22CSE104 COMPUTER ARCHITECTURE AND ORGANIZATION

MODULE 1: Fundamentals and Basics

CPU, Memory, Input-Output Subsystems, Control Unit, Functional units, Basic operational concepts, Bus structures, Software, Performance, Multiprocessors and Multi-Computers.

Encoders, Demultiplexers, Programmable Logic Arrays (PLAs), Digital Logic Circuits: Basic LogicFunctions, Synthesis of Logic Functions Using AND, OR, and NOT Gates, Minimization of Logic Expression, Synthesis with NAND and NOR Gates, Flip-Flops.

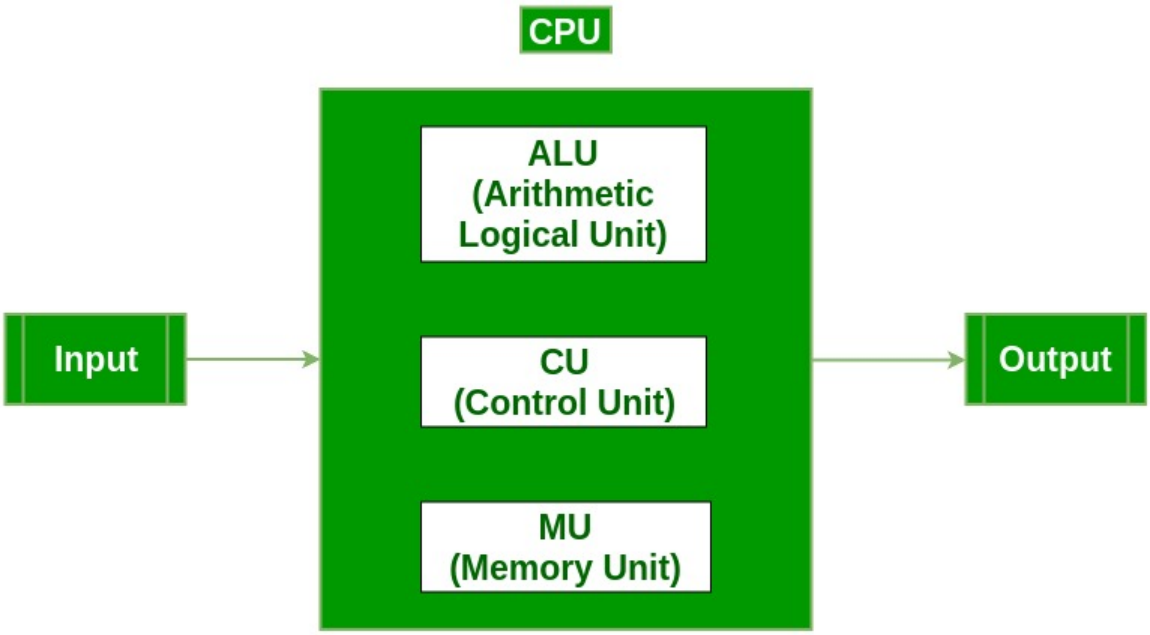
Functional Components of a Computer

Computer: A computer is a combination of **hardware and software** resources which integrate together and provides various functionalities to the user. Hardware are the physical components of a computer like the processor, memory devices, monitor, keyboard etc. while software is the set of programs or instructions that are required by the hardware resources to function properly.

There are a few basic components that aids the working-cycle of a computer i.e. the Input- Process- Output Cycle and these are called as the functional components of a computer. It needs certain input, processes that input and produces the desired output. The input unit takes the input, the central processing unit does the processing of data and the output unit produces the output. The memory unit holds the data and instructions during the processing.

Digital Computer: A digital computer can be defined as a programmable machine which reads the binary data passed as instructions, processes this binary data, and displays a calculated digital output. Therefore, Digital computers are those that work on the digital data.

Details of Functional Components of a Digital Computer



- **Input Unit :**The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.

- **Central Processing Unit (CPU) :** Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers

- **Arithmetic and Logic Unit (ALU) :** The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.

- **Control Unit :** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

- **Memory Registers :** A register is a temporary unit of memory in the CPU. These are used to store the data which is directly used by the processor. Registers can be of different sizes(16 bit, 32 bit, 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.

- **Memory :** Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. when a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM).
- **Output Unit :** The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, plotter etc

Basic Operational Concepts

- Instructions take a vital role for the proper working of the computer.
- An appropriate program consisting of a list of instructions is stored in the memory so that the tasks can be started.
- The memory brings the Individual instructions into the processor, which executes the specified operations.
- Data which is to be used as operands are moreover also stored in the memory.

Example:

Add LOCA, R0

- This instruction adds the operand at memory location LOCA to the operand which will be present in the Register R0.
- The above mentioned example can be written as follows:

Load LOCA, R1

Add R1, R0

- First instruction sends the contents of the memory location LOCA into processor Register R0, and meanwhile the second instruction adds the contents of Register R1 and R0 and places the output in the Register R1.

The memory and the processor are are swapped and are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals.

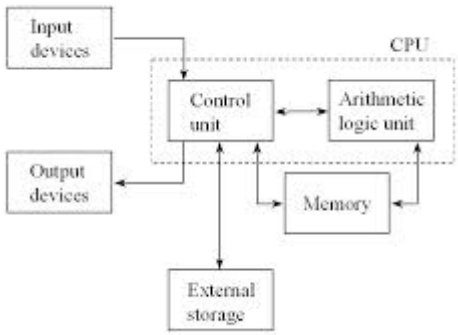
- The data is then transferred to or from the memory.

Analysing how processor and memory are connected :–

- Processors have various registers to perform various functions :-
- Program Counter :- It contains the memory address of next instruction to be fetched.
- Instruction Register:- It holds the instruction which is currently being executed.
- MDR :- It facilities communication with memory. It contains the data to be written into or read out of the addressed location.
- MAR :- It holds the address of the location that is to be accessed
- There are n general purpose registers that is R0 to Rn-1

Performance :-

- Performance means how quickly a program can be excecuted.



Computer organization

- In order to get the best performance it is required to design the compiler, machine instruction set & hardware in a coordinated manner.

<u>Connection B / W Processor & Memory</u>

Connection B/W Processor & Memory

- The above mentioned block diagram consists of the following components

1) Memory
2) MAR
3) MDR
4) PC
5) IR
6) General Purpose Registers
7) Control Unit
8) ALU

- The instruction that is currently being executed is held by the Instruction Register.

- IR output is available to the control circuits, which generates the timing signal that control the various processing elements involved in executing the instruction.
- The Memory address of the next instruction to be fetched and executed is contained by the Program Counter.
- It is a specialized register.
- It keeps the record of the programs that are executed.
- Role of these registers is to handle the data available in the instructions. They store the data temporarily.
- Two registers facilitate the communication with memory.

These registers are:

1) MAR (Memory Address Register)
2) MDR (Memory Data Register)

Memory Address Register:

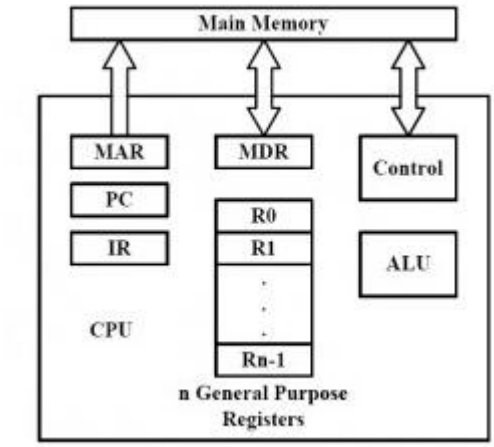
- The address of the location to be accessed is held by MAR.

Memory Data Register:

- It contains the data to be written into or to be read out of the addressed location.

Working Explanation

A **PC** is set to point to the first instruction of the program. The contents of the **PC** are transferred to the **MAR** and a Read control signal is sent to the memory. The addressed word is fetched from the location which is mentioned in the **MAR** and loaded into **MDR**. This post thus contains all the important basic operational concepts.



BUS STRUCTURE:

A Bus is a collection of wires that connects several devices.

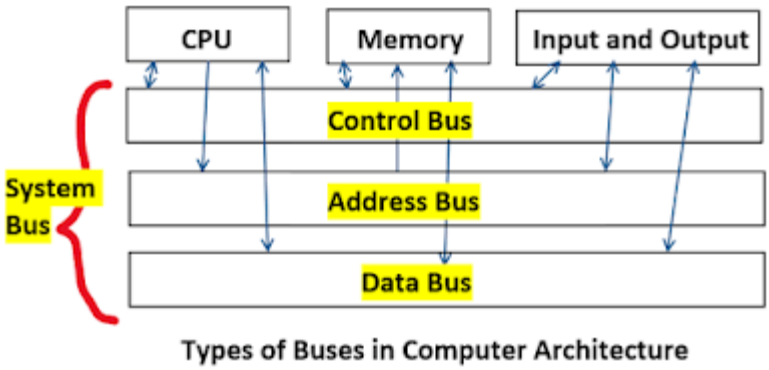
Buses are used to send control signals and data between the processor and other components

This is to achieve a reasonable speed of operation.

In computer system all the peripherals are connected to microprocessor through Bus.

Types of Bus structure:

1. Address bus
2. Data bus
3. Control bus



Types of Buses in Computer Architecture

1. Address Bus:

1. Address bus carry the memory address while reading from writing into memory.
2. Address bus caary I/O post address or device address from I/O port.
3. In uni-directional address bu only the CPU could send address and other units could not address the microprocessor.
4. Now a days computers are haing bi-directional address bus.

2. Data Bus:

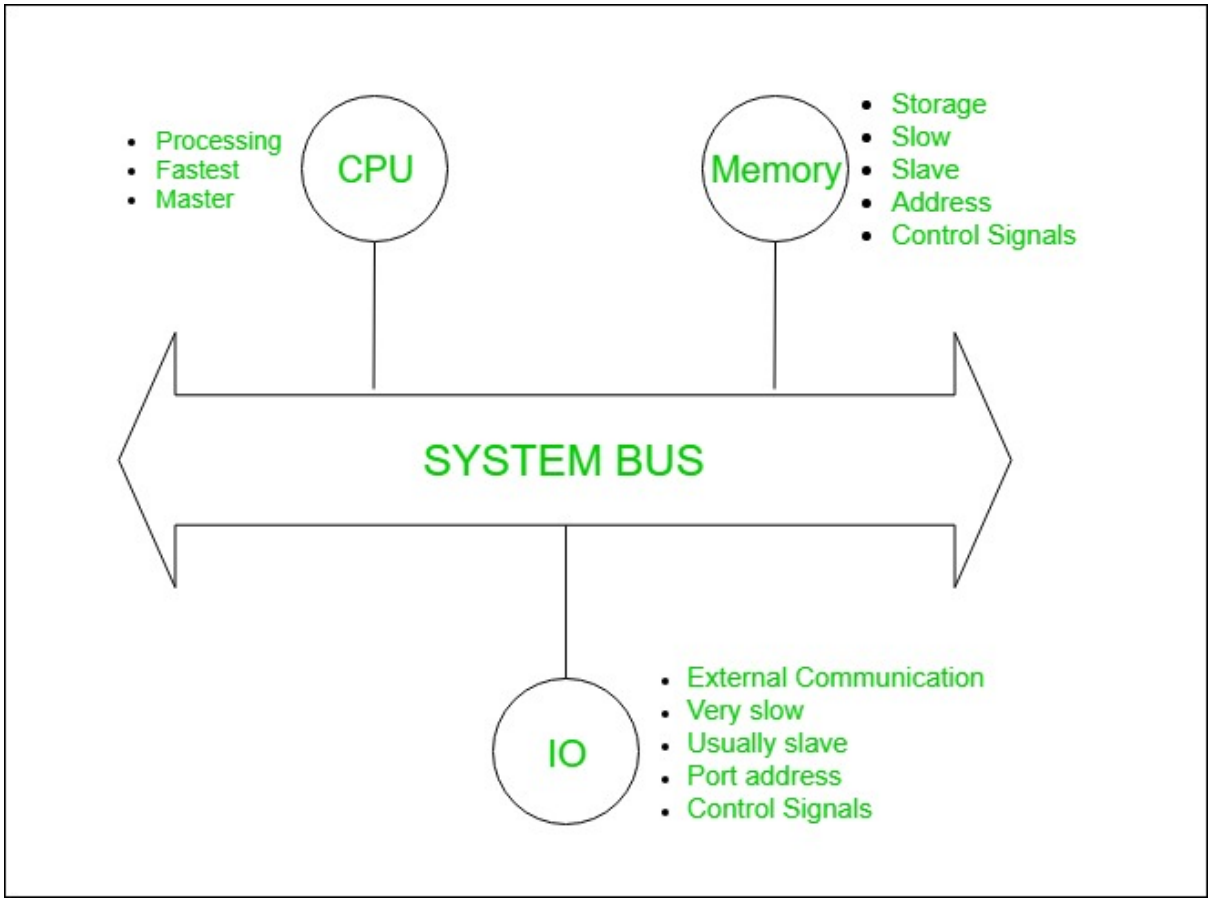
1. Data bus carry the data.
2. Data bus is a bidirectional bus.
3. Data bus fetch the instructions from memory.
4. Data bus used to store the result of an instruction into memory.
5. Data bus carry commands to an I/O device controller or port.
6. Data bus carry data from a device controller or port.
7. Data bus issue data to a device controller or port.

3. Control Bus:

Different types of control signals are used in a bus:

1. Memory Read: This signal, is issued by the CPU or DMA controller when performing a read operation with the memory.
2. MemoryWrite: This signal isissued by the CPU or DMAcontroller when performing a write operation with the memory.
3. I/O Read: This signal isissued by the CPU when it is reading from an input port.
4. I/O Write: This signal is issued by the CPU when writing into an output port.
5. Ready: The ready is an input signal to the CPU generated in order to synchronize the show memory or I/O ports with the fast CPU.

A **system bus** is a single computer bus that connects the major components of a computer system, combining the functions of a data bus to carry information, an address bus to determine where it should be sent, and a control bus to determine its operation.



Performance:

In computer organization, performance refers to the speed and efficiency at which a computer system can execute tasks and process data. A high-performing computer system is one that can perform tasks quickly and efficiently, while minimizing the amount of time and resources required to complete these tasks.

There are several factors that can impact the performance of a computer system, including:

1. Processor speed: The speed of the processor, measured in GHz (gigahertz), determines how quickly the computer can execute instructions and process data.
2. Memory: The amount and speed of the memory, including RAM (random access memory) and cache memory, can impact how quickly data can be accessed and processed by the computer.
3. Storage: The speed and capacity of the storage devices, including hard drives and solid-state drives (SSDs), can impact the speed at which data can be stored and retrieved.
4. I/O devices: The speed and efficiency of input/output devices, such as keyboards, mice, and displays, can impact the overall performance of the system.
5. Software optimization: The efficiency of the software running on the system, including operating systems and applications, can impact how quickly tasks can be completed.

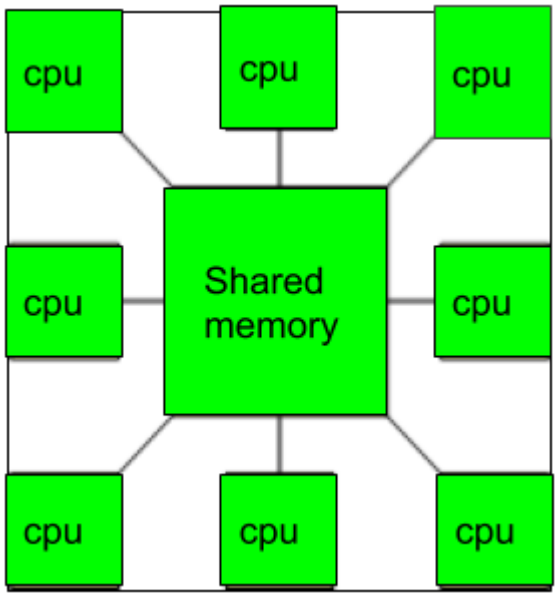
Improving the performance of a computer system typically involves optimizing one or more of these factors to reduce the time and resources required to complete tasks. This can involve upgrading hardware components, optimizing software, and using specialized performance tuning tools to identify and address bottlenecks in the system.

Computer performance is the amount of work accomplished by a computer system. The word performance in computer performance means "How well is the computer doing the work it is supposed to do?". It basically depends on response time, throughput and execution time of a computer system. **Response time** is the time from start to completion of a task. This also includes:

- Operating system overhead.
- Waiting for I/O and other processes
- Accessing disk and memory
- Time spent executing on the CPU or execution time.

Throughput is the total amount of work done in a given time. **CPU execution time** is the total time a CPU spends computing on a given task. It also excludes time for I/O or running other programs. This is also referred to as simply CPU time. Performance is determined by execution time as performance is inversely proportional to execution time.

1.Multiprocessor: A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching. There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory multiprocessor. In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own memory. .



Applications of Multiprocessor

1. As a uniprocessor, such as single instruction, single data stream (SISD).
2. As a multiprocessor, such as single instruction, multiple data stream (SIMD), which is usually used for vector processing.
3. Multiple series of instructions in a single perspective, such as multiple instruction, single data stream (MISD), which is used for describing hyper-threading or pipelined processors.
4. Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).
5.

Benefits of using a Multiprocessor

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

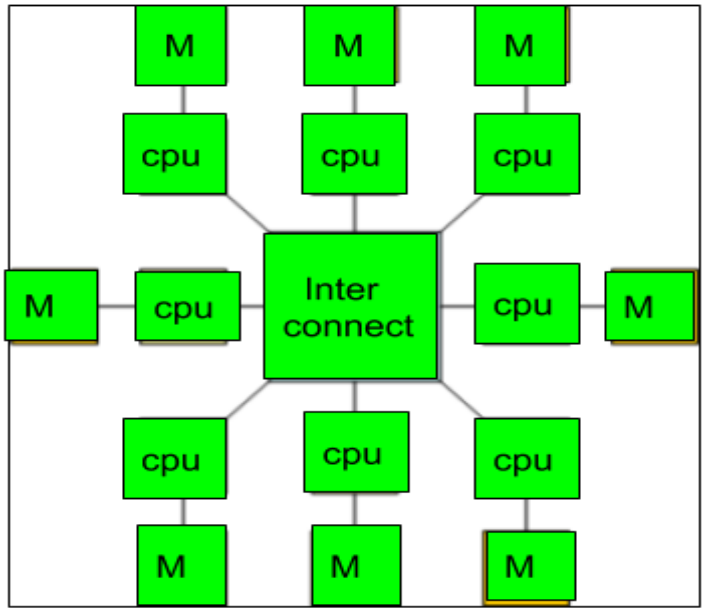
Advantages:

Improved performance: Multiprocessor systems can execute tasks faster than single-processor systems, as the workload can be distributed across multiple processors.
Better scalability: Multiprocessor systems can be scaled more easily than single-processor systems, as additional processors can be added to the system to handle increased workloads.
Increased reliability: Multiprocessor systems can continue to operate even if one processor fails, as the remaining processors can continue to execute tasks.
Reduced cost: Multiprocessor systems can be more cost-effective than building multiple single-processor systems to handle the same workload.
Enhanced parallelism: Multiprocessor systems allow for greater parallelism, as different processors can execute different tasks simultaneously.

Disadvantages:

Increased complexity: Multiprocessor systems are more complex than single-processor systems, and they require additional hardware, software, and management resources.
Higher power consumption: Multiprocessor systems require more power to operate than single-processor systems, which can increase the cost of operating and maintaining the system.
Difficult programming: Developing software that can effectively utilize multiple processors can be challenging, and it requires specialized programming skills.
Synchronization issues: Multiprocessor systems require synchronization between processors to ensure that tasks are executed correctly and efficiently, which can add complexity and overhead to the system.
Limited performance gains: Not all applications can benefit from multiprocessor systems, and some applications may only see limited performance gains when running on a multiprocessor system.

2. Multicomputer: A multicomputer system is a computer system with multiple processors that are connected together to solve a problem. Each processor has its own memory and it is accessible by that particular processor and those processors can communicate with each other via an interconnection



network.

As the multicomputer is capable of messages passing between the processors, it is possible to divide the task between the processors to complete the task. Hence, a multicomputer can be used for distributed computing. It is cost effective and easier to build a multicomputer than a multiprocessor.

Difference between multiprocessor and Multicomputer:

1. Multiprocessor is a system with two or more central processing units (CPUs) that is capable of performing multiple tasks where as a multicomputer is a system with multiple processors that are attached via an interconnection network to perform a computation task.
2. A multiprocessor system is a single computer that operates with multiple CPUs where as a multicomputer system is a cluster of computers that operate as a singular computer.
3. Construction of multicomputer is easier and cost effective than a multiprocessor.
4. In multiprocessor system, program tends to be easier where as in multicomputer system, program tends to be more difficult.
5. Multiprocessor supports parallel computing, Multicomputer supports distributed computing.

Advantages:

Improved performance: Multicomputer systems can execute tasks faster than single-computer systems, as the workload can be distributed across multiple computers.
Better scalability: Multicomputer systems can be scaled more easily than single-computer systems, as additional computers can be added to the system to handle increased workloads.
Increased reliability: Multicomputer systems can continue to operate even if one computer fails, as the remaining computers can continue to execute tasks.
Reduced cost: Multicomputer systems can be more cost-effective than building a single large computer system to handle the same workload.

Enhanced parallelism: Multicomputer systems allow for greater parallelism, as different computers can execute different tasks simultaneously.

Disadvantages:

Increased complexity: Multicomputer systems are more complex than single-computer systems, and they require additional hardware, software, and management resources.
Higher power consumption: Multicomputer systems require more power to operate than single-computer systems, which can increase the cost of operating and maintaining the system.
Difficult programming: Developing software that can effectively utilize multiple computers can be challenging, and it requires specialized programming skills.
Synchronization issues: Multicomputer systems require synchronization between computers to ensure that tasks are executed correctly and efficiently, which can add complexity and overhead to the system.
Network latency: Multicomputer systems rely on a network to communicate between computers, and network latency can impact system performance.
An encoder is a digital circuit that converts a set of binary inputs into a unique binary code. The binary code represents the position of the input and is used to identify the specific input that is active. Encoders are commonly used in digital systems to convert a parallel set of inputs into a serial code.

Encoder:

The basic principle of an encoder is to assign a unique binary code to each possible input. For example, a 2-to-4 line encoder has 2 input lines and 4 output lines and assigns a unique 4-bit binary code to each of the 2^2 = 4 possible input combinations. The output of an encoder is usually active low, meaning that only one output is active (low) at any given time, and the remaining outputs are inactive (high). The active low output is selected based on the binary code assigned to the active input.

There are different types of encoders, including priority encoders, which assign a priority to each input, and binary-weighted encoders, which use a binary weighting system to assign binary codes to inputs. In summary, an encoder is a digital circuit that converts a set of binary inputs into a unique binary code that represents the position of the input. Encoders are widely used in digital systems to convert parallel inputs into serial codes.

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

Encoder

Types of Encoders
There are different types of Encoders which are mentioned below.

- 4 to 2 Encoder
- Octal to Binary Encoder (8 to 3 Encoder)

- Decimal to BCD Encoder
- Priority Encoder

4 to 2 Encoder

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

4 to 2 Encoder

The Truth table of 4 to 2 encoders is as follows.

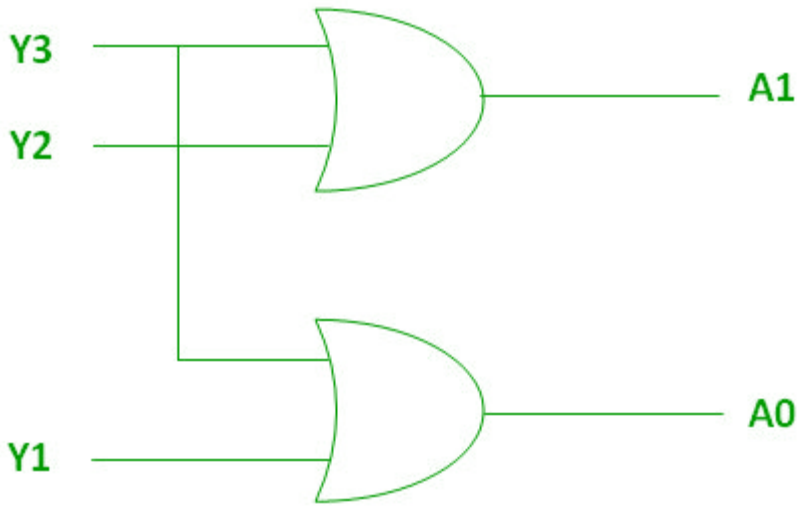
INPUTS				**OUTPUTS**	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Logical expression for A1 and A0:

A1 = Y3 + Y2

A0 = Y3 + Y1

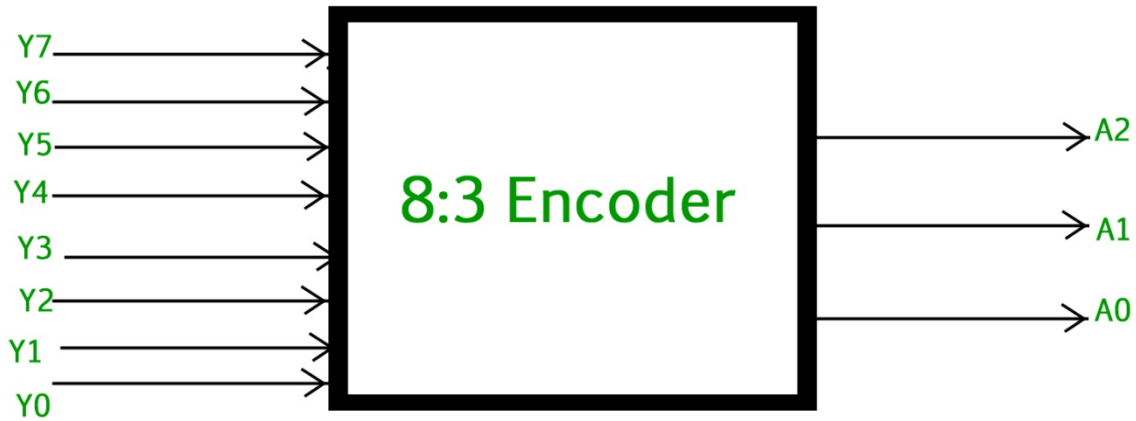
The above two Boolean functions A1 and A0 can be implemented using two input OR gates :



Implementation using OR Gate

Octal to Binary Encoder (8 to 3 Encoder)

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



Octal to Binary Encoder (8 to 3 Encoder)

The truth table for the 8 to 3 encoder is as follows.

INPUTS	OUTPUTS

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

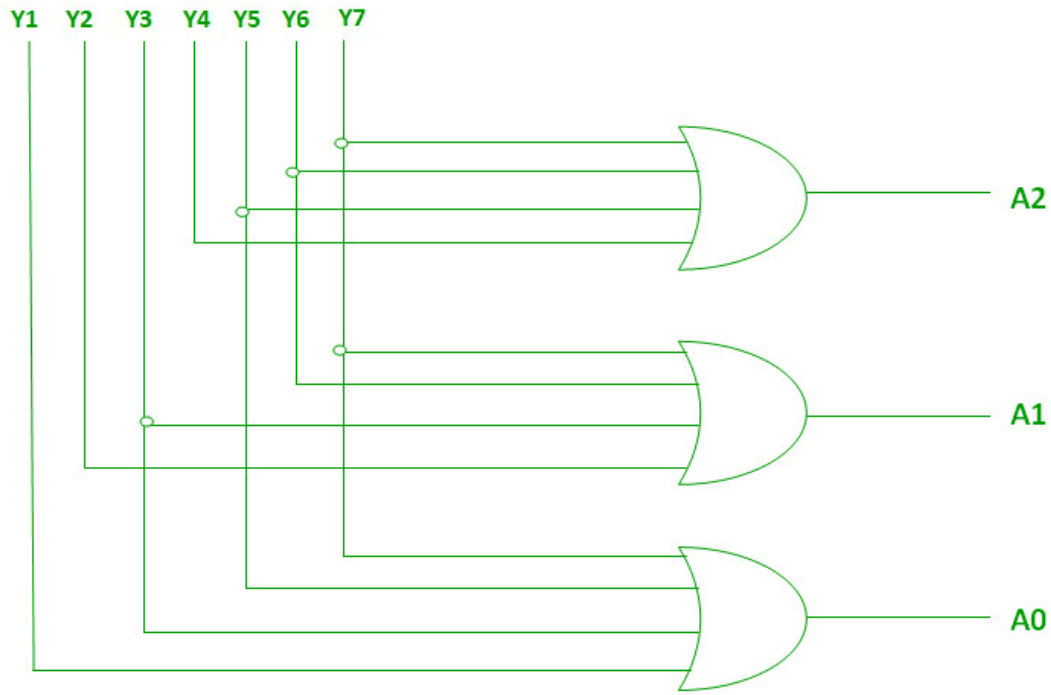
Logical expression for A2, A1, and A0.

A2 = Y7 + Y6 + Y5 + Y4

A1 = Y7 + Y6 + Y3 + Y2

A0 = Y7 + Y5 + Y3 + Y1

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



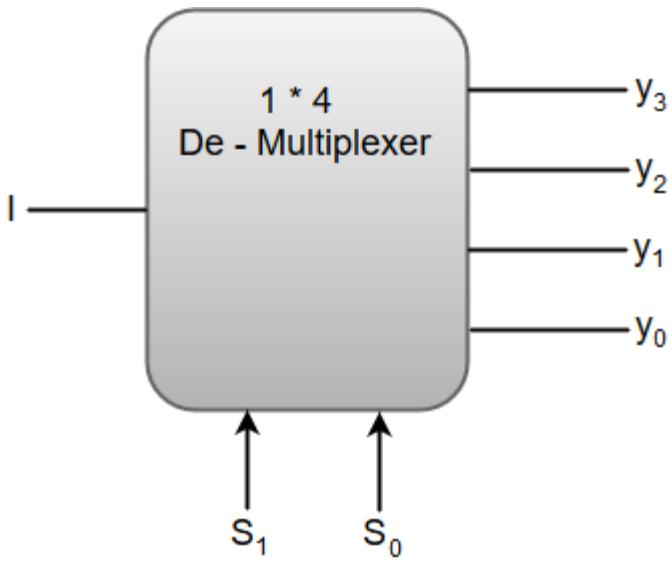
Implementation using OR Gate

De-Multiplexers

A De-multiplexer (De-Mux) can be described as a combinational circuit that performs the reverse operation of a Multiplexer.

A De-multiplexer has a single input, 'n' selection lines and a maximum of 2^n outputs.

The following image shows the block diagram of a 1 * 4 De-multiplexer.



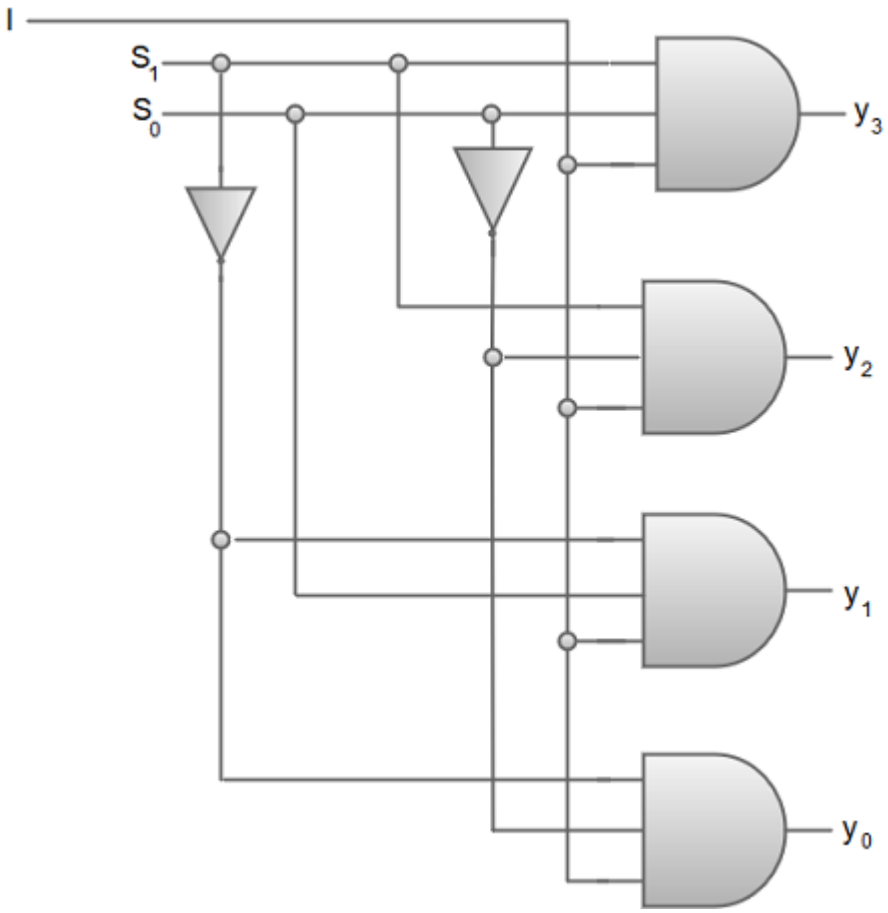
The function table for a 1 * 4 De - Multiplexer can be represented as:

S1	S0	y3	y2	y1	y0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

From the above function table, we can write the Boolean function for each output as:

y3 = S1S0 I, y2 = S1S0' I, y1 = S1' S0 I, y0 = S1'S0' I

The above equations can be implemented using inverters and three-input AND gates.

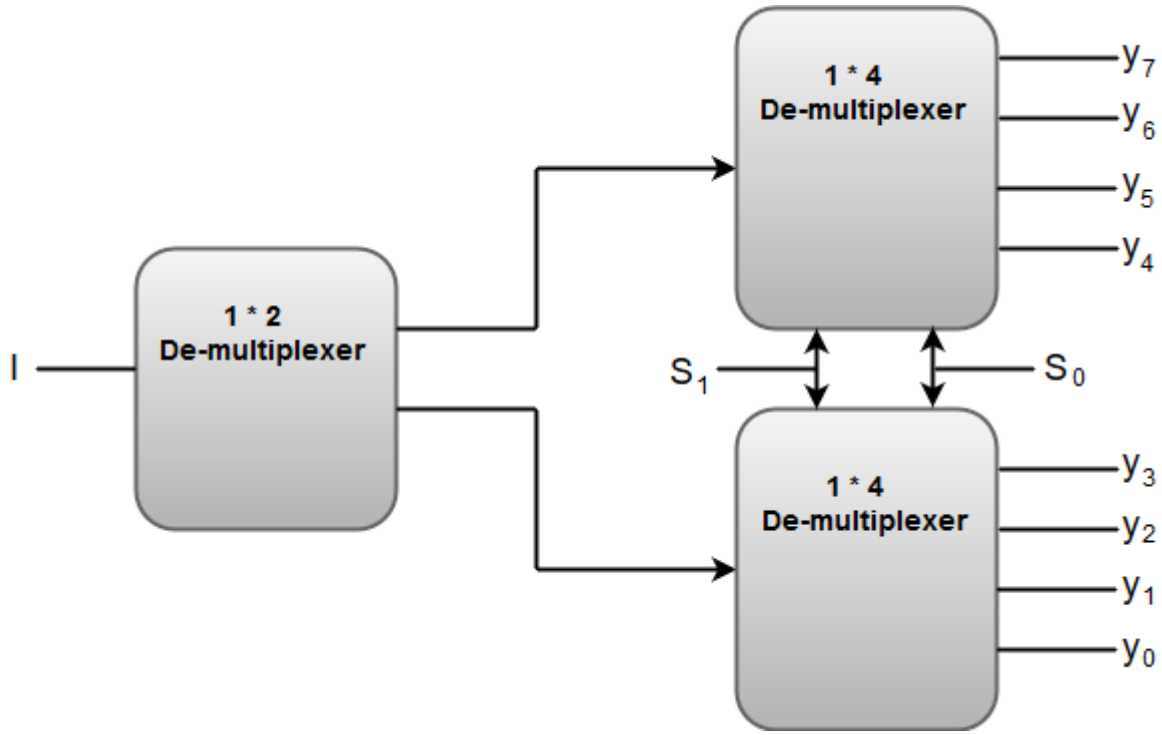


We can also implement higher order De-multiplexers using lower order De-multiplexers. For instance, let us implement a 1 * 8 De-multiplexer using 1 * 2 De-multiplexer in the first stage followed by two 1 * 4 De-multiplexers in the second stage.

The function table for a 1 * 8 De-multiplexer can be represented as:

S2	S1	S0	y7	y6	y5	y4	y3	y2	y1	y0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

The block diagram for a 1 * 8 De-multiplexer can be represented as:



The Selection lines 'S1' and 'S0' are common for both of the 1 * 4 De-multiplexers.

Programmable Logic Array(PLA)

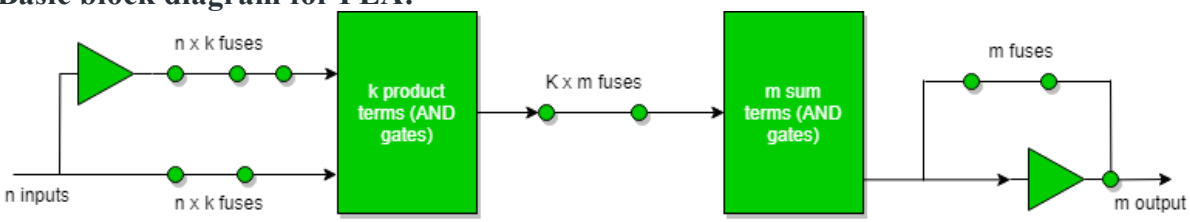
Programmable Logic Array(PLA) is a fixed architecture logic device with programmable AND gates followed by programmable OR gates. PLA is basically a type of programmable logic device used to build a reconfigurable digital circuit. PLDs have an undefined function at the time of manufacturing, but they are programmed before being made into use. PLA is a combination of memory and logic.

Comparison with other Programmable Logic Devices:

- PLA has a programmable AND gate array and programmable OR gate array.
- PAL has a programmable AND gate array but a fixed OR gate array.
- ROM has a fixed AND gate array but programmable OR gate array.

PLA is similar to a ROM in concept; however, it does not provide full decoding of variables and does not generate all minterms as in the ROM. Though its name consists of the word “programmable”, it does not require any type of programming like in C and C++.

Basic block diagram for PLA:



Following Truth table will be helpful in understanding function on no of inputs:

A	B	C	F1	F2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

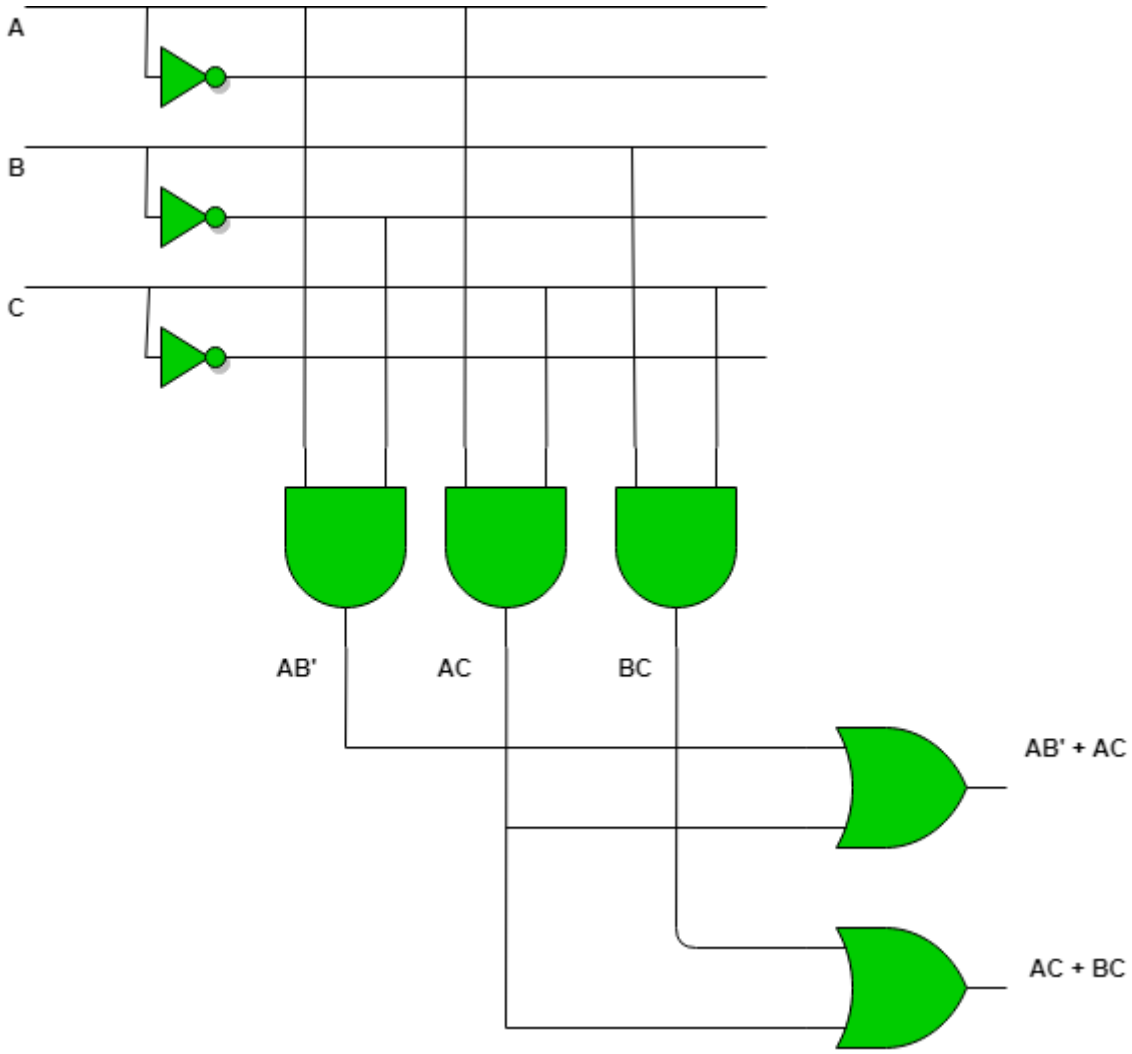
F1 = AB'C' + ABC' + ABC

on simplifying we get : F1 = AB + AC'

F2 = A'BC + AB'C + ABC

on simplifying we get: F2 = BC + AC

For the realization of the above function following circuit diagram will be used.



PLA is used for the implementation of various combinational circuits using a buffer, AND gate, and OR gate. In PLA, all the minterms are not realized but only required minterms are implemented. As PLA has a programmable AND gate array and a programmable OR gate array, it provides more flexibility but the disadvantage is, it is not easy to use.

The operation of a PLA can be summarized in three steps:
1. **Programming**: The user defines the logic function to be implemented by the PLA by programming the input and output configurations into the device.
2. **Product term generation**: The inputs are applied to the AND gate array to produce a set of product terms.
3. Sum term generation: The product terms are then applied to the OR gate array to generate the final output.

PLAs are often used in digital systems as they are versatile and allow complex functions to be implemented easily. They are particularly useful for implementing Boolean expressions with many variables as the arrays of AND gates and OR gates can be configured to handle large numbers of inputs.

Applications:

- PLA is used to provide control over datapath.
- PLA is used as a counter.

- PLA is used as a decoder.
- PLA is used as a BUS interface in programmed I/O.

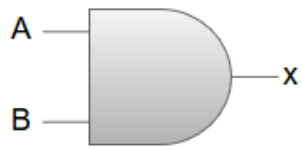
Logic Gates

- The logic gates are the main structural part of a digital system.
- Logic Gates are a block of hardware that produces signals of binary 1 or 0 when input logic requirements are satisfied.
- Each gate has a distinct graphic symbol, and its operation can be described by means of algebraic expressions.
- The seven basic logic gates includes: AND, OR, XOR, NOT, NAND, NOR, and XNOR.
- The relationship between the input-output binary variables for each gate can be represented in tabular form by a truth table.
- Each gate has one or two binary input variables designated by A and B and one binary output variable designated by x.

AND GATE:

The AND gate is an electronic circuit which gives a high output only if all its inputs are high. The AND operation is represented by a dot (.) sign.

AND Gate:



Algebraic Function: x = AB

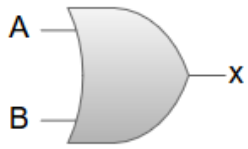
Truth Table:

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

OR GATE:

The OR gate is an electronic circuit which gives a high output if one or more of its inputs are high. The operation performed by an OR gate is represented by a plus (+) sign.

OR Gate:



Algebraic Function: x = A + B

Truth Table:

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

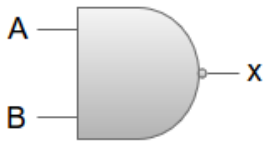
NOT GATE:

The NOT gate is an electronic circuit which produces an inverted version of the input at its output. It is also known as an **Inverter**.

NAND GATE:

The NOT-AND (NAND) gate which is equal to an AND gate followed by a NOT gate. The NAND gate gives a high output if any of the inputs are low. The NAND gate is represented by a AND gate with a small circle on the output. The small circle represents inversion.

NAND Gate:



Algebraic Function: x = (AB)'

Truth Table:

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

NOR GATE:

The NOT-OR (NOR) gate which is equal to an OR gate followed by a NOT gate. The NOR gate gives a low output if any of the inputs are high. The NOR gate is represented by an OR gate with a small circle on the output. The small circle represents inversion.

NOR Gate:

Algebraic Function: x = (A+B)'

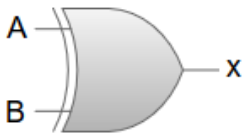
Truth Table:

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

Exclusive-OR/ XOR GATE:

The 'Exclusive-OR' gate is a circuit which will give a high output if one of its inputs is high but not both of them. The XOR operation is represented by an encircled plus sign.

XOR Gate:



Algebraic Function: $x = A \oplus B$

or

$x = A'B + AB'$

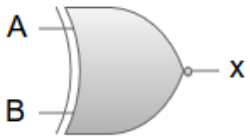
Truth Table:

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

EXCLUSIVE-NOR/Equivalence GATE:

The 'Exclusive-NOR' gate is a circuit that does the inverse operation to the XOR gate. It will give a low output if one of its inputs is high but not both of them. The small circle represents inversion.

Exclusive-NOR Gate:



Algebraic Function: $x = (A \oplus B)'$

or

$x = A'B' + AB$

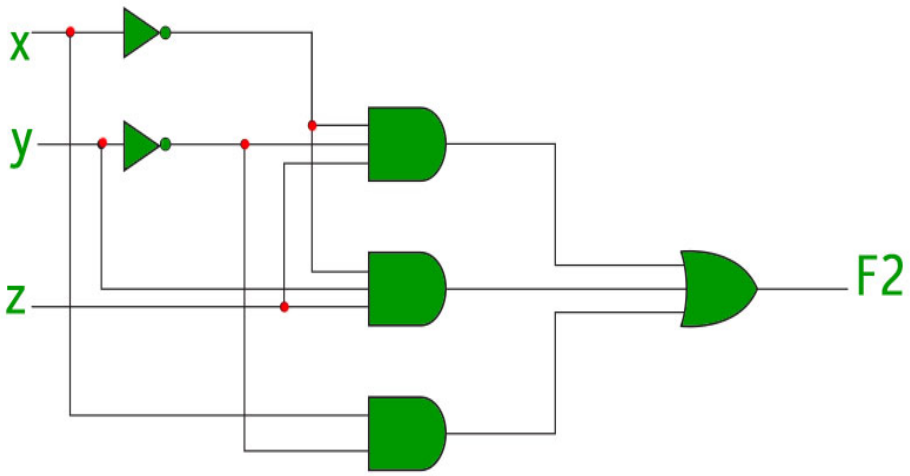
Truth Table:

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

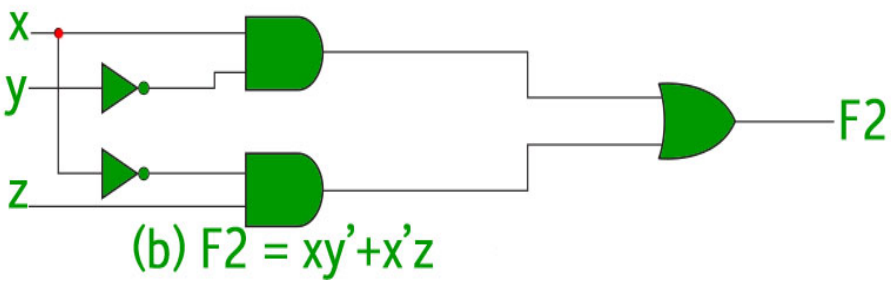
Minimization:

The process of simplifying the algebraic expression of a boolean function is called **minimization**. Minimization is important since it reduces the cost and complexity of the associated circuit.

For example, the function can be minimized . The circuits associated with above expressions is –



(a) F2 = x'y'z+x'yz+xy'



(b) F2 = xy'+x'z

It is clear from the above image that the minimized version of the expression takes a less number of logic gates and also reduces the complexity of the circuit substantially.

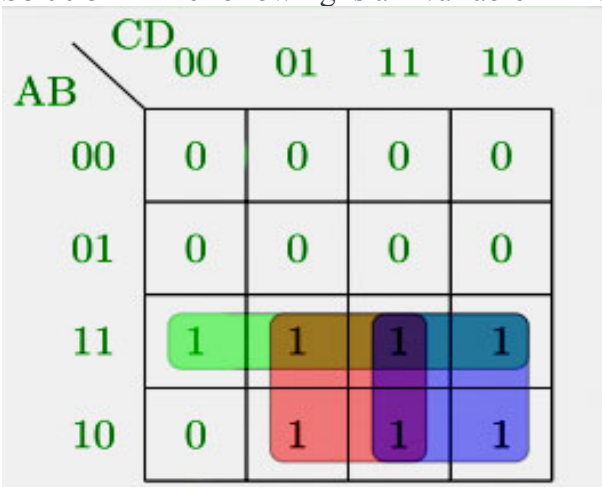
Minimization is hence important to find the most economic equivalent representation of a boolean function.

Minimization can be done using Algebraic Manipulation or K-Map method. Each method has it's own merits and demerits.

Minimization using Algebraic Manipulation
This method is the simplest of all methods used for minimization. It is suitable for medium sized expressions involving 4 or 5 variables. Algebraic manipulation is a manual method, hence it is prone to human error.

Minimization using K-Map –
The Algebraic manipulation method is tedious and cumbersome. The K-Map method is faster and can be used to solve boolean functions of upto 5 variables. **Example 2** – Consider the same expression from example-1 and minimize it using K-Map.

- **Solution –** The following is a 4 variable K-Map of the given expression.



The above figure highlights the prime implicants in green, red and blue.
The green one spans the whole third row, which gives us AB
The red one spans 4 squares, which gives us AC
The blue one spans 4 squares, which gives us AD

- So, the minimized boolean expression is-AD+AC+AB

- **FLIP FLOP**

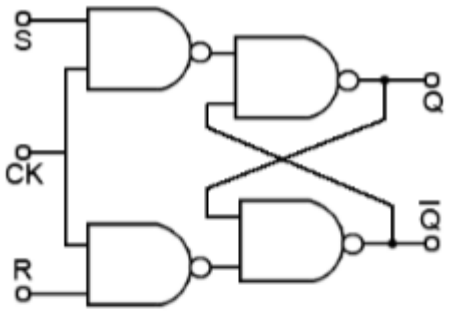
Flip-flop is a circuit that maintains a state until directed by input to change the state. A basic flip-flop can be constructed using four-NAND or four-NOR gates.

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 :

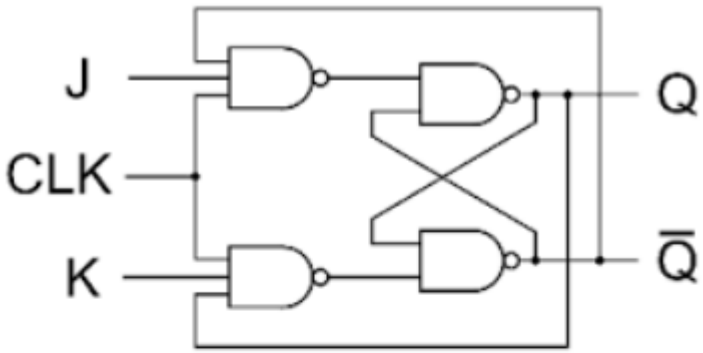


TRUTH TABLE

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:

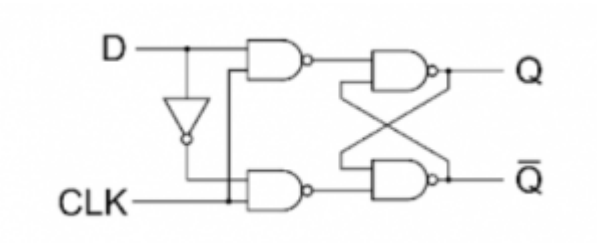


TRUTH TABLE

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: $Q_{N+1} = JQ'_N + K'Q_N$

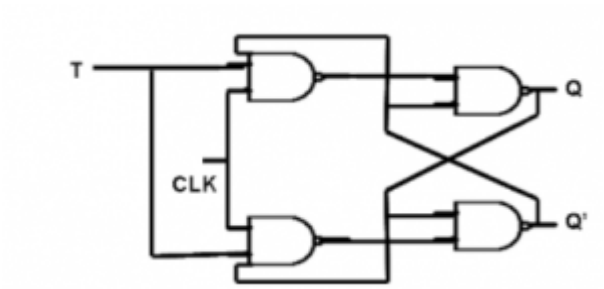
D Flip Flop:



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

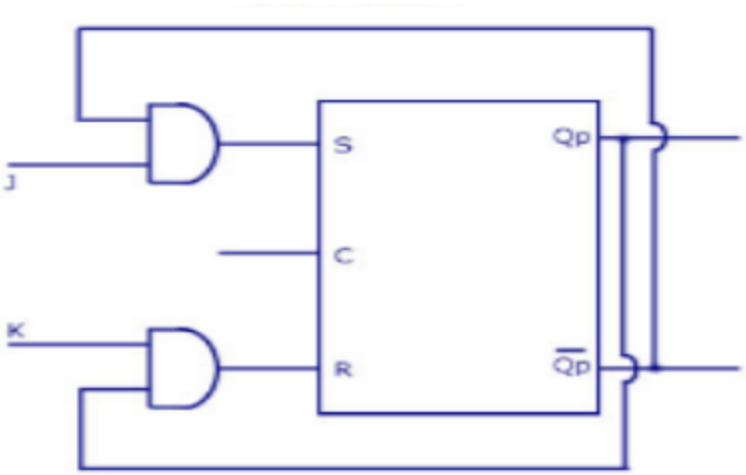
Excitation Functions:

$S = JQ_N'$

KQ_N / J			
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

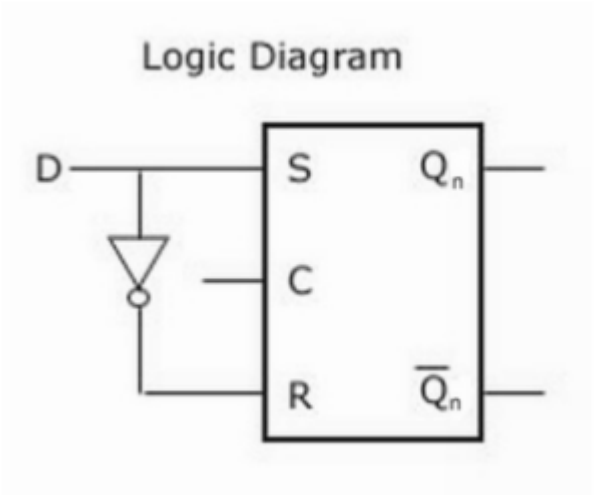
Excitation Functions: S = D, R = D'

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

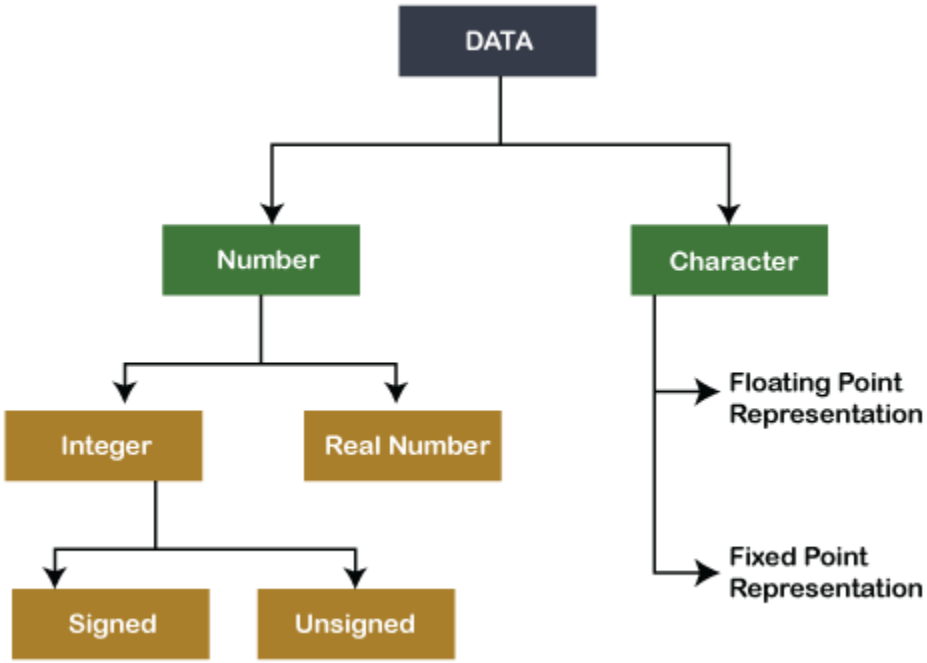
MODULE 2 Data Representation and Operations
Data types, Complements, Other binary codes, Error Detection codes, Register and Micro operations, Register Transfer Language, Register Transfer Bus and Memory Transfers, Arithmetic Micro Operations, Logic Micro Operations, Shift Micro Operations, Arithmetic Logic Shift Unit, Circuits for all Micro Operations.

Data Formats

In the digital world, everything is based on digits. In every method, the calculation is processed on the digits, and most importantly, the computer only understands the binary language (digits 0 & 1). These digits are input Data for the system to fetch, process and execute the instruction.

Data is the raw information or collection of instruction provided to the system to execute that instruction to gain a required output. So it becomes necessary to enter that data in a particular format according to the system's requirement, which is called data formatting.

Types of Data input in Computer



Number
Numbers are digits (0-9) used for calculating (subtraction, addition, multiplication, division and other mathematical calculation). Mathematics is originated with numbers. In a computer system, numbers are represented in different ways.
Ways of number representation are:-

- **Integer**
- **Real Number**

Integer

Integers are the whole Number (not a fraction) represented in the computer as a group of binary digits (bits). These are some of the data types commonly used in computers. Integers can be positive, negative or zero-like 34, -567 or 0, 45563.

Different integers are divided into two parts.

Unsigned type of integer

An unsigned integer can hold a whole number (0-n). Number (0-n) can be zero and positive Number like 0, 36277. Unsigned integers cannot use negative values or numbers.

If the computer has an 8-bit register to store Number, the range of unsigned integer is:-

- Smallest Number = 0000000 =0
- Largest Number = 1111111 = 2^8-1= 255

If the computer has a 16-bit register to store Number, the range of unsigned integer is:-

- Smallest Number = 000000000000000 =0
- Largest Number = 111111111111111 = 2^{16}-1= 65535

If the computer has a 32-bit register to store Number, the range of unsigned integer is:-

- Smallest Number =0
- Largest Number = 2^{32}-1= 4,294,967,295 or about 4 billion.

Generalize formula for n- bit= 0 to 2^n -1

Signed Integer

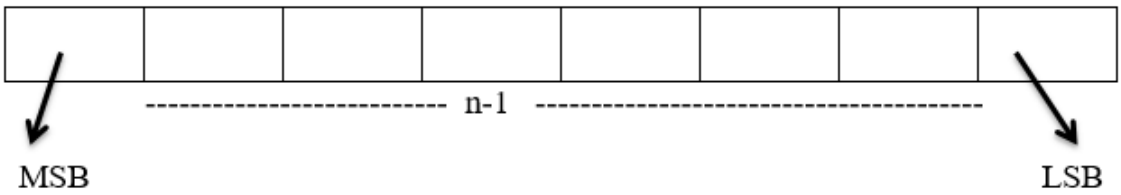
Like unsigned integer, signed integer also represents zero (0), positive value, but it can also represent Negative Number (-234). It is representing real Numbers as fixed-point representation in the system and will discuss further in detail. In a computer system, there are three ways to represent signed integer:-

1. SMR- Sign Magnitude Representation.
2. 1's Complement Representation.
3. 2's Complement representation.

Here, the MSB (Most significant bit) and LSB (least significant bit) are used.
MSB (Most significant bit)- The leftmost and higher-order bit in the binary Number have a higher number. For example, in binary number 1001, the MSB is 1, and in binary number 0011, the MSB is 0.

- If MSB is 0, then the Number evaluated is positive.
- If MSB is one, then the Number evaluated is negative.

LSB (least significant bit) – Least significant bit is the smallest bit in the binary number series at the rightmost side. For example, in binary number 1010, the LSB is 0, and in binary number 0111, the LSB is 1.
The computer only understands binary numbers where 0 is for +ve (positive) and 1 for negative. In representation, 1 bit is reserved for the Number.



Let, Total number of bits = n
Sign bit = 0 (it means number is positive)
Sign Bit = 1(It means Number is negative)
Important Point – For all positive signed integer numbers, all the three ways (sign-magnitude, 1's complement, 2's complement) have the same representation.
Representation of 25 in 8 bit using SMR, 1's complement and 2's complement.

0	**0**	**0**	**1**	**1**	**0**	**0**	**1**

Largest +ve(positive) number (n bits)= $2^{n-1} - 1$

Largest Unsigned Integer = $2^n - 1$

Representation of -25 in 8-bit using SMR.

1	**0**	**0**	**1**	**1**	**0**	**0**	**1**

1's complement for negative Number

The working rule for finding 1's complement of negative Number.
Step 1: Represent the positive Binary of the given Number.
Step 2: Inverse the digits (make 0 =>1 and 1 =>0). Hence we get 1's compliment of that negative number.
Representation of -45 (10101101) in 8 bit using 1's complement.

1	**0**	**1**	**0**	**1**	**1**	**0**	**1**

0	**1**	**0**	**1**	**0**	**0**	**1**	**0**

2's complement for negative Number
The working rule for finding 2's complement of negative Number.
Step 1: Find the Binary of positive Numbers.
Step 2: Scan that binary number from right to left , when 1 appears first then write the 1 as it is and inverse rest of the remaining digits . For example :-

Number = 1 0 1 1 1 0 1 0 0 0

Solution = 1 1 0 0 0 1 1 0 0 0

General formula for 'n' bits

SMR = $-2^{n-1}-1$ to $2^{n-1}-1$

1's Compliment = $-2^{n-1}-1$ to $2^{n-1}-1$

2's Compliment = -2^{n-1} to $2^{n-1}-1$

Characters

Characters are words(letters, alphabet) that we input in the system, but we cannot represent characters directly on a computer as we do with binary numbers. So to solve this, Scientists have given some different methods in Binary to represent the characters.
Method of representing a character in Binary called ALPHA NUMERIC CODES and following are some codes:-

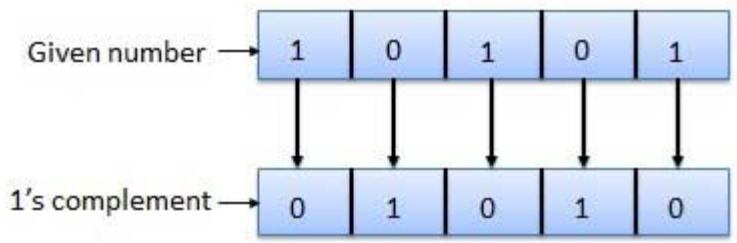
- **MORSE CODE-** It is the first alphanumeric code invented by FB morse. This symbol is used to represent some tasks.

Binary system complements

As the binary system has base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.

1's complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.

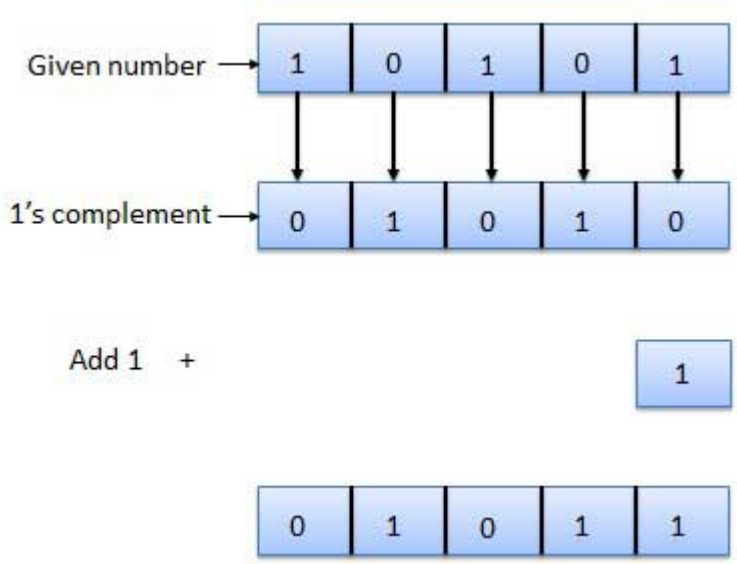


2's complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

2's complement = 1's complement + 1

Example of 2's Complement is as follows.



Binary Codes:

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **binary code**. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

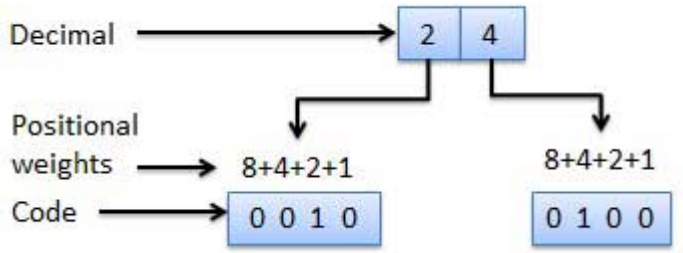
Classification of binary codes

The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes

Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

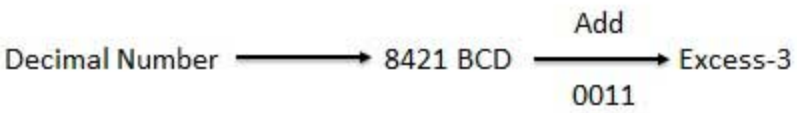


Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Excess-3 code

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows −



Example

Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Gray Code

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

Application of Gray code

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

Binary Coded Decimal (BCD) code

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

Alphanumeric codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).
- Five bit Baudot Code.

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

Error Detection Codes:

That means a bit 0 may change to 1 or a bit 1 may change to 0. We can't avoid the interference of noise. But, we can get back the original data first by detecting whether any errors present and then correcting those errors. For this purpose, we can use the following codes.

- Error detection codes
- Error correction codes

Error detection codes − are used to detect the errors present in the received data bitstream. These codes contain some bits, which are included appended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bitstream. **Example** − Parity code, Hamming code.

Error correction codes − are used to correct the errors present in the received data bitstream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes. **Example** − Hamming code.
Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

Parity Code

It is easy to include append one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in **even parity code**. Even parity code contains the data bits and even parity bit.

The following table shows the **even parity codes** corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- If the other system receives other than even parity codes, then there will be an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Odd Parity Code

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.

The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Hamming Code

Hamming code is useful for both detection and correction of error present in the received data. This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.

The **minimum value of 'k'** for which the following relation is correct valid is nothing but the required number of parity bits.

$$2k \geq n+k+12 \geq ++1$$

Where,

'n' is the number of bits in the binary code information

'k' is the number of parity bits

Therefore, the number of bits in the Hamming code is equal to n + k.

Let the **Hamming code** is bn+kbn+k−1.....b3b2b1−1.....1 & parity bits pk,pk−1,....p1−1,....1. We can place the 'k' parity bits in powers of 2 positions only. In remaining bit positions, we can place the 'n' bits of binary code.

Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.

Follow this procedure for finding **parity bits**.

- Find the value of $\mathbf{p_1}$, based on the number of ones present in bit positions b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of $\mathbf{p_2}$, based on the number of ones present in bit positions b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of $\mathbf{p_3}$, based on the number of ones present in bit positions b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of parity bits.

Follow this procedure for finding **check bits**.

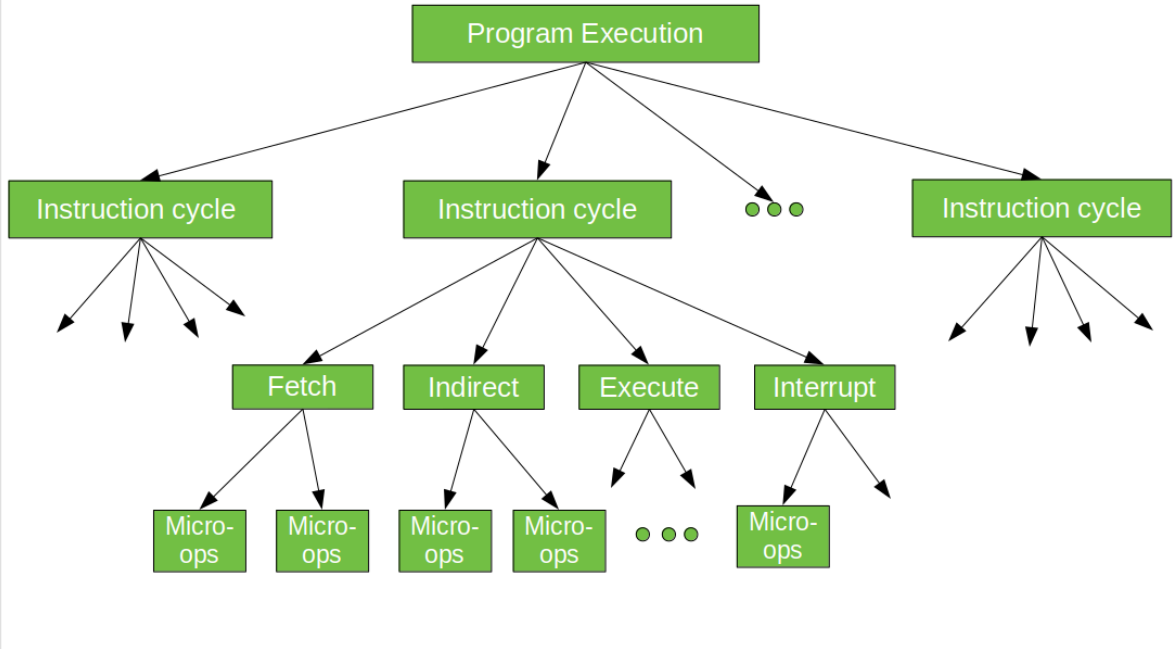
- Find the value of c_1, based on the number of ones present in bit positions b_1, b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of c_2, based on the number of ones present in bit positions b_2, b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of c_3, based on the number of ones present in bit positions b_4, b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of check bits.

The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present. Just complement the value present in that bit position. Therefore, we will get the original binary code after removing parity bits.

Micro Operations

In computer central processing units, **micro-operations** (also known as micro-ops) are the functional or atomic, operations of a processor. These are low level instructions used in some designs to implement complex machine instructions. They generally perform operations on data stored in one or more registers. They transfer data between registers or between external buses of the CPU, also performs arithmetic and logical operations on registers.

In executing a program, operation of a computer consists of a sequence of instruction cycles, with one machine instruction per cycle. Each instruction cycle is made up of a number of smaller units – *Fetch, Indirect, Execute and Interrupt cycles.* Each of these cycles involves series of steps, each of which involves the processor registers. These steps are referred as micro-operations. the prefix micro refers to the fact that each of the step is very simple and accomplishes very little. Figure below depicts the concept being discussed here.



Miniature tasks are performed on the information put away in the registers inside the computer chip. They are utilized to perform math and intelligent activities, as well as to move information among registers and memory. A few instances of miniature tasks include:

1.Load: This miniature activity loads information from memory into a register.
2.Store: This miniature activity stores information from a register into memory.
3.Add: This miniature activity adds two qualities and stores the outcome in a register.
4.Subtract: This miniature activity deducts two qualities and stores the outcome in a register.
5.And: This miniature activity plays out a legitimate AND procedure on two qualities and stores the outcome in a register.
6.Or: This miniature activity plays out a legitimate OR procedure on two qualities and stores the outcome in a register.
7.Not: This miniature activity plays out a legitimate NOT procedure on a worth and stores the outcome in a register.
8.Shift: This miniature activity moves the pieces of a worth to the left or right.
9.Rotate: This miniature activity pivots the pieces of a worth to the left or right.

Miniature activities are consolidated to frame more elevated level guidelines and tasks. For instance, an option activity might be executed utilizing various miniature tasks, including a heap activity to stack the qualities into registers, an add activity to play out the option, and a store activity to store the outcome in memory.

Register Transfer Language (RTL)

In symbolic notation, it is used to describe the micro-operations transfer among registers. It is a kind of intermediate representation (IR) that is very close to assembly language, such as that which is used in a compiler. The term "Register Transfer" can perform micro-operations and transfer the result of operation to the same or other register.

Micro operations:

The operation executed on the data store in registers are called micro-operations. They are detailed low-level instructions used in some designs to implement complex machine instructions.

Register Transfer:

The information transformed from one register to another register is represented in symbolic form by replacement operator is called Register Transfer.

Replacement Operator:

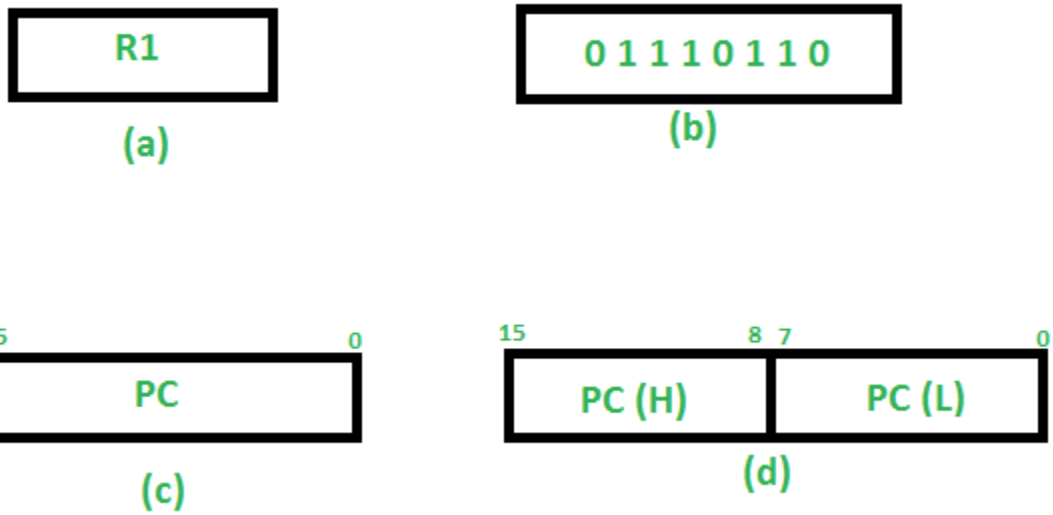
In the statement, R2 <- R1, **<-** acts as a replacement operator. This statement defines the transfer of content of register R1 into register R2.
There are various methods of RTL –

1. General way of representing a register is by the name of the register enclosed in a rectangular box as shown in (a).

2. Register is numbered in a sequence of 0 to (n-1) as shown in (b).

3. The numbering of bits in a register can be marked on the top of the box as shown in (c).

4. A 16-bit register PC is divided into 2 parts- Bits (0 to 7) are assigned with lower byte of 16-bit address and bits (8 to 15) are assigned with higher bytes of 16-bit address as shown in (d).



Basic symbols of RTL :

Symbol	Description	Example
Letters and Numbers	Denotes a Register	MAR, R1, R2
( )	Denotes a part of register	R1(8-bit) R1(0-7)
<-	Denotes a transfer of information	R2 <- R1
,	Specify two micro-operations of Register Transfer	R1 <- R2 R2 <- R1
:	Denotes conditional operations	P : R2 <- R1 if P=1
Naming Operator (:=)	Denotes another name for an already existing register/alias	Ra := R1

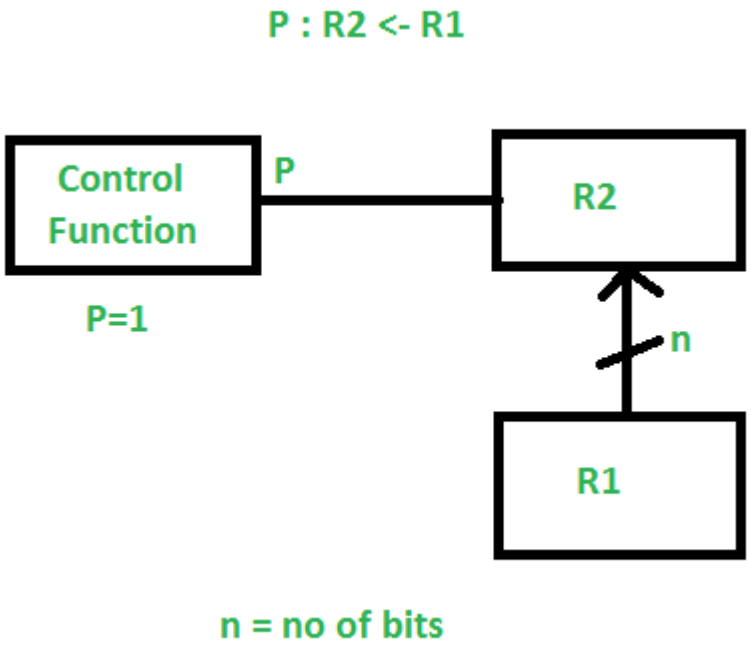
Register Transfer Operations:
The operation performed on the data stored in the registers are referred to as register transfer operations.

There are different types of register transfer operations:

1. Simple Transfer – R2 <- R1

The content of R1 are copied into R2 without affecting the content of R1. It is an unconditional type of transfer operation.

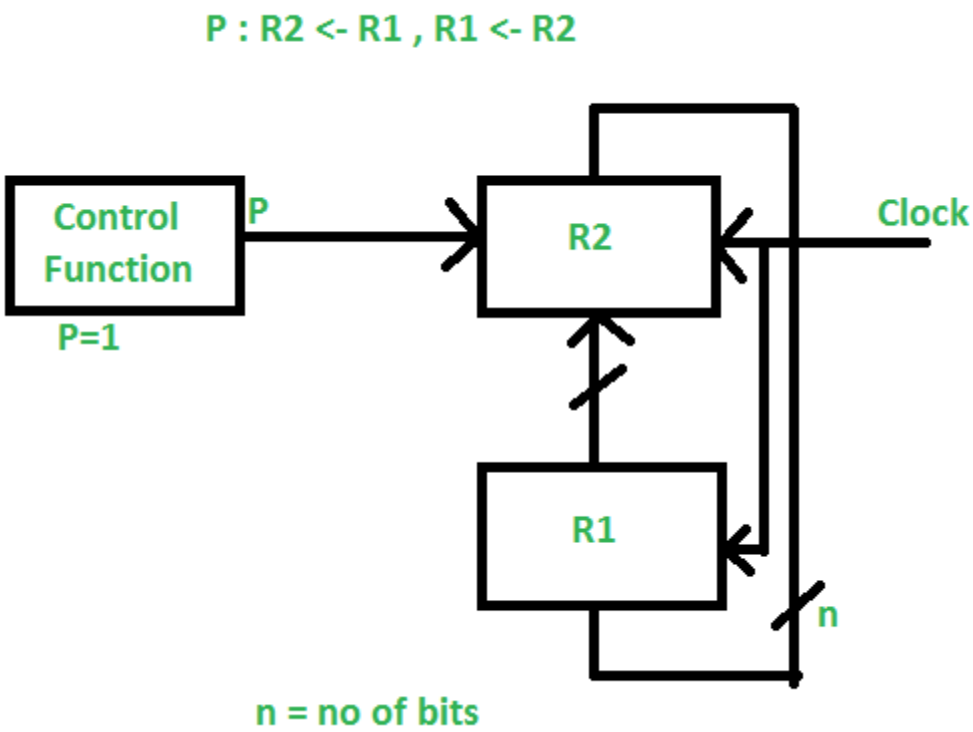
2. Conditional Transfer –



It indicates that if P=1, then the content of R1 is transferred to R2. It is a unidirectional operation.

3. Simultaneous Operations –

If 2 or more operations are to occur simultaneously then they are separated with comma **(,)**.



If the control function P=1, then load the content of R1 into R2 and at the same clock load the content of R2 into R1.

We can perform arithmetic operations on the numeric data which is stored inside the registers.

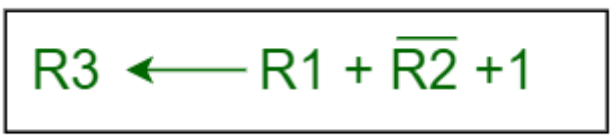
Example :
R3 <- R1 + R2

The value in register R1 is added to the value in the register R2 and then the sum is transferred into register R3. Similarly, other arithmetic micro-operations are performed on the registers.

- **Addition –**
  In addition micro-operation, the value in register R1 is added to the value in the register R2 and then the sum is transferred into register R3.

- **Subtraction –**
  In subtraction micro-operation, the contents of register R2 are subtracted from contents of the register R1, and then the result is transferred into R3.

There is another way of doing the subtraction. In this, 2's complement of R2 is added to R1, which is equivalent to **R1 – R2**, and then the result is transferred into register R3.



- **Increment –**
  In Increment micro-operation, the value inside the R1 register is increased by 1.

- **Decrement –**
  In Decrement micro-operation, the value inside the R1 register is decreased by 1.

- **1's Complement –**
  In this micro-operation, the complement of the value inside the register R1 is taken.

- **2's Complement –**
  In this micro-operation, the complement of the value inside the register R2 is taken and then 1 is added to the value and then the final result is transferred into the register R2. This process is also called Negation. It is equivalent to **-R2**.

Arithmetic micro-operations are the basic building blocks of arithmetic operations performed by a computer's central processing unit (CPU). These micro-operations are executed on the data stored in registers, which are small, high-speed storage units within the CPU.

There are several types of arithmetic micro-operations that can be performed on register data, including:

1. Addition: This micro-operation adds two values together and stores the result in a register.
2. Subtraction: This micro-operation subtracts one value from another and stores the result in a register.
3. Increment: This micro-operation adds 1 to the value in a register.
4. Decrement: This micro-operation subtracts 1 from the value in a register.
5. Multiplication: This micro-operation multiplies two values together and stores the result in a register.
6. Division: This micro-operation divides one value by another and stores the quotient and remainder in separate registers.
7. Shift: This micro-operation shifts the bits in a register to the left or right, depending on the direction specified.

These arithmetic micro-operations are used in combination with logical micro-operations, such as AND, OR, and NOT, to perform more complex calculations and manipulate data within the CPU.

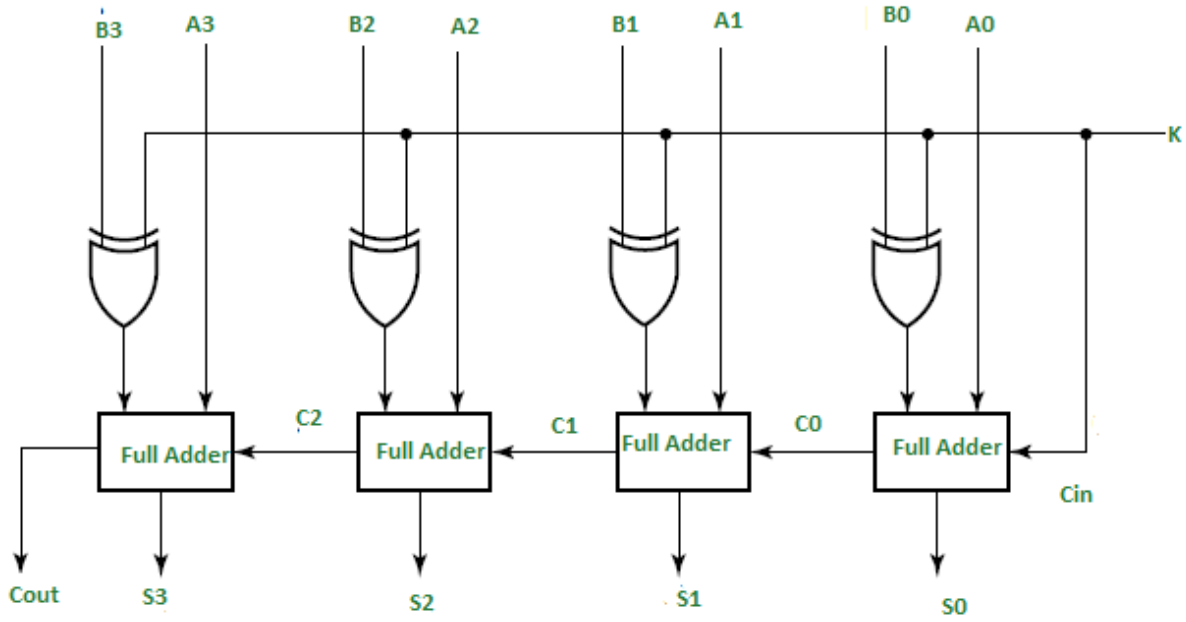
In Digital Circuits, A **Binary Adder-Subtractor** is capable of both the addition and subtraction of binary numbers in one circuit itself. The operation is performed depending on the binary value the control signal holds. It is one of the components of the ALU (Arithmetic Logic Unit).

Let's consider two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

A0 A1 A2 A3 for A

B0 B1 B2 B3 for B

The circuit consists of 4 full adders since we are performing operations on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation is carried out is addition or subtraction.



As shown in the figure, the first full adder has a control line directly as its input(input carry Cin), The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the exor of B0 and K. The two outputs produced are Sum/Difference (S0) and Carry (C0).

If the value of K (Control line) is 1, the output of B0(exor)K=B0′(Complement B0). Thus the operation would be A+(B0′). Now 2's complement subtraction for two numbers A and B is given by A+B'+Cin. This suggests that when K=1, the operation being performed on the four-bit numbers is subtraction.

Similarly If the Value of K=0, B0 (exor) K=B0. The operation is A+B which is simple binary addition. This suggests that When K=0, the operation is performed on the four-bit numbers in addition.

Then C0 is serially passed to the second full adder as one of it's outputs. The sum/difference S0 is recorded as the least significant bit of the sum/difference. A1, A2, A3 are direct inputs to the second, third and fourth full adders. Then the third input is the B1, B2, B3 EXOR ed with K to the second, third and fourth full adder respectively. The carry C1, C2 are serially passed to the successive full adder as one of the inputs. C3 becomes the total carry to the sum/difference. S1, S2, S3 are recorded to form the result with S0.

For an n-bit binary adder-subtractor, we use n number of full adders.

Binary Adder

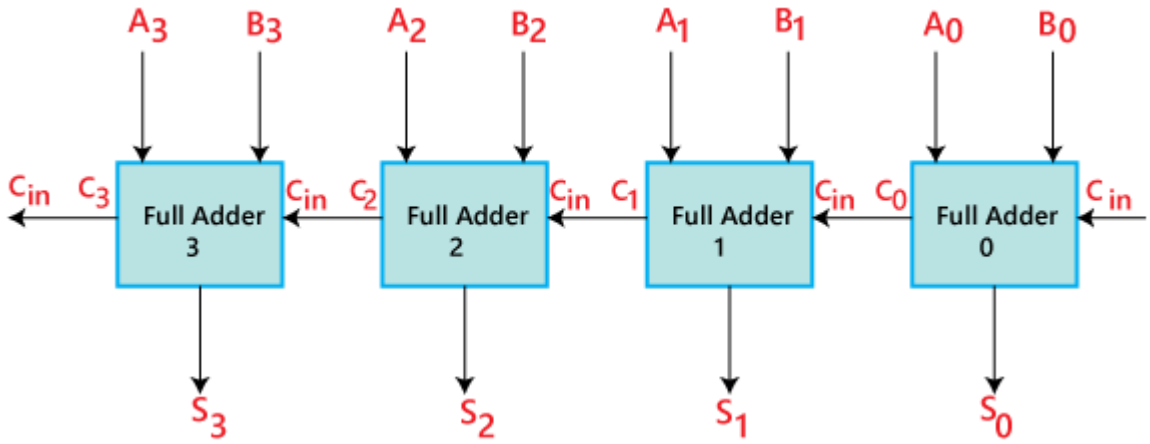
The registers play an important role in performing the micro-operations. The registers hold the digital component and the data which performs the arithmetic operation. The Binary Adder is a logical circuit which is used to perform the addition operation of two binary number of any length.

The Binary Adder is formed with the help of the Full-Adder circuit. The Full-Adders are connected in series, and the output carry of the first Adder will be treated as the input carry of the next Full-Adder.

N-Bit Parallel Adder

The Full Adder is used to sum two single-bit binary numbers with carry input. In digital calculation, we need to add two n-bit binary numbers rather than only single-bit binary numbers. For this purpose, we need to use n-bit parallel Adder. In order to get N-bit parallel adder, we cascade the n number of Full Adders. The carry output of the first Adder is treated as the carry input of the second Adder.

4-bit Binary Adder

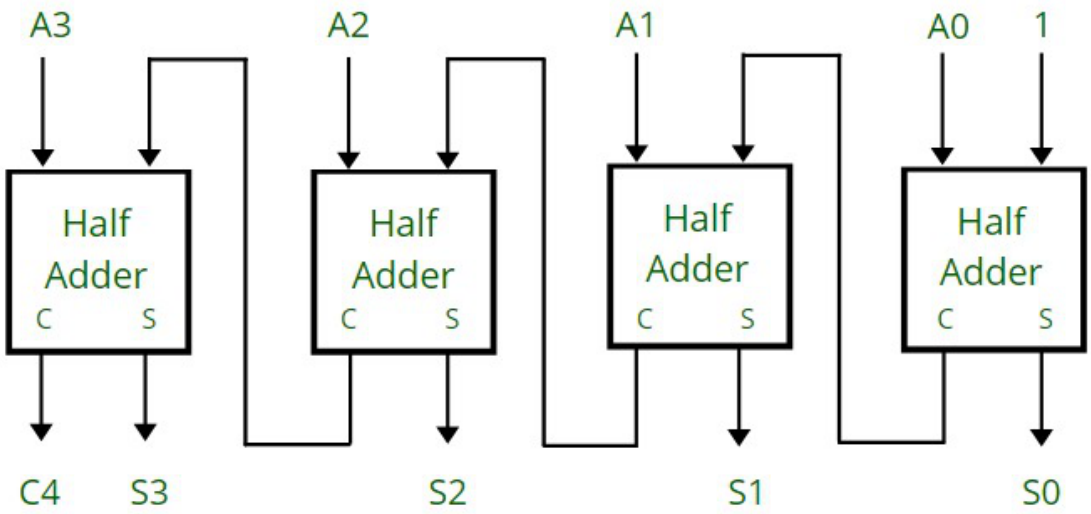


- The 'A' and 'B' are the augend, and addend bits are defined by the subscript numbers. The subscripts start from right to left, and the lower-order bit is defined by subscript '0'.
- The C0, C1, C2, and C3 are the carry inputs which are connected together as a chain using Full Adder. The C4 is the carry output produced by the last Full-Adder.
- The C_{out} of the first Adder is connected as the C_{in} of the next Full-Adder.
- The S0, S1, S2, and S3 are the sum outputs that produce the sum of augend and addend bits.
- The inputs for the input variable 'A' and 'B' are fetched from different source registers. For example, the bit for the input variable 'A' comes from register 'R1', and a bit for the input variable 'B' comes from register 'R2'.

- The outcome produced by adding both input variables is stored into either third register or to one of the source registers.

What is 4 Bit Binary Incrementer ?

It adds 1 binary value to the existing binary value stored in the register or in other words we can simply say that it increases the value stored in the register by 1. For any n-bit binary incrementer,'n' refers to the storage capacity of the register which needs to be incremented by 1. So we require 'n' number of half adders. Thus, in case of 4 bit binary incrementer we require 4 half adders.



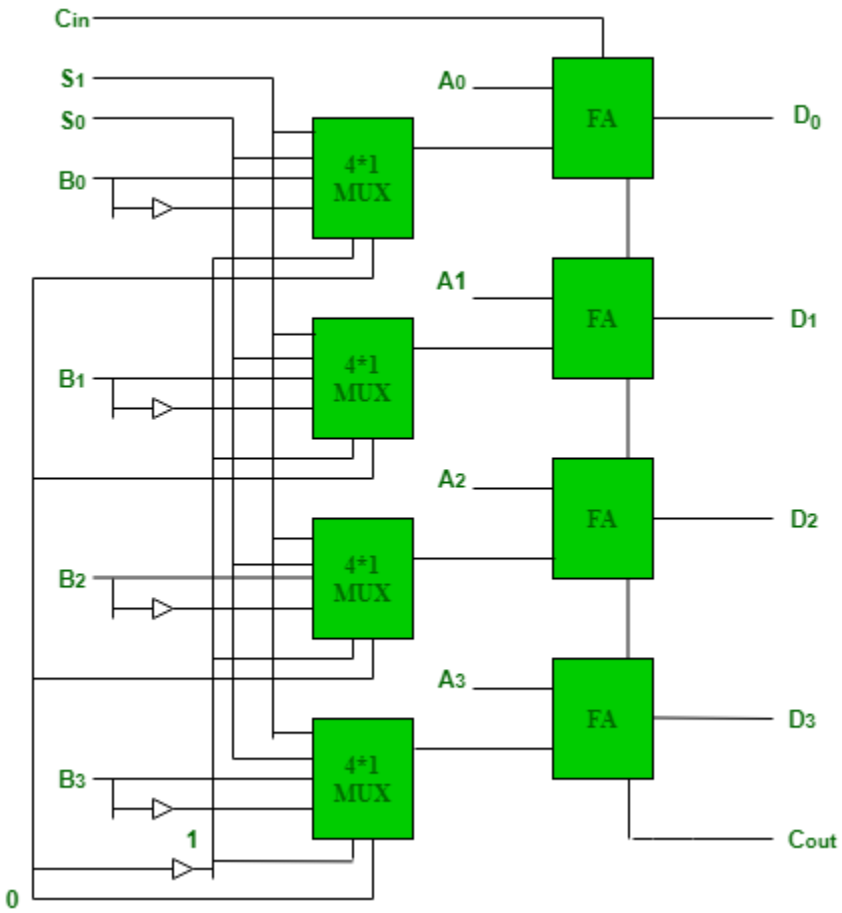
4- Bit Binary Incrementer

Working:

- The half adders are connected one after the other , as it has 2 inputs and 2 outputs , so for the LSB (least significant bit) half adder or the right most half adder is given 1 as direct input(first input) and A0 which is the first bit of the register (second input) , so we get the two output : sum (S0) and carry (C).
- The carry(C) from previous half adder is propagated to the next half adder, so the carry output of the previous half adder becomes the input of the next higher order half adder.
- So considering the case for 4 half adders the circuit gets in total 4 bits (A0, A1, A2, A3), 1 is added and we get an incremented output.

Arithmetic circuits can perform seven different arithmetic operations using a single composite circuit. It uses a full adder (FA) to perform these operations. A multiplexer (MUX) is used to provide different inputs to the circuit in order to obtain different arithmetic operations as outputs.

4-bit Arithmetic Circuit :
Consider the following 4-bit Arithmetic circuit with inputs A and B. It can perform seven different arithmetic operations by varying the inputs of the multiplexer and the carry (C_0).



Truth Table for the above Arithmetic Circuit :

S_0	S_1	C_0	MUX Output	Full Adder Output
0	0	0	B	A + B
0	0	1	B	A + B + 1
0	1	0	B’	A + B’
0	1	1	B’	A + B’ + 1 = A – B
1	0	0	0	A
1	0	1	0	A + 1
1	1	0	1	A – 1

1	1	1	1	A – 1 + 1 = A

Hence, the different operations for the inputs A and B are –

1. A + B (adder)
2. A + B + 1
3. A + B’
4. A – B (subtracter)
5. A
6. A + 1 (incrementer)
7. A – 1 (decrementer)

Logic Micro operations

Logic operations are binary micro-operations implemented on the bits saved in the registers. These operations treated each bit independently and create them as binary variables.

For example, the exclusive-OR micro-operation with the contents of two registers R1 and R2 is denoted by the statement

P: R1←R1⊕⊕R2

It determines a logic micro-operation to be implemented on the single bits of the registers supported that the control variable P = 1. Consider that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100.

The exclusive-OR micro-operation stated above represent the following logic computation −

1010 Content of R1

1100 Content of R2

0110 Content of R1 after P = 1

The content of R1, after the implementation of the micro-operation, is similar to the bit-by-bit exclusive-OR operation on pairs of bits in R2 and previous values of R1.

Special Symbols

Special symbols will be approved for the logic micro-operations OR, AND, and complement, to categorize them from the matching symbols that can define Boolean functions. The symbol V can indicate an OR micro-operation and the symbol can indicate an AND micro-operation.

The complement micro-operation is similar to the 1's complement and supports a bar on the highest of the symbol that indicates the registered name. There are various symbols, and it will be applicable to differentiate between a logic micro-operation and a control (or Boolean) function.

There is another sense for supporting two sets of symbols can that recognize the symbol +, when can symbolize arithmetic plus, from a logic OR operation. Although the + symbol has two meanings, it will be available to determine between them by observing where the symbol appears.

When the symbol + appears in a micro-operation, it will indicate an arithmetic plus. When it appears in a control (or Boolean) function, it will indicate an OR operation. We cannot use it to symbolize an OR micro-operation.

For example, in the statement

P+Q: R1←R2+R3,R 4←R5V R6

The + between P and Q is an OR operation between two binary variables of a control function. The + between R2 and R3 determines an add micro-operation. The OR micro-operation is named by the symbol V between registers R5 and R6.

SHIFT MICRO-OPERATIONS:

Shift micro-operations are those micro-operations that are used for the serial transfer of information. These are also used in conjunction with arithmetic micro-operation, logic micro-operation, and other data-processing operations. There are three types of shift micro-operations:

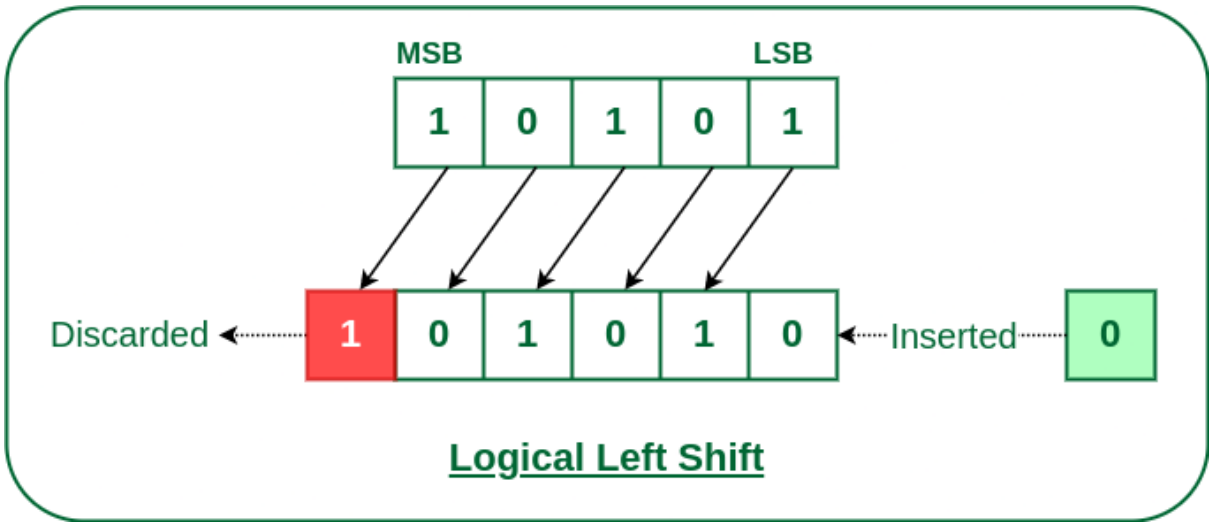
1. Logical Shift:

It transfers the 0 zero through the serial input. We use the symbols '<<' for the logical left shift and '>>' for the logical right shift.

Logical Left Shift:

In this shift, one position moves each bit to the left one by one. The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected.

The left shift operator is denoted by the double left arrow key (<<). The general syntax for the left shift is shift-expression << k.

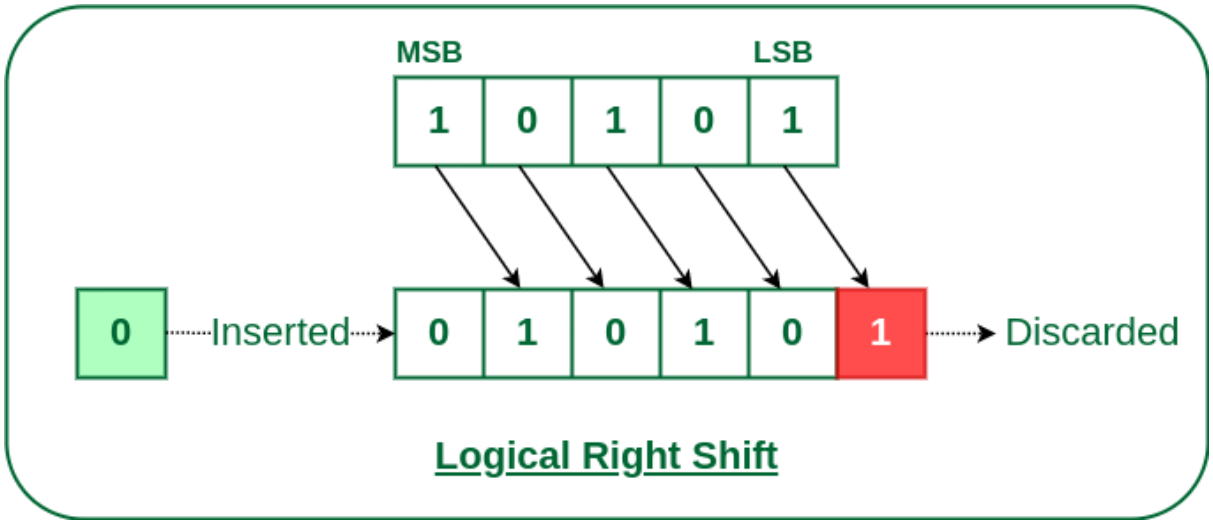


Logical Left Shift

Logical Right Shift

In this shift, each bit moves to the right one by one and the least significant bit(LSB) is rejected and the empty MSB is filled with zero.

The right shift operator is denoted by the double right arrow key (>>). The general syntax for the right shift is "shift-expression >> k".



Logical Right Shift

2. Arithmetic Shift:

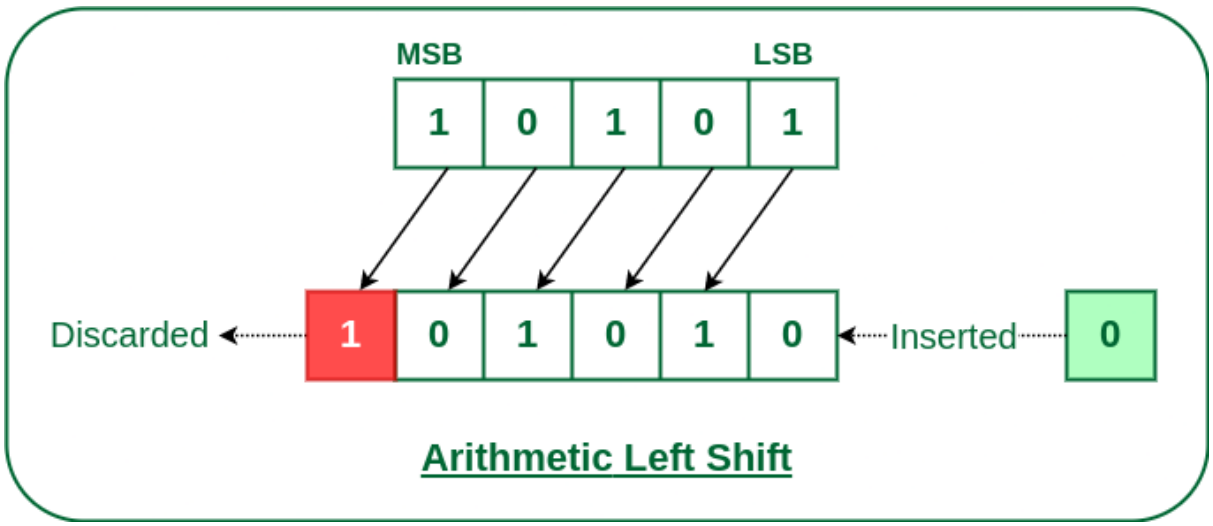
The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position.

Following are the two ways to perform the arithmetic shift.

1. Arithmetic Left Shift
2. Arithmetic Right Shift

Arithmetic Left Shift:

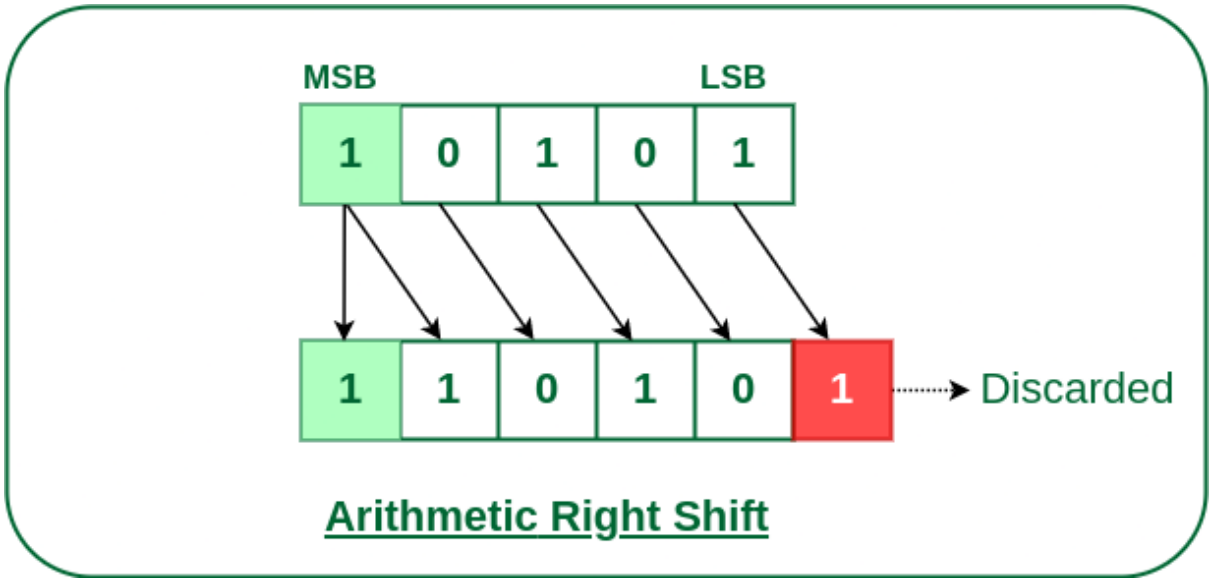
In this shift, each bit is moved to the left one by one. The empty least significant bit (LSB) is filled with zero and the most significant bit (MSB) is rejected. Same as the Left Logical Shift.



Arithmetic Left Shift

Arithmetic Right Shift:

In this shift, each bit is moved to the right one by one and the least significant(LSB) bit is rejected and the empty most significant bit(MSB) is filled with the value of the previous MSB.



Arithmetic Right Shift

3. Circular Shift:

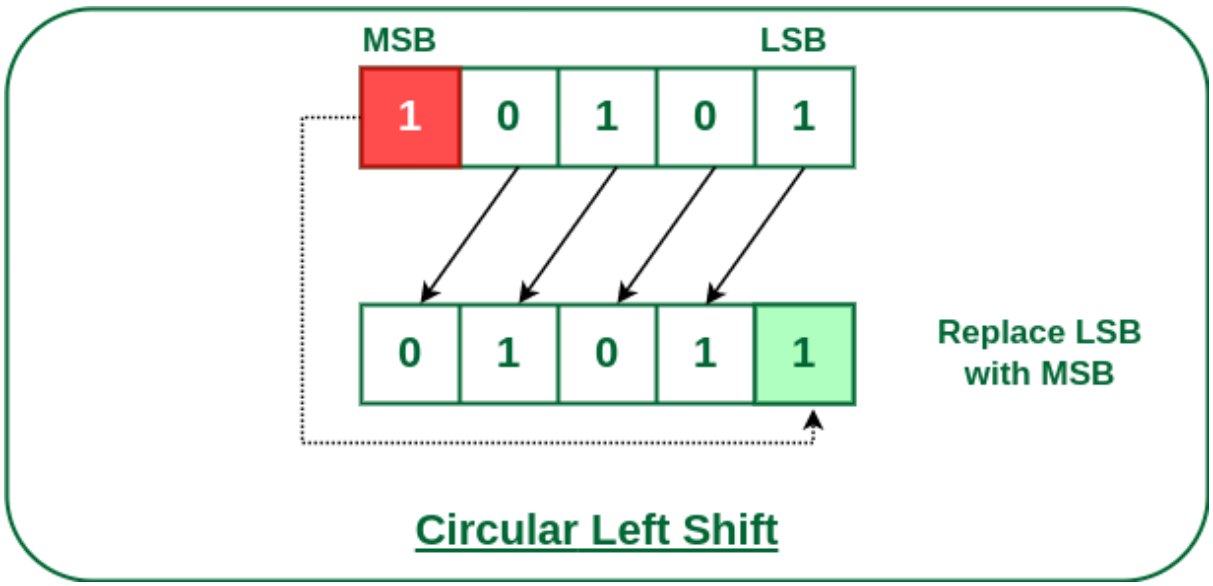
The circular shift circulates the bits in the sequence of the register around both ends without any loss of information.

Following are the two ways to perform the circular shift.

1. Circular Shift Left
2. Circular Shift Right

Circular Left Shift:

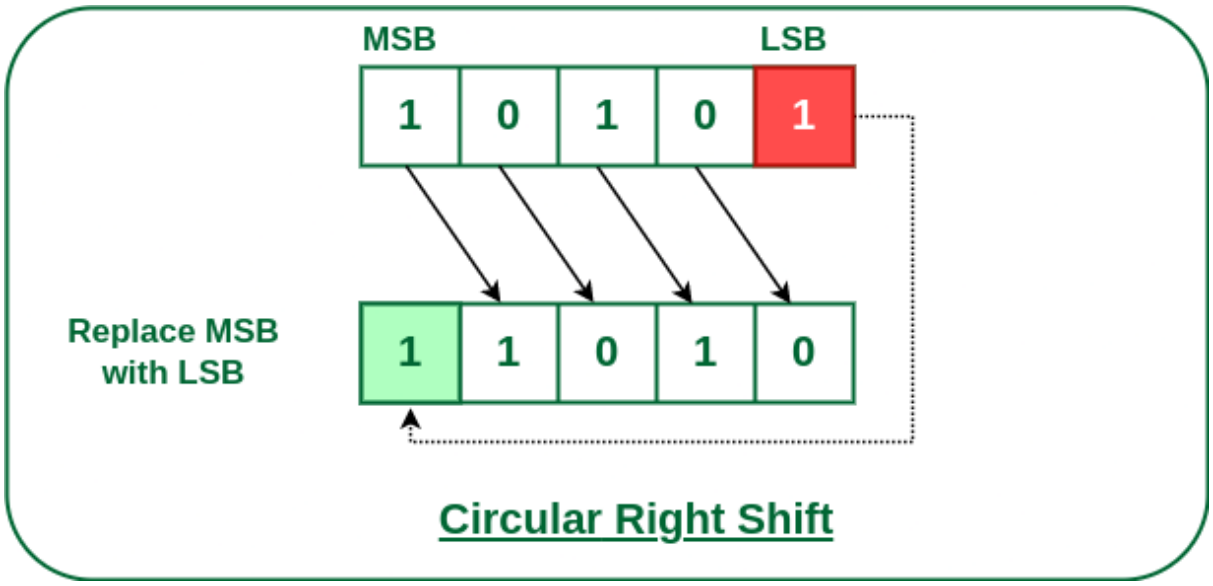
In this micro shift operation each bit in the register is shifted to the left one by one. After shifting, the LSB becomes empty, so the value of the MSB is filled in there.



Circular Left Shift

Circular Right Shift:

In this micro shift operation each bit in the register is shifted to the right one by one. After shifting, the MSB becomes empty, so the value of the LSB is filled in there.

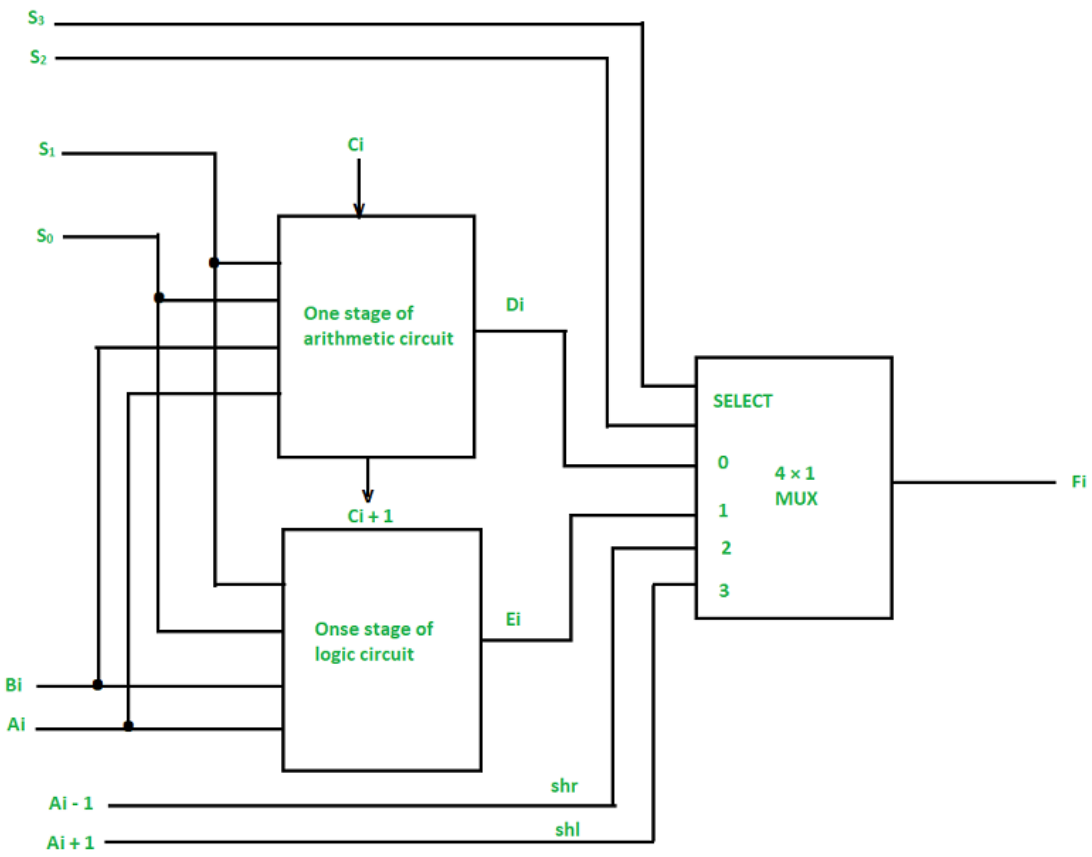


Circular Right Shift

Arithmetic Logic Shift Unit

Arithmetic Logic Shift Unit (ALSU) is a member of the **Arithmetic Logic Unit (ALU)** in a computer system. It is a digital circuit that performs logical, arithmetic, and shift operations. Rather than having individual registers calculating the micro operations directly, the computer deploys a number of storage registers which is connected to a common operational unit known as an arithmetic logic unit or ALU.

The Arithmetic Logic Unit performs an operation that leads as a result and gets transferred to a destination register. Arithmetic Logic Unit may be a combinatory circuit in order that the complete register transfer operation from the supply registers through the ALU and into the destination register is performed throughout one clock pulse amount. Sometimes, the shift micro operations are performed in a separate unit, but sometimes it is made as a part of full ALU.



One stage of ALSU

We can combine and make one ALU with common selection variables by adding arithmetic, logic, and shift circuits. We can see the, One stage of an arithmetic logic shift unit in the diagram below. Some particular micro operations are selected through the inputs S1 and S0.

4 x 1 multiplexer at the output chooses between associate arithmetic output between Ei and a logic output in Hi. The data in the multiplexer are selected through inputs S3 and S2 and the other two data inputs to the multiplexer obtain the inputs Ai – 1 for the ***shr*** operation and Ai + 1 for the ***shl*** operation.

Process: The output carry Ci + 1 of a specified arithmetic stage must be attached to the input carry Ci of the next stage in the sequence.
The circuit whose one stage is given in the below diagram provides 8 arithmetic operations, 4 logic operations, and 2 shift operations, and Each operation is selected by the 5 variables S3, S2, S1, S0, and Cin.

The below table shows the 14 operations perform by the Arithmetic Logic Unit:

1. The first 8 are arithmetic operations which are selected by S3 S2 = 00
2. The next 4 are logic operations which are selected by S3 S2 = 01
3. The last two are shift operations which are selected by S3 S2 = 10 & 11

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B'	Subtract with borrow
0	0	1	0	1	F = A + B' + 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F= A	Transfer A
0	1	0	0	×	F = A ^ B	AND
0	1	0	1	×	F = A v B	OR
0	1	1	0	×	F = A XOR B	XOR
0	1	1	1	×	F = A'	Complement A
1	0	×	×	×	F = shr A	Shift right A into F
1	1	×	×	×	F = shl A	Shift left A into F

Function table of ALSU