B.C.A study

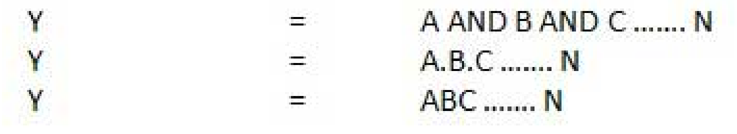
Unit -1 :Logic gates and circuit

Logic Gates

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

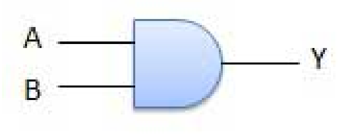
AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

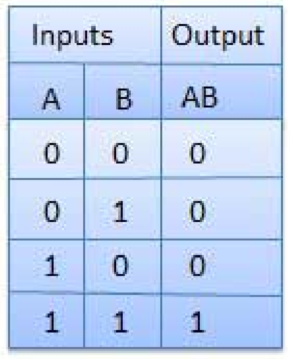


Y	=	A AND B AND C .... N
Y	=	A.B.C .... N
Y	=	ABC .... N

Logic diagram



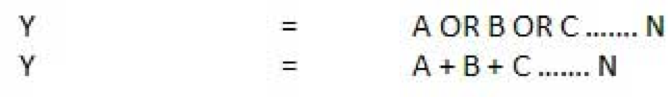
Truth Table



Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

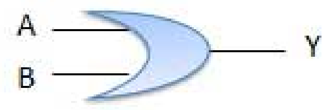
OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

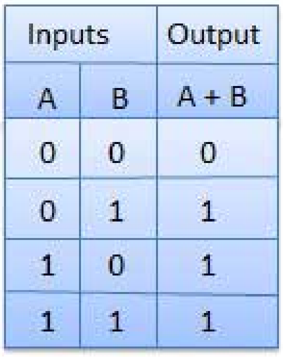


Y	=	A OR B OR C .... N
Y	=	A + B + C .... N

Logic diagram



Truth Table



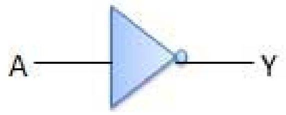
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Y	=	NOT A
Y	=	$\overline{A}$

Logic diagram



Truth Table

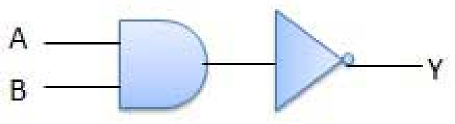
Inputs	Output
A	B
0	1
1	0

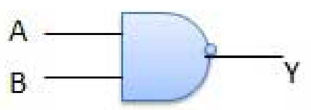
NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y	=	A NOT AND B NOT AND C .... N
Y	=	A NAND B NAND C .... N

Logic diagram





Truth Table

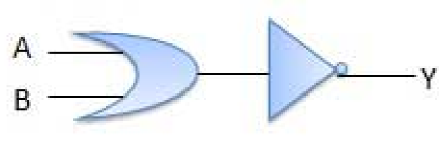
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

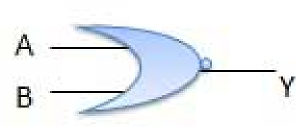
NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

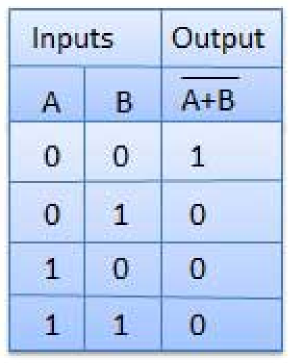
Y	=	A NOT OR B NOT OR C .... N
Y	=	A NOR B NOR C .... N

Logic diagram





Truth Table



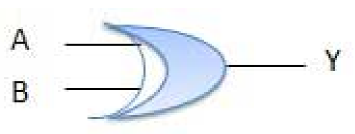
Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

XOR Gate

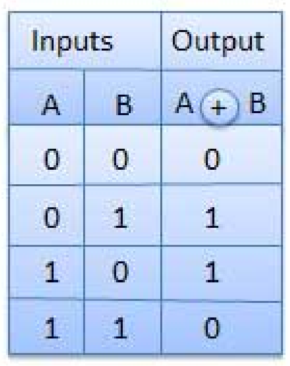
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$
$$Y = A \oplus B \oplus C \ldots\ldots N$$
$$Y = \overline{A}B + A\overline{B}$$

Logic diagram



Truth Table



Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

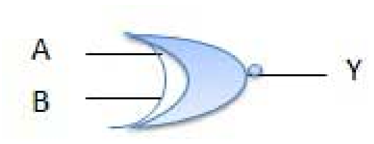
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

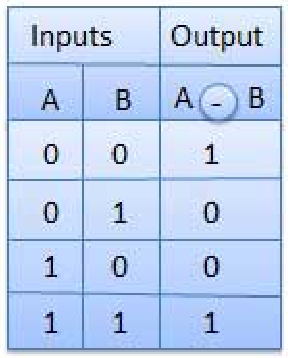
$$Y = A \odot B \odot C \ldots\ldots N$$

$$Y = \overline{A}\,\overline{B} + AB$$

Logic diagram



Truth Table



Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Laws of Boolean Algebra

Boolean Algebra uses a set of Laws and Rules to define the operation of a digital logic circuit **(https://www.electronics-tutorials.ws/boolean/bool_6.html#) (https://www.electronics-tutorials.ws/boolean/bool_6.html#)**

As well as the logic symbols "0" and "1" being used to represent a digital input or output, we can also use them as constants for a permanently "Open" or "Closed" circuit or contact respectively.

A set of rules or Laws of Boolean Algebra expressions have been invented to help reduce the number of logic gates needed to perform a particular logic operation resulting in a list of functions or theorems known commonly as the **Laws of Boolean Algebra**.

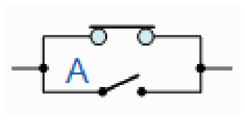
Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these "Laws of Boolean" to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic that has its own set of rules or laws which are used to define and reduce Boolean expressions.

The variables used in **Boolean Algebra** only have one of two possible values, a logic "0" and a logic "1" but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression, For example, variables A, B, C etc, giving us a logical expression of A + B = C, but each variable can ONLY be a 0 or a 1.

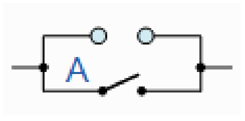
Examples of these individual laws of Boolean, rules and theorems for Boolean Algebra are given in the following table.

Truth Tables for the Laws of Boolean

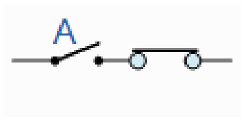
Switching CircuitBoolean Algebra
Law or Rule A + 1 = 1 A in parallel with
closed = "CLOSED"



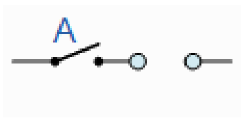
AnnulmentA + 0 = AA in parallel with
open = "A"



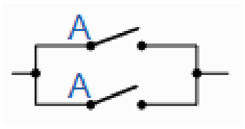
IdentityA . 1 = AA in series with
closed = "A"



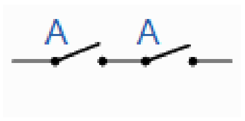
IdentityA . 0 = 0A in series with
open = “OPEN”



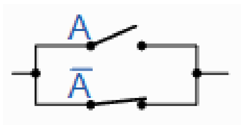
AnnulmentA + A = AA in parallel with
A = “A”



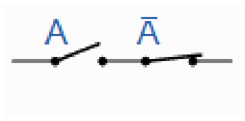
IdempotentA . A = AA in series with
A = “A”



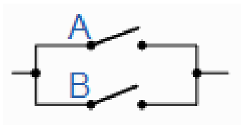
IdempotentNOT A = ANOT NOT A
(double negative) = “A” Double NegationA + A = 1A in parallel with
NOT A = “CLOSED”



ComplementA . A = 0A in series with
NOT A = “OPEN”



ComplementA+B = B+AA in parallel with B =
B in parallel with A



CommutativeA.B = B.AA in series with B =
B in series with A

The basic **Laws of Boolean Algebra** that relate to the *Commutative Law* allowing a change in position for addition and multiplication, the *Associative Law* allowing the removal of brackets for addition and multiplication, as well as the *Distributive Law* allowing the factoring of an expression, are the same as in ordinary algebra.

Each of the *Boolean Laws* above are given with just a single or two variables, but the number of variables defined by a single law is not limited to this as there can be an infinite number of variables as inputs too the expression. These Boolean laws detailed above can be used to prove any given Boolean expression as well as for simplifying complicated digital circuits.

A brief description of the various **Laws of Boolean** are given below with A representing a variable input.

Description of the Laws of Boolean Algebra

- de Morgan´s Theorem – There are two "de Morgan´s" rules or theorems,
- (1) Two separate terms NOR´ed together is the same as the two terms inverted (Complement) and AND´ed for example: A+B = A . B
- (2) Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example: A.B = A + B

Karnaugh Map

In this tutorial we will learning about Karnaugh Map.

We know that truth table is used to represent values of aboolean function. Similarly, **Karnaugh map** is another way of representing the values of aboolean function. So, K-map is a table consisting of cells which represents a Minterm or Maxterm. Karnaugh map or K-map is named after Mayrice **(https://en.wikipedia.org/wiki/Maurice_Karnaugh)**Karnough.

Karnaugh Map for Sum of Products

For SOP or Sum of Products, each cells in a K-map represents a Minterm. If there are n variables for a given boolean function then, the K-map will have 2^n cells. And we fill the cells with 1s whose Minterms output is 1. Lets check the K-map for 2, 3 and 4 variables.

2 variables K-map for Sum of Products

Say we have two variables X and Y then, there will be $2^2 = 4$ cells in the K-map.

Each cell has a subscripted number at the bottom right corner. It is the value of the minterm. So, if a cell has number 2 then it represent minterm m_2.

3 variables K-map for Sum of Products

Say we have three variables X, Y and Z then there will be $2^3 = 8$ cells in the K-map.

Each cell has a subscripted number at the bottom right corner it is the value of the minterm. So, if a cell has number 7 then it represent minterm m_7.

Now if we look at the above K-map we will see that the numbering scheme is 0, 1, 3, 2 in the first row and 4, 5, 7, 6 in the second row. So, the numbers differ in one place when moving from left to right.

00, 01, 11, 10 this is done so that only one variable change at a time from complement to un-complement or un-complement to complement every row.

Example, in the first row we have **X'Y'Z', X'Y'Z, X'YZ and X'YZ'** so, only one variable change at a time as we move from left to right.

1st row: moving left to right cell-wise
X'Y'Z' —> X'Y'Z [Z' changed to Z]
X'Y'Z —> X'YZ [Y' changed to Y]
X'YZ —> X'YZ' [Z changed to Z']

4 variables K-map for Sum of Products

Say we have four variables W, X, Y and Z then there will be 2^4 = 16 cells in the K-map.

Each cell has a subscripted number at the bottom right corner. It is the value of the minterm. So, if a cell has number 12 then, it represent minterm m_{12}.

And in this case the numbering scheme is 0, 1, 3, 2 in the first row 4, 5, 7, 6 in the second row 12, 13, 15, 14 in the third row and 8, 9, 11, 10 in the fourth row.

How to fill values in the K-map for Sum of Products?

Say we have a boolean function F defined on two variables A and B and F = 1 i.e., output is 1 only when exactly one of the input is 1. So, we can draw the following truth table with all possible input and output values.

To express F using Minterm shorthand notation we have to consider only those minterms for which F = 1.

$F = m_1 + m_2$

This can also be written as
$F = \sum(1, 2)$

Now, we have $F = m_1 + m_2 = \sum(1, 2)$
Draw an empty K-map having 4 cells (since there are 2 variables so 2_2 = 4 cells).

$F = m_1 + m_2$
So, fill the cells marked with subscript 1 and 2 with value 1. And fill rest of the cells of the K-map with value 0.

In a similarly way we can fill 3 and 4 variables K-map.

Karnaugh Map for Product of Sums

For POS each cells in a K-map represents a Maxterm. If there are n variables for a given boolean function then the K-map will have 2n cells. And we fill the cells with 0s whose Maxterm output is 0. Lets check the K-map for 2, 3 and 4 variables.

2 variables K-map for Product of Sums

Say we have two variables X and Y then there will be 2^2 = 4 cells in the K-map.

Each cell has a subscripted number at the bottom right corner. It is the value of the maxterm. So, if a cell has number 2 then, it represent maxterm M_2.

3 variables K-map for Product of Sums

Say we have three variables X, Y and Z then there will be 2^3 = 8 cells in the K-map. So, if a cell has number 0 then it represent maxterm M_0.

Now if we look at the above K-map we will see that the numbering scheme is 0, 1, 3, 2 in the first row and 4, 5, 7, 6 in the second row. So, the numbers differ in one place when moving from left to right.

00, 01, 11, 10 this is done so that only one variable change at a time from complement to un-complement or un-complement to complement every row.

Example, in the first row we have **X+Y+Z, X+Y+Z′, X+Y′+Z′ and X+Y′+Z** so, only one variable change at a time as we move from left to right.

1st row: moving left to right cell-wise
X+Y+Z —> X+Y+Z′ [Z changed to Z′]
X+Y+Z′ —> X+Y′+Z′ [Y changed to Y′]
X+Y′+Z′ —> X+Y′+Z [Z′ changed to Z]

4 variables K-map for Product of Sums

Say we have four variables W, X, Y and Z then there will be $2^4 = 16$ cells in the K-map.

Each cell has a subscripted number at the bottom right corner. It is the value of the maxterm. So, if a cell has number 15 then it represent maxterm M_{15}.

And in this case the numbering scheme is 0, 1, 3, 2 in the first row 4, 5, 7, 6 in the second row 12, 13, 15, 14 in the third row and 8, 9, 11, 10 in the fourth row.

How to fill values in the K-map for Product of Sums

Say we have a boolean function F defined on two variables A and B and F = 1 i.e., output is 1 only when exactly one of the input is 1. So we can draw the following truth table with all possible input and output values.

To express F using Maxterm shorthand notation we have to consider only those maxterms for which F = 0.

$F = M_0 . M_3$

This can also be written as
$F = \prod(0, 3)$

Now that we have $F = M_0 . M_3 = \prod(0, 3)$.

Draw an empty K-map having 4 cells (since there are 2 variables so $2^2 = 4$ cells)

$F = M_0 . M_3$
So, fill the cells marked with subscript 0 and 3 with value 0. And fill rest of the cells of the K-map with value 1.

In a similarly way we can fill 3 and 4 variables K-map.

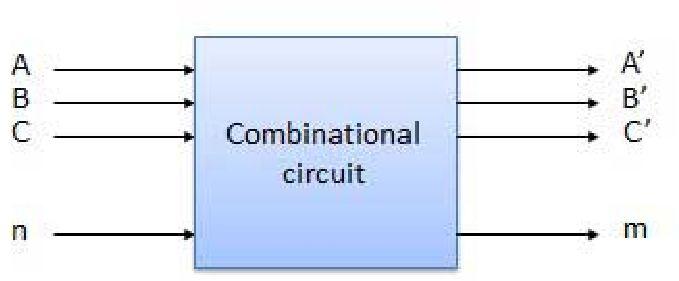
<u>B.C.A study</u>

Unit -2 :Combinational Building Blocks

Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following −

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.

Block diagram

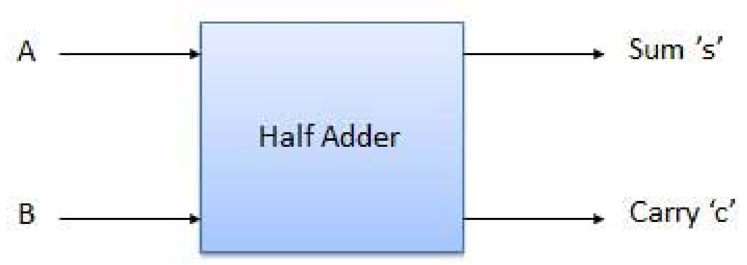


We're going to elaborate few important combinational circuits as follows.

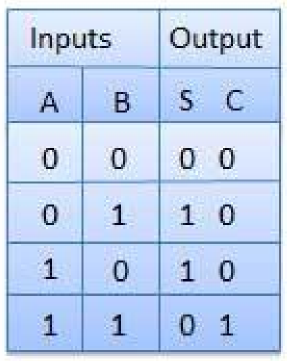
Half Adder

Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

Block diagram

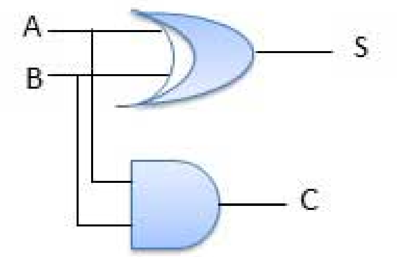


Truth Table



Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

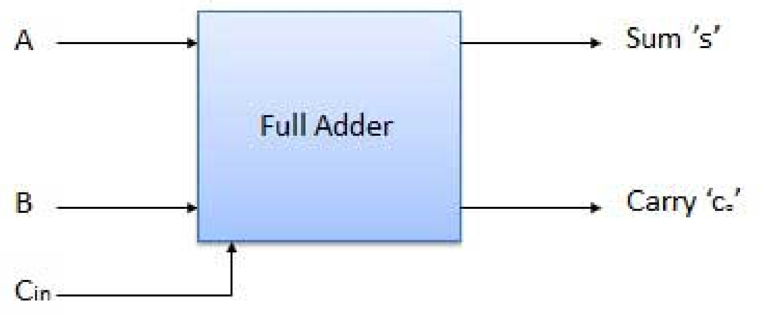
Circuit Diagram



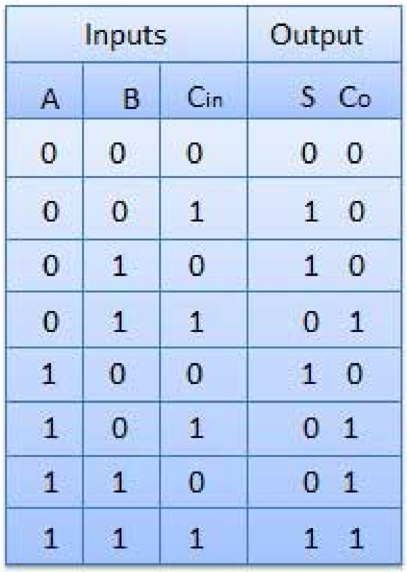
Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

Block diagram

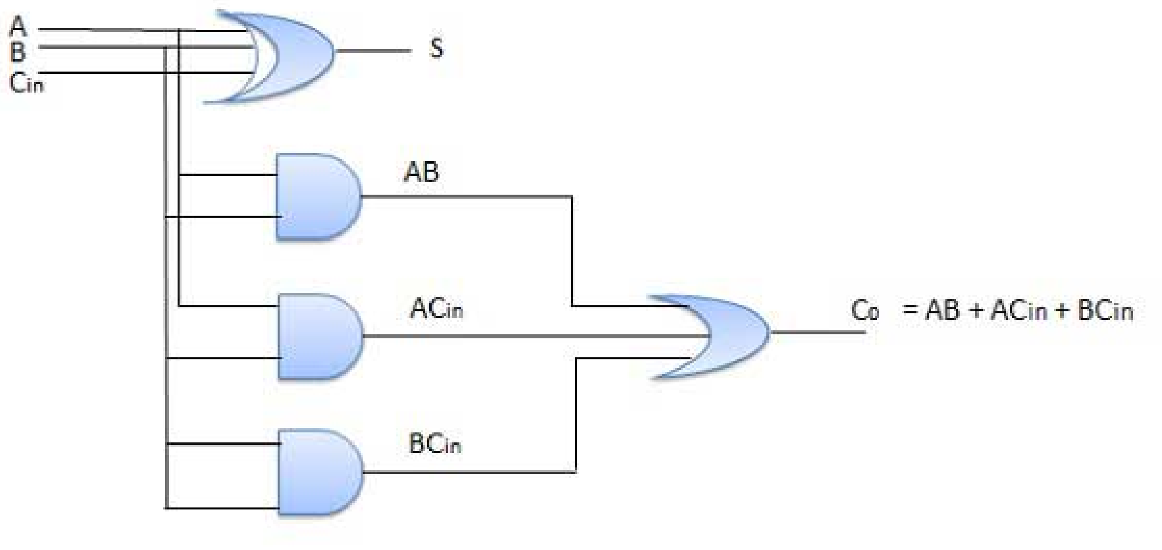


Truth Table



Inputs			Output	
A	B	C_{in}	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Circuit Diagram



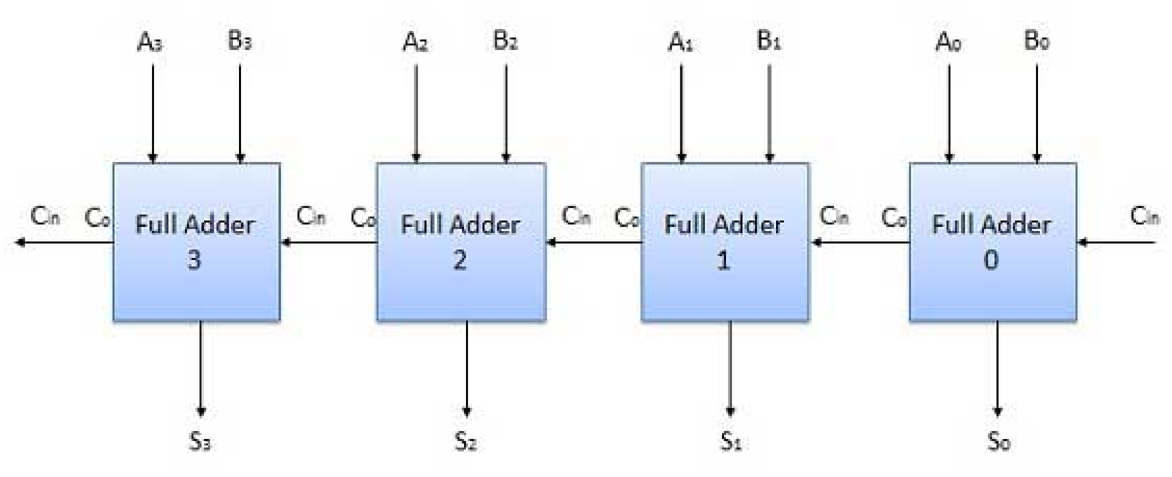
N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

Block diagram



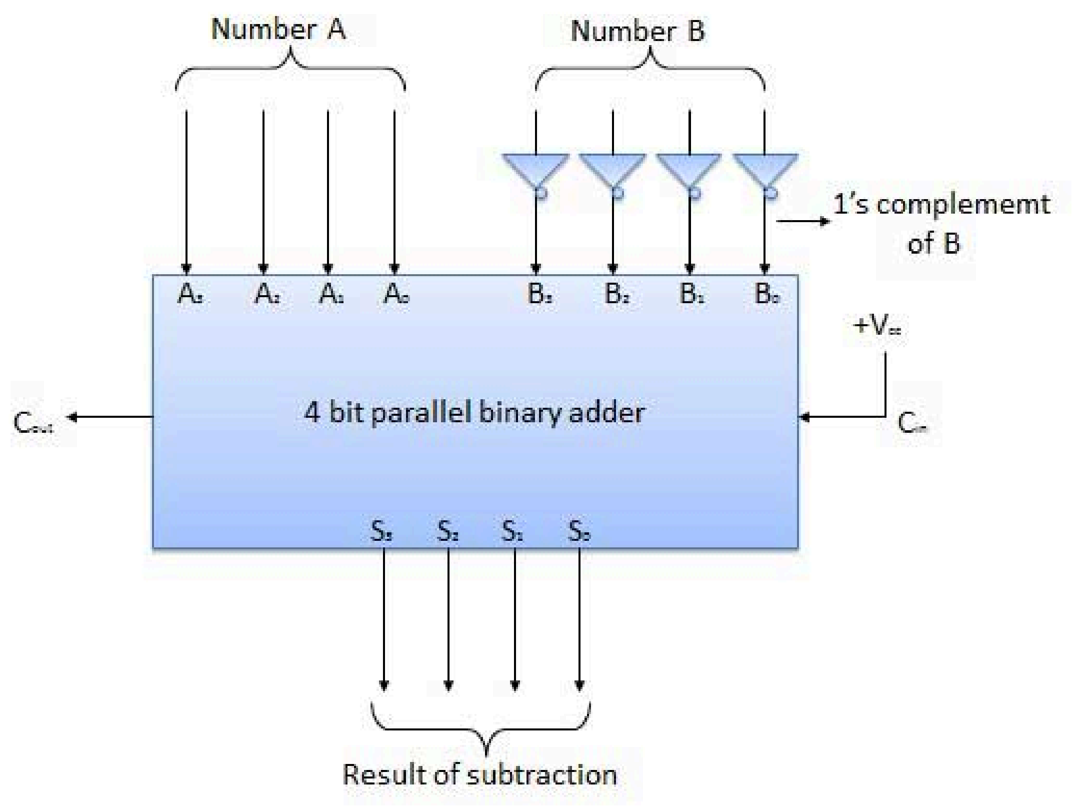
N-Bit Parallel Subtractor

The subtraction can be carried out by taking the 1′s or 2′s complement of the number to be subtracted. For example we can perform the subtraction (A-B) by adding either 1′s or 2′s complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted (B) is first passed through inverters to obtain its 1′s complement. The 4-bit adder then adds A and 2′s complement of B to produce the subtraction. S_3 S_2 S_1 S_0represents the result of binary subtraction (A-B) and carry output C_{out} represents the polarity of the result. If A > B then Cout = 0 and the result of binary form (A-B) then C_{out} = 1 and the result is in the 2’s complement form.

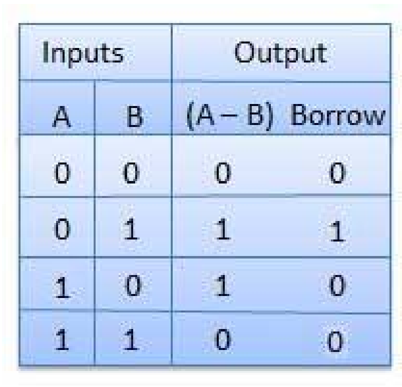
Block diagram



Half Subtractors

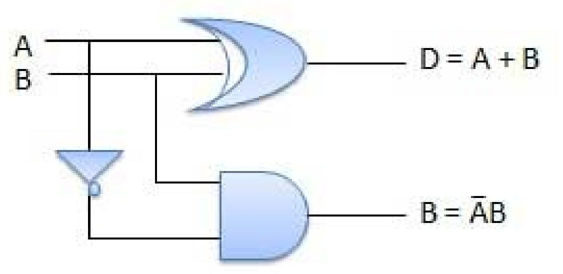
Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Truth Table



Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

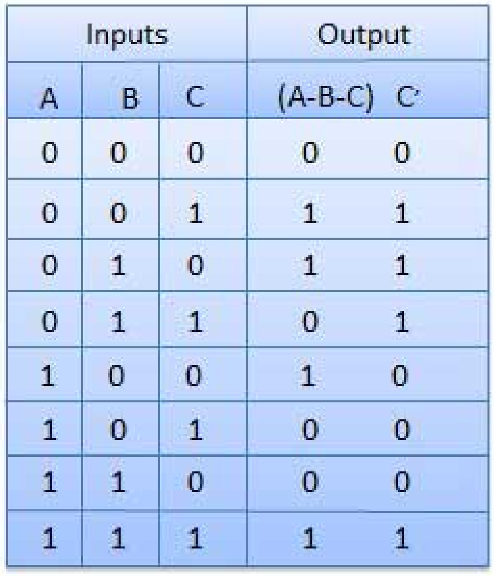
Circuit Diagram



Full Subtractors

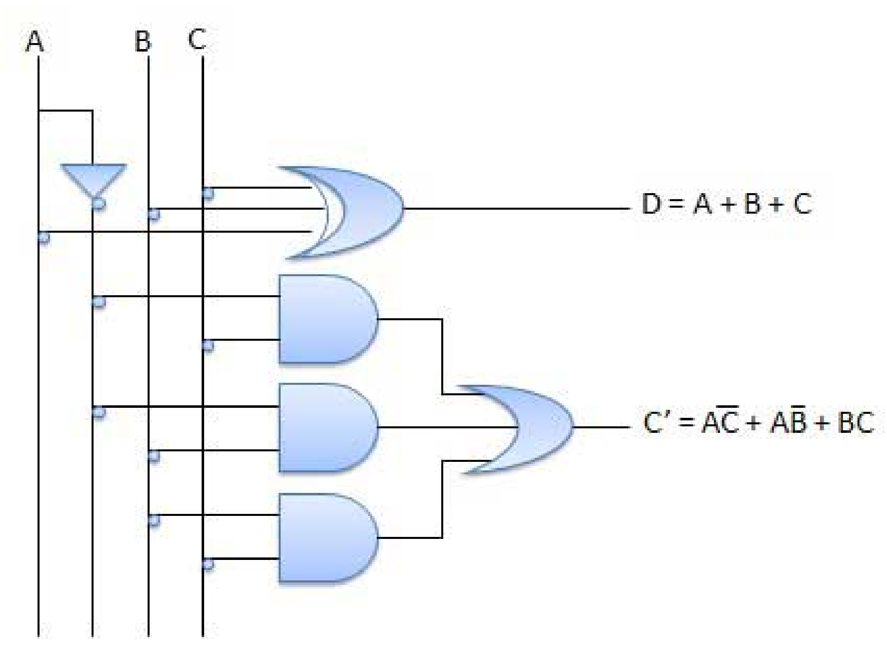
The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table



Inputs			Output	
A	B	C	(A-B-C)	C'
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

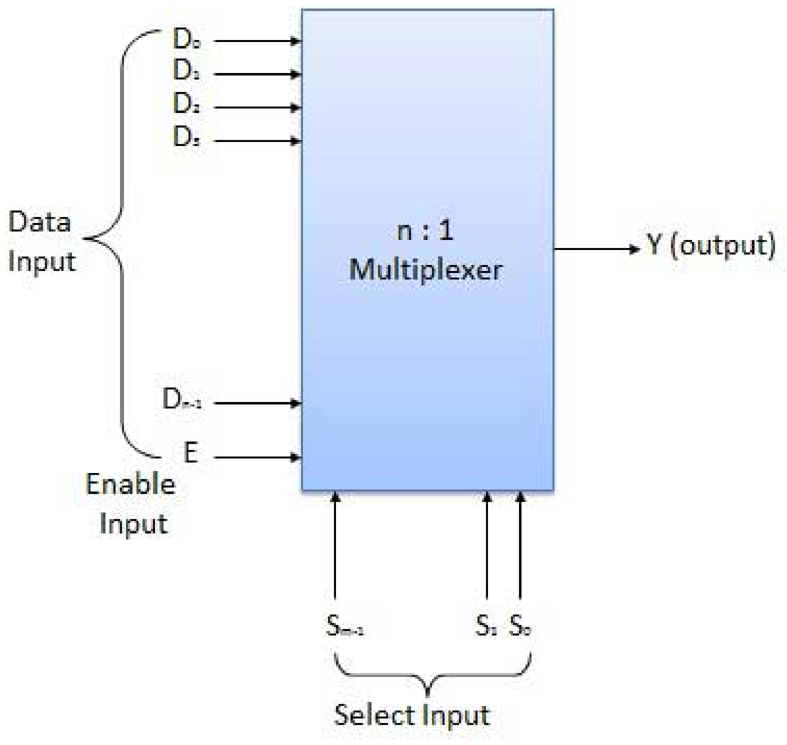
Circuit Diagram



Multiplexers

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with 2m = n. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the strobe or enable input which is useful for the cascading. It is generally an active low terminal that means it will perform the required operation when it is low.

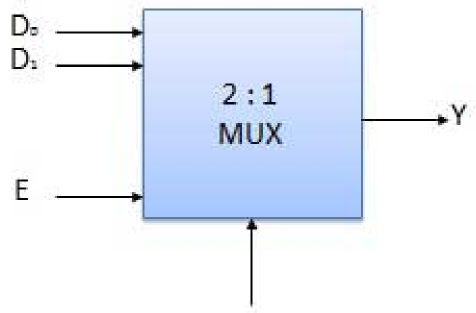
Block diagram



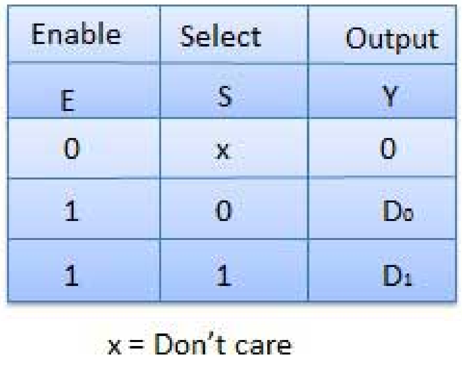
Multiplexers come in multiple variations

- 2 : 1 multiplexer
- 4 : 1 multiplexer
- 16 : 1 multiplexer
- 32 : 1 multiplexer

Block Diagram



Truth Table



Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

x = Don't care

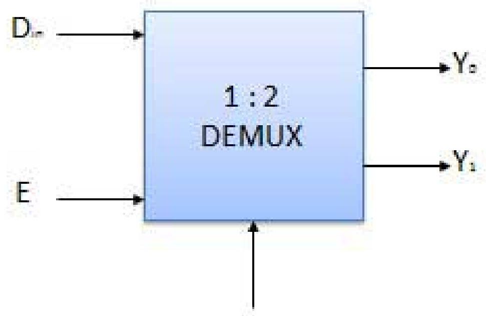
Demultiplexers

A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in fig.

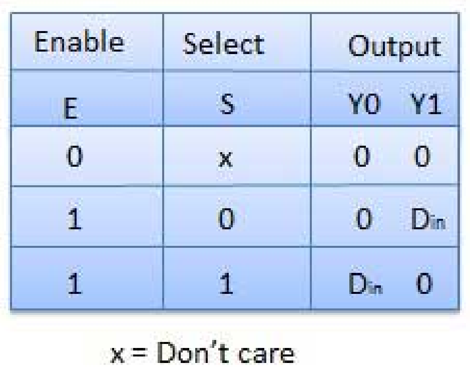
Demultiplexers comes in multiple variations.

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

Block diagram



Truth Table



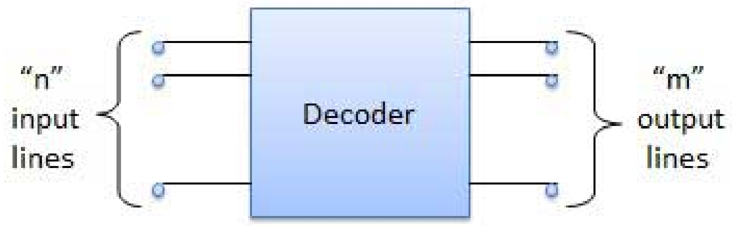
Enable	Select	Output	
E	S	Y0	Y1
0	x	0	0
1	0	0	D_{in}
1	1	D_{in}	0

x = Don't care

Decoder

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Block diagram



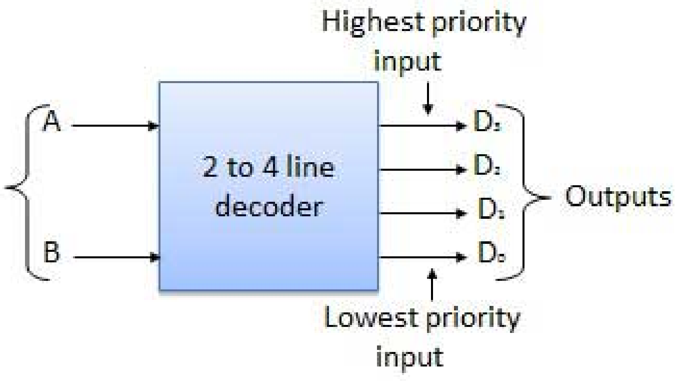
Examples of Decoders are following.

- Code converters
- BCD to seven segment decoders
- Nixie tube decoders
- Relay actuator

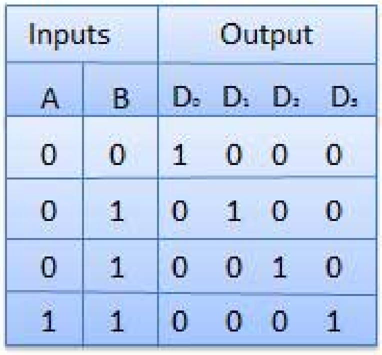
2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

Block diagram

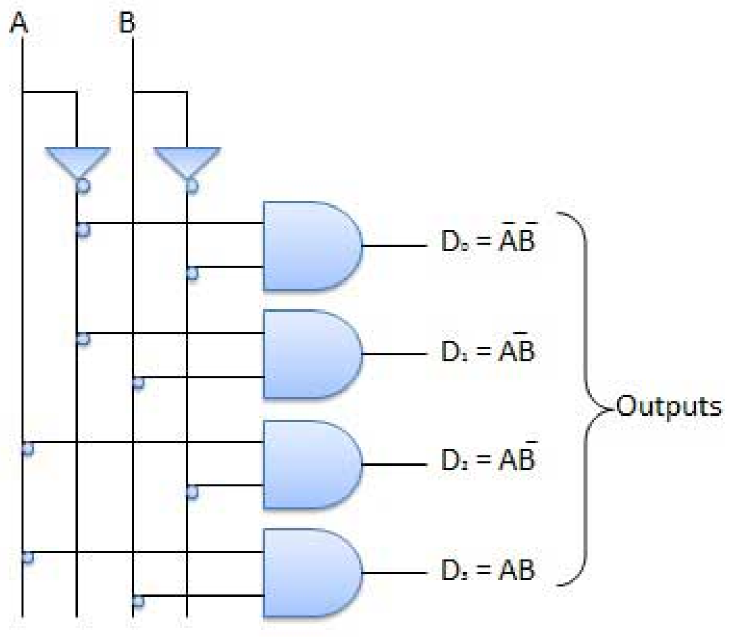


Truth Table



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

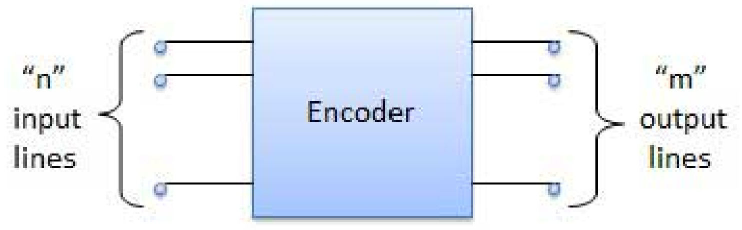
Logic Circuit



Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.

Block diagram



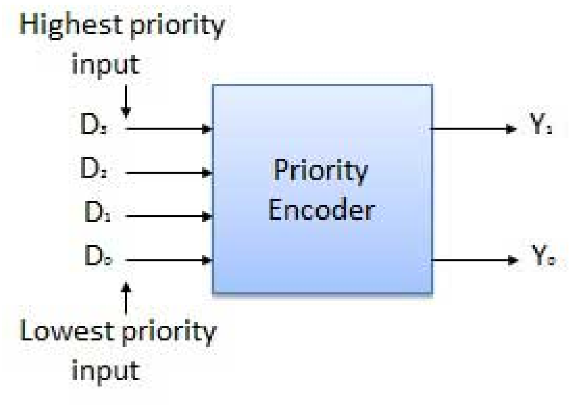
Examples of Encoders are following.

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

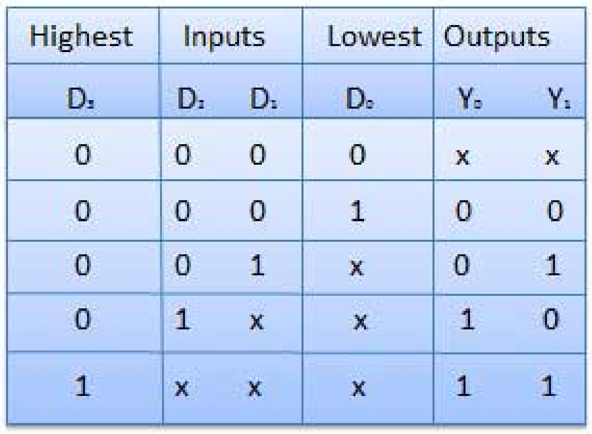
Priority Encoder

This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered. There are four input D_0, D_1, D_2, D_3 and two output Y_0, Y_1. Out of the four input D_3 has the highest priority and D_0 has the lowest priority. That means if $D_3 = 1$ then $Y_1Y_1 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then Y_1 $Y_0 = 10$ irrespective of the other inputs.

Block diagram

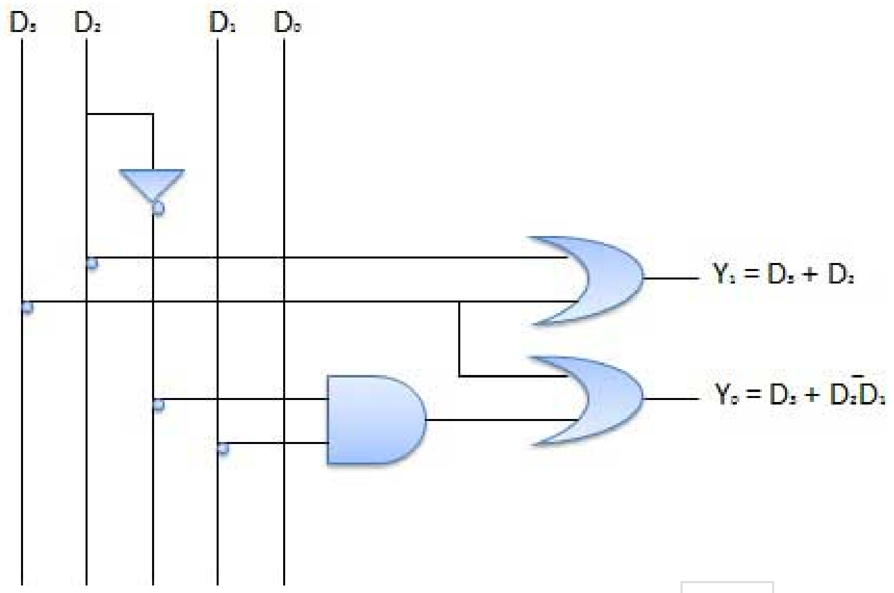


Truth Table



Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_0	Y_1
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

Logic Circuit



•••

A WordPress.com Website.

B.C.A study

Unit -3 :Memories

Memory Devices

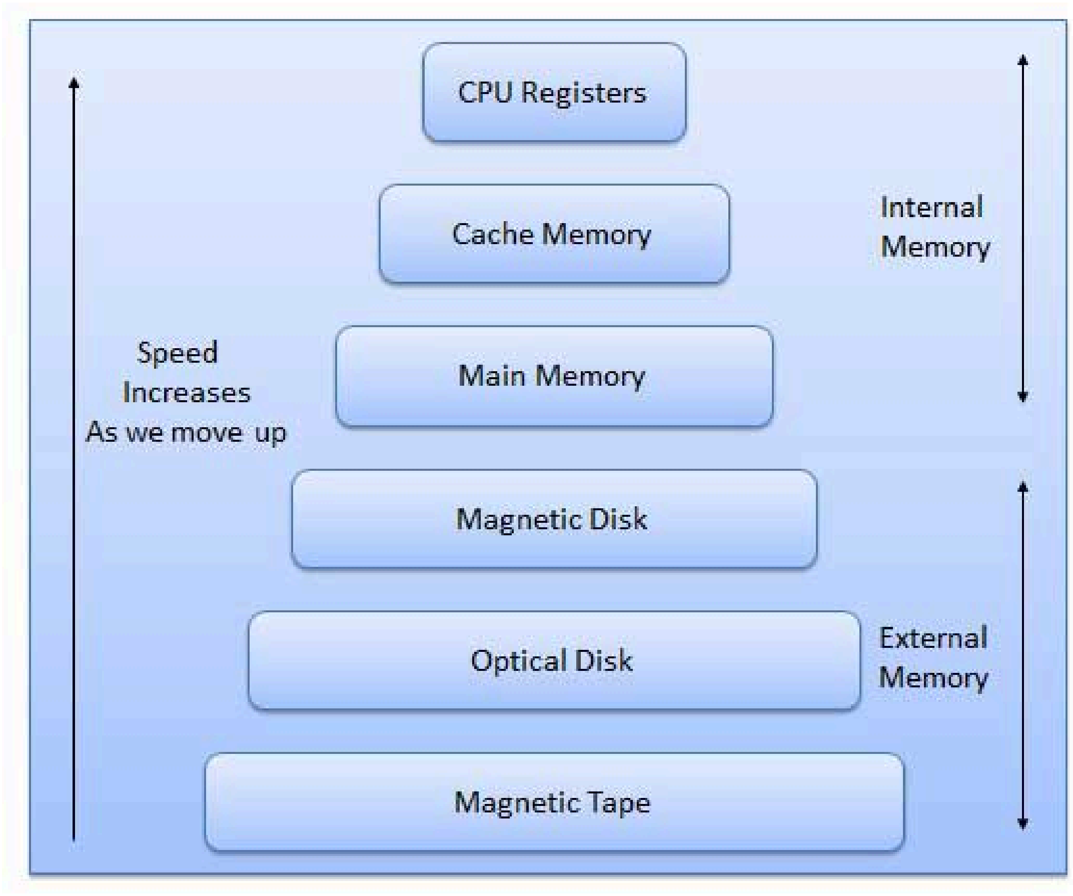
A memory is just like a human brain. It is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.

The memory is divided into large number of small parts. Each part is called a cell. Each location or cell has a unique address which varies from zero to memory size minus one.

For example if computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory location. The address of these locations varies from 0 to 65535.

Memory is primarily of two types

- **Internal Memory** − cache memory and primary/main memory
- **External Memory** − magnetic disk / optical disk etc.



Characteristics of Memory Hierarchy are following when we go from top to bottom.

- Capacity in terms of storage increases.
- Cost per bit of storage decreases.
- Frequency of access of the memory by the CPU decreases.
- Access time by the CPU increases.

RAM

A RAM constitutes the internal memory of the CPU for storing data, program and program result. It is read/write memory. It is called random access memory (RAM).

Since access time in RAM is independent of the address to the word that is, each storage location inside the memory is as easy to reach as other location & takes the same amount of time. We can reach into the memory at random & extremely fast but can also be quite 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 remains applied. 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 have to be refreshed on a regular basis.

Because of the extra space in the matrix, SRAM uses more chips than DRAM for the same amount of storage space, thus making the manufacturing costs higher.

Static RAM is used as cache memory needs to be very fast and small.

Dynamic RAM (DRAM)

DRAM, unlike SRAM, must be continually **refreshed** in order for it 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 because it is cheap and small. All DRAMs are made up of memory cells. These cells are composed of one capacitor and one transistor.

ROM

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 instruction as are required to start computer when electricity is first turned on, this operation is referred to as bootstrap. ROM chip are not only used in the computer but also in other electronic items like washing machine and microwave oven.

Following are the various types of ROM −

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. It is inexpensive ROM.

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 programmer. 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)

The EPROM can be erased by exposing it to ultra-violet light for a duration of upto 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 ten 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)

The 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 re-programming is flexible but slow.

Serial Access Memory

Sequential access means the system must search the storage device from the beginning of the memory address until it finds the required piece of data. Memory device which supports such access is called a Sequential Access Memory or Serial Access Memory. Magnetic tape is an example of serial access memory.

Direct Access Memory

Direct access memory or Random Access Memory, refers to conditions in which a system can go directly to the information that the user wants. Memory device which supports such access is called a Direct Access Memory. Magnetic disks, optical disks are examples of direct access memory.

Cache Memory

Cache memory is a very high speed semiconductor memory which can speed up CPU. It acts as a buffer between the CPU and main memory. It is used to hold those parts of data and program which are most frequently used by CPU. The parts of data and programs, are transferred from disk to cache memory by operating system, from where CPU can access them.

Advantages

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.
- It stores data for temporary use.

Disadvantages

- Cache memory has limited capacity.
- It is very expensive.

How Does Cache Memory Work?

Cache memory works by taking data or instructions at certain memory addresses in RAM and copying them into the cache memory, along with a record of the original address of those instructions or data.

This results in a table containing a small number of RAM memory addresses, and copies of the instructions or data that those RAM memory address contain.

Memory Cache “Hit”

When the processor requires instructions or data from a given RAM memory address, then before retrieving them from RAM it checks to see if the cache memory contains a reference to that RAM memory address. If it does, then it reads the corresponding data or instructions from the cache memory instead of from RAM. This is known as a “cache hit”. Since the cache memory is faster than RAM, and because it is located closer to the CPU, it can get and start processing the instructions and data much more quickly.

The same procedure is carried out when data or instructions need to be written back to memory. However, in this case there is an additional step because if anything is written to cache memory than ultimately it must also be written RAM.

Memory Cache “Miss”

If data or instructions at a given RAM memory address are not found in cache memory, then this is known as a “cache miss.” In this case, the CPU is forced to wait while the information is retrieved from RAM.

In fact, the data or instructions are retrieved from RAM and written to cache memory, and then sent on to the CPU. The reason for this is that data or instructions that have been recently used are very likely to be required again in the near future. So anything that the CPU requests from RAM is always copied to cache memory.

This begs the question of what happens if the cache memory is already full. The answer is that some of the contents of the cache memory has to be “evicted” to make room for the new information that needs to be written there.

If a decision needs to be made then the memory cache will apply a "replacement policy" to decide which information is evicted.

There are a number of possible replacement policies. One of the most common ones is a least recently used (LRU) policy. This policy uses the principal that if data or instructions have not been used recently, then they are less likely to be required in the immediate future than data or instructions that have been required more recently.

Types of Cache Memory

- **Primary Cache** Most cache memory is physically located on the same die as the CPU itself, and the part closest to the CPU cores is sometimes called primary cache, although the term is not commonly used any more.
- **Secondary Cache** This often refers to a further piece of cache memory, which is located on a separate chip on the motherboard close to the CPU. This term is also not commonly used anymore, because most cache memory is now located on the CPU die itself.

Levels of Cache Memory

Modern computer systems have more than one piece of cache memory, and these caches vary in size and proximity to the processor cores, and therefore also in speed. These are known as cache levels.

The smallest and fastest cache memory is known as Level 1 cache, or L1 cache, and the next is L2 cache. Most systems now have L3 cache, and since the introduction of its Skylake chips, Intel has added L4 cache to some of its processors as well.

Level 1

L1 cache is cache memory that is built into the CPU itself. It runs at the same clock speed as the CPU. It is the most expensive type of cache memory so its size is extremely limited. But because it is very fast it is the first place that a processor will look for data or instructions that may have been buffered there from RAM.

In fact, in most modern CPUs, the L1 cache is divided into two parts: a data section (L1d) and an instruction section (L1i). These hold data and instructions, respectively.

A modern CPU may have a cache size on the order of 32 KB of L1i and L1d per core.

Level 2

L2 cache may also be located in the CPU chip, although not as close to the core as L1 cache. Or more rarely, it may be located on a separate chip close to the CPU. L2 caches are less expensive and larger than L1 caches, so L2 cache sizes tend to be larger, and may be of the order of 256 KB per core.

Level 3

Level 3 cache tends to be much larger than either L1 or L2 cache, but it also different in another important way. Whereas L1 and L2 caches are private to each core of a processor, L3 tends to be a shared cache that is common to all the cores. This allows it to play an important role in data sharing and inter-core communication. L3 cache may be of the order of 2 MB per core.

Auxiliary Memory

Auxiliary memory is much larger in size than main memory but is slower. It normally stores system programs, instruction and data files. It is also known as secondary memory. It can also be used as an overflow/virtual memory in case the main memory capacity has been exceeded. Secondary memories cannot be accessed directly by a processor. First the data/information of auxiliary memory is transferred to the main memory and then that information can be accessed by the CPU. Characteristics of Auxiliary Memory are following −

- **Non-volatile memory** − Data is not lost when power is cut off.
- **Reusable** − The data stays in the secondary storage on permanent basis until it is not overwritten or deleted by the user.
- **Reliable** − Data in secondary storage is safe because of high physical stability of secondary storage device.
- **Convenience** − With the help of a computer software, authorised people can locate and access the data quickly.
- **Capacity** − Secondary storage can store large volumes of data in sets of multiple disks.
- **Cost** − It is much lesser expensive to store data on a tape or disk than primary memory.

- **Floppy Disk:** A floppy disk is a flexible disk with a magnetic coating on it. It is packaged inside a protective plastic envelope. These are one of the oldest type of portable storage devices that could store up to 1.44 MB of data but now they are not used due to very less memory storage.
- **Hard disk:** A hard disk consists of one or more circular disks called platters which are mounted on a common spindle. Each surface of a platter is coated with a magnetic material. Both surfaces of each disk are capable of storing data except the top and bottom disk where only the inner surface is used. The information is recorded on the surface of the rotating disk by magnetic read/write heads. These heads are joined to a common arm known as access arm.**Hard disk drive components:**
  Most of the basic types of hard drives contains a number of disk platters that are placed around a spindle which is placed inside a sealed chamber. The chamber also includes read/write head and motors. Data is stored on each of these disks in the arrangement of concentric circles called tracks which are divided further into sectors. Though internal Hard drives are not very portable and used internally in a computer system, external hard disks can be used as a substitute for portable storage. Hard disks can store data upto several terabytes.

CD- ROM -It stands for Compact Disk – Read Only Memory.Data is written on these disks at the time of manufacture. This data cannot be changed, once is it written by the manufacturer, but can only be read. CD- ROMs are used for text, audio and video distribution like games, encyclopedias and application software.

B.C.A study

Unit -4 :Sequential building Blocks:

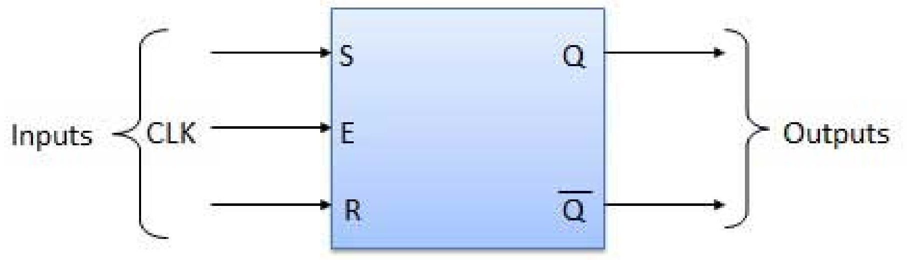
Flip Flop

Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip flop is said to be edge sensitive or edge triggered rather than being level triggered like latches.

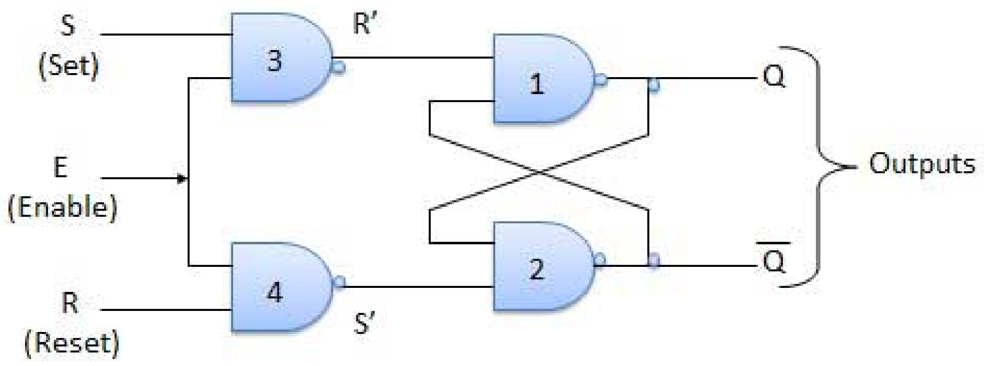
S-R Flip Flop

It is basically S-R latch using NAND gates with an additional **enable** input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

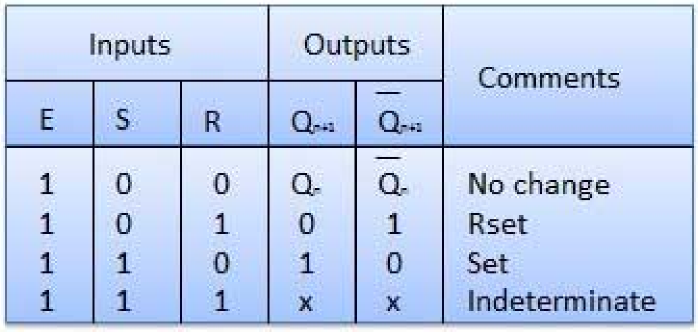
Block Diagram



Circuit Diagram



Truth Table



Inputs			Outputs		Comments
E	S	R	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	x	x	Indeterminate

Operation

S.N.ConditionOperation1**S = R = 0 : No change**

If S = R = 0 then output of NAND gates 3 and 4 are forced to become 1.

Hence R′ and S′ both will be equal to 1. Since S′ and R′ are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.2**S = 0, R = 1, E = 1**

Since S = 0, output of NAND-3 i.e. R′ = 1 and E = 1 the output of NAND-4 i.e. S′ = 0.

Hence Q_{n+1} = 0 and Q_{n+1}bar = 1. This is reset condition.3**S = 1, R = 0, E = 1**

Output of NAND-3 i.e. R′ = 0 and output of NAND-4 i.e. S′ = 1.

Hence output of S-R NAND latch is $Q_{n+1} = 1$ and Q_{n+1} bar = 0. This is the reset condition.4**S = 1, R = 1, E = 1**

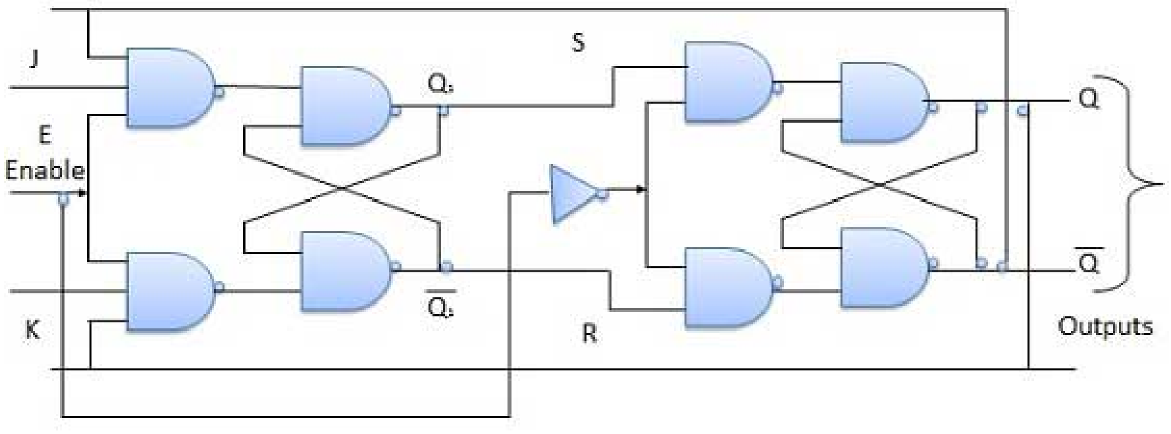
As S = 1, R = 1 and E = 1, the output of NAND gates 3 and 4 both are 0 i.e. S′ = R′ = 0.

Hence the **Race** condition will occur in the basic NAND latch.

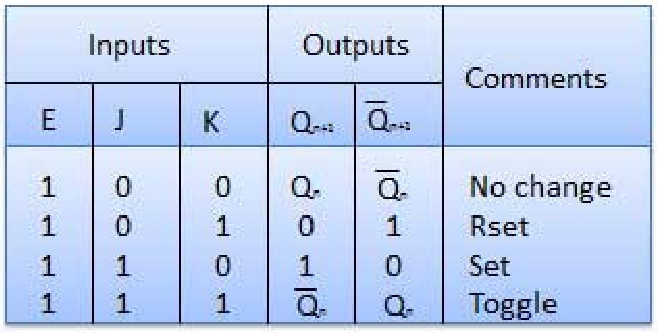
Master Slave JK Flip Flop

Master slave JK FF is a cascade of two S-R FF with feedback from the output of second to input of first. Master is a positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and master is inactive.

Circuit Diagram



Truth Table



Inputs			Outputs		Comments
E	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	$\overline{Q}_n$	Q_n	Toggle

Operation

S.N.ConditionOperation1**J = K = 0 (No change)**

When clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged. Therefore outputs will not change if J = K =0.2**J = 0 and K = 1 (Reset)**

Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q_1 = 0 and Q_1 bar = 1. That means S = 0 and R =1.

Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 0 and Q bar = 1.

Again clock = 1 − Master active, slave inactive. Therefore even with the changed outputs Q = 0 and Q bar = 1 fed back to master, its output will be Q1 = 0 and Q1 bar = 1. That means S = 0 and R = 1.

Hence with clock = 0 and slave becoming active the outputs of slave will remain Q = 0 and Q bar = 1. Thus we get a stable output from the Master slave.3**J = 1 and K = 0 (Set)**

Clock = 1 − Master active, slave inactive. Therefore outputs of the master become Q_1 = 1 and Q_1 bar = 0. That means S = 1 and R =0.

Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 1 and Q bar = 0.

Again clock = 1 − then it can be shown that the outputs of the slave are stabilized to Q = 1 and Q bar = 0.4**J = K = 1 (Toggle)**

Clock = 1 − Master active, slave inactive. Outputs of master will toggle. So S and R also will be inverted.

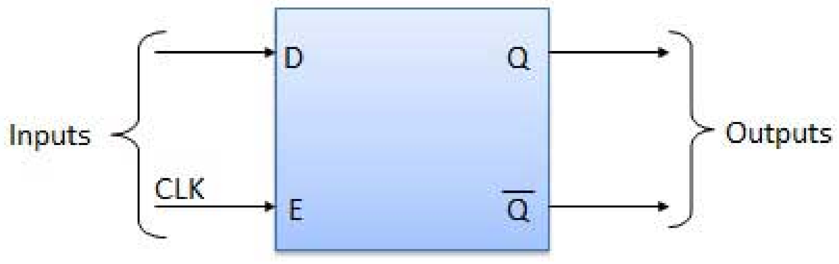
Clock = 0 − Slave active, master inactive. Outputs of slave will toggle.

These changed output are returned back to the master inputs. But since clock = 0, the master is still inactive. So it does not respond to these changed outputs. This avoids the multiple toggling which leads to the race around condition. The master slave flip flop will avoid the race around condition.

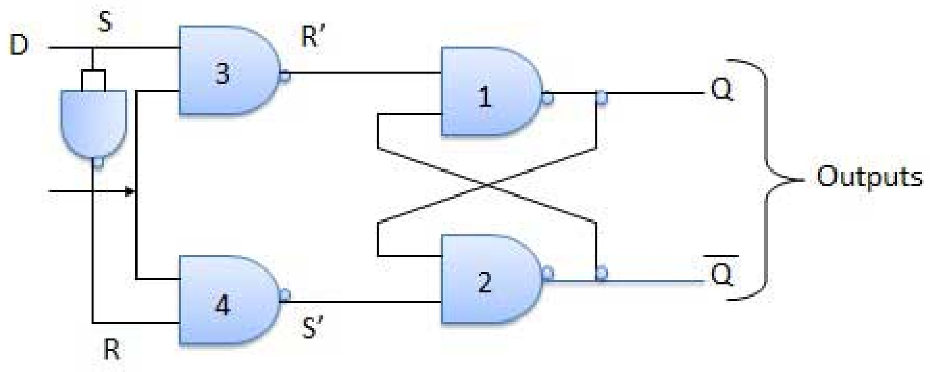
Delay Flip Flop / D Flip Flop

Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs. It has only one input. The input data is appearing at the output after some time. Due to this data delay between i/p and o/p, it is called delay flip flop. S and R will be the complements of each other due to NAND inverter. Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoid by SR = 00 and SR = 1 conditions.

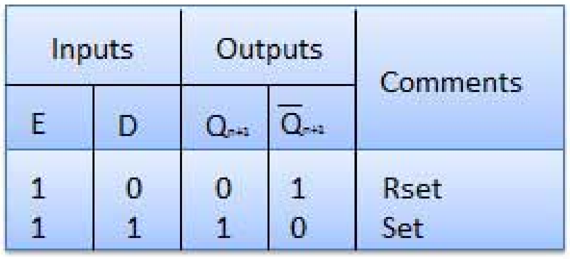
Block Diagram



Circuit Diagram



Truth Table



Inputs		Outputs		Comments
E	D	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	1	Rset
1	1	1	0	Set

Operation

S.N.ConditionOperation1**E = 0**

Latch is disabled. Hence no change in output.2**E = 1 and D = 0**

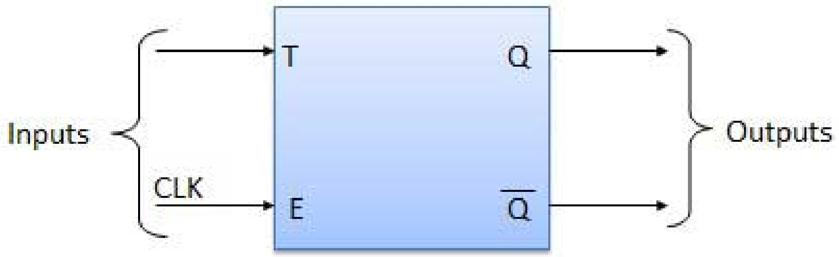
If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is Q_{n+1}= 0 and Q_{n+1} bar = 1. This is the reset condition.3**E = 1 and D = 1**

If E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and Q_{n+1} = 1 and Q_{n+1}bar = 0 irrespective of the present state.

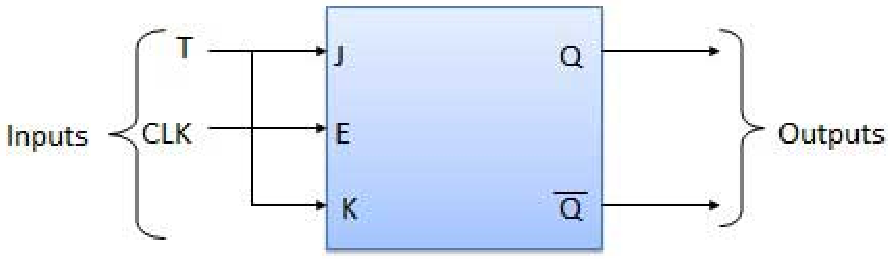
Toggle Flip Flop / T Flip Flop

Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only input denoted by **T** as shown in the Symbol Diagram. The symbol for positive edge triggered T flip flop is shown in the Block Diagram.

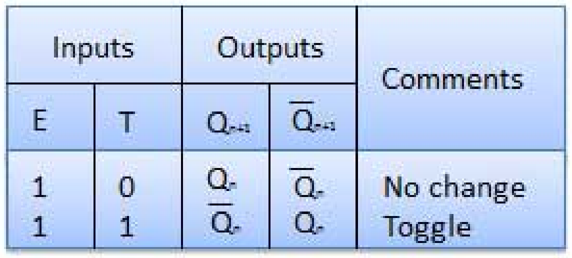
Symbol Diagram



Block Diagram



Truth Table



Inputs		Outputs		Comments
E	T	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	Q_n	$\overline{Q}_n$	No change
1	1	$\overline{Q}_n$	Q_n	Toggle

Operation

S.N.ConditionOperation1**T = 0, J = K = 0**The output Q and Q bar won't change2**T = 1, J = K = 1**Output will toggle corresponding to every leading edge of clock signal.

Digital Registers

Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a **Register**. The **n-bit register** will consist of **n** number of flip-flop and it is capable of storing an **n-bit** word.

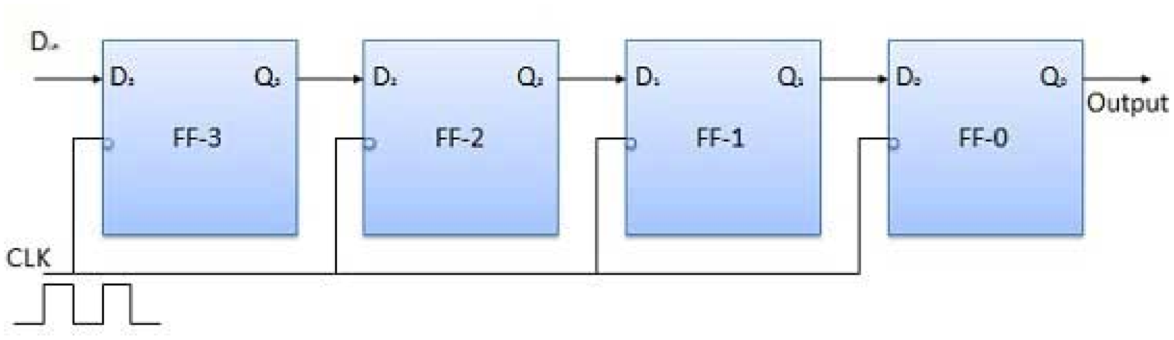
The binary data in a register can be moved within the register from one flip-flop to another. The registers that allow such data transfers are called as **shift registers**. There are four mode of operations of a shift register.

- Serial Input Serial Output
- Serial Input Parallel Output
- Parallel Input Serial Output
- Parallel Input Parallel Output

Serial Input Serial Output

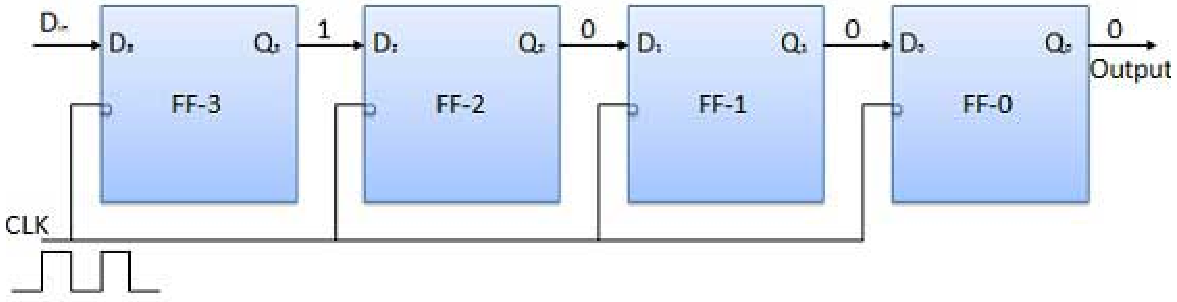
Let all the flip-flop be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$. If an entry of a four bit binary number 1 1 1 1 is made into the register, this number should be applied to $\mathbf{D_{in}}$ bit with the LSB bit applied first. The D input of FF-3 i.e. D_3 is connected to serial data input $\mathbf{D_{in}}$. Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

Block Diagram

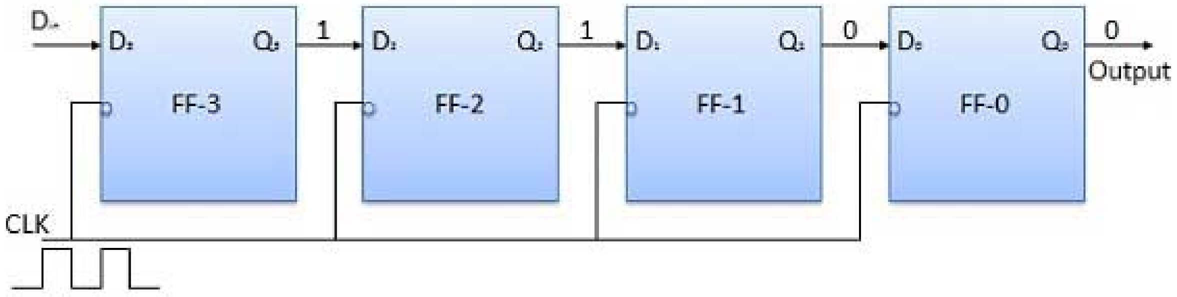


Operation

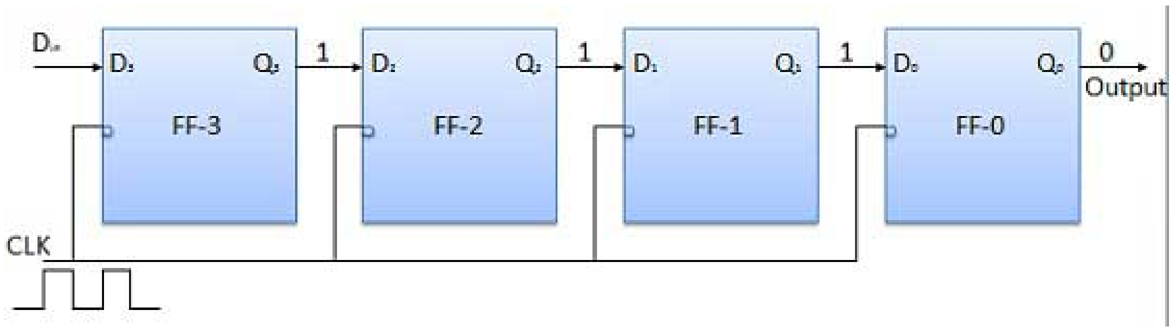
Before application of clock signal, let $Q_3\ Q_2\ Q_1\ Q_0$= 0000 and apply LSB bit of the number to be entered to D_{in}. So $D_{in} = D_3 = 1$. Apply the clock. On the first falling edge of clock, the FF-3 is set, and stored word in the register is $Q_3\ Q_2\ Q_1\ Q_0 = 1000$.



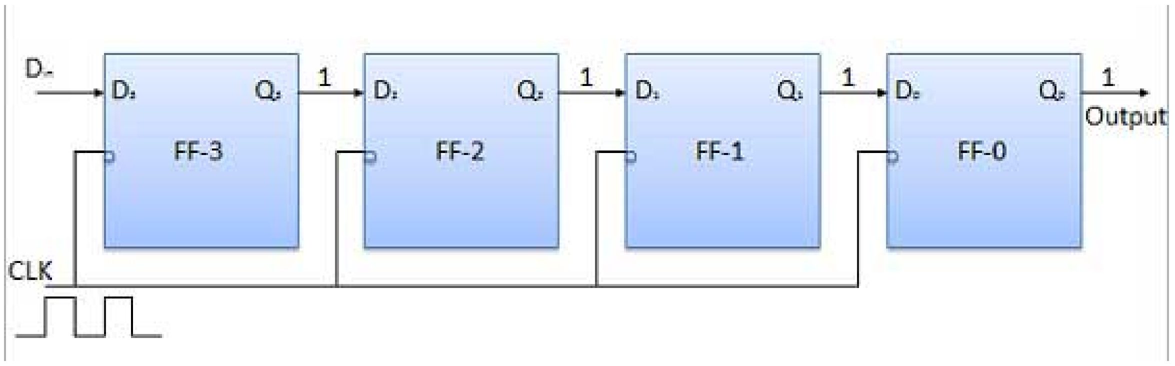
Apply the next bit to D_{in}. So $D_{in} = 1$. As soon as the next negative edge of the clock hits, FF-2 will set and the stored word change to $Q_3\ Q_2\ Q_1\ Q_0 = 1100$.



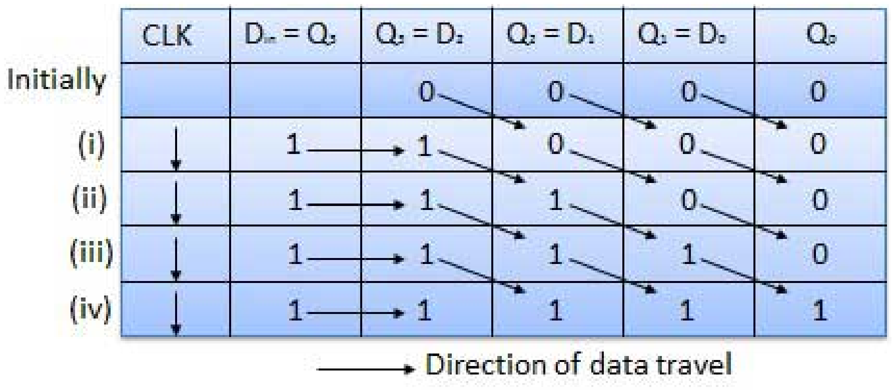
Apply the next bit to be stored i.e. 1 to D_{in}. Apply the clock pulse. As soon as the third negative clock edge hits, FF-1 will be set and output will be modified to $Q_3\ Q_2\ Q_1\ Q_0 = 1110$.



Similarly with $D_{in} = 1$ and with the fourth negative clock edge arriving, the stored word in the register is $Q_3\ Q_2\ Q_1\ Q_0 = 1111$.



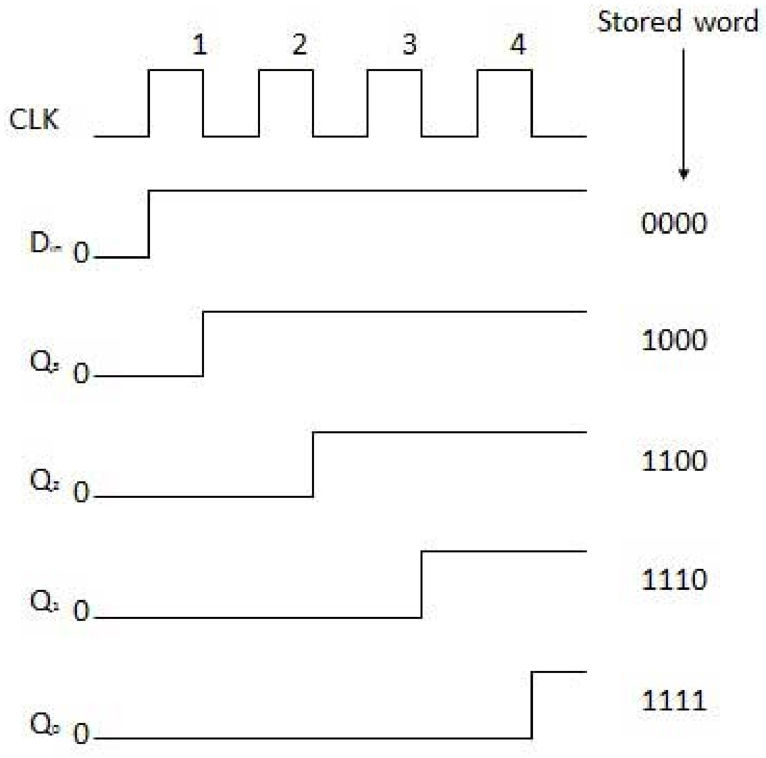
Truth Table



	CLK	$D_{in} = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(i)	↓	1	1	0	0	0
(ii)	↓	1	1	1	0	0
(iii)	↓	1	1	1	1	0
(iv)	↓	1	1	1	1	1

→ Direction of data travel

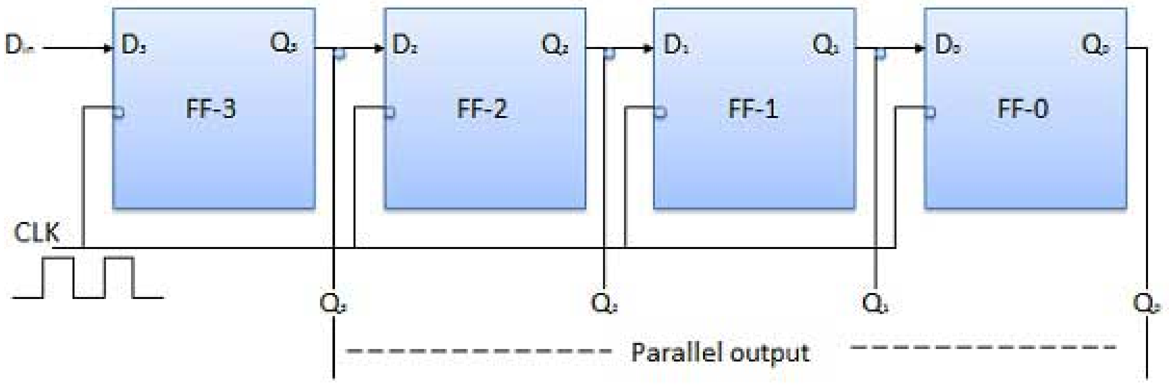
Waveforms



Serial Input Parallel Output

- In such types of operations, the data is entered serially and taken out in parallel fashion.
- Data is loaded bit by bit. The outputs are disabled as long as the data is loading.
- As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines at the same time.
- 4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

Block Diagram



Parallel Input Serial Output (PISO)

- Data bits are entered in parallel fashion.
- The circuit shown below is a four bit parallel input serial output register.
- Output of previous Flip Flop is connected to the input of the next one via a combinational circuit.
- The binary input word B_0, B_1, B_2, B_3 is applied though the same combinational circuit.
- There are two modes in which this circuit can work namely – shift mode or load mode.

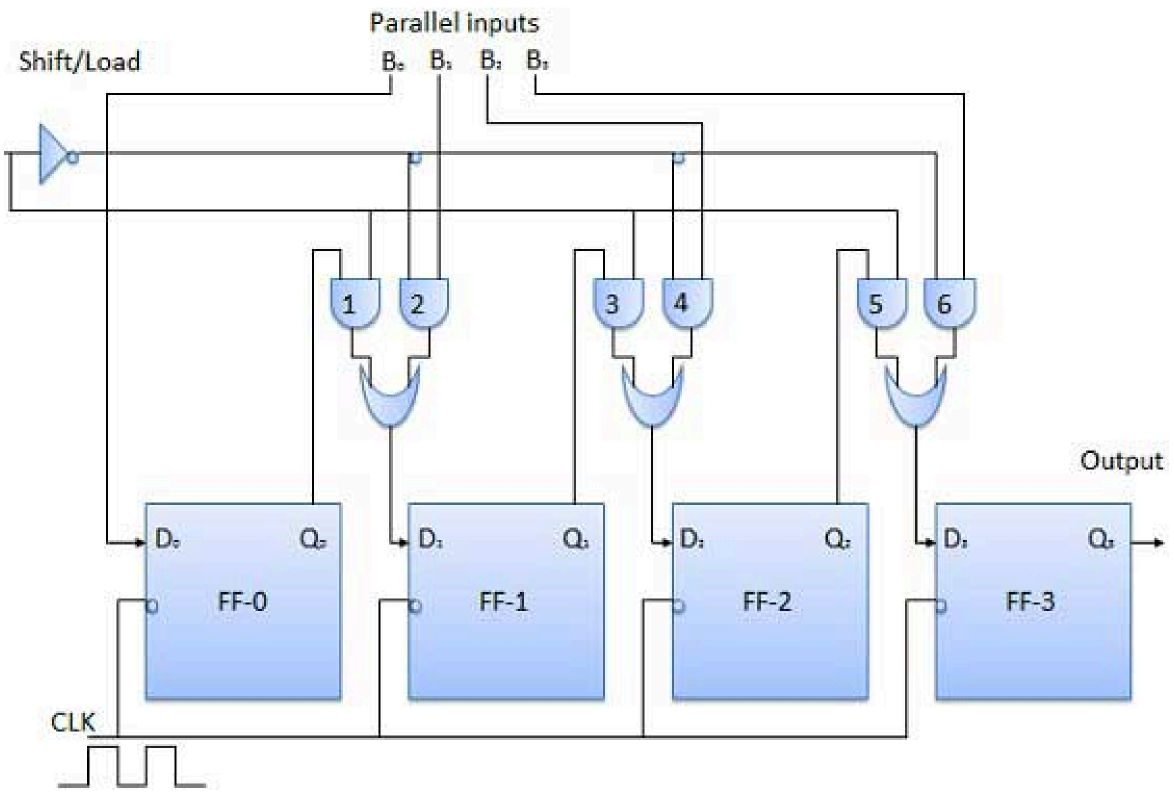
Load mode

When the shift/load bar line is low (0), the AND gate 2, 4 and 6 become active they will pass B_1, B_2, B_3 bits to the corresponding flip-flops. On the low going edge of clock, the binary input B_0, B_1, B_2, B_3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode

When the shift/load bar line is low (1), the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

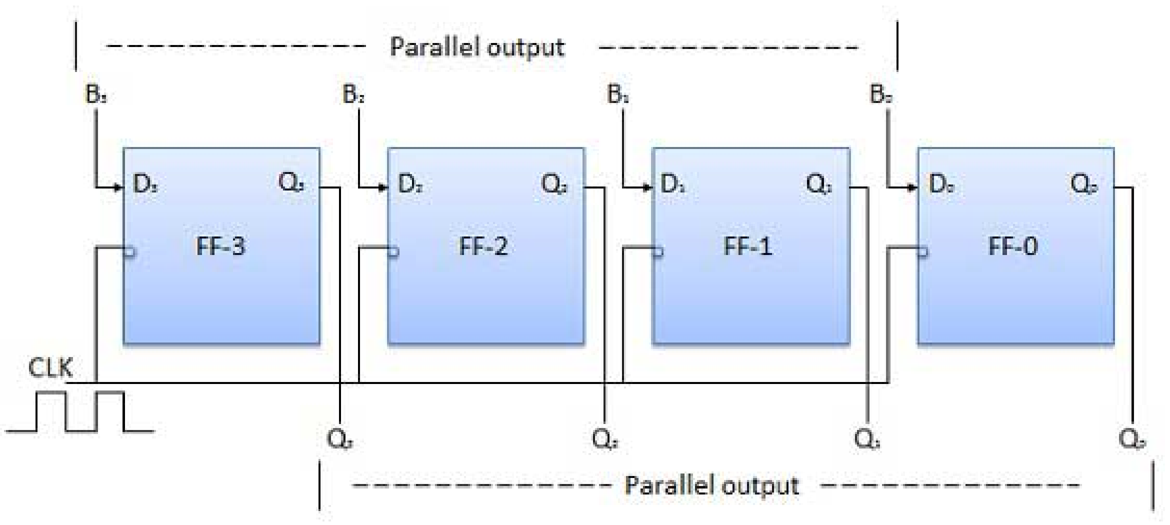
Block Diagram



Parallel Input Parallel Output (PIPO)

In this mode, the 4 bit binary input B_0, B_1, B_2, B_3 is applied to the data inputs D_0, D_1, D_2, D_3respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits.

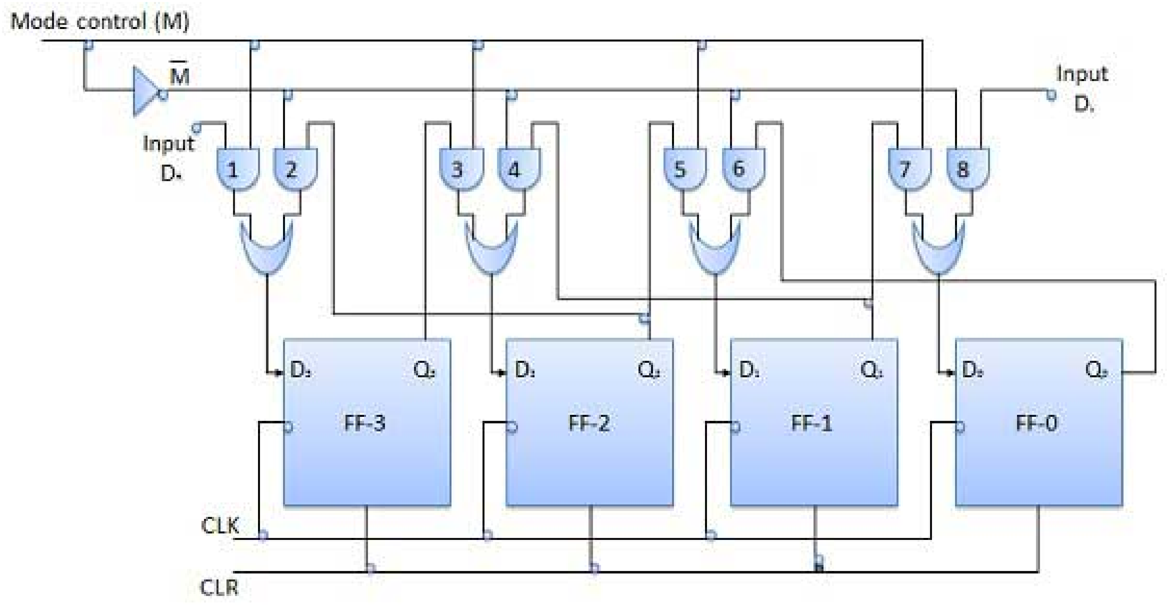
Block Diagram



Bidirectional Shift Register

- If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2. Similarly if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2.
- Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction.
- Such a register is called bi-directional register. A four bit bi-directional shift register is shown in fig.
- There are two serial inputs namely the serial right shift data input DR, and the serial left shift data input DL along with a mode select input (M).

Block Diagram



Operation

S.N.ConditionOperation1**With M = 1 – Shift right operation**

If M = 1, then the AND gates 1, 3, 5 and 7 are enabled whereas the remaining AND gates 2, 4, 6 and 8 will be disabled.

The data at D_R is shifted to right bit by bit from FF-3 to FF-0 on the application of clock pulses. Thus with M = 1 we get the serial right shift operation.2**With M = 0 – Shift left operation**

When the mode control M is connected to 0 then the AND gates 2, 4, 6 and 8 are enabled while 1, 3, 5 and 7 are disabled.

The data at D_L is shifted left bit by bit from FF-0 to FF-3 on the application of clock pulses. Thus with M = 0 we get the serial right shift operation.

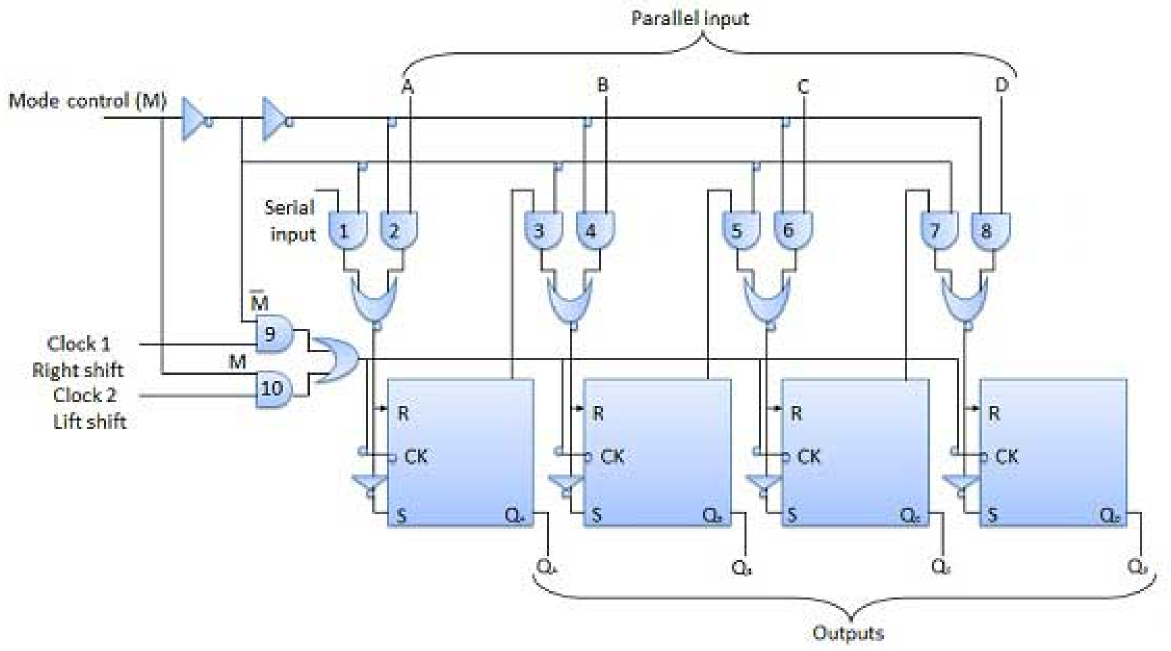
Universal Shift Register

A shift register which can shift the data in only one direction is called a uni-directional shift register. A shift register which can shift the data in both directions is called a bi-directional shift register. Applying the same logic, a shift register which can shift the data in both directions as well as load it parallely, is known as a universal shift register. The shift register is capable of performing the following operation −

- Parallel loading
- Left Shifting
- Right shifting

The mode control input is connected to logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting. With mode control pin connected to ground, the universal shift register acts as a bi-directional register. For serial left operation, the input is applied to the serial input which goes to AND gate-1 shown in figure. Whereas for the shift right operation, the serial input is applied to D input.

Block Diagram



Digital Counters

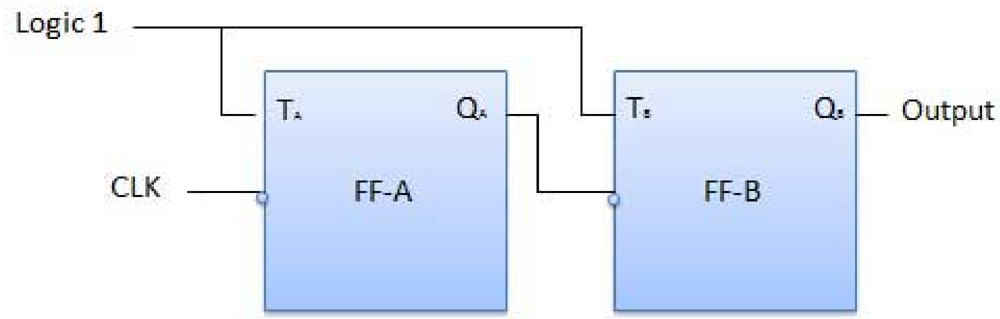
Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters.

Asynchronous or ripple counters

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

Logical Diagram



Operation

S.N.ConditionOperation

1**Initially let both the FFs be in the reset state**Q_BQ_A = 00 initially

2**After 1st negative clock edge**

As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will be equal to 1.

Q_A is connected to clock input of FF-B. Since $Q_A$has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF.

Q_BQ_A = 01 after the first clock pulse.3**After 2nd negative clock edge**

On the arrival of second negative clock edge, FF-A toggles again and Q_A = 0.

The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and $Q_B$will be 1.

Q_BQ_A = 10 after the second clock pulse.4**After 3rd negative clock edge**

On the arrival of 3rd negative clock edge, FF-A toggles again and $Q_A$become 1 from 0.

Since this is a positive going change, FF-B does not respond to it and remains inactive. So $Q_B$does not change and continues to be equal to 1.

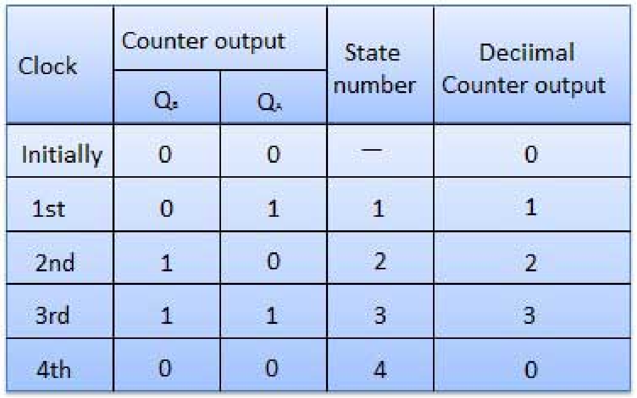
Q_BQ_A = 11 after the third clock pulse.5**After 4th negative clock edge**

On the arrival of 4th negative clock edge, FF-A toggles again and $Q_A$becomes 1 from 0.

This negative change in $Q_A$acts as clock pulse for FF-B. Hence it toggles to change $Q_B$from 1 to 0.

Q_BQ_A = 00 after the fourth clock pulse.

Truth Table



Clock	Counter output		State number	Deciimal 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

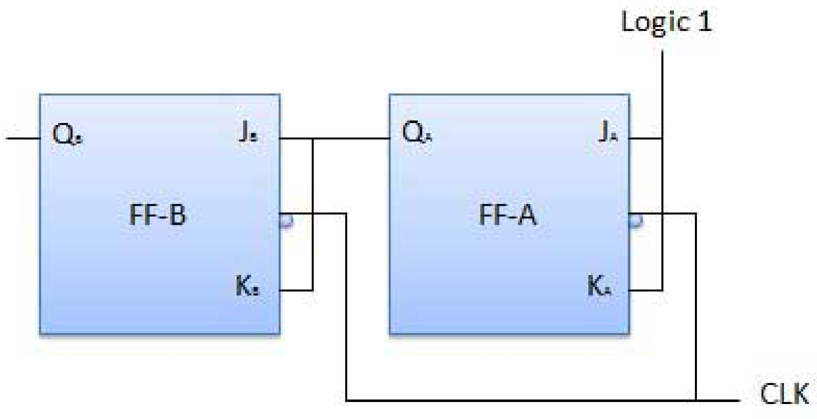
Synchronous counters

If the “clock” pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

2-bit Synchronous up counter

The J_A and K_A inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A.

Logical Diagram



Operation

S.N.ConditionOperation1**Initially let both the FFs be in the reset state**Q_BQ_A = 00 initially.2**After 1st negative clock edge**

As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1.

But at the instant of application of negative clock edge, Q_A , J_B = K_B = 0. Hence FF-B will not change its state. So $Q_B$will remain 0.

Q_BQ_A = 01 after the first clock pulse.3**After 2nd negative clock edge**

On the arrival of second negative clock edge, FF-A toggles again and $Q_A$changes from 1 to 0.

But at this instant Q_A was 1. So J_B = K_B= 1 and FF-B will toggle. Hence Q_B changes from 0 to 1.

Q_BQ_A = 10 after the second clock pulse.4**After 3rd negative clock edge**

On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B.

Q_BQ_A = 11 after the third clock pulse.5**After 4th negative clock edge**

On application of the next clock pulse, $Q_A$will change from 1 to 0 as Q_B will also change from 1 to 0.

Q_BQ_A = 00 after the fourth clock pulse.

Classification of counters

Depending on the way in which the counting progresses, the synchronous or asynchronous counters are classified as follows −

- Up counters
- Down counters
- Up/Down counters

UP/DOWN Counter

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

- Type of up/down counters
- UP/DOWN ripple counters
- UP/DOWN synchronous counter

UP/DOWN Ripple Counters

In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used. The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from (Q = Q bar) output of the previous FF.

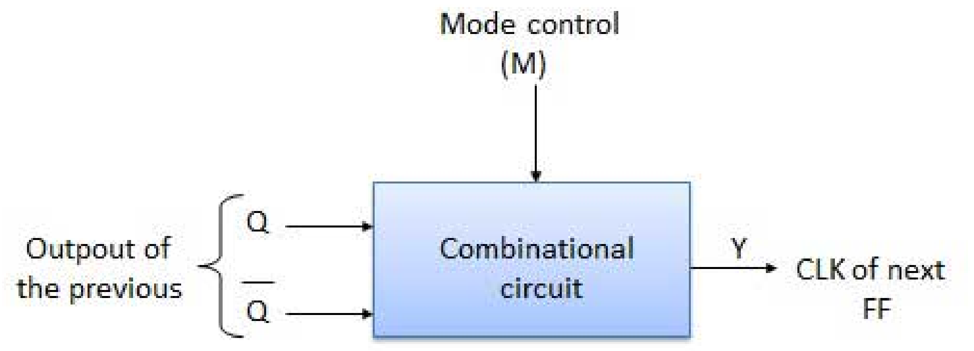
- **UP counting mode (M=0)** − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).
- **DOWN counting mode (M=1)** − If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

Example

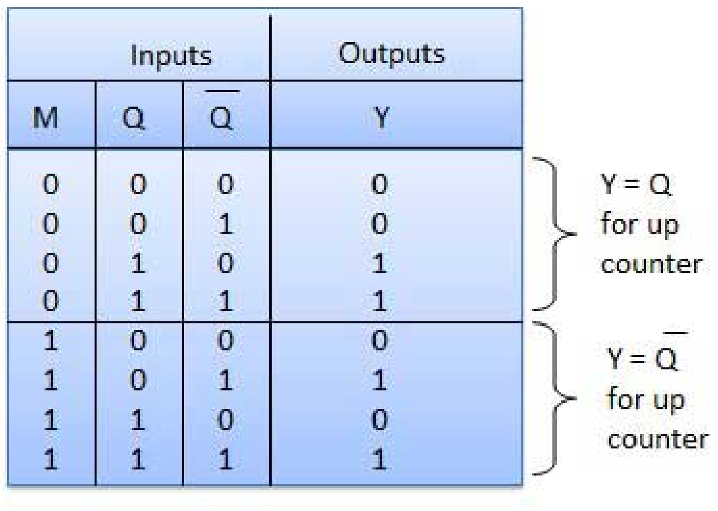
3-bit binary up/down ripple counter.

- 3-bit – hence three FFs are required.
- UP/DOWN – So a mode control input is essential.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple down counter, the Q bar output of preceding FF is connected to the clock input of the next one.
- Let the selection of Q and Q bar output of the preceding FF be controlled by the mode control input M such that, If M = 0, UP counting. So connect Q to CLK. If M = 1, DOWN counting. So connect Q bar to CLK.

Block Diagram



Truth Table



Inputs			Outputs
M	Q	$\overline{Q}$	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$Y = Q$ for up counter

$Y = \overline{Q}$ for up counter

Operation

S.N.ConditionOperation1**Case 1 – With M = 0 (Up counting mode)**

If M = 0 and M bar = 1, then the AND gates 1 and 3 in fig. will be enabled whereas the AND gates 2 and 4 will be disabled.

Hence Q_A gets connected to the clock input of FF-B and $Q_B$gets connected to the clock input of FF-C.

These connections are same as those for the normal up counter. Thus with M = 0 the circuit work as an up counter.2**Case 2: With M = 1 (Down counting mode)**

If M = 1, then AND gates 2 and 4 in fig. are enabled whereas the AND gates 1 and 3 are disabled.

Hence Q_A bar gets connected to the clock input of FF-B and $Q_B$bar gets connected to the clock input of FF-C.

These connections will produce a down counter. Thus with M = 1 the circuit works as a down counter.

Modulus Counter (MOD-N Counter)

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n.

Type of modulus

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)

Application of counters

- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator

<u>B.C.A study</u>

Unit-6:Memory Organization

Associative Memory

An associative memory can be considered as a memory unit whose stored data can be identified for access by the content of the data itself rather than by an address or memory location.

Associative memory is often referred to as **Content Addressable Memory (CAM)**.

When a write operation is performed on associative memory, no address or memory location is given to the word. The memory itself is capable of finding an empty unused location to store the word.

On the other hand, when the word is to be read from an associative memory, the content of the word, or part of the word, is specified. The words which match the specified content are located by the memory and are marked for reading.

The following diagram shows the block representation of an Associative memory.

Associative Memory

From the block diagram, we can say that an associative memory consists of a memory array and logic for ‘m’ words with ‘n’ bits per word.

The functional registers like the argument register **A** and key register **K** each have **n** bits, one for each bit of a word. The match register **M** consists of **m** bits, one for each memory word.

The words which are kept in the memory are compared in parallel with the content of the argument register.

The key register (K) provides a mask for choosing a particular field or key in the argument word. If the key register contains a binary value of all 1’s, then the entire argument is compared with each memory word. Otherwise, only those bits in the argument that have 1’s in their corresponding position of the key register are compared. Thus, the key provides a mask for identifying a piece of information which specifies how the reference to memory is made.

The following diagram can represent the relation between the memory array and the external registers in an associative memory.

Associative Memory

The cells present inside the memory array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. For instance, the cell C_{ij} is the cell for bit **j** in word **i**.

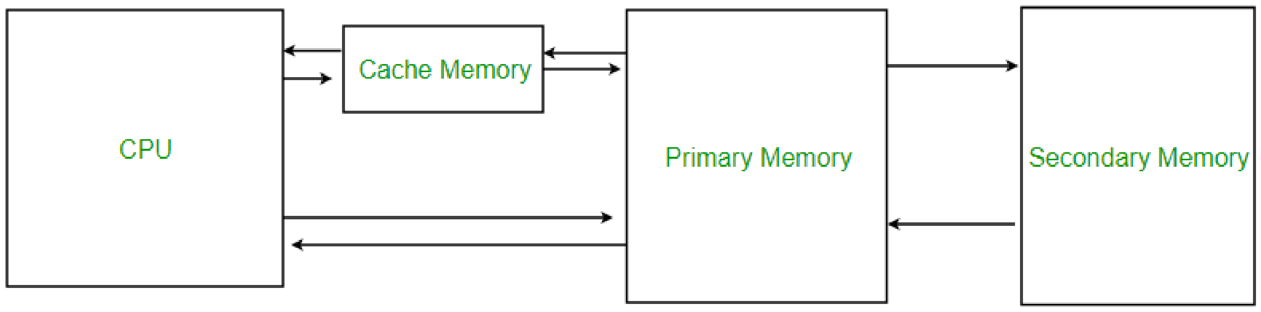
A bit A_j in the argument register is compared with all the bits in column **j** of the array provided that K_j = 1. This process is done for all columns **j** = 1, 2, 3……, n.

If a match occurs between all the unmasked bits of the argument and the bits in word **i**, the corresponding bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

Cache memory Organization

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Cache Performance:
When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

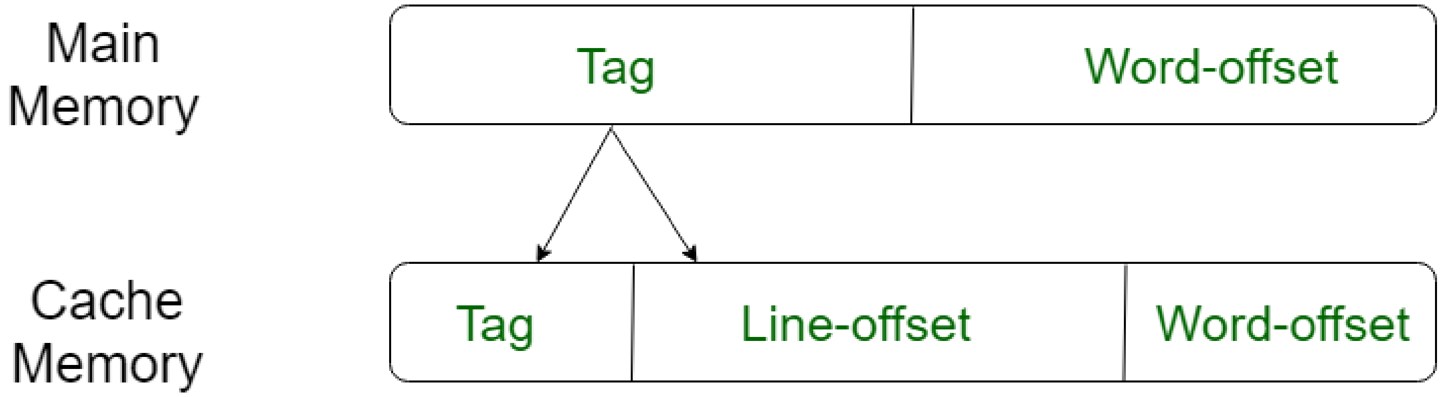
```
Hit ratio = hit / (hit + miss) =  no. of hits/total accesses
```

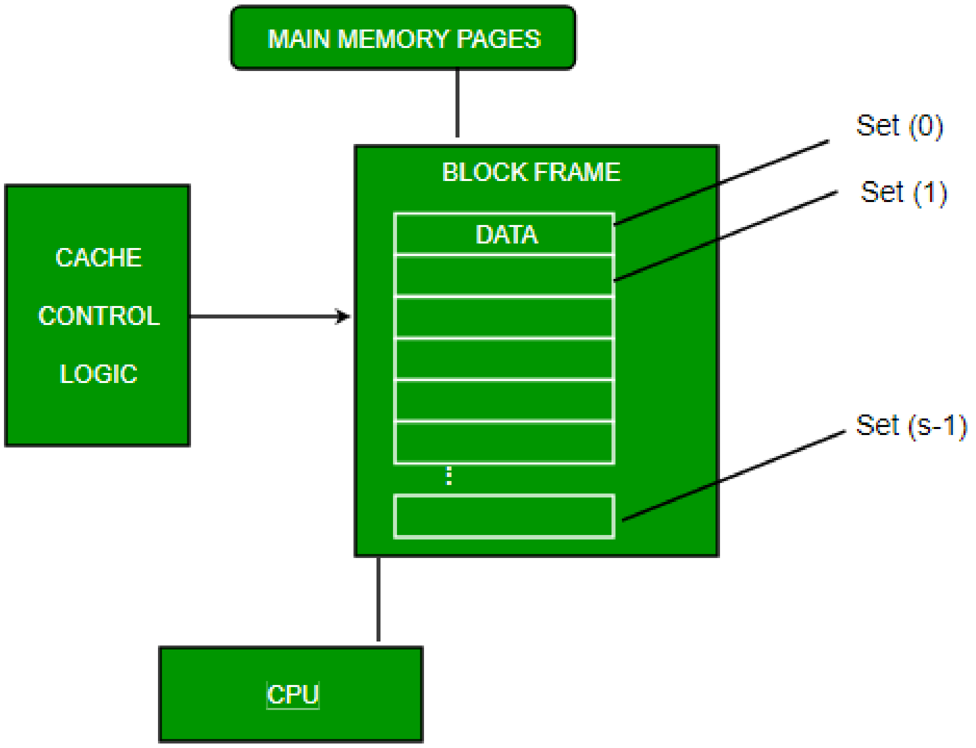
We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce Reduce the time to hit in the cache.

Cache Mapping:
There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

1. **Direct Mapping –**
   The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or
   In Direct mapping, assigne each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^S blocks of main memory. The cache logic

interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the m=2^r lines of the cache.





Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

•••

A WordPress.com Website.