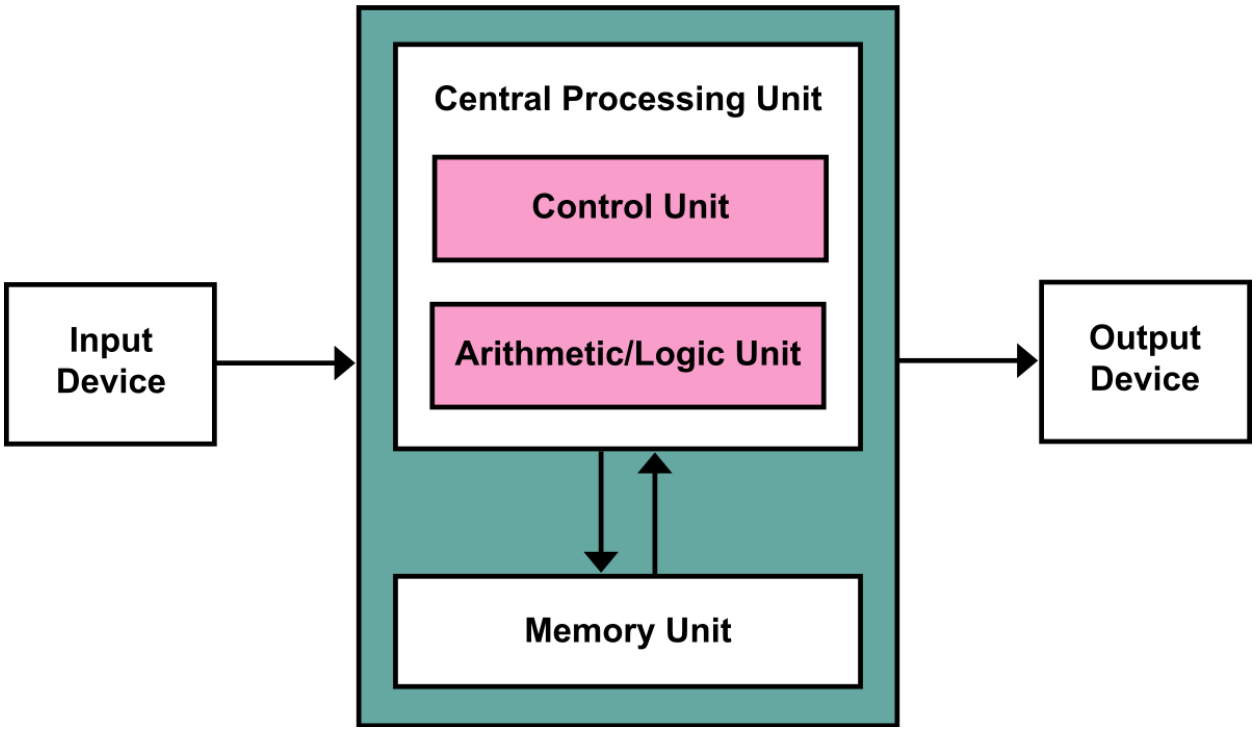
Computer Organization

UNIT 1

TOPIC 1:-

Von Nueman Architecture :



Von-Neumann proposed a computer architecture design in 1945 which was later known as Von-Neumann Architecture. It consisted of a Memory Unit and Central processing Unit which intern consist of Control Unit, Arithmetic and Logical Unit (ALU) and Memory Unit .

Von Neumann architecture is based on the stored-program computer concept, where instructions and program data are stored in the same memory.

Input Device: An **input device** is any hardware device that sends data to a computer, allowing you to interact with and control it. Examples- Keyboard, Mouse, scanner etc.

Output Device: An output device is any hardware device used to send data from a computer to another device or user. Usually, most output peripherals are meant for human use, so they receive the processed data from the computer and transform it in the form of audio, video, or physical reproductions.

Memory Unit: Memory unit is a component of a computer system. It is used to store data, instructions and information. It is actually a work area of computer, where the CPU stores the data and instruction. It is also known as a Primary memory.

Read only memory (ROM):- ROM is a part of the memory unit. This is read only memory. It cannot be used to written. ROM is used in situations where the data must be held permanently.

Random access memory (RAM):- RAM is also part of memory unit. It is used for temporary storage of program data. Its data is lost when power is turned off.

Central Processing Unit: It is the main component of computer, which perform all the processing on input data .The Central Processing Unit can also be defined as an electric circuit responsible for executing the instructions of a computer program. In microcomputers the CPU is a single chip or IC and is called Microprocessor. The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer. The major components of CPU are Arithmetic and Logic Unit (ALU), Control Unit (CU) and a variety of registers.

***Arithmetic Logic Unit (ALU): -** The Arithmetic and logic unit is used for all arithmetic operation like addition, subtraction, multiplication operation such as less then, equal to and greater then. Actually, all calculations and comparisons are performed in the arithmetic logic unit.

***Control Unit (CU):-** The CU is used for controlling the transfer of data and instructions among other units of computer.

***Registers: -** Registers are the small high speed circuits. It is used to store data, instruction and memory address when ALU performs arithmetic and logical operations. Resistor can be divided into four categories: -

1. **Accumulator Register** : This register holds the intermediate arithmetic and logic results.

2. **Program counter Register**: This Register Contains the Address Of the next instruction to be executed.

3. **Instruction Register**: This register contains the current instruction during processing.

4. **Memory Register**: This Register holds the data that is being transferred to or from the memory.

5. **Memory Data Register**: This register holds the memory location of the data that needs to be accessed.

INFORMAL EXPLANATION (FOR UNDERSTANDING PURPOSE ONLY)-

*Von Nueman Architecture works on **Stored Program Concept**, Now what do we mean by this ?*

All the system programs (such as compilers, Operating system etc.) and application Programs (such as paint, MS office etc.) and user defined programs for example C , JAVA programs that we usually build on compiler etc. must be present in Primary memory (which is essentially RAM). So all in all every software or program that executes, its code must be present in RAM and if it is not then it will be bring to RAM from Hard disk.

SO when we double click on any file, program etc. it will be brought in to RAM from Hard disk otherwise it will not execute. So for execution code must be in RAM.

Again Question 2 – Now how do you make a computer? So first we need a blue print and then we will follow that blue print design and will arrange the hardware. Now that logical design is what we call architecture and when we implement that design in real life then it is called Organization.

Now let us see the architecture or the design so to make a computer we need 1. CPU so for processing and to do various types of calculations 2. Some devices to feed or give the data to CPU known as input devices 3. Then we need Memory to hold the data . 3. To show the output we need output devices. 4. Now for all these things to communicate properly there is control circuit or control unit.

Now what is Register ?

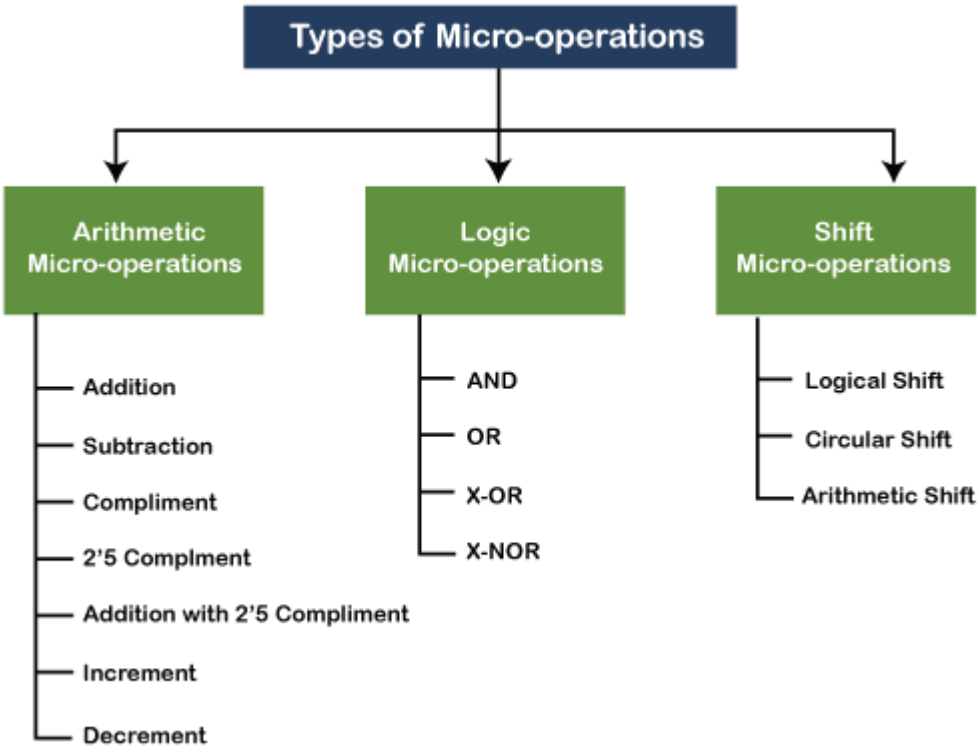
A **Register** is a group of flip-flops with each flip-flop capable of storing **one bit** of information. An *n-bit* register has a group of *n flip-flops* and is capable of storing binary information of *n-bits*.

A register consists of a group of flip-flops and gates. The flip-flops hold the binary information and gates control when and how new information is transferred into a register.

Think of registers as storage space like a variable which holds data and registers reside in the processor.

TOPIC 2:-

MICRO OPERATIONS:-



Micro-Operations

The operations executed on data stored in registers are called micro-operations. A micro-operation is an elementary operation performed on the information stored in one or more registers.

Example: Shift, count, clear and load.

Types of Micro-Operations

The micro-operations in digital computers are of 4 types:

1. **Register transfer micro-operations** transfer binary information from one register to another.

2. **Arithmetic micro-operations** perform arithmetic operations on numeric data stored in registers.
3. **Logic micro-operations** perform bit manipulation operation on non-numeric data stored in registers.
4. **Shift micro-operations** perform shift micro-operations performed on data.

Arithmetic Micro-Operations

Some of the basic micro-operations are addition, subtraction, increment and decrement.

Add Micro-Operation
It is defined by the following statement:

```
R3 → R1 + R2
```

The above statement instructs the data or contents of register R1 to be added to data or content of register R2 and the sum should be transferred to register R3.

Subtract Micro-Operation
Let us again take an example:

```
R3 → R1 + R2' + 1
```

In subtract micro-operation, instead of using minus operator we take **1's compliment** and add 1 to the register which gets subtracted, i.e **R1 - R2** is equivalent to **R3 → R1 + R2' + 1**

Increment/Decrement Micro-Operation
Increment and decrement micro-operations are generally performed by adding and subtracting 1 to and from the register respectively.

```
R1 → R1 + 1

R1 → R1 − 1
```

Symbolic Designation	Description
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.

R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 - 1	Decrement the contents of R1 by 1.

Logic Micro-Operations

These are binary micro-operations performed on the bits stored in the registers. These operations consider each bit separately and treat them as binary variables.

Let us consider the X-OR micro-operation with the contents of two registers R1 and R2.

```
P: R1 ← R1 X-OR R2
```

In the above statement we have also included a Control Function.

Assume that each register has 3 bits. Let the content of R1 be **010** and R2 be **100**. The X-OR micro-operation will be:

010 → R1

100 → R2

110 → R1 after P=1

Shift Micro-Operations

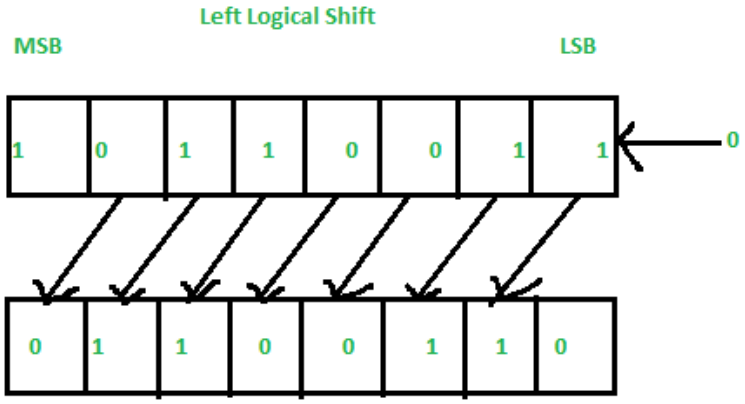
These are used for serial transfer of data. That means we can shift the contents of the register to the left or right. In the **shift left** operation the serial input transfers a bit to the right most position and in **shift right** operation the serial input transfers a bit to the left most position.

There are three types of shifts as follows:

a) Logical Shift

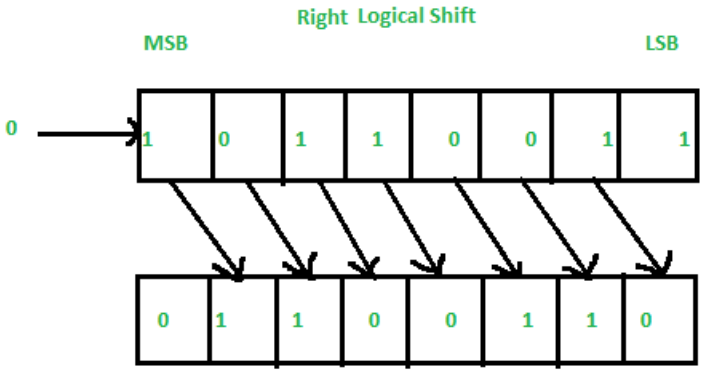
It transfers 0 through the serial input. The symbol **"shl"** is used for logical shift left and **"shr"** is used for logical shift right.

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.



Right Logical Shift -

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




```
R1 ← shl R1

R2 ← shr R2
```

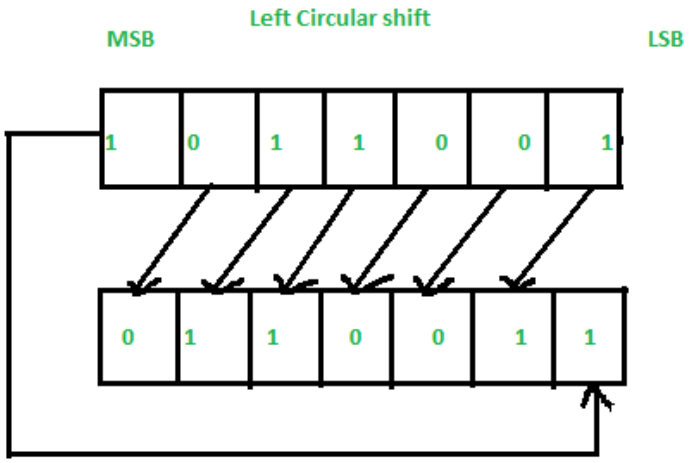
The register symbol must be same on both sides of arrows.

b) Circular Shift

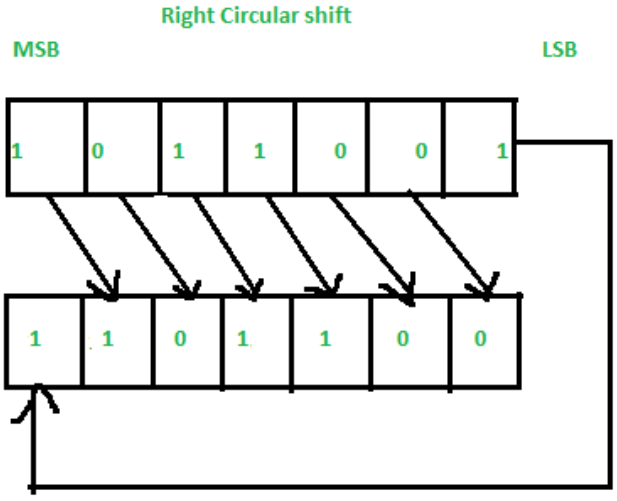
This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively.

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

1. **Left Circular Shift –**



2. **Right Circular Shift –**

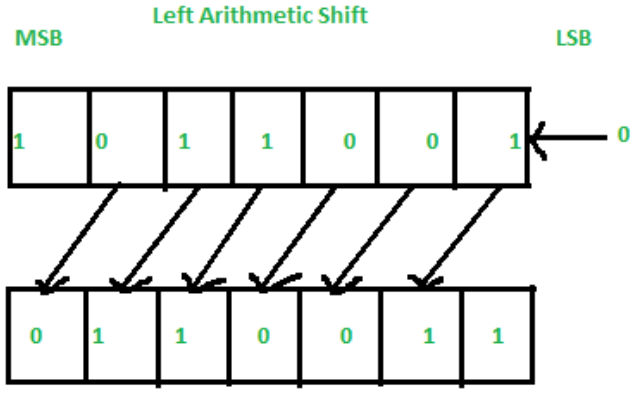


c) Arithmetic Shift

This shifts a signed binary number to left or right. An **arithmetic shift left** multiplies a signed binary number by 2 and **shift left** divides the number by 2. Arithmetic shift micro-operation leaves the sign bit unchanged because the signed number remains same when it is multiplied or divided by 2.

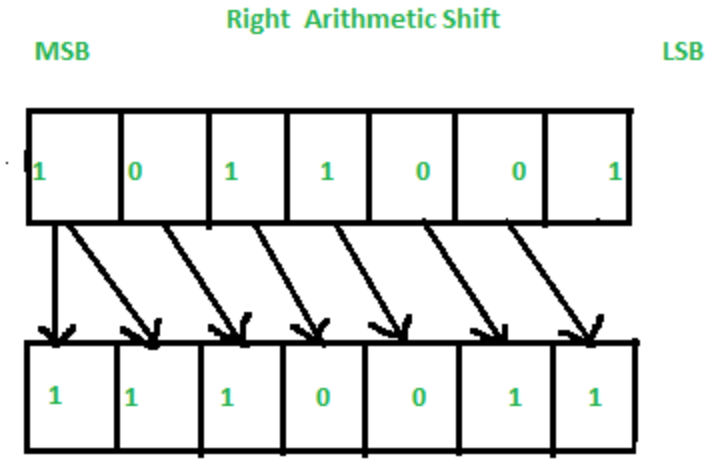
This micro-operation shifts a signed binary number to the left or to the right position. In an arithmetic shift-left, it multiplies a signed binary number by 2 and In an arithmetic shift-right, it divides the number by 2.

1. **Left Arithmetic Shift –**
   In this one position moves each bit 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.



Right Arithmetic Shift –

In this one position moves each bit to the right one by one and the least significant bit is rejected and the empty MSB is filled with the value of the previous MSB.



What Does System Bus Mean?

The system bus is a pathway composed of cables and connectors used to carry data between a computer microprocessor and the main memory. The bus provides a communication path for the data and control signals moving between the major components of the computer system. The system bus works by combining the functions of the three main buses: namely, the data, address and control buses. Each of the three buses has its separate characteristics and responsibilities.

The system bus connects the CPU with the main memory and, in some systems, with the level 2 (L2) cache. Other buses, such as the IO buses, branch off from the system bus to provide a communication channel between the CPU and the other peripherals.

The system bus combines the functions of the three main buses, which are as follows:

- The control bus carries the control, timing and coordination signals to manage the various functions across the system.

- The address bus is used to specify memory locations for the data being transferred.

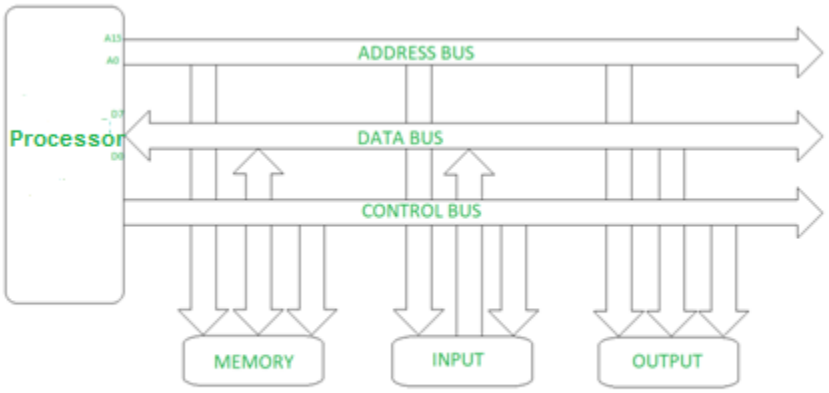
- The data bus, which is a bidirectional path, carries the actual data between the processor, the memory and the peripherals.

The design of the system bus varies from system to system and can be specific to a particular computer design or may be based on an industry standard. One advantage of using the industry standard is the ease of upgrading the computer using standard components such as the memory and IO devices from independent manufacturers.

System bus characteristics are dependent on the needs of the processor, the speed, and the word length of the data and instructions. The size of a bus, also known as its width, determines how much data can be transferred at a time and indicates the number

of available wires. A 32-bit bus, for example, refers to 32 parallel wires or connectors that can simultaneously transmit 32 bits.

The design and dimensions of the system bus are based on the specific processor technology of the motherboard. This, in effect, affects the speed of the motherboard, with faster system buses requiring that the other components on the system be equally fast for the best performance.



Bus organization system of 8085 Microprocessor

TOPIC 3 : ALU

What is an arithmetic-logic unit (ALU)?

An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands in computer instruction words. In some processors, the ALU is divided into two units: an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one AU -- for example, one for fixed-point operations and another for floating-point operations.

How does an arithmetic-logic unit work?

Typically, the ALU has direct input and output access to the processor controller, main memory (random access memory or RAM in a personal computer) and input/output devices. Inputs and outputs flow along an electronic path that is called a bus. The input consists of an instruction word, sometimes called a machine instruction word, that contains an operation

code or "opcode," one or more operands and sometimes a format code. The operation code tells the ALU what operation to perform and the operands are used in the operation.

For example, two operands might be added together or compared logically. The format may be combined with the opcode and tells, for example, whether this is a fixed-point or a floating-point instruction.

The output consists of a result that is placed in a storage register and settings that indicate whether the operation was performed successfully. If it isn't, some sort of status will be stored in a permanent place that is sometimes called the machine status word.

In general, the ALU includes storage places for input operands, operands that are being added, the accumulated result (stored in an accumulator) and shifted results. The flow of bits and the operations performed on them in the subunits of the ALU are controlled by gated circuits.

The gates in these circuits are controlled by a sequence logic unit that uses a particular algorithm or sequence for each operation code. In the arithmetic unit, multiplication and division are done by a series of adding or subtracting and shifting operations.

ALU CHIP :

It is a digital circuit that performs arithmetic calculations, logical and shift operations. Instead of having individual registers performing the micro operations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU. ALU consists of arithmetic, logical and shift circuits. In each stage, the circuit can perform 8 arithmetic operations, 16 logical operations and 2 shift operations.

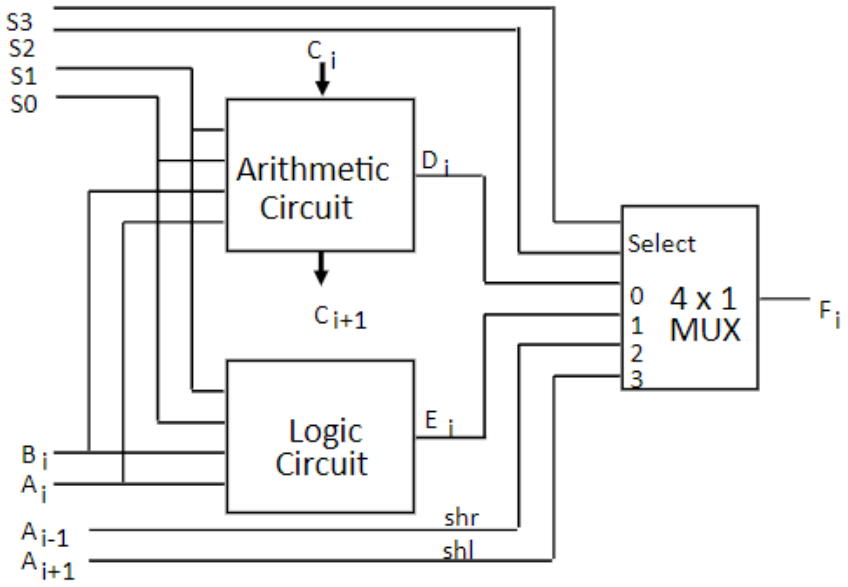


Figure 1 — Arithmetic Logic Unit

Characteristics of the ALU

1. Arithmetic logic unit in digital circuit performs all the logical and arithmetic operations.
2. The operations like addition, subtraction, multiplication and division are referred as Arithmetic operations.
3. The logical operations refer to operations on numbers and special character operations.

The ALU is responsible for the conditions given below:

1. Equal to
2. Less than
3. Greater than

Functions of the ALU

A particular micro operation is selected with inputs S1 and S0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in Ei and a logic output in Hi. The data in the

multiplexer are selected with inputs S3 and S2. The other two data inputs to the multiplexer receive inputs Ai — 1 for the shift-right operation and Ai + 1 for the shift-left operation.

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F

Table 1 — Functions of Arithmetic Logic Unit

The circuit of Figure 1 must be repeated n times for an n-bit ALU. The output carry Ci+1 of a given arithmetic stage must be connected to the input carry Ci of the next stage in sequence. The input carry to the first stage is the input carry Cin, which provides a selection variable for the arithmetic operations.

TOPIC 4 - CODES

In the computer system, digits, numbers, letters or special symbols are represented by a specific 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.

Different types of code:

- BCD code
- Alphanumeric Code

- ASCII
- EBCDIC
- Unicode

Binary Coded Decimal (BCD):

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.

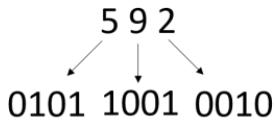
Different Types of BCD code-

- BCD 8421(NBCD- Natural Binary Coded Decimal)
- BCD 7421
- BCD 5421
- BCD 2421
- Excess-3

The following table shows different BCD code of 0-9:

Table 1.6.3					
Decimal	7421	5421	5211	2421	Excess 3
0	0000	0000	0000	0000	0011
1	0001	0001	0010	0001	0100
2	0010	0010	0011	0010	0101
3	0011	0011	0101	0011	0110
4	0100	0100	0111	0100	0111
5	0101	1000	1000	1011	1000
6	0110	1001	1010	1100	1001
7	1000	1010	1100	1101	1010
8	1001	1011	1101	1110	1011
9	1010	1100	1111	1111	1100

Example: Convert $(592)_{10}$ into BCD code.



So, $(592)_{10}$ = $(010110010010)_{BCD}$

Convert $(807)_{10}$ into BCD code.

Alphanumeric codes:

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 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols. The following three alphanumeric codes are very commonly used for the data representation.

- EBCDIC- Extended Binary Coded Decimal Interchange Code.
- ASCII- American Standard Code for Information Interchange.
- Unicode- Universal Code

EBCDIC:

Extended binary coded decimal interchange code (EBCDIC) is an 8-bit binary code for numeric and alphanumeric characters. It was developed and used by IBM. It is a coding representation in which symbols, letters and numbers are presented in binary language.

ASCII:

ASCII is the acronym for the ***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange*. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another. ANSI(American National Standard Institute) invented this code. ASCII two types-

- ASCII-7

- ASCII-8

ASCII-7: It is a code for representing 128 English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase *M* is 77.

ASCII-8: It is a code for representing 256 English characters as numbers

TOPIC 4 NUMBER SYSTEM :

Base of Number System-

- Base of a number system is the total number of digits used in that number system.
- Number system with base 'b' has its digits in the range [0 , b-1].
- It is also called as **radix** of a number system.

Examples-

Consider the following examples-

Base 10 Number System-

Consider base 10 number system popularly called as decimal number system-

- Total number of digits used in this number system are 10 since it has base 10.
- These digits are 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9.
- These digits clearly lie in the range [0 , base-1] = [0 , 9].

Base 2 Number System-

Consider base 2 number system popularly called as binary number system-

- Total number of digits used in this number system are 2 since it has base 2.
- These digits are 0 and 1.
- These digits clearly lie in the range [0 , base-1] = [0 , 1].

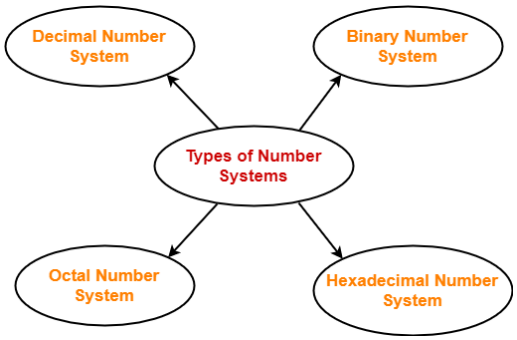
Important Note-

It is important to note that-

- All the digits of any number system with base 'b' are always less than 'b'.
- This is clear from the range [0 , base-1] in which the digits of any number system lie.

Types of Number System-

Four most commonly used number systems are-



1. Decimal Number System
2. Binary Number System

3. Octal Number System
4. Hexadecimal Number System

The following table shows the base and digits used in these number systems-

Number System	Base	Digits Used
Decimal Number System	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (Total 10 digits)
Binary Number System	2	0, 1 (Total 2 digits)
Octal Number System	8	0, 1, 2, 3, 4, 5, 6, 7 (Total 8 digits)
Hexadecimal Number System	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (Total 16 digits)

Number System Conversions-

Before you go through this article, make sure that you have gone through the previous article on Basics of Number System.

In number system,

- It is very important to have a good knowledge of how to convert numbers from one base to another base.
- Here, we will learn how to convert any given number from any base to base 10.

Converting to Base 10-

A given number can be converted from any base to base 10 using **Expansion Method**.

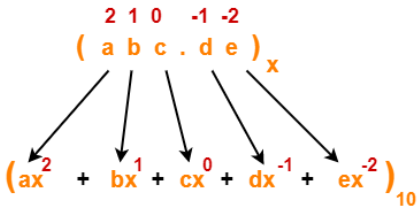
According to expansion method, if abc.de is any given number in base x, then its value in base 10 is given as-

$$(abc.de)_x = (ax^2 + bx + c + dx^{-1} + ex^{-2})_{10}$$

Explanation-

To use expansion method for conversion,

- Assign position number to each digit of the given number.
- Digits to the left of decimal are numbered starting from 0.
- Digits to the right of decimal are numbered starting from -1.
- Write a term for each digit as digit x (base of given number)$^{\text{position number of digit}}$
- Perform the addition of all terms to obtain the number in base 10.
- This formula can be expanded for any number of digits.



Expansion Method

Decimal to Binary Conversion-

A given number can be converted from base 10 to any other base using division method and multiplication method.

Following two cases are possible-

Case-01: For Numbers Carrying No Fractional Part-

- Division Method is used to convert such numbers from base 10 to another base.
- The division is performed with the required base.

Steps To Convert From Base 10 to Base 2-

- Divide the given number (in base 10) with 2 until the result finally left is less than 2.
- Traverse the remainders from bottom to top to get the required number in base 2.

Case-02: For Numbers Carrying A Fractional Part-

To convert such numbers from base 10 to another base, real part and fractional part are treated separately.

For Real Part-

The steps involved in converting the real part from base 10 to another base are same as above.

For Fractional Part-

- Multiplication Method is used to convert fractional part from base 10 to another base.
- The multiplication is performed with the required base.

Steps To Convert From Base 10 To Base 2-

- Multiply the given fraction (in base 10) with 2.
- Write the real part and fractional part of the result so obtained separately.
- Multiply the fractional part with 2.
- Write the real part and fractional part of the result so obtained separately.
- Repeat this procedure until the fractional part remains 0.
- If fractional part does not terminate to 0, find the result up to as many places as required.

Required Number in Base 2

= Series of real part of multiplication results obtained in the above steps from top to bottom

$(18.625)_{10}$

$$(18.625)_{10} \rightarrow (\ ?\ )_2$$

Here, we treat the real part and fractional part separately-

For Real Part-

- The real part is $(18)_{10}$
- We convert the real part from base 10 to base 2 using division method same as above.

So, $(18)_{10} = (10010)_2$

For Fractional Part-

- The fractional part is $(0.625)_{10}$
- We convert the fractional part from base 10 to base 2 using multiplication method.

Using multiplication method, we have-

	Real part	Fractional Part
0.625 x 2	1	0.25
0.25 x 2	0	0.50
0.50 x 2	1	0

Explanation

Step-01:

Multiply 0.625 with 2. Result = 1.25.
Write 1 in real part and 0.25 in fractional part.

Step-02:

Multiply 0.25 with 2. Result = 0.50.
Write 0 in real part and 0.50 in fractional part.

Step-03:

Multiply 0.50 with 2. Result = 1.0.
Write 1 in real part and 0.0 in fractional part.

Since fractional part becomes 0, so we stop.

- The fractional part terminates to 0 after 3 iterations.
- Traverse the real part column from top to bottom to obtain the required number in base 2.

From here, $(0.625)_{10} = (0.101)_2$

Combining the results of real part and fractional part, we have-

$(18.625)_{10} = (10010.101)_2$

Decimal to Octal Conversion-

A given number can be converted from base 10 to any other base using division method and multiplication method.

Following two cases are possible-

Case-01: For Numbers Carrying No Fractional Part-

- Division Method is used to convert such numbers from base 10 to another base.

- The division is performed with the required base.

Steps To Convert From Base 10 to Base 8-

- Divide the given number (in base 10) with 8 until the result finally left is less than 8.
- Traverse the remainders from bottom to top to get the required number in base 8.

Case-02: For Numbers Carrying A Fractional Part-

To convert such numbers from base 10 to another base, real part and fractional part are treated separately.

For Real Part-

The steps involved in converting the real part from base 10 to another base are same as above.

For Fractional Part-

- Multiplication Method is used to convert fractional part from base 10 to another base.
- The multiplication is performed with the required base.

Steps To Convert From Base 10 To Base 8-

- Multiply the given fraction (in base 10) with 8.
- Write the real part and fractional part of the result so obtained separately.
- Multiply the fractional part with 8.
- Write the real part and fractional part of the result so obtained separately.
- Repeat this procedure until the fractional part remains 0.
- If fractional part does not terminate to 0, find the result up to as many places as required.

Required Number in Base 8

= Series of real part of multiplication results obtained in the above steps from top to bottom

$(1032.6875)_{10}$

$$(1032.6875)_{10} \rightarrow (\ ?\ )_{8}$$

Here, we treat the real part and fractional part separately-

For Real Part-

- The real part is $(1032)_{10}$
- We convert the real part from base 10 to base 8 using division method same as above.

So, $(1032)_{10} = (2010)_8$

For Fractional Part-

- The fractional part is $(0.6875)_{10}$
- We convert the fractional part from base 10 to base 8 using multiplication method.

Using multiplication method, we have-

	Real part	Fractional Part
0.6875 x 8	5	0.5
0.5 x 8	4	0.0

Explanation

Step-01:

Multiply 0.6875 with 8. Result = 5.5.
Write 5 in real part and 0.5 in fractional part.

Step-02:

Multiply 0.5 with 8. Result = 4.0.
Write 4 in real part and 0.0 in fractional part.

Since fractional part becomes 0, so we stop.

- The fractional part terminates to 0 after 2 iterations.
- Traverse the real part column from top to bottom to obtain the required number in base 8.

From here, $(0.6875)_{10} = (0.54)_8$

Combining the result of real and fractional parts, we have-
$(1032.6875)_{10} = (2010.54)_8$

Decimal to Hexadecimal Conversion-

A given number can be converted from base 10 to any other base using division method and multiplication method.

Following two cases are possible-

Case-01: For Numbers Carrying No Fractional Part-

- Division Method is used to convert such numbers from base 10 to another base.
- The division is performed with the required base.

Steps To Convert From Base 10 to Base 16-

- Divide the given number (in base 10) with 16 until the result finally left is less than 16.
- Traverse the remainders from bottom to top to get the required number in base 16.

Case-02: For Numbers Carrying A Fractional Part-

To convert such numbers from base 10 to another base, real part and fractional part are treated separately.

For Real Part-

The steps involved in converting the real part from base 10 to another base are same as above.

For Fractional Part-

- Multiplication Method is used to convert fractional part from base 10 to another base.
- The multiplication is performed with the required base.

Steps To Convert From Base 10 To Base 16-

- Multiply the given fraction (in base 10) with 16.
- Write the real part and fractional part of the result so obtained separately.
- Multiply the fractional part with 16.
- Write the real part and fractional part of the result so obtained separately.
- Repeat this procedure until the fractional part remains 0.
- If fractional part does not terminate to 0, find the result up to as many places as required.

Required Number in Base 16

= Series of real part of multiplication results obtained in the above steps from top to bottom

$(2020.65625)_{10}$

$$(2020.65625)_{10} \rightarrow (\ ?\ )_{8}$$

Here, we treat the real part and fractional part separately-

For Real Part-

- The real part is $(2020)_{10}$
- We convert the real part from base 10 to base 16 using division method same as above.

So, $(2020)_{10} = (7E4)_{16}$

For Fractional Part-

- The fractional part is $(0.65625)_{10}$
- We convert the fractional part from base 10 to base 16 using multiplication method.

Using multiplication method, we have-

	Real part	Fractional Part
0.65625 x 16	10 = A	0.5
0.5 x 16	8	0.0

Explanation

Step-01:

Multiply 0.65625 with 16. Result = 10.5.
Write 10 (= A in hexadecimal) in real part and 0.5 in fractional part.

Step-02:

Multiply 0.5 with 16. Result = 8.0.
Write 8 in real part and 0.0 in fractional part.

Since fractional part becomes 0, so we stop.

- The fractional part terminates to 0 after 2 iterations.
- Traverse the real part column from top to bottom to obtain the required number in base 16.

From here, $(0.65625)_{10} = (0.A8)_{8}$

Combining the result of real and fractional parts, we have-

$(2020.65625)_{10} = (7E4.A8)_{16}$

Conversion of Bases-

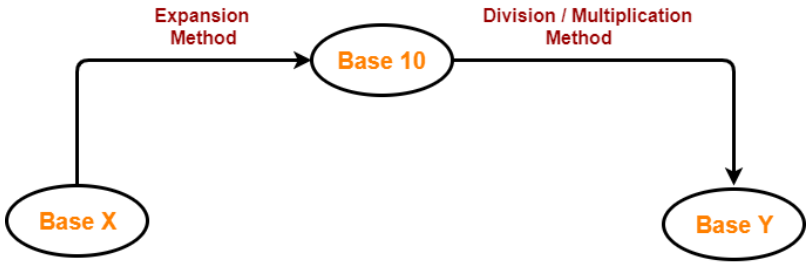
A given number in base x can be converted to any other base y using the following steps-

Step-01:

Convert the number from base x to base 10 using expansion method.

Step-02:

Convert the number from base 10 to base y using division & multiplication method.



Conversion of Bases

TOPIC 5 – LOGIC GATES

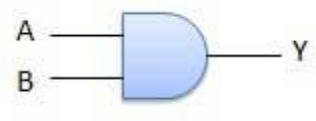
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

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

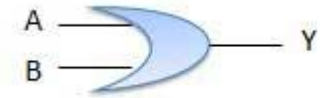
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

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

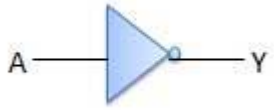
NOT Gate

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

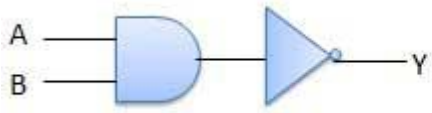
NAND Gate

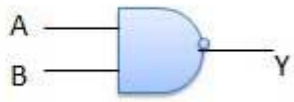
A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

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

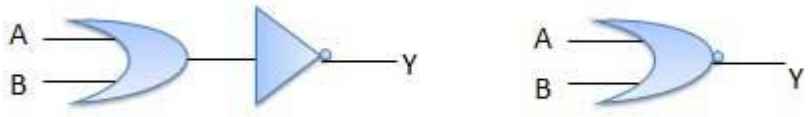
NOR Gate

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

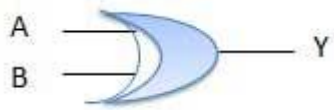
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y = A XOR B XOR C ..... N

Y = $A \oplus B \oplus C$ ..... N

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

Logic diagram



Truth Table

Inputs		Output
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

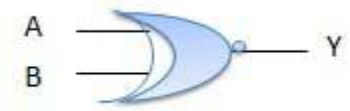
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

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

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

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

Logic diagram



Truth Table

Inputs		Output
A	B	A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1