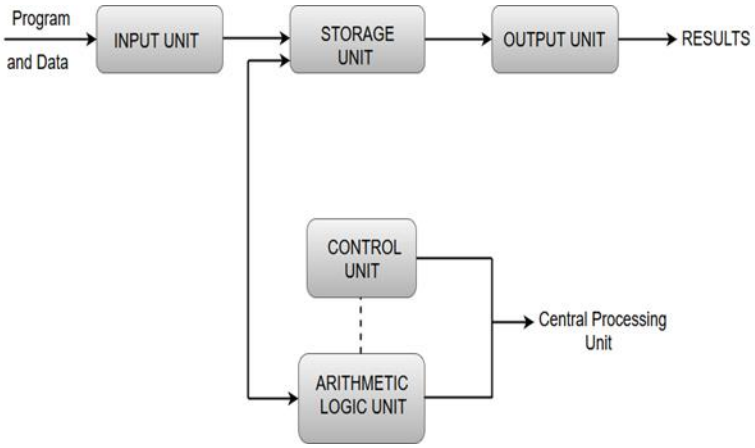
Unit-III

Functional Units of Digital System

- A computer organization describes the functions and design of the various units of a digital system.
- A general-purpose computer system is the best-known example of a digital system. Other examples include telephone switching exchanges, digital voltmeters, digital counters, electronic calculators and digital displays.
- Computer architecture deals with the specification of the instruction set and the hardware units that implement the instructions.
- Computer hardware consists of electronic circuits, displays, magnetic and optic storage media and also the communication facilities.
- Functional units are a part of a CPU that performs the operations and calculations called for by the computer program.
- Functional units of a computer system are parts of the CPU (Central Processing Unit) that performs the operations and calculations called for by the computer program. A computer consists of five main components namely, Input unit, Central Processing Unit, Memory unit Arithmetic & logical unit, Control unit and an Output unit.



Input unit

- Input units are used by the computer to read the data. The most commonly used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
- However, the most well-known input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Central processing unit

- Central processing unit commonly known as CPU can be referred as an electronic circuitry within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Memory unit

- The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.
- Primary storage is the fastest memory that operates at electronic speeds. Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
- It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
- The most common examples of primary memory are RAM and ROM.
- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
- The most common examples of secondary memory are magnetic disks, magnetic tapes, and optical disks.

Arithmetic & logical unit

- Most of all the arithmetic and logical operations of a computer are executed in the ALU (Arithmetic and Logical Unit) of the processor. It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

Control unit

- The control unit is a component of a computer's central processing unit that coordinates the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.
- The control unit is also known as the nerve center of a computer system.
- Let's us consider an example of addition of two operands by the instruction given as Add LOCA, RO. This instruction adds the memory location LOCA to the operand in the register RO and places the sum in the register RO. This instruction internally performs several steps.

Output Unit

- The primary function of the output unit is to send the processed results to the user. Output devices display information in a way that the user can understand.
- Output devices are pieces of equipment that are used to generate information or any other response processed by the computer. These devices display information that has been held or generated within a computer.
- The most common example of an output device is a monitor.

Bus and Memory Transfers

A digital system composed of many registers, and paths must be provided to transfer information from one register to another. The number of wires connecting all of the registers will be excessive if separate lines are used between each register and all other registers in the system.

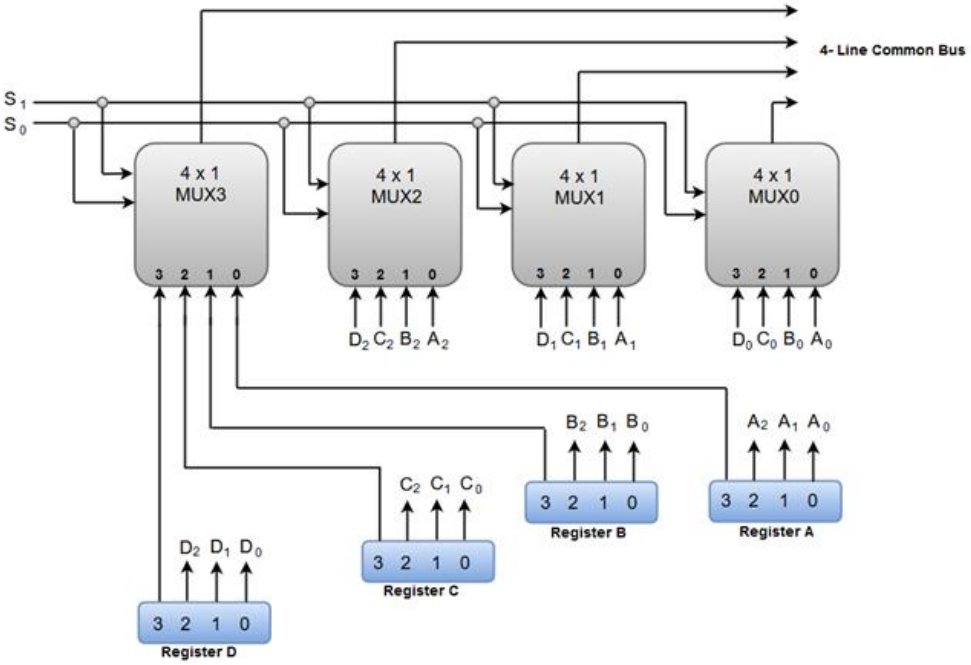
A bus structure, on the other hand, is more efficient for transferring information between registers in a multi-register configuration system.

A bus consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during a particular register transfer.

The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S1 and S2).

We have used labels to make it more convenient for you to understand the input-output configuration of a Bus system for four registers. For instance, output 1 of register A is connected to input 0 of MUX1.

Bus System for 4 Registers:



The two selection lines S1 and S2 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus.

When both of the select lines are at low logic, i.e. S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that forms the bus. This, in turn, causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

Similarly, when S1S0 = 01, register B is selected, and the bus lines will receive the content provided by register B.

The following function table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.

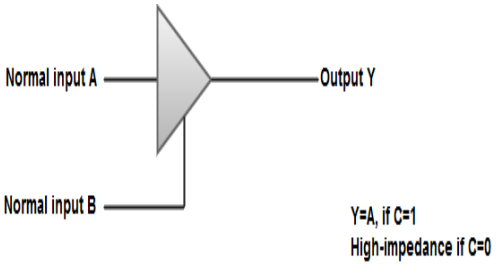
S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

A bus system can also be constructed using **three-state gates** instead of multiplexers.

The **three state gates** can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high-impedance state.

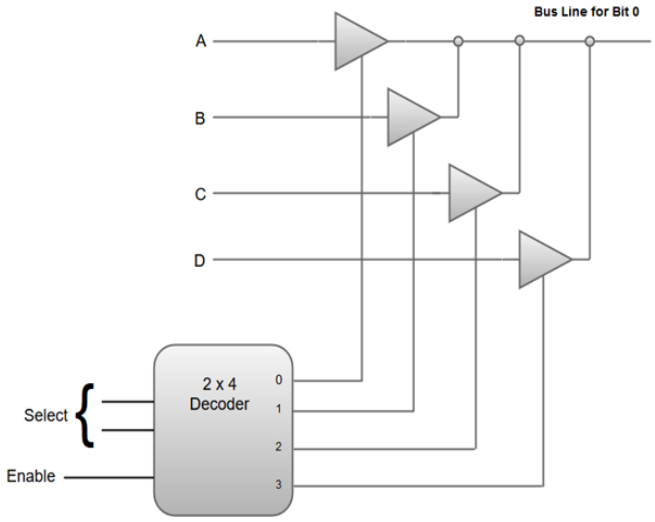
The most commonly used three state gates in case of the bus system is a **buffer gate**.

The graphical symbol of a three-state buffer gate can be represented as:



The following diagram demonstrates the construction of a bus system with three-state buffers.

Bus line with three state buffer:



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A 2 * 4 decoder ensures that no more than one control input is active at any given point of time.

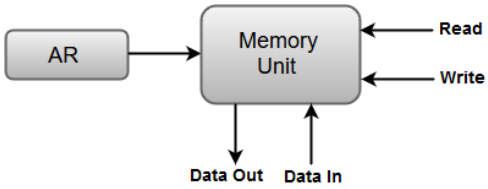
Memory Transfer

Most of the standard notations used for specifying operations on memory transfer are stated below.

- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.
- We must specify the address of memory word while writing the memory transfer operations.
- The **address register** is designated by **AR** and the **data register** by **DR**.
- Thus, a read operation can be stated as:

1. Read: DR ← M [AR]
   - The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
   - And the corresponding write operation can be stated as:

1. Write: M [AR] ← R1
   - The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



Register Transfer

The term Register Transfer refers to the availability of hardware logic circuits that can perform a given micro-operation and transfer the result of the operation to the same or another register.

Most of the standard notations used for specifying operations on various registers are stated below.

- The memory address register is designated by **MAR**.
- Program Counter **PC** holds the next instruction's address.
- Instruction Register **IR** holds the instruction being executed.

- **R1** (Processor Register).
- We can also indicate individual bits by placing them in parenthesis. For instance, PC (8-15), R2 (5), etc.
- Data Transfer from one register to another register is represented in symbolic form by means of replacement operator. For instance, the following statement denotes a transfer of the data of register R1 into register R2.

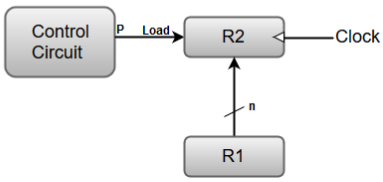
1. R2 ← R1

- Typically, most of the users want the transfer to occur only in a predetermined control condition. This can be shown by following if-then statement: If (P=1) then (R2 ← R1); Here P is a control signal generated in the control section.
- It is more convenient to specify a control function (P) by separating the control variables from the register transfer operation. For instance, the following statement defines the data transfer operation under a specific control function (P).

1. P: R2 ← R1

The following image shows the block diagram that depicts the transfer of data from R1 to R2.

Transfer from R1 to R2 when P = 1:

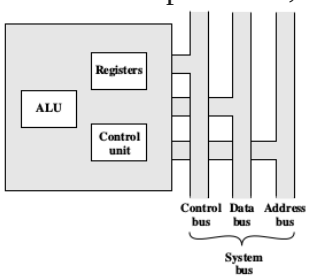


Here, the letter 'n' indicates the number of bits for the register. The 'n' outputs of the register R1 are connected to the 'n' inputs of register R2.

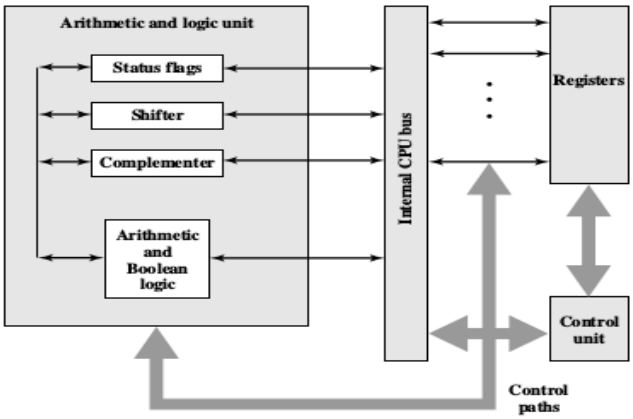
A load input is activated by the control variable 'P' which is transferred to the register R2.

Processor Organization

Figure below is a simplified view of a processor, indicating its connection to the rest of the system via the system bus.



The ALU does the actual computation or processing of data. The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers. Figure below depicts is a slightly more detailed view of the processor.



The data transfer and logic control paths are indicated, including an element labeled internal processor bus. This element is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory. The figure also shows typical basic elements of the ALU. Note the similarity between the internal structure of the computer as a whole and the internal structure of the processor. In both cases, there is a small collection of major elements (computer: processor, I/O, memory; processor:

Register organization of 8086

General 16-bit registers

The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation.

BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register: It is used as default counter or count register in case of string and loop instructions.

DX Register: Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:
To complete 1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.5. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.
Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.

Pointers and index registers
The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Conditional Flags
Conditional flags are as follows:

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AC flag is set i.e. carry given by D3 bit to D4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity flag is reset.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags

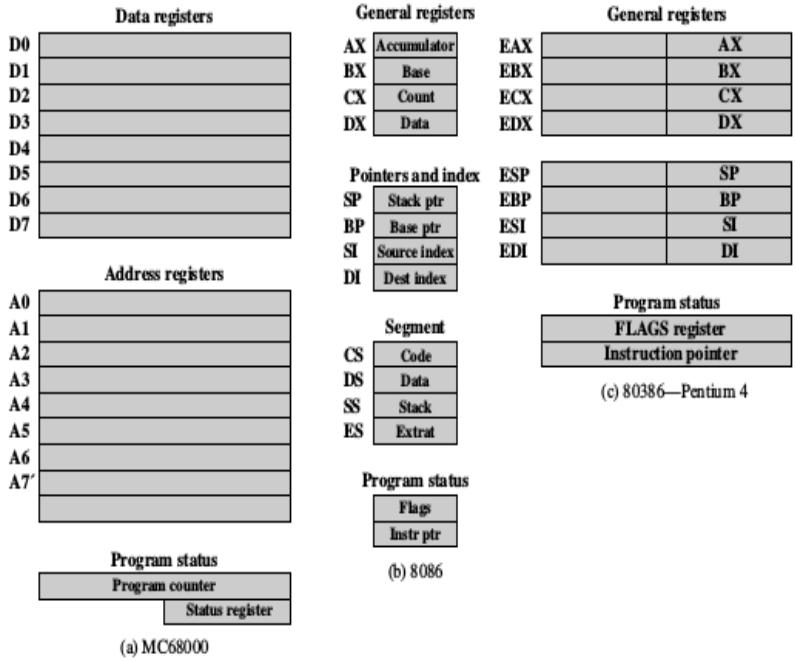
Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

Sample microprocessor register organizations are illustrated bellow.



Stack Organization

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called ***Stack Pointer***.

Stack Operations

The two operations of a stack are:

1. **Push:** Inserts an item on top of stack.
2. **Pop:** Deletes an item from top of stack.

Implementation of Stack

In digital computers, stack can be implemented in two ways:

1. Register Stack
2. Memory Stack

Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

Memory Stack

A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as *stack pointer*.

Addressing Modes

Addressing Modes– The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

Addressing modes for 8086 instructions are divided into two categories:

1) Addressing modes for data

2) Addressing modes for branch

The 8086 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types. The key to good assembly language programming is the proper use of memory addressing modes.

IMPORTANT TERMS

- **Starting address** of memory segment.
- **Effective address or Offset**: An offset is determined by adding any combination of three address elements: **displacement, base and index.**
    - **Displacement:** It is an 8 bit or 16 bit immediate value given in the instruction.
    - **Base**: Contents of base register, BX or BP.
    - **Index**: Content of index register SI or DI.

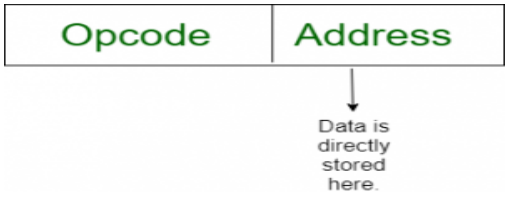
According to different ways of specifying an operand by 8086 microprocessor, different addressing modes are used by 8086.

Addressing modes used by 8086 microprocessor are discussed below:

- **Implied mode:**: In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction.Zero address instruction are designed with implied addressing mode.

Example: CLC (used to reset Carry flag to 0)

- **Immediate addressing mode (symbol #):**In this mode data is present in address field of instruction .Designed like one address instruction format. **Note:**Limitation in the immediate mode is that the range of constants are restricted by size of address field.



Example: MOV AL, 35H (move the data 35H into AL register)

- **Register mode:** In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction. *Here one register reference is required to access the data.*

- Example: MOV AX,CX (move the contents of CX register to AX register)
- **Register Indirect mode**: In this addressing the operand's offset is placed in any one of the registers BX,BP,SI,DI as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction. *Here two register reference is required to access the data.*

The 8086 CPUs let you access memory indirectly through a register using the register indirect addressing modes.

- MOV AX, [BX](move the contents of memory location s

  addressed by the register BX to the register AX)

- **Auto Indexed (increment mode)**: Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.**(R1)+**.

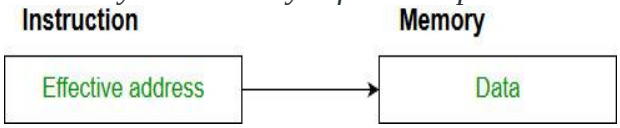
Here one register reference,one memory reference and one ALU operation is required to access the data.
Example:

- Add R1, (R2)+ // OR
- R1 = R1 +M[R2]

  R2 = R2 + d

  Useful for stepping through arrays in a loop. R2 – start of array d – size of an element
- **Auto indexed (decrement mode)**: Effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. –**(R1)**
  Here one register reference,one memory reference and one ALU operation is required to access the data.
  Example:
  Add R1,-(R2) //OR

  R2 = R2-*d*
  R1 = R1 + M[R2]
  Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop . Auto increment and Auto decrement modes are useful for implementing "Last-In-First-Out" data structures.
- **Direct addressing/ Absolute addressing Mode (symbol [ ]):** The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element. In this addressing mode the 16 bit effective address of the data is the part of the instruction. *Here only one memory reference operation is required to access the data.*



Example:ADD AL,[0301] //add the contents of offset address 0301 to AL

- **Indirect addressing Mode (symbol @ or ())**:In this mode address field of instruction contains the address of effective address.Here two references are required.
  1st reference to get effective address.
  2nd reference to access the data.
  Based on the availability of Effective address, Indirect mode is of two kind:

  1. Register Indirect:In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction. *Here one register reference,one memory reference is required to access the data.*
  2. Memory Indirect:In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.
     Here two memory reference is required to access the data.

  Indexed addressing mode: The operand's offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement.
  Example:MOV AX, [SI +05]

  Based Indexed Addressing: The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.
  Example: ADD AX, [BX+SI]

Based on Transfer of control, addressing modes are:

PC relative addressing mode: PC relative addressing mode is used to implement intra segment transfer of control, In this mode effective address is obtained by adding displacement to PC.

EA= PC + Address field value

PC= PC + Relative value.

Base register addressing mode: Base register addressing mode is used to implement inter segment transfer of control.In this mode effective address is obtained by adding base register value to address field value.

EA= Base register + Address field value.

PC= Base register + Relative value.

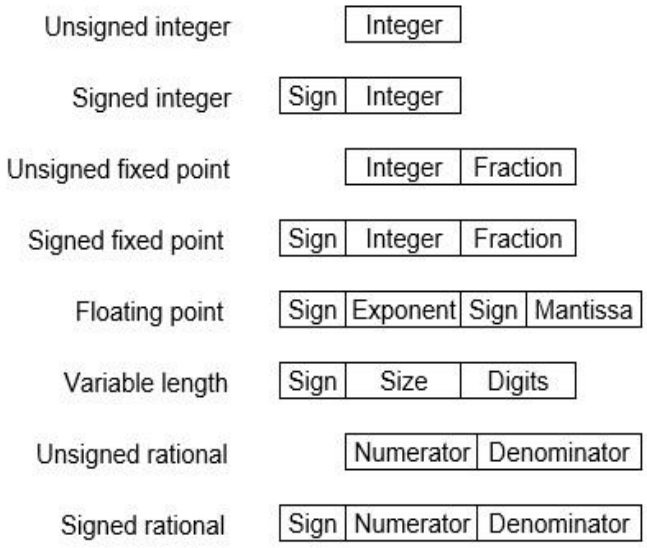
Fixed and floating point number representation

Digital Computers use Binary number system to represent all types of information inside the computers. Alphanumeric characters are represented using binary bits (i.e., 0 and 1). Digital representations are easier to design, storage is easy, and accuracy and precision are greater.

There are various types of number representation techniques for digital number representation, for example: Binary number system, octal number system, decimal number system, and hexadecimal number system etc. But Binary number system is most relevant and popular for representing numbers in digital computer system.

Storing Real Number

These are structures as following below −



There are two major approaches to store real numbers (i.e., numbers with fractional component) in modern computing. These are (i) Fixed Point Notation and (ii) Floating Point Notation. In fixed point notation, there are a fixed number of digits after the decimal point, whereas floating point number allows for a varying number of digits after the decimal point.

Fixed-Point Representation −

This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field.

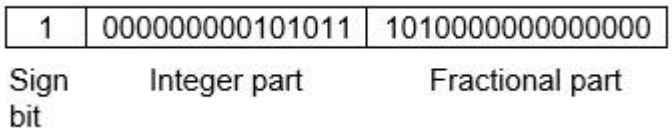
We can represent these numbers using:

- Signed representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 1's complement representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 2's complementation representation: range from $-(2^{(k-1)})$ to $(2^{(k-1)}-1)$, for k bits.

2's complementation representation is preferred in computer system because of unambiguous property and easier for arithmetic operations.

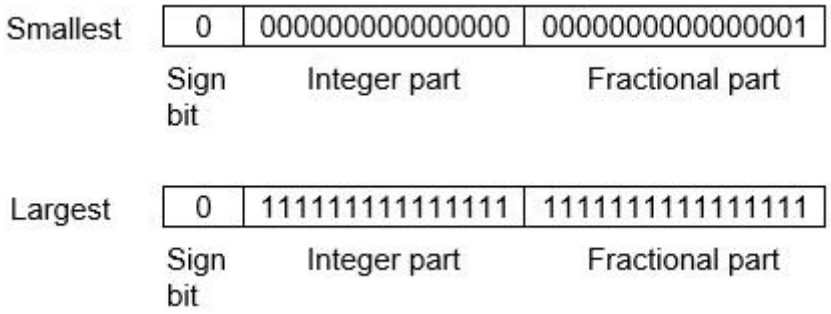
Example −Assume number is using 32-bit format which reserve 1 bit for the sign, 15 bits for the integer part and 16 bits for the fractional part.

Then, -43.625 is represented as following:



Where, 0 is used to represent + and 1 is used to represent -. 000000000101011 is 15 bit binary value for decimal 43 and 1010000000000000 is 16 bit binary value for fractional 0.625.

The advantage of using a fixed-point representation is performance and disadvantage is relatively limited range of values that they can represent. So, it is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy. A number whose representation exceeds 32 bits would have to be stored inexactly.



These are above smallest positive number and largest positive number which can be store in 32-bit representation as given above format. Therefore, the smallest positive number is $2^{-16} \approx 0.000015$ approximate and the largest positive number is $(2^{15}-1)+(1-2^{-16})=2^{15}(1-2^{-16})=32768$, and gap between these numbers is 2^{-16}.

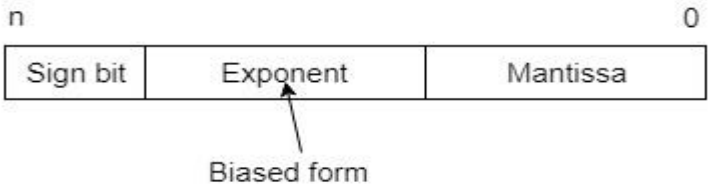
We can move the radix point either left or right with the help of only integer field is 1.

Floating-Point Representation −

This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two part: the first part represents a signed fixed point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be fraction or an integer. Floating -point is always interpreted to represent a number in the following form: Mxr^e.

Only the mantissa m and the exponent e are physically represented in the register (including their sign). A floating-point binary number is represented in a similar manner except that is uses base 2 for the exponent. A floating-point number is said to be normalized if the most significant digit of the mantissa is 1.



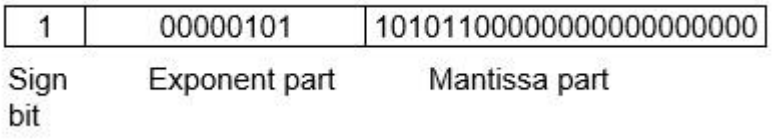
So, actual number is $(-1)^s(1+m)x2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number.

Note that signed integers and exponent are represented by either sign representation, or one's complement representation, or two's complement representation.

The floating point representation is more flexible. Any non-zero number can be represented in the normalized form of $\pm (1.b_1b_2b_3 ...)_{2x2n}$ this is normalized form of a number x.

Example −Suppose number is using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. The leading bit 1 is not stored (as it is always 1 for a normalized number) and is referred to as a *"hidden bit*".

Then −53.5 is normalized as $-53.5=(-110101.1)_2=(-1.101011)x2^5$, which is represented as following below,

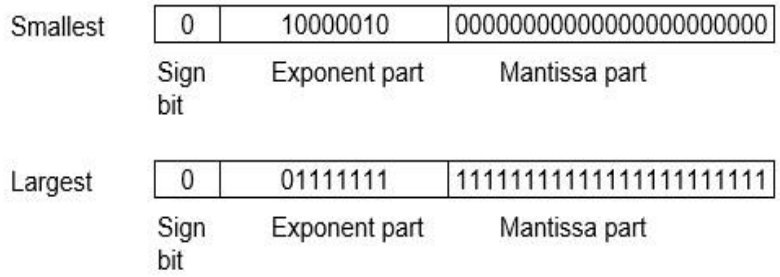


Where 00000101 is the 8-bit binary value of exponent value +5.

Note that 8-bit exponent field is used to store integer exponents $-126 \leq n \leq 127$.

The smallest normalized positive number that fits into 32 bits is $(1.00000000000000000000000)_2x2^{-126}=2^{-126}\approx1.18x10^{-38}$, and largest normalized positive

number that fits into 32 bits is $(1.11111111111111111111111)_2 x 2^{127} = (2^{24}-1)x2^{104} \approx 3.40x10^{38}$. These numbers are represented as following below,



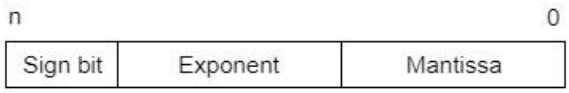
The precision of a floating-point format is the number of positions reserved for binary digits plus one (for the hidden bit). In the examples considered here the precision is 23+1=24.

The gap between 1 and the next normalized floating-point number is known as machine epsilon. the gap is $(1+2^{-23})-1=2^{-23}$for above example, but this is same as the smallest positive floating-point number because of non-uniform spacing unlike in the fixed-point scenario.

Note that non-terminating binary numbers can be represented in floating point representation, e.g., 1/3 = $(0.010101\ ...)_2$ cannot be a floating-point number as its binary representation is non-terminating.

IEEE Floating point Number Representation −

IEEE (Institute of Electrical and Electronics Engineers) has standardized Floating-Point Representation as following diagram.



So, actual number is $(-1)^s(1+m)x2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number. The sign bit is 0 for positive number and 1 for negative number. Exponents are represented by or two's complement representation.

According to IEEE 754 standard, the floating-point number is represented in following ways:

- Half Precision (16 bit): 1 sign bit, 5 bit exponent, and 10 bit mantissa
- Single Precision (32 bit): 1 sign bit, 8 bit exponent, and 23 bit mantissa
- Double Precision (64 bit): 1 sign bit, 11 bit exponent, and 52 bit mantissa
- Quadruple Precision (128 bit): 1 sign bit, 15 bit exponent, and 112 bit mantissa

Special Value Representation −

There are some special values depended upon different values of the exponent and mantissa in the IEEE 754 standard.

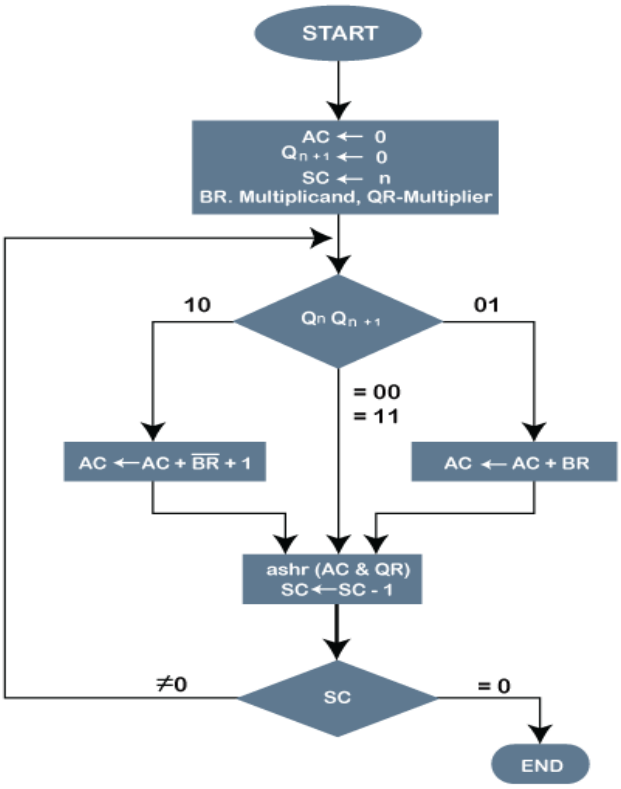
- All the exponent bits 0 with all mantissa bits 0 represents 0. If sign bit is 0, then +0, else -0.
- All the exponent bits 1 with all mantissa bits 0 represents infinity. If sign bit is 0, then +∞, else -∞.

- All the exponent bits 0 and mantissa bits non-zero represents denormalized number.
- All the exponent bits 1 and mantissa bits non-zero represents error.

Booth's Multiplication Algorithm

The booth algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement, respectively. It is also used to speed up the performance of the multiplication process. It is very efficient too. It works on the string bits 0's in the multiplier that requires no additional bit only shift the right-most string bits and a string of 1's in a multiplier bit weight 2^k to weight 2^m that can be considered as $\mathbf{2^{k+1} - 2^m}$.

Following is the pictorial representation of the Booth's Algorithm:



In the above flowchart, initially, **AC** and $\mathbf{Q_{n+1}}$ bits are set to 0, and the **SC** is a sequence counter that represents the total bits set **n,** which is equal to the number of bits in the multiplier. There are **BR** that represent the **multiplicand bits,** and QR represents the **multiplier bits**. After that, we encountered two bits of the multiplier as Q_n and Q_{n+1}, where Qn represents the last bit of QR, and Q_{n+1} represents the incremented bit of Qn by 1. Suppose two bits of the multiplier is equal to 10; it means that we have to subtract the multiplier from the partial product in the accumulator AC and then perform the arithmetic shift operation (ashr). If the two of the multipliers equal to 01, it means we need to perform the addition of the multiplicand to the partial product in accumulator AC and then perform the arithmetic shift operation (**ashr**), including $\mathbf{Q_{n+1}}$. The arithmetic shift operation is used in Booth's algorithm to shift AC and QR bits to the right by one and remains the sign bit in AC unchanged. And the sequence counter is continuously decremented till the computational loop is repeated, equal to the number of bits (n).

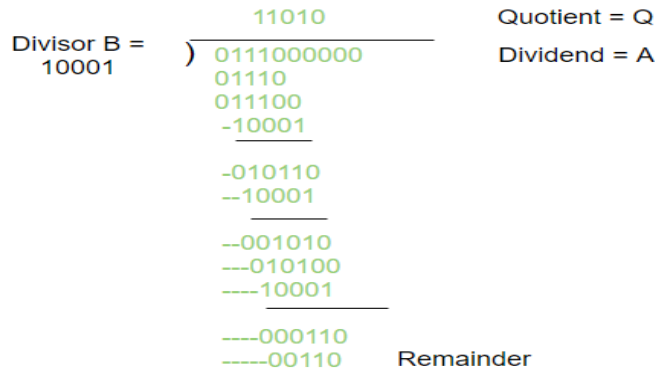
Working on the Booth Algorithm

1. Set the Multiplicand and Multiplier binary bits as M and Q, respectively.
2. Initially, we set the AC and Q_{n+1} registers value to 0.
3. SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
4. A Q_n represents the last bit of the Q, and the Q_{n+1} shows the incremented bit of Q_n by 1.
5. On each cycle of the booth algorithm, Q_n and Q_{n+1} bits will be checked on the following parameters as follows:
    i. When two bits Q_n and Q_{n+1} are 00 or 11, we simply perform the arithmetic shift right operation (ashr) to the partial product AC. And the bits of Qn and Q_{n+1} is incremented by 1 bit.
    ii. If the bits of Q_n and Q_{n+1} is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
    iii. If the bits of Q_n and Q_{n+1} is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
6. The operation continuously works till we reached n - 1 bit in the booth algorithm.
7. Results of the Multiplication binary bits will be stored in the AC and QR registers.

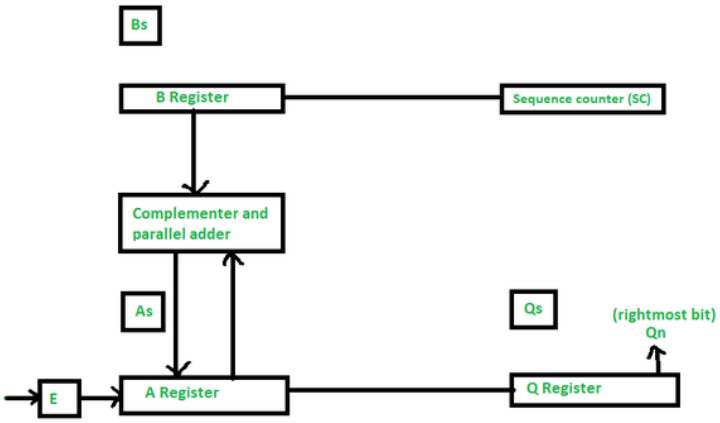
Division Algorithm in Signed Magnitude

The Division of two fixed-point binary numbers in the signed-magnitude representation is done by the cycle of successive compare, shift, and subtract operations.

The binary division is easier than the decimal division because the quotient digit is either 0 or 1. Also, there is no need to estimate how many times the dividend or partial remainders adjust to the divisor.

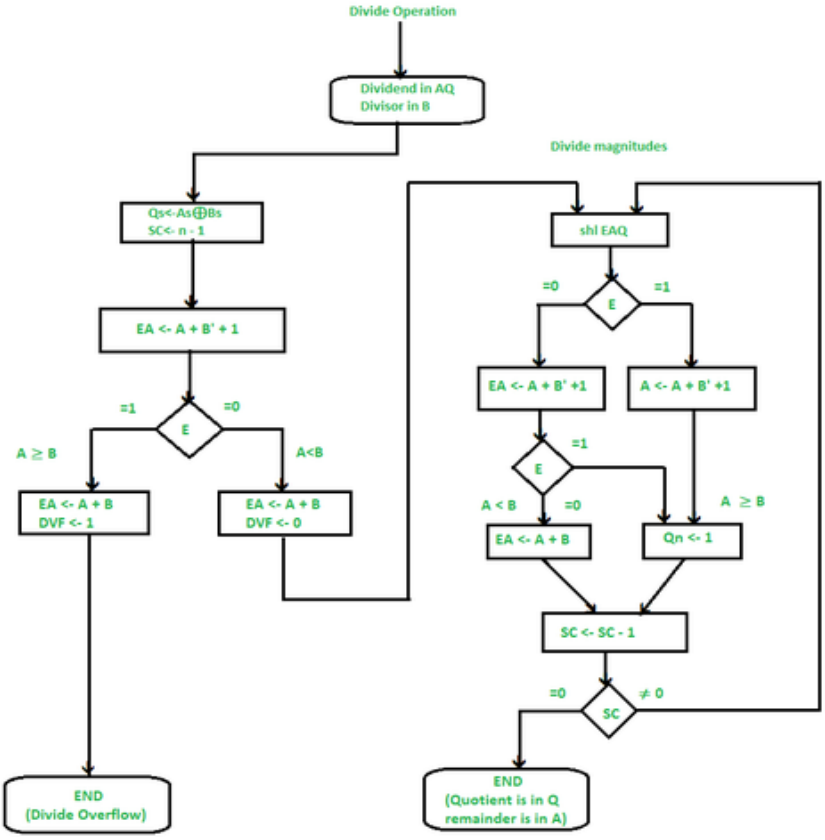


Hardware Implementation :



The hardware implementation in the division operation is identical to that required for multiplication and consists of the following components –

- Here, Registers B is used to store divisor and the double-length dividend is stored in registers A and Q
- The information for the relative magnitude is given in E.
- Sequence Counter register (SC) is used to store the number of bits in the dividend.



- Initially, the dividend is in A & Q and the divisor is in B.
- Sign of result is transferred into Q, to be the part of quotient. Then a constant is set into the SC to specify the number of bits in the quotient.
- Since an operand must be saved with its sign, one bit of the word will be inhabited by the sign, and the magnitude will be composed of n -1 bits.
- The condition of divide-overflow is checked by subtracting the divisor in B from the half of bits of the dividend stored in A. If $A \geq B$, DVF is set and the operation is terminated before time. If $A < B$, no overflow condition occurs and so the value of the dividend is reinstated by adding B to A.
- The division of the magnitudes starts by shl dividend in AQ to left in the high-order bit shifted into E.
  (Note – If shifted a bit into E is equal to 1, and we know that EA > B as EA comprises a 1 followed by n -1 bits whereas B comprises only n -1 bits). In this case, B must be subtracted from EA, and 1 should insert into Q, for the quotient bit.
- If the shift-left operation (shl) inserts a 0 into E, the divisor is subtracted by adding its 2's complement value and the carry is moved into E. If E = 1, it means that $A \geq B$; thus, Q, is set to 1. If E = 0, it means that $A < B$ and the original number is reimposed by adding B into A.

Divisor B = 10001	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B}$ + 1		01111		
E = 1	1	01011		
Set Q_n = 1	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B}$ + 1		01111		
E = 1	1	00101		
Set Q_n = 1	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B}$ + 1		01111		
E = 0; leave Q_n = 0	0	11001	00110	
Add B		10001		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\overline{B}$ + 1		01111		
E = 1	1	00011		
Set Q_n = 1	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B}$ + 1		01111		
E = 0; leave Q_n = 0	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Arithmetic Logic Unit (ALU) is the heart of any CPU. An ALU performs three kinds of operations, i.e.

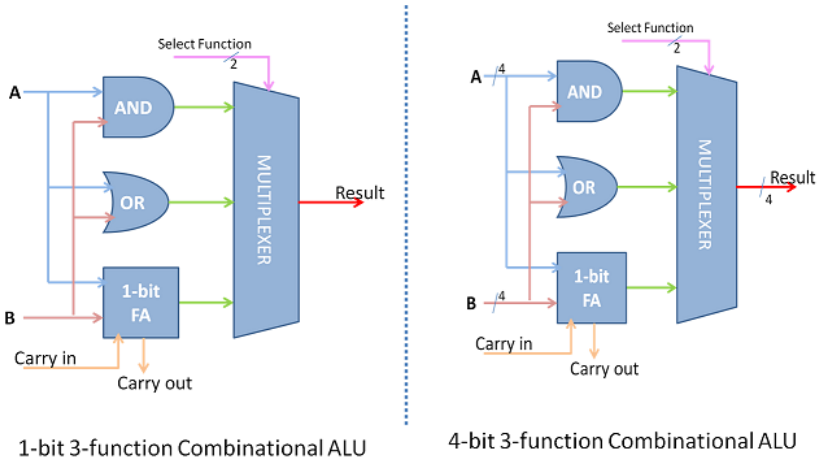
- Arithmetic operations such as Addition/Subtraction,
- Logical operations such as AND, OR, etc. and
- Data movement operations such as Load and Store

ALU derives its name because it performs arithmetic and logical operations. A simple ALU design is constructed with Combinational circuits. ALUs that perform multiplication and division are designed around the circuits developed for these operations while implementing the desired algorithm. More complex ALUs are designed for executing Floating point, Decimal operations and other complex numerical operations. These are called Coprocessors and work in tandem with the main processor.

The design specifications of ALU are derived from the Instruction Set Architecture. The ALU must have the capability to execute the instructions of ISA. An instruction execution in

a CPU is achieved by the movement of data/datum associated with the instruction. This movement of data is facilitated by the Data path. For example, a LOAD instruction brings data from memory location and writes onto a GPR. The navigation of data over datapath enables the execution of LOAD instruction. We discuss Datapath more in details in the next chapter on Control Unit Design. The trade-off in ALU design is necessitated by the factors like Speed of execution, hardware cost, the width of the ALU.

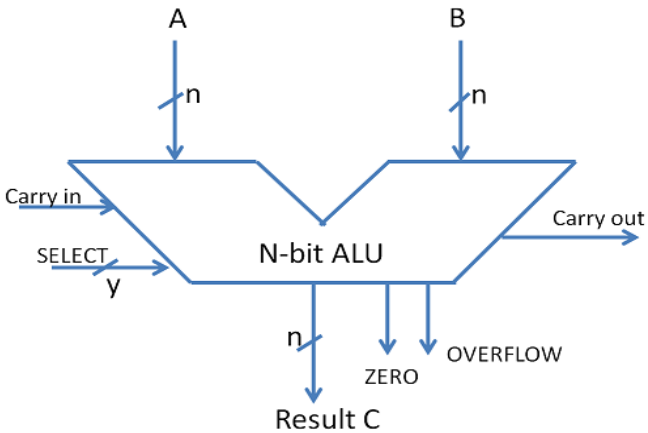
Combinational ALU :A primitive ALU supporting three functions AND, OR and ADD is explained in figure 11.1. The ALU has two inputs A and B. These inputs are fed to AND gate, OR Gate and Full ADDER. The Full Adder also has CARRY IN as an input. The combinational logic output of A and B is statically available at the output of AND, OR and Full Adder. The desired output is chosen by the Select function, which in turn is decoded from the instruction under execution. Multiplexer passes one of the inputs as output based on this select function. Select Function essentially reflects the operation to be carried out on the operands A and B. Thus A and B, A or B and A+B functions are supported by this ALU. When ALU is to be extended for more bits the logic is duplicated for as many bits and necessary cascading is done. The AND and OR logic are part of the logical unit while the adder is part of the arithmetic unit.



1-bit 3-function Combinational ALU

4-bit 3-function Combinational ALU

The simplest ALU has more functions that are essential to support the ISA of the CPU. Therefore the ALU combines the functions of 2's complement, Adder, Subtractor, as part of the arithmetic unit. The logical unit would generate logical functions of the form f(x,y) like AND, OR, NOT, XOR etc. Such a combination supplements most of a CPU's fixed point data processing instructions.



So far what we have seen is a primitive ALU. ALU can be as complex as the variety of functions that are carried out by the ALU. The powerful modern CPUs have powerful and versatile ALUs. Modern CPUs have multiple ALU to improve efficiency.