, the Boolean functions would be:
X = D4 + D5 + D6 + D7

Y = D2 +D3 + D6 + D7

Z = D1 + D3 + D5 + D7

Hence, the encoder can be realised with OR gates as follows:



One limitation of this encoder is that only one input can be active at any given time. If more than one inputs are active, then the output is undefined. For example, if D6 and D3 are both active, then, our output would be 111 which is the output for D7. To overcome this, we use Priority Encoders.

Another ambiguity arises when all inputs are 0. In this case, encoder outputs 000 which actually is the output for D0 active. In order to avoid this, an extra bit can be added to the output, called the valid bit which is 0 when all inputs are 0 and 1 otherwise.

Priority Encoder –
A priority encoder is an encoder circuit in which inputs are given priorities. When more than one inputs are active at the same time, the input with higher priority takes precedence and the output corresponding to that is generated.

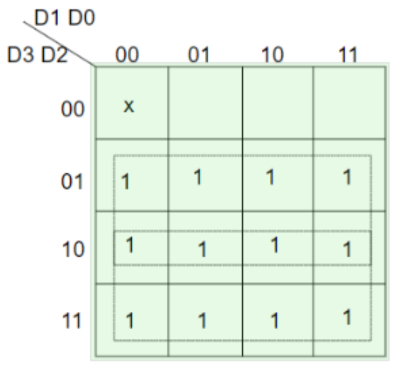
Let us consider the 4 to 2 priority encoder as an example.
From the truth table, we see that when all inputs are 0, our V bit or the valid bit is zero and outputs are not used. The x's in the table show the don't care condition, i.e, it may either be 0 or 1. Here, D3 has highest priority, therefore, whatever be the other inputs, when D3 is high, output has to be 11. And D0 has the lowest priority, therefore the output would be 00 only when D0 is high and the other input lines are low. Similarly, D2 has higher priority over D1 and D0 but lower than D3 therefore the output would be 010 only when D2 is high and D3 are low (D0 & D1 are don't care).

Truth Table –

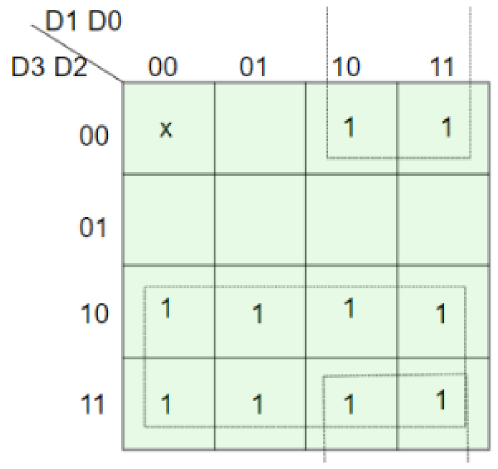
D3	D2	D1	D0	X	Y	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Implementation –
It can clearly be seen that the condition for valid bit to be 1 is that at least any one of the inputs should be high. Hence,
V = D0 + D1 + D2 + D3

For X:

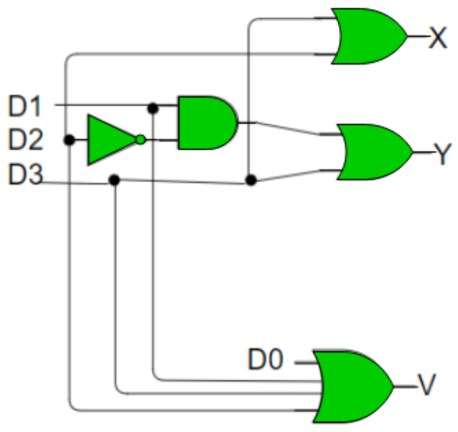


=> X = D2 + D3
For Y:



=> Y = D1 D2’ + D3

Hence, the priority 4-to-2 encoder can be realized as follows:



2. Decoders –
A decoder does the opposite job of an encoder. It is a combinational circuit that converts n lines of input into 2^n lines of output.
Let’s take an example of 3-to-8 line decoder.

Truth Table –

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementation –
D0 is high when X = 0, Y = 0 and Z = 0. Hence,
D0 = X’ Y’ Z’

Similarly,

D1 = X’ Y’ Z

D2 = X’ Y Z’

D3 = X’ Y Z

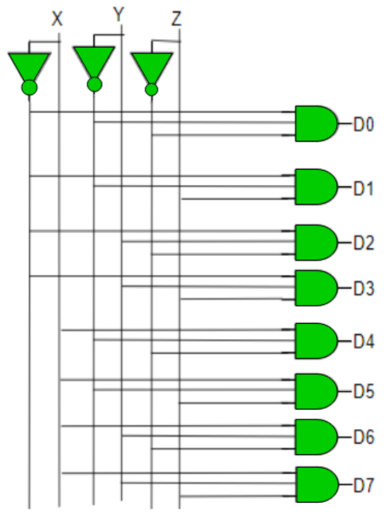
D4 = X Y’ Z’

D5 = X Y’ Z

D6 = X Y Z’

D7 = X Y Z

Hence,



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.

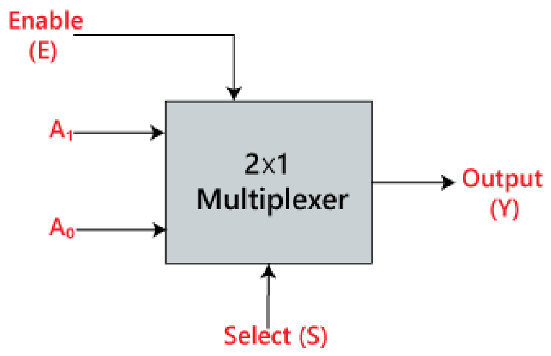
Unlike encoder and decoder, there are n selection lines and 2^n input lines. So, there is a total of 2^N possible combinations of inputs. A multiplexer is also treated as **Mux**.

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:



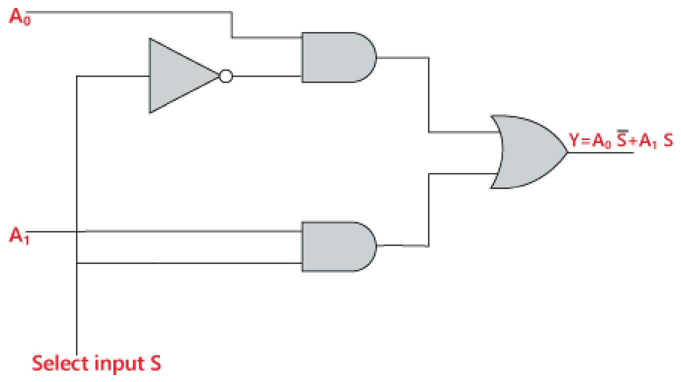
Truth Table:

INPUTS	Output
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

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

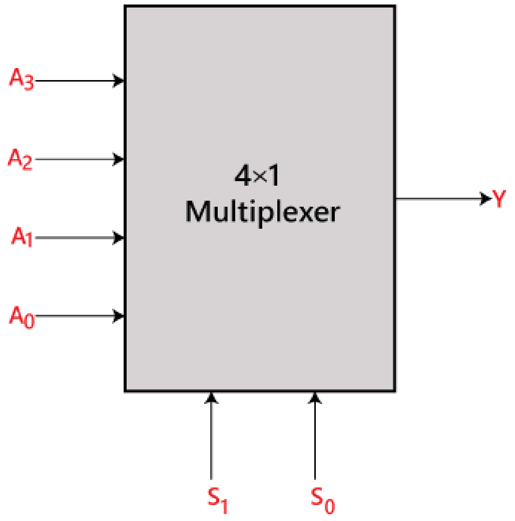
Logical circuit of the above expression is given below:



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:



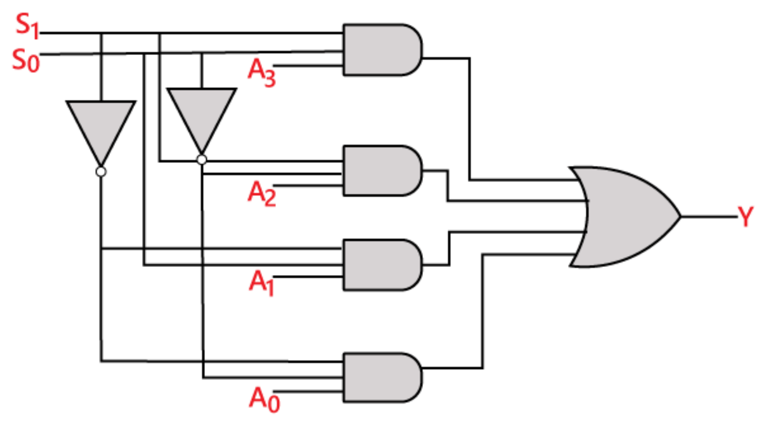
Truth Table:

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:

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

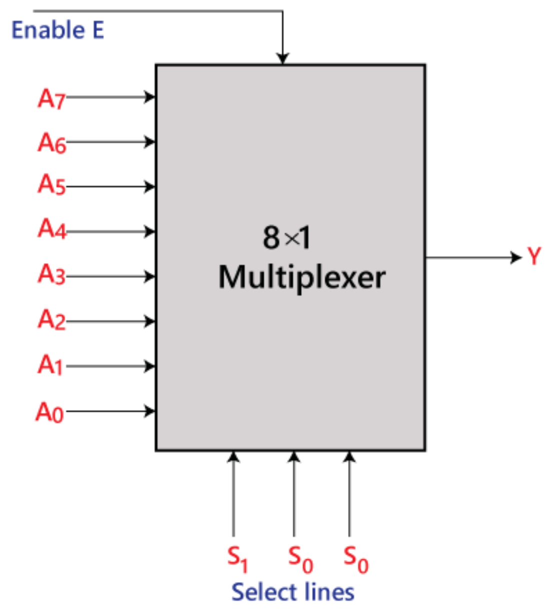
Logical circuit of the above expression is given below:



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:



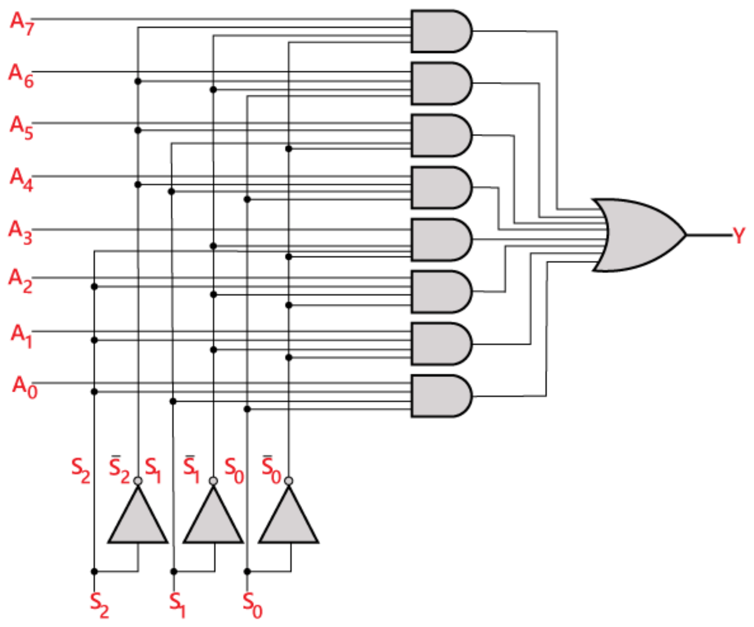
Truth Table:

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:

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

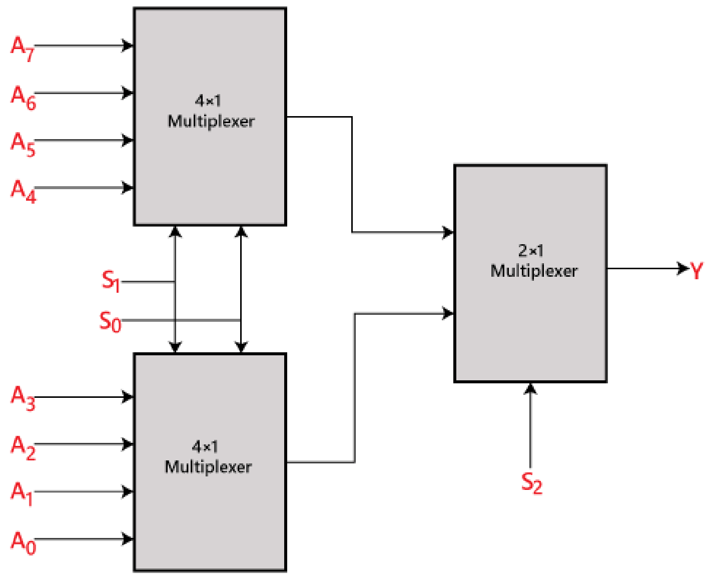
Logical circuit of the above expression is given below:



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.

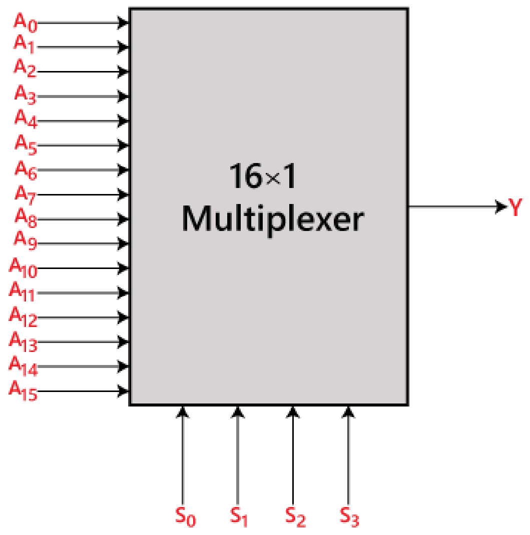
For getting 8 data inputs, we need two 4×1 multiplexers. The 4×1 multiplexer produces one output. So, in order to get the final output, we need a 2×1 multiplexer. The block diagram of 8×1 multiplexer using 4×1 and 2×1 multiplexer is given below.



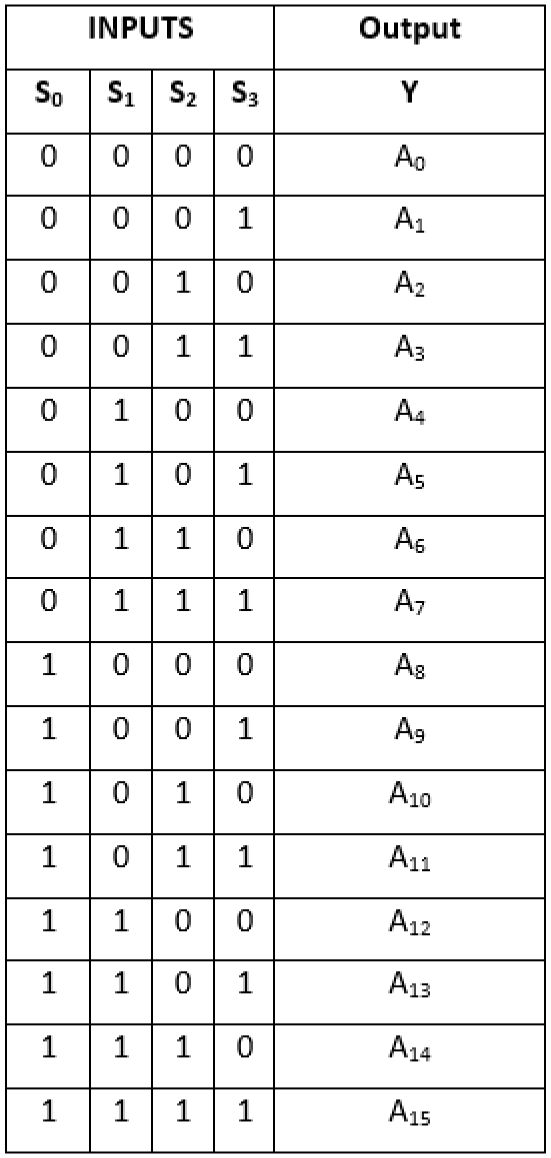
16 to 1 Multiplexer

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:



Truth Table:

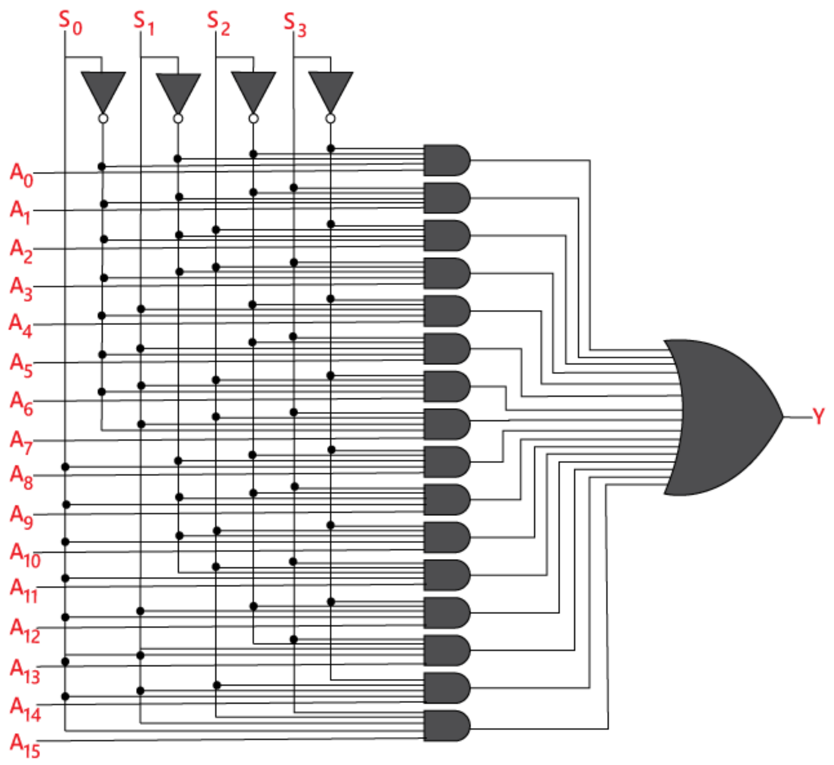


INPUTS				Output
S_0	S_1	S_2	S_3	Y
0	0	0	0	A_0
0	0	0	1	A_1
0	0	1	0	A_2
0	0	1	1	A_3
0	1	0	0	A_4
0	1	0	1	A_5
0	1	1	0	A_6
0	1	1	1	A_7
1	0	0	0	A_8
1	0	0	1	A_9
1	0	1	0	A_{10}
1	0	1	1	A_{11}
1	1	0	0	A_{12}
1	1	0	1	A_{13}
1	1	1	0	A_{14}
1	1	1	1	A_{15}

The logical expression of the term Y is as follows:

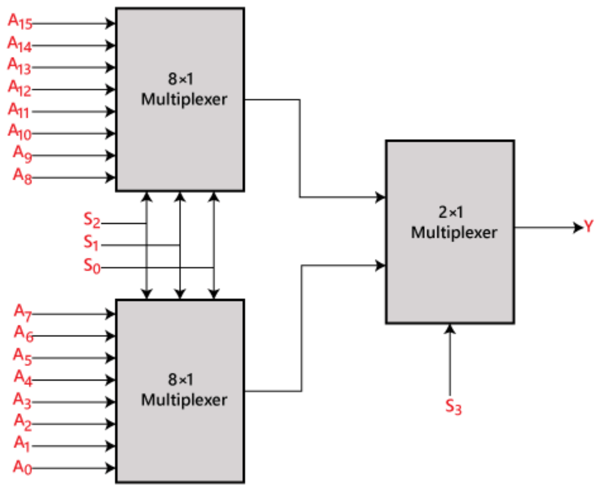
$Y=A_0.S_0'.S_1'.S_2'.S_3'+A_1.S_0'.S_1'.S_2\ '.S_3+A_2.S_0'.S_1'.S_2.S_3'+A_3.S_0'.S_1\ '.S_2.S_3+A_4.S_0'.S_1.S_2'.S_3'+A_5.S_0\ '.S$
$_1.S_2'.S_3+A_6.S_1.S_2.S_3'+A_7.S_0\ '.S_1.S_2.S_3+A_8.S_0.S_1'.S_2'.S_3'+A_9\ .S_0.S_1'.S_2'.S_3+Y_10.S_0.S_1'.S_2.S_3\ '+A_11.S_0.$
$S_1'.S_2.S_3+A_12\ S_0.S_1.S_2\ '.S_3'+A_13.S_0.S_1.S_2'.S_3+A_14.S_0.S_1\ .S_2.S_3'+A_15.S_0.S_1.S_2'.S_3$

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.



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:

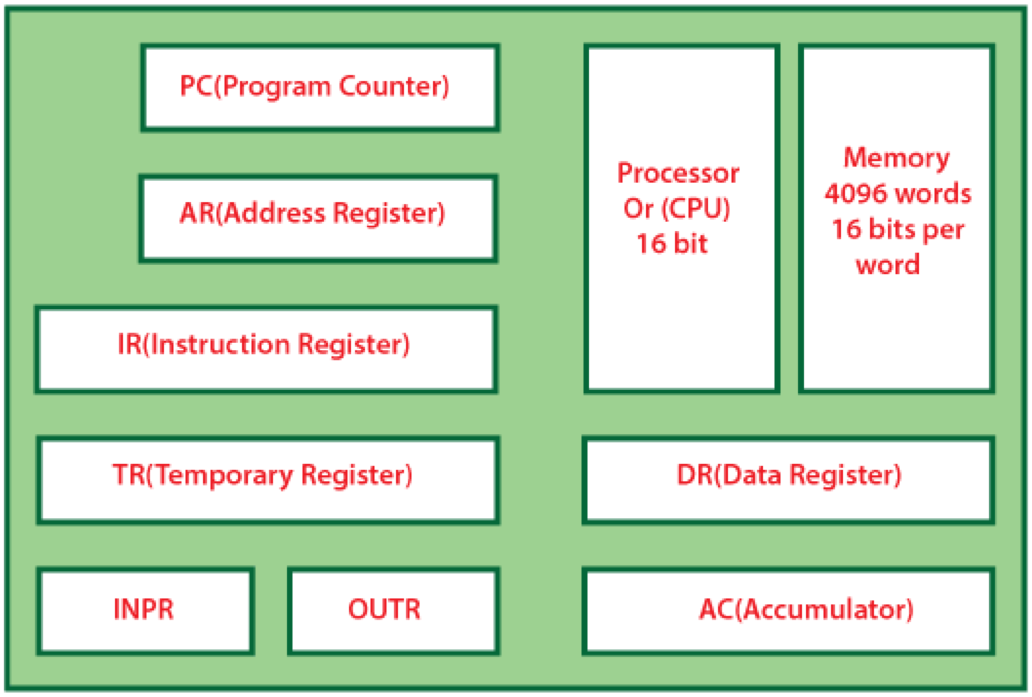
Fetch:

It is used

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

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:

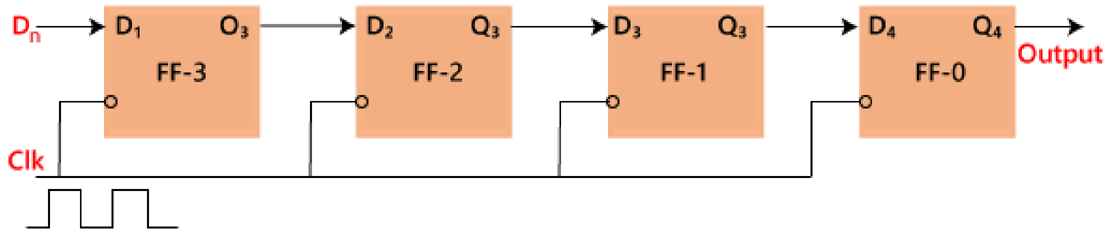
- Serial In Serial Out
- Serial In Parallel Out
- Parallel In Serial Out
- Parallel In Parallel Out
- Bi-directional Shift Register
- Universal Shift Register

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.

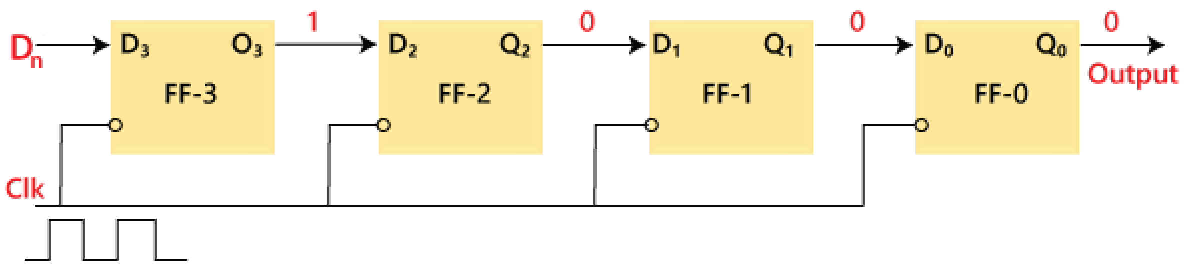
Initially, all the flip-flops are set in "reset" condition i.e. $Y_3 = Y_2 = Y_1 = Y_0 = 0$. If we pass the binary number 1111, the LSB bit of the number is applied first to the Din bit. The D3 input of the third flip flop, i.e., FF-3, is directly connected to the serial data input D3. The output Y_3 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:

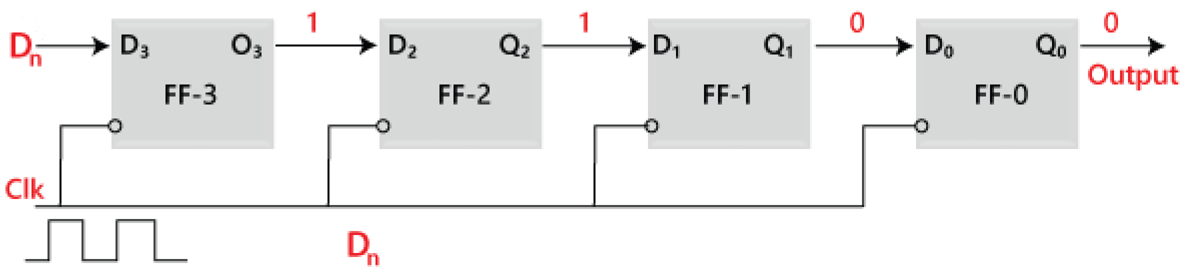


Operation

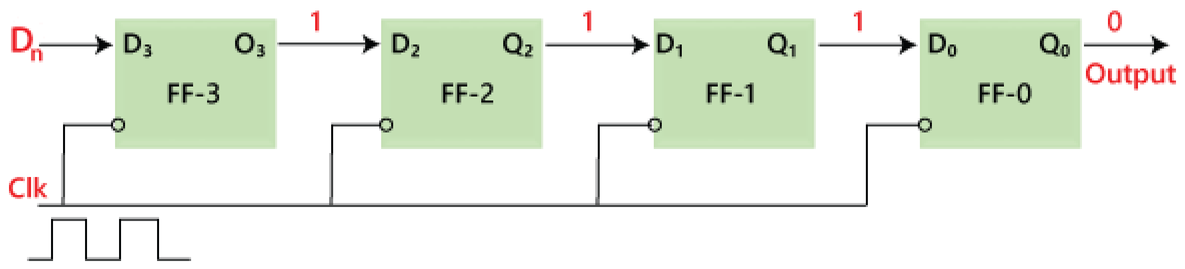
When the clock signal application is disabled, the outputs $Y_3\ Y_2\ Y_1\ Y_0 = 0000$. The LSB bit of the number is passed to the data input D_{in}, i.e., D_3. We will apply the clock, and this time the value of D_3 is 1. The first flip flop, i.e., FF-3, is set, and the word is stored in the register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



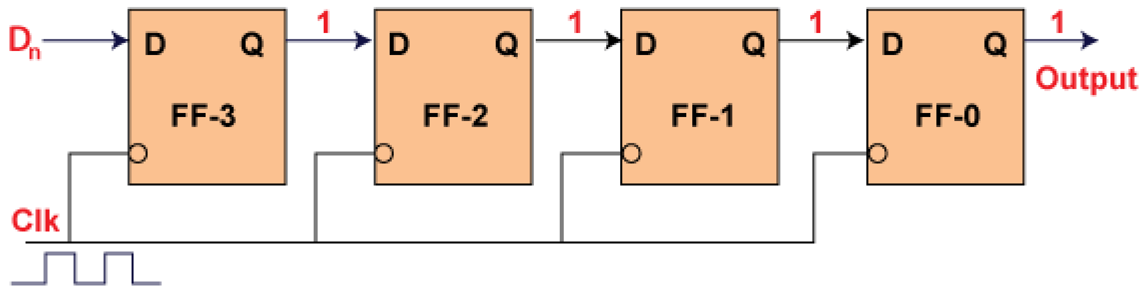
The next bit of the binary number, i.e., 1, is passed to the data input D_2. The second flip flop, i.e., FF-2, is set, and the word is stored when the next negative edge of the clock hits. The stored word is changed to 1100.



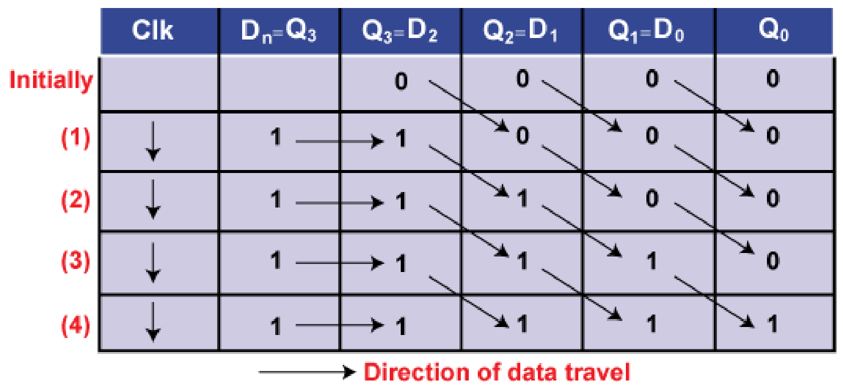
The next bit of the binary number, i.e., 1, is passed to the data input D_1, and the clock is applied. The third flip flop, i.e., FF-1, is set, and the word is stored when the negative edge of the clock hits again. The stored word is changed to 1110.



Similarly, the last bit of the binary number, i.e., 1, is passed to the data input D_0, 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.



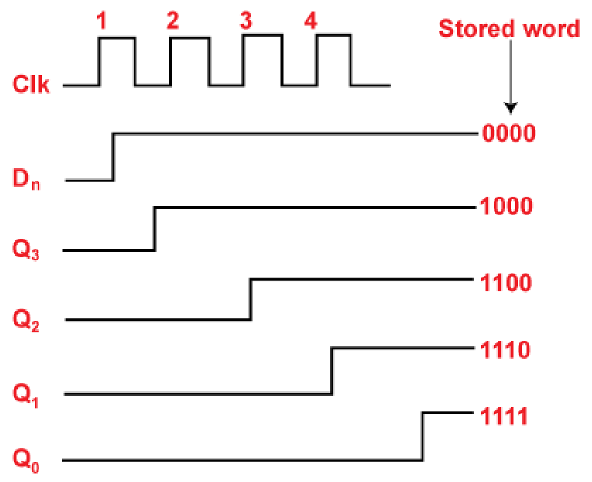
Truth Table



	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

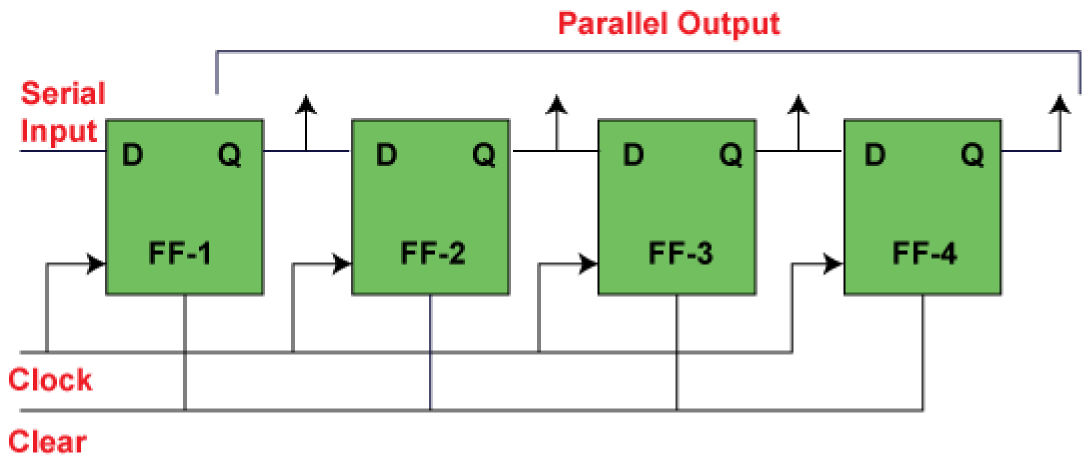


Serial IN Parallel OUT

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.

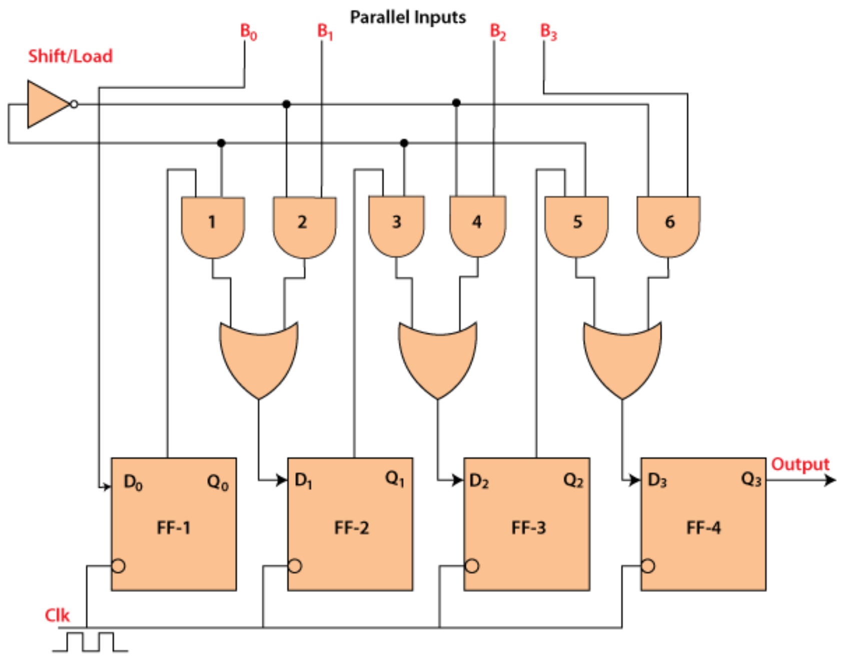
Load mode

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.

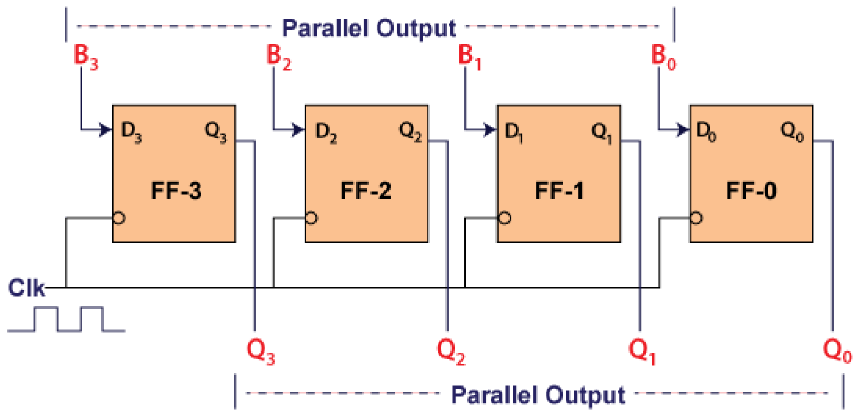
Block Diagram



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



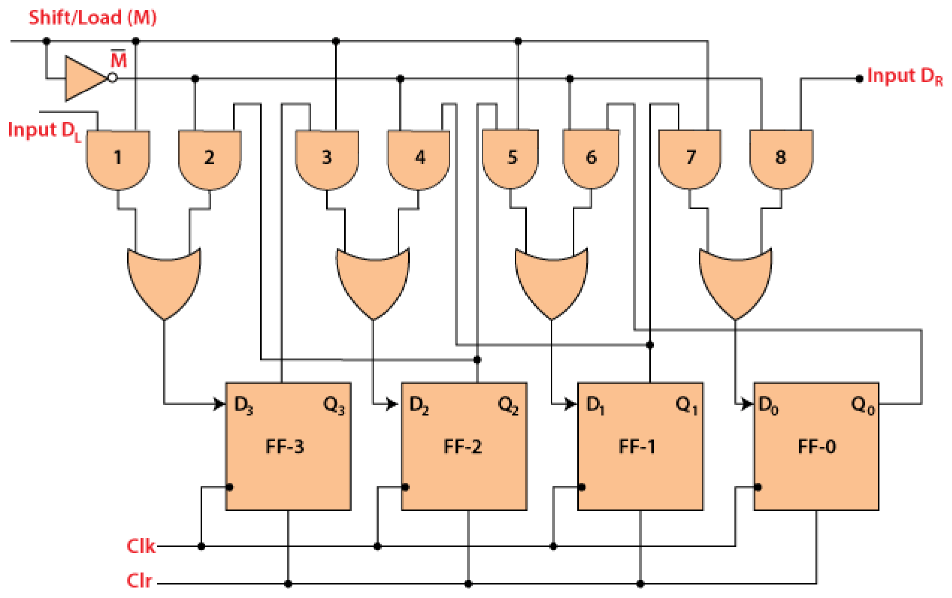
Bidirectional Shift Register

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 $\mathbf{D_R}$ is the **"serial right shift data input"**, $\mathbf{D_L}$ 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)

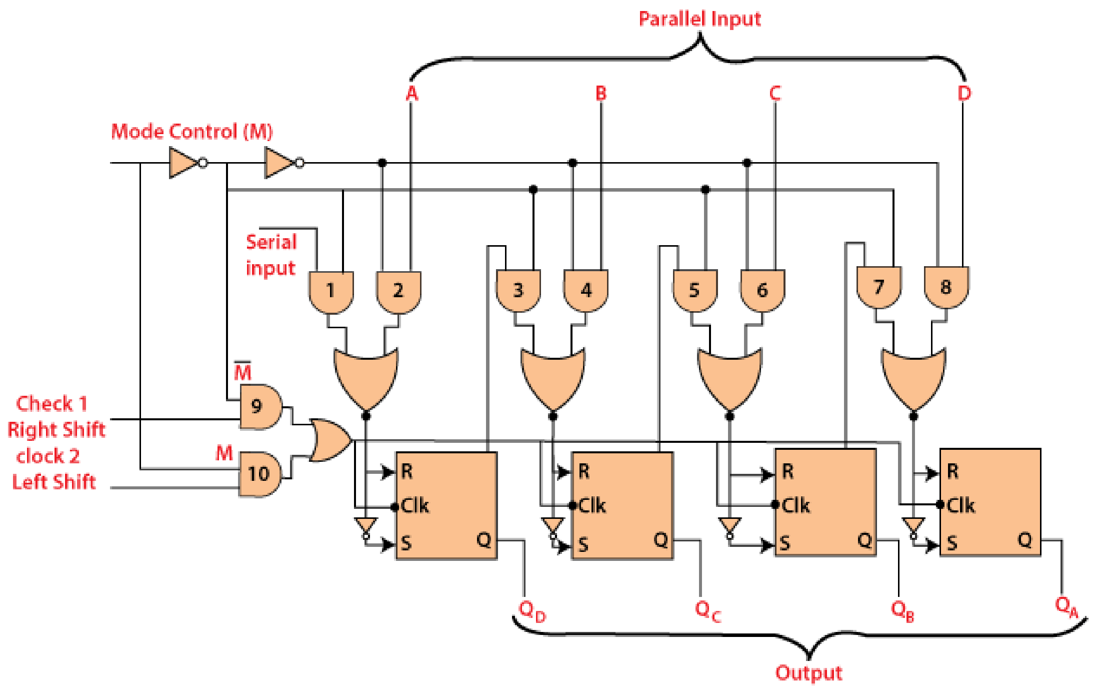
- The second, fourth, sixth and eighth AND gates will be enabled, but the AND gates first, third, fifth, and seventh will be disabled.
- The data present on the data input DR is shifted bit by bit from the first flip flop to the fourth flip flop when the clock pulse is applied. In this way, the shift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

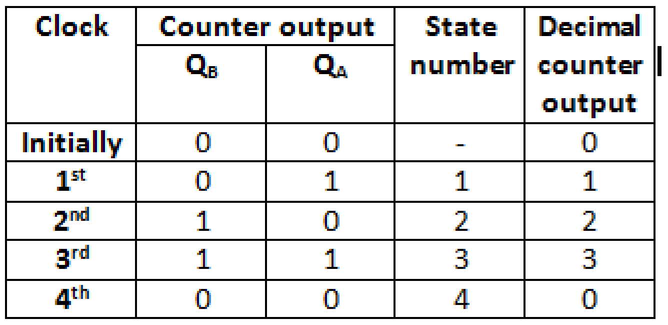


Counters

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

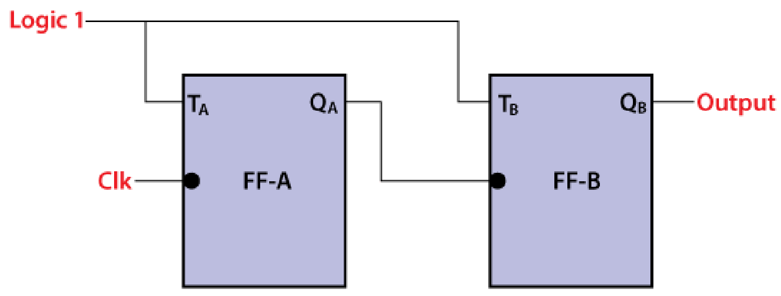
There are the following types of counters:

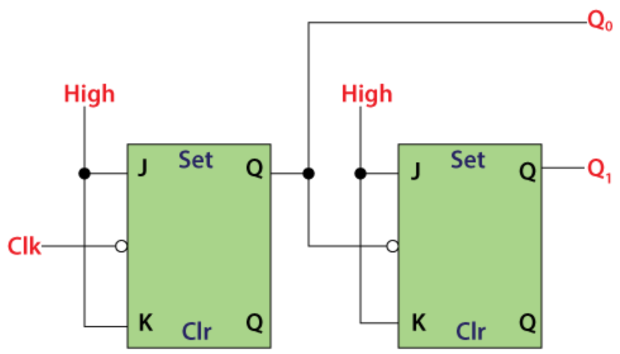
- Asynchronous Counters
- Synchronous Counters

Asynchronous or ripple counters

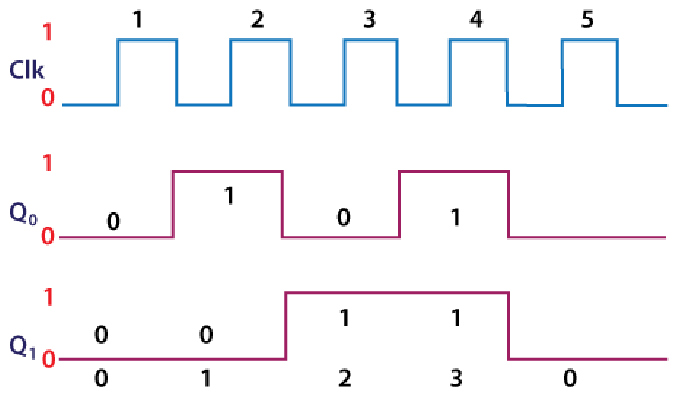
The **Asynchronous counter** is also known as the **ripple counter**. Below is a diagram of the 2-bit **Asynchronous counter** in which we used two T flip-flops. Apart from the T flip flop, we can also use the JK flip flop by setting both of the inputs to 1 permanently. The external clock pass to the 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.

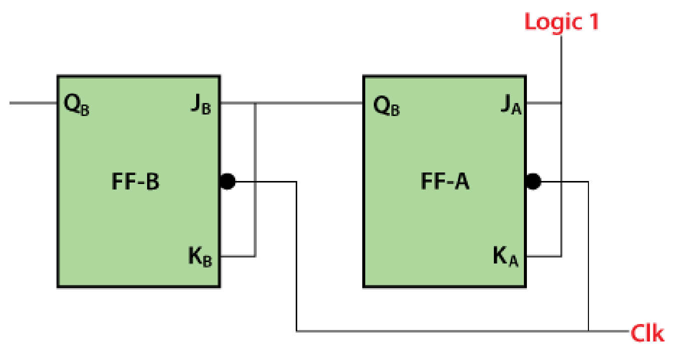
   So, $Q_A = 0$ and $Q_B = 0$

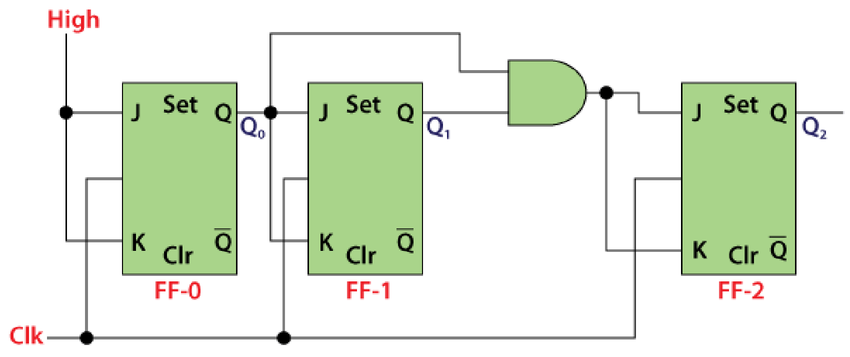
Synchronous counters

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.

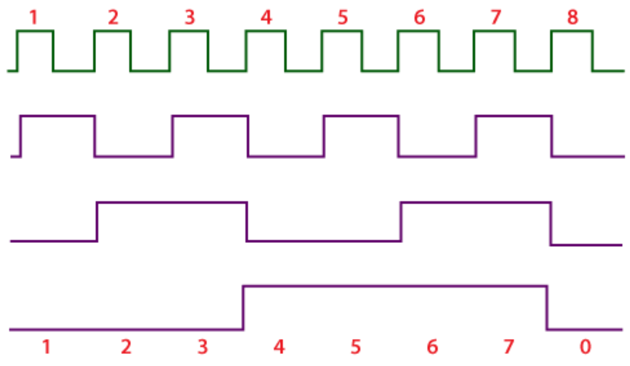
In the **synchronous counter**, the same clock pulse is passed to the clock input of all the flip flops. The clock signals produced by all the flip flops are the same as each other. Below is the diagram of a 2-bit synchronous counter in which the inputs of the first flip flop, i.e., FF-A, are set to 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, **Q_A = 0** and **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.

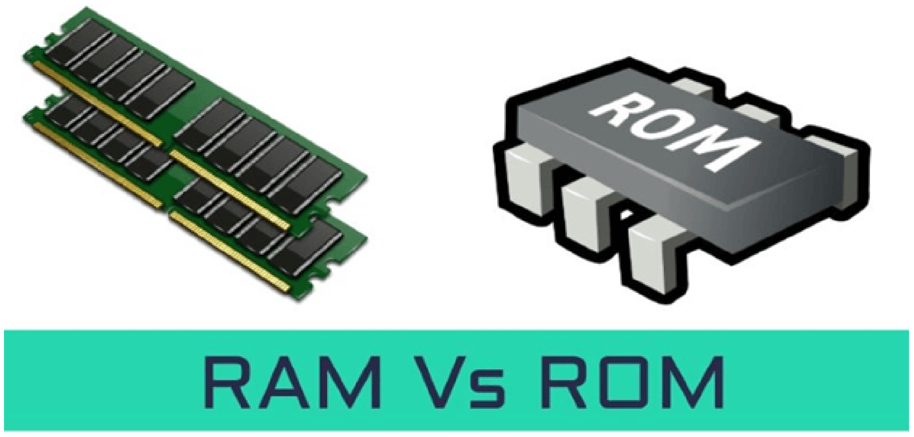
   So, **Q_A = 1** and **Q_B = 1**

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, **Q_A = 0** and **Q_B = 0**

Difference between RAM and ROM



Although RAM and ROM both are the internal memories of the computer, they are different from each other in terms of their uses, storage capacity, physical size, and more. Let us see how they differ from each other.

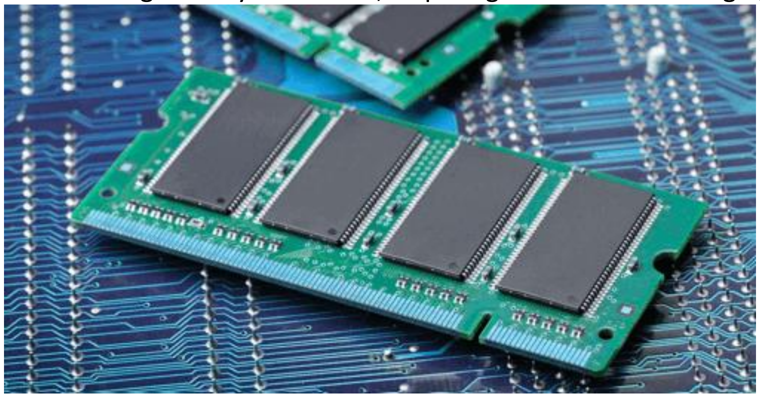
What is RAM:

RAM stands for Random Access Memory. It is the internal memory of the CPU in the form of a hardware device located on the motherboard of a computer. It is designed to store data, programs, and results of a program when a computer is switched on. It is the read and write memory of a computer as we can write information to it as well as read from it.

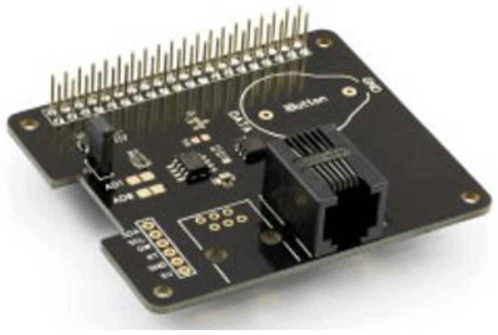
Furthermore, RAM is a volatile memory as it can?t store data and instructions permanently. For example, when we switch on a computer, the instructions from the hard disk are stored in the RAM. These instructions include the operating system (OS) and other programs which are needed to run a computer. CPU uses these instructions to perform the tasks required to run the computer. This data is retained by the RAM as long as the computer is on, the moment you shut it down, the RAM loses the data. The reason for transferring the data to RAM is that it is easy and fast to read data from RAM as compared to reading it from the hard drive.

Different Types of RAM:

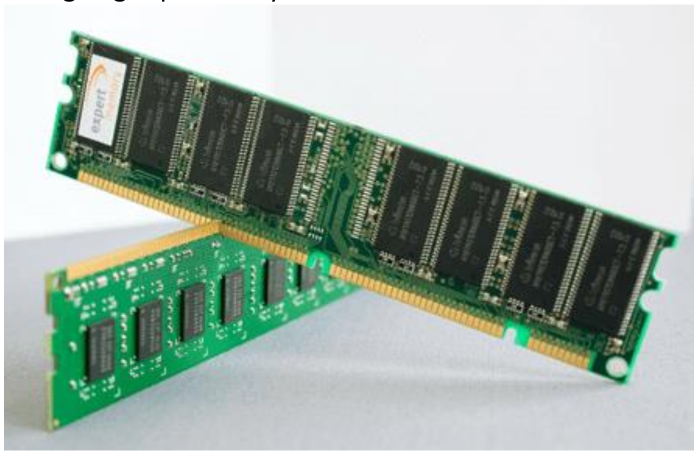
- **DRAM (Dynamic Random-access memory):** DRAM, or dynamic random-access memory, is the most common RAM type in computer systems nowadays. Every single bit of data is stored separately in a capacitor using an integrated circuit. But because of the nature of capacitors, the stored charge slowly drains out, requiring continuous recharging.



- **SRAM (Static Random-access memory):** Static Random-access memory, or SRAM, is a costly and fast kind of RAM. It does not require continual refreshing because flip-flops are used to store all the data.



- **DDR SDRAM (Double data rate Synchronous Dynamic Random-access memory):** DDR SDRAM is an improved version of DRAM. Transferring information on both the clock signal's growing and falling edges practically doubles the records transfer rate compared to traditional SDRAM.



DDR2, DDR3, and DDR4 are several DDR SDRAM generations, each of which gives faster speeds and better performance than the one earlier. New technology and faster data transfer speeds are introduced with each technology. The most recent DDR SDRAM version frequently used in current computers is DDR4.

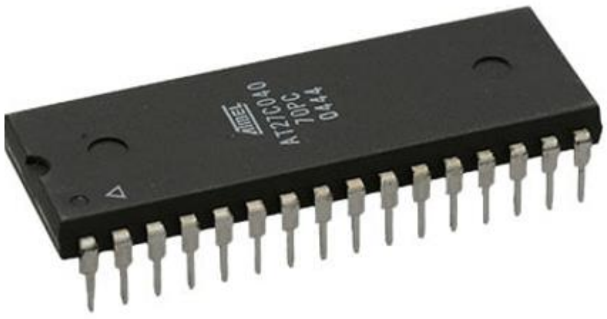
What is ROM:

ROM stands for read only memory. It is a non-volatile memory that stores information permanently, even when the power is turned off. Like RAM, it is also the primary memory of a computer. It is called read only memory as the programs and data stored in it can be read but cannot be written on it.

At the time of manufacturing, the manufacturer fills the ROM with programs that can?t be altered later. So, you cannot reprogram, rewrite, or erase its data after it is manufactured. However, in some types of ROM, you can modify the stored data. Some common examples of ROM include cartridge used in video game consoles, the data stored permanently on personal computers, and other electronic devices like smartphones, tablets, TV, AC, etc.

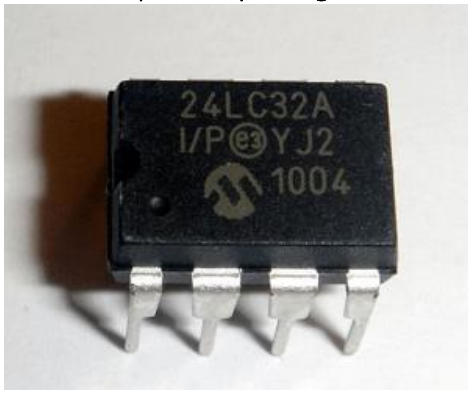
Different Types of ROM:

- **PROM (Programmable Read-Only Memory):** A PROM is a type of ROM that the user or the manufacturer may customize after creating the memory chip. A special tool called a PROM programmer is used to modify it; it changes or burns fuses in the memory cells to store the necessary data. The data in PROM cannot be modified after it has been programmed.



- **EPROM:** In contrast to PROM, EPROM can be cleaned and reprogrammed using ultraviolet (UV) radiation. Transparent windows on EPROM chips allow UV light exposure, erasing the inside data. For developing and testing firmware, EPROMs are frequently utilized.

- **EEPROM (Electrically Erasable Programmable Read-Only Memory):** EEPROM is an improved version of EPROM that can be erased and reprogrammed with electricity rather than UV light. EEPROM can keep its recorded data even when the power is off. It is frequently used to store small amounts of important information that is difficult to lose, such as BIOS settings that may occasionally need updating.



- **Flash Memory:** It is a kind of EEPROM that can erase or write to many memory cells simultaneously, making it quicker and more effective. It is often used in portable storage devices, including USB drives, solid-state drives (SSDs), memory cards, etc. Flash memory gives a large quantity of storage with comparatively quick access times and is non-volatile.
- **Mask ROM:** Mask ROM is a ROM that is programmed during the memory chip's manufacturing process. The information is kept in the memory cells forever and cannot be altered or deleted by the user. Frequently, firmware and other crucial data that should be kept the same are stored in mask ROM.

Some of the **key differences between RAM and ROM** are as follows:

RAM	**ROM**

It is a temporary memory of the computer.	It is the permanent memory of the computer.
It is a read-write memory. The data can be written and read.	It is a read only memory. The data can only be read.
It is a volatile memory as it temporarily stores the files as long as the computer is on and working.	It is a non-volatile memory as it permanently stores the files even when the power is turned off, such as game cartridge and BIOS program stored in the memory of a computer, etc.
The storage capacity ranges from 1 to 256 GB.	Its storage capacity ranges from 4 to 8 MB.
It is large in size than ROM. It comes in two different sizes for use in desktop computers and laptops. A desktop ram is around 5.5 inches in length and 1 inch in width. Whereas, the Laptop RAM is around half the length of desktop RAM.	Its size ranges from less than an inch in length to multiple inches in length and width based on their use. It has less capacity than RAM.
Data stored in RAM can be retrieved and altered.	We can only read the data stored in ROM. It cannot be altered.
It is faster than ROM as it is a high-speed memory.	It is slower than the RAM.
The data stored in RAM is used by the CPU in real-time to run the computer.	The data stored in ROM is used by CPU only when it is transferred to RAM.
It temporarily stores the files and data that the CPU needs to process the current instructions or work.	It stores the BIOS program on the motherboard of a computer, which is needed to bootstrap the computer.
Examples: It is used as CPU Cache, Primary Memory in a computer.	Examples: It is used as Firmware by micro-controllers.
The stored data is easy to access.	The stored data is not as easy to access as it is in ROM.

It is costlier than ROM.	It is cheaper than RAM.
Types: DRAM (Dynamic Random Access Memory), SRAM (Static Random Access Memory).	Types: PROM (programmable read-only memory), EPROM (erasable programmable read-only memory), EEPROM(electrically erasable programmable ROM), Mask ROM.

Characteristics of RAM

Here are some key characteristics of RAM:

- **Volatility:** RAM is a volatile memory that needs constant power to maintain data. The data kept in RAM is lost when the power is interrupted or shut off.
- **Speed:** Data may be accessed quickly, thanks to RAM. It is essential for a computer system's speedy and effective operation because it helps the CPU read and write information quickly.
- **Read and Write Operations:** Read and write operations are supported by RAM. It permits the reading and writing of data to and from its storage cells. The CPU can obtain and alter data using this capability as needed.
- **Storage Capacity:** Compared to other forms of memory, RAM generally has a bigger storage capacity.
- **Temporary Storage:** While the computer functions, RAM is a temporary storage location for data and program instructions. It contains the data that the processor is presently working on. The information saved in RAM is deleted whenever the power is switched off.

Characteristics of ROM

Here are some key characteristics of ROM:

- **Non-Volatility:** ROM is non-volatile memory, in contrast to RAM. Data is kept even when there is no constant power source. This quality makes it useful for storing crucial information and instructions that must be kept.
- **Stability:** As its contents cannot be readily changed or altered, ROM serves as a reliable storage medium.

- **Read-Only Access:** ROM is a type of memory that only enables data to be read from it, as the name suggests. The machine or user cannot alter or replace the stored information.
- **Storage Capacity:** When compared to RAM, ROM typically has a reduced storage capacity. It is often expressed in megabytes (MB) or kilobits (KB).

Advantages of RAM (Random Access Memory):

1. RAM provides quick access times, enabling the computer's CPU to access and alter data quickly. This speed influences the system's overall responsiveness and performance.
2. RAM is a volatile form of memory, requiring constant power to maintain data. Data may be quickly edited and updated in real-time, enabling quick and effective read and writes operations.
3. RAM is used by the computer when it is running as temporary storage for data and program instructions. This transient character allows for flexibility and adaptability.
4. RAM makes it possible for the computer to execute numerous programs at once.
5. RAM is quite simple to increase in most computer systems. The system's memory capacity may be increased by adding more RAM modules.

Advantages of ROM (Read-Only Memory):

1. ROM preserves its recorded data even when power is switched off or interrupted. It is useful for storing crucial firmware and instructions that shouldn't be altered or lost because of this property.
2. The information saved in ROM stays intact and unaffected by unintentional alterations or software mistakes, maintaining the integrity of essential instructions and firmware.
3. ROM offers long-term storage for important information, boot instructions, and firmware. This guarantees that crucial software elements needed for the system's functionality are constantly accessible.
4. ROM may help to increase system security. It is more difficult for hostile assaults or software vulnerabilities to attack the system since the data in ROM is read-only and cannot be changed. As a result, it prohibits unauthorized modification or manipulation of crucial program components.