Module - 1

Basic Structure of Computers

* Introduction

* Digital Computer or Simply Computer

Computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions, and produces the resulting output information.

* The list of instructions is called a Computer program.

* The internal storage is called Computer memory.

* Computer organization

* It describes the function and design of the various units of digital computers that store and process information.

* It also deals with the units of the Computer that receive information from external sources and send Computed results to external destinations.

* Computer hardware

* Computer hardware consists of electronic circuits, displays, magnetic and optical storage media, electromechanical equip-ment and Communication facilities.

* Computer Architecture

* Computer architecture encompasses the specification of an instruction set and

the hardware units that implement the instructions.

(1.1)* Computer Types

* Many types of computers exist that differ widely in size, cost, computational power and intended use.
* The most common computer are

1) personal Computer
which has found wide use in homes, schools and business offices

2) Desktop Computer
It have processing and storage units, visual display and audio output units and a keyboard that can all be located easily on a home or office desk.
The storage media include hard disks, CD-ROMS and diskettes.

3) portable notebook Computers
These are a Compact version of the personal Computer with all of these components packaged into a single unit the size of a thin briefcase.

4) Workstations
* with high resolution graphics I/o capability
* with more Computation power than pc
* It is used to solve Complex problems which arises in engineering applications (especially for interactive design work-graphics, CAD/CAM etc)
* It is costlier

5) Enterprise system (mainframes)
* More Computational power
* large storage capacity
* used for business data processing in large organisation
* Commonly reffered as servers or super Computers

i) Server System
* It supports large volume of data which frequently need to be accessed (or) to be modified.
* In many cases, servers are widely accessible to the education, business and personal user communities.
* It supports request - response operation

6) Super Computers
* Faster than mainframes
* helps in calculating large scale numerical and algorithm calculation in short span of time
* It is used for aircraft design and testing military application, weather forecasting.

7) Handheld Computer

* It also called PDA (personal digital Assistant)
* A Computer that fits into a pocket, runs on batteries
* Typically used as an appointment book, Calculator and notepad.

(1.2)* Functional units

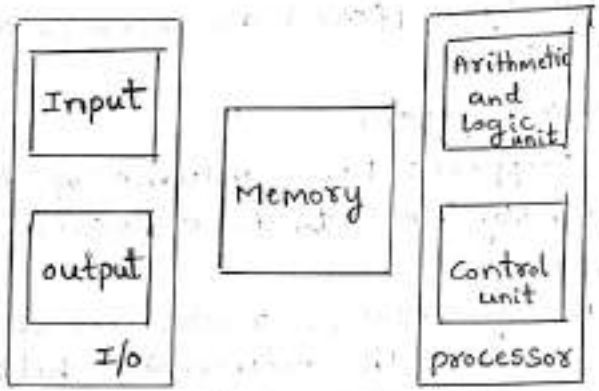


Figure 1.1 – Basic functional units of a Computer.

* Computer consists of 5 functionally independent main parts
  1) Input
  2) output
  3) Memory
  4) ALU (Arithmetic and logical unit)
  5) CU (Control unit)

1) Input unit

* It accepts coded information from human operators, from electromechanical devices such as keyboard (or) from other Computer over digital communication lines
* It converts the external world data to a binary format, which can be under-stood by cpu.

  Ex - keyboard, mouse, joysticks etc

* whenever a key is pressed, the corresponding letter (or) digit is autom-atically translated into its correspond code and transmitted over a cable to either memory (or) processor.

2) output unit

* Converts the binary format data to a format that a common man can understand.

  Ex - Monitor, printer, LCD, LED etc

* It function is to send processed results to the outside world.

3) memory

* It function is to store program and data.
* There are two classes of memory
  i) primary memory / storage
  ii) Secondary memory / storage

	primary memory (storage)	Secondary memory (Storage)
1	It is also known as main memory (or) internal memory	It is also known as External memory (or) Auxilliary memory
2	It has limited storage capacity	It can store bulk amounts of data in a single unit
3	It is a temporary memory	It is a permanent memory
4	It is volatile in nature	It is non-volatile in nature
5	The memory has fast access time	The memory has low access time
6	It is expensive	It is inexpensive
7	Example – RAM, ROM, Cache memory, EPROM, Registers etc	Example – Hard DISK, Floppy DISK, Magnetic Tapes. etc

4) Arithmetic and Logical unit (ALU)

* most Computer operations are executed in ALU.
* It present inside the processor
* It is a Combination circuit
* It doesn't store any value.

5) Control unit (CU)

* operations of all the units are Controlled by Control unit
* It is a nerve Centre that sends control signals to other units.
* Transfers are govern by timing signals (determine when a given action takes place) that are generated by CU (Control unit).
* Data transfer between the processor and memory are also controlled by the Control unit through timing signals.

* operations of a Computer (Summarized)

* The Computer accepts information in the form of programs and data through an input unit and stores it in the memory.
* Information stored in the memory is fetched, under program control, into arithmetic and logic unit, where it is processed.
* processed information leaves the Computer through an output unit.
* All activities inside the machine are directed by the control unit.

(1.3)* Basic operational Concepts

* The operation of a Computer can be Summarised as.
* The Computer accepts information in the form of programs and data through an input unit and stores it in memory.
* Information stored in the memory is fetched, under program control, into an

ALU where it is processed.

* processed information leaves the Comp -uter through an output unit

* All activities inside the machine are directed by control unit

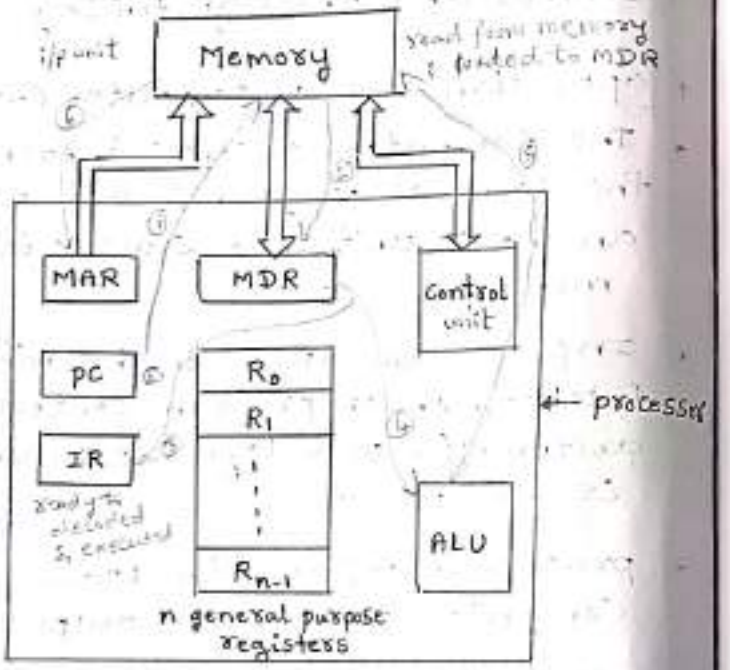


Figure 1.2 – Connections between the processor and the memory

* processor description

* In addition to the ALU and CU, processor contains a number of registers used for different purposes

* IR (Instruction Register)

* It holds the instructions that is currently being executed.

* pc (program counter)

* keep track of the instruction from execution of a program

* Contains the memory address of the next instruction to be fetch and executed

* It contains n general purpose registers which is used to store data and inter -mediate result during instruction execution

* Two register facilitate communication with memory.

  * MAR (memory Address Register)
  * MDR (Memory Data Register)

MAR holds the address of the location to be accessed (entered)

MDR holds the data to be written into (or) read out from the memory.

* operating steps

* programs reside a memory through input unit.

* Execution of the program starts when the pc is set to point to the first instruction of the program.

* First instruction of the program is read out of the memory and loaded into MDR.

* The content of MDR are transffered to IR at this point this instruction is ready to be decoded and executed.

* If the instruction involves an operation

to be performed by ALU, ALU required operands. if operand resides in memory (or) general purpose registers, it has to be fetched by sending its address to the MAR and indicating read cycle.

when the operand has been read from the memory into the MDR. it is transferred from MDR to ALU.

* If the result of this operation is to be stored in the memory, then the result is sent to MDR. the address of the location where the result is to be stored is sent to MAR and write cycle is initiated.

Ex - Add Loc A, R_0

→ This instruction adds the operand at memory location Loc A to the operand in the Register in the processor R_0 and places the sum into Register R_0

→ The original Contents of Loc A are preserved where as those of R_0 also overwritten.

Steps

1) instruction fetched from the memory into the processor
2) The operand at loc A is fetched and added to the contents of R_0
3) The resulting Sum is stored in Register R_0

(1.5)

Bus Structures

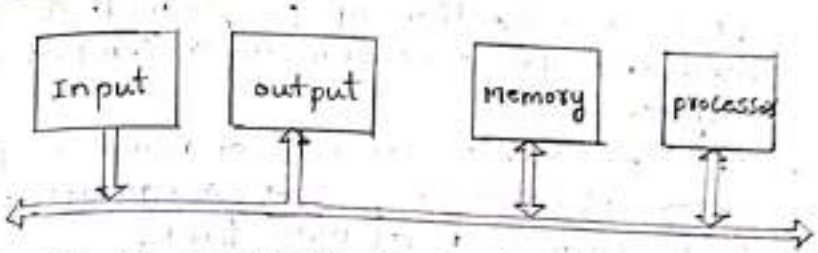


Figure 1-3 - single-bus structure.

* A group of lines that serves as a connecting path for several devices is called a bus.
* Figure 1.3 shows single-bus structure here all units are connected to this bus. because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time.
* Bus Control lines are used to arbitrate multiple requests for use of the bus.
* The main virtue of the single-bus structure is its low cost and its flexibility for attaching peripheral devices.
* The number of lines in the bus affects the speed at which the data travels between different components.
* The devices connected to a bus vary widely in the speed of operation.
* A common approach is to include "buffer registers" with the devices to hold the information during transfers for fast transfers.

(1.5) Software

* It is a collection of program that are available for execution on a computer system.
* This programs consists of number of instructions arranged in a specific order to perform a particular task.
* There are two types of software
  1) System Software
  2) Application Software

Application Software – is a program or collection of programs used by end-user (designed for end users)

System Software (system program).

* It consist of a collection of program and purpose of this program is to make use of Computer more efficient and easier.
→ It plays an very important role in the operation of Computer system.
→ system Software usually stored in ROM during the process of manufacture.
→ system Software is a collection of programs that are executed as needed to perform.

Functions such as

1) Receiving and interpreting user commands
2) Entering and editing application programs and storing them as files in secondary storage devices. Ex- Memory Card, CD, DVD
3) Managing the storage and retrieval of files in secondary storage devices.
4) Running standard application programs such as word processors, spreadsheets or games with data supplied by the user.
5) Controlling I/o units to receive input information and produce output results.
6) Translating programs from source form prepared by the user into object form Consisting of machine instructions.
7) Linking and running user-written application programs with existing standard library routines, such as numerical Computation packages.

* system Software includes
  i) Compiler → software tool
  ii) Interpreter
  iii) Assembler
  iv) Text editor
  v) operating system

i) Compiler

* It Compiles whole program at a time
* It checks all kinds of errors but program execution time is more as if occupies Large amount of memory space.
* It is a program which translates high level languages to machine language.

ii) Interpreter

* It is a program that translates high level languages machine language line by

line and execute it also by line

iii) Assembler

* It is a program to translate Assembly level language into machine level language.

iv) Text editor

* entering and editing application programs
* The user of this program interactively executes commands that allow statements of a source program entered at a keyboard to be accumulated in a file.

v) operating System

* Interface between user and computer hardware
* manage all the resources of computer System.
* It includes assigning memory and magnetic disk space to program and data files.
* moving data between memory and disk units.
* Handling I/o operation.

Difference between System Software and Application Software

	System Software	Application Software
1	It is used for operating computer hardware	It is used by user to perform specific task.
2	It installed on the computer when operating system is installed	It installed according to user's requirements
3	In general, the user does not interact with system software because it works in the background	In general, the user interacts with application softwares
4	It can run independently. It provides platform for running application softwares	It can't run independently, they can't run without the presence of system software
5	Examples are Compiler, assembler, debugger, driver etc	Examples are word processor, web browser, media player etc

performance

(1.6)

* The most important measure of the performance of a computer is how quickly it can execute programs.
* The speed with which a computer executes programs is affected by the design of its hardware and its machine language instr-uctions. Because programs are usually

written in high-level language, perfor-mance is also affected by the compiler that translates programs into machine language.

* performance of the computer is affected by the speed of processor, disk and I/o devices.

* A program will be executed faster, if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.

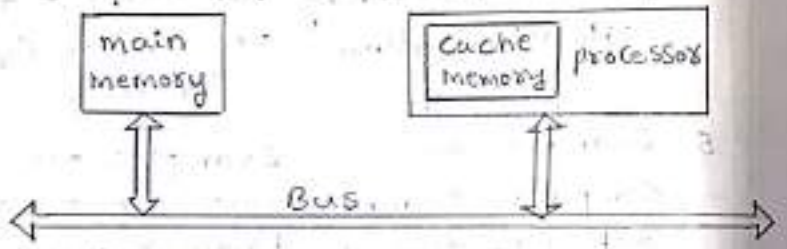


Figure 1.4 — processor cache

* Suppose a number of instructions are executed repeatedly for a short period of time as happens in a programming loop. if these instructions are available in the cache, they can be fetched quickly during the period of repeated use.

* cache memory — small sized type of volatile memory that provide high speed data access to a processor and stores frequently used computer programs, applications and data.

→ fastest memory in a computer

→ typically integrated onto the mother-board and directly embedded in the processor (or) main RAM.

→ clock cycle is an important parameter that affects processor performance.

→ Flow of program instruction and data between memory and the processor

* At the start of execution, all program instruction and the required data are stored in the main memory.

* As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache.

* when the execution of an instruction calls for data located in the main memory, the data are fetched and a copy is placed in cache for frequent access (quick access)

(1.7) processor clock

* processor circuits are controlled by a timing signal called a 'clock'.

* The clock defines regular time intervals Called clock cycles

* To execute a machine instruction the processor divides the action to be performed into a sequence of basic steps.

such that each step can be completed in one clock cycle

* The length p of one clock cycle is an important parameter that affects processor performance.
* Its inverse is the clock rate, $R = \frac{1}{p}$, which is measured in cycles per second.
* processors used in today's personal computers and workstations have clock rates that range from a few hundred million to over a billion cycles per second.
* In standard electrical engineering terminology, the term "cycles per second" is called hertz (Hz).
The term "million" is denoted by the prefix mega (M), and
The term "billion" is denoted by the prefix Giga (G).

(1.8) Basic performance Equation

* The Basic performance Equation is

$$T = \frac{N \times S}{R}$$

where,
T → processor time required to execute a program
N → program contain N microcontroller language instruction
S → Average number of basic steps needed to execute one instructions
R → clock rate

$$R = \frac{1}{p}, \quad p = \text{clock}$$

* To achieve high performance, computer designer must seek (try to find) ways to reduce the value of T. which means reducing N and S. and increasing R.
* The value of N is reduced, if the source program is completed with a fewer micro controller instructions.
* The value of S is reduced, if the instruction have a smaller number of basic steps to perform (or) if the execution of instruction is overlapped.
* using a higher frequency clock increases the value of R, which means that the time required to complete a basic execution step is reduced.

CHAPTER - 2

* Machine Instructions and programs

(1.9) Numbers, Arithmetic operations and characters

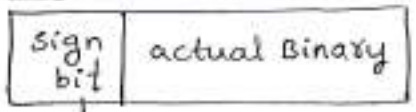
- Computers are built using logic circuits that operate on information represented by two-valued electrical signals
- we label the two values as 0 and 1 bit where bit stands for binary digit.
- The most natural way to represent a number in a computer system is by a string of bits called a binary number.
- A text character can also be represented by a string of bits called a character code

(1.9.1) * Number Representation

- 3 systems are used for representing signed (+ve and -ve) numbers
  1) Sign and magnitude
  2) 1's Complement
  3) 2's Complement
- In all three system, the leftmost bit is 0 for positive numbers and 1 for negative numbers.

1) Signed magnitude form

Syntax :-



0 → for all +ve numbers

1 → for all -ve numbers

7 → 111 is the binary value for 7

if it is

+7 → 0111

-7 → 1111

Ex - +5 → 0101

-5 → 1101

2) 1's Complement

Ex - 9 → 1001 is the binary value of 9

1's complement means changing the value 1 → 0, 0 → 1

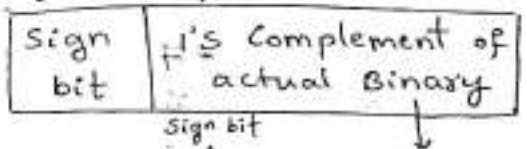
1's complement of 9 is

1 0 0 1

↓ ↓ ↓ ↓

0 1 1 0

for signed numbers



Ex - +7 ⟹ 0111 ⟹ 0000

-7 ⟹ 1111 ⟹ 1000

3) 2's Complement

Add 1 to 1's Complement value.

Ex - 9 ⟹ 1001

↓ ↓ ↓ ↓ 1's complement

0 1 1 0

+ 1 add 1

0 1 1 1

For signed numbers

Sign bit	2's complement of actual binary

Ex - $7 \rightarrow 111$

1's com $\rightarrow 000$

$+1$

2's comp of 7 $\rightarrow 001$

Ex - $-7 \Rightarrow 1001$

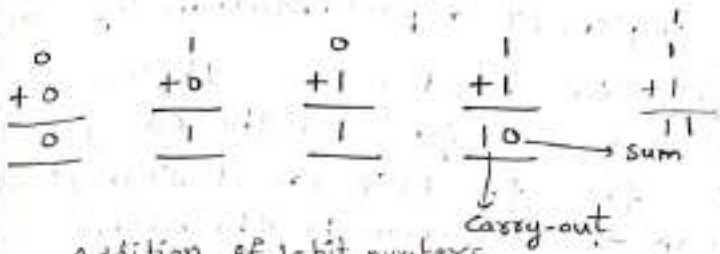
$+7 \Rightarrow 0001$

B	Values represented		
$b_3 b_2 b_1 b_0$	sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	-0	-7	-8
1 0 0 1	-1	-6	-7
1 0 1 0	-2	-5	-6
1 0 1 1	-3	-4	-5
1 1 0 0	-4	-3	-4
1 1 0 1	-5	-2	-3
1 1 1 0	-6	-1	-2
1 1 1 1	-7	-0	-1

figure - 1.5 - Binary, Signed-integer representations

(1.9.2) Addition of positive numbers

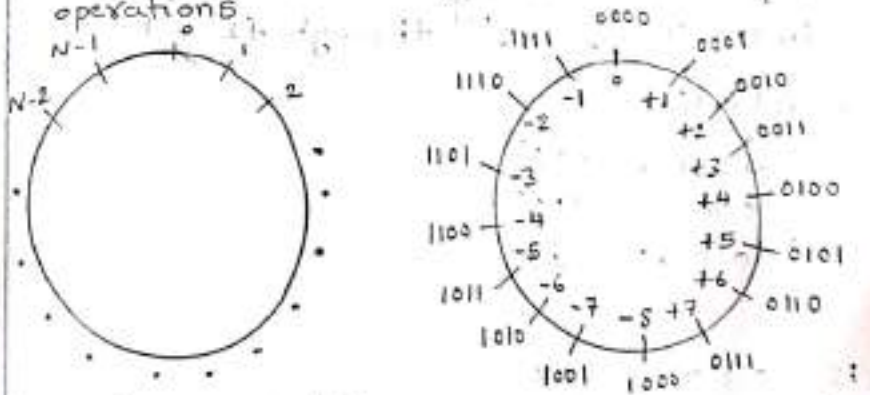
Addition of 1-bit numbers



Addition of 1-bit numbers

(1.9.3) Addition and Subtraction of Signed numbers

* Sign and magnitude system - It is the simplest representation, it is also the most awkward for addition and subtraction operations
* 1's Complement method - Some what better
* 2's Complement method - is the most efficient method for performing addition and subtraction operations



(a) Circle representation of integers mod N

(b) Mod 16 system for 2's-Complement numbers

Figure - 1.6 - Modular number systems and the 2's-Complement system

- Rules governing the addition and subtraction of n-bit signed numbers using the 2's Complement representation system.

1) To <u>add</u> two numbers, add their n-bit representation ignoring the carry-out signal from the MSB (most significant bit) position, the sum will be the algebraically correct value in the 2's complement representation as long as the answer is in the range -2^{n-1} through $+2^{n-1}-1$.

2) To <u>Subtract</u> two numbers x and y, i.e to perform $x - y$, form the 2's complement of y and then add it to x, as in rule 1. Again the result will be the algebraically correct value in the 2's complement representation system if the answer is in the range -2^{n-1} through $+2^{n-1}-1$.

- <u>Addition</u>

(a)
$$\begin{array}{rl} 0\,0\,1\,0 & (+2) \\ +0\,0\,1\,1 & (+3) \\ \hline 0\,1\,0\,1 & (+5) \end{array}$$

(b)
$$\begin{array}{rl} 0\,1\,0\,0 & (+4) \\ +1\,0\,1\,0 & (-6) \\ \hline 1\,1\,1\,0 & (-2) \end{array}$$

c)
$$\begin{array}{rl} 1\,0\,1\,1 & (-5) \\ +1\,1\,1\,0 & (-2) \\ \hline \boxed{1}\,1\,0\,0\,1 & (-7) \end{array}$$

d)
$$\begin{array}{rl} 0\,1\,1\,1 & (+7) \\ +1\,1\,0\,1 & (-3) \\ \hline \boxed{1}\,0\,1\,0\,0 & (+4) \end{array}$$

- <u>Subtraction</u>

(e)
$$\begin{array}{rl} 1\,1\,0\,1 & (-3) \\ -1\,0\,0\,1 & (-7) \end{array} \Longrightarrow \begin{array}{rl} 1\,1\,0\,1 & (-3) \\ +0\,1\,1\,1 & \text{2's complement of } -7 \\ \hline \boxed{1}\,0\,1\,0\,0 & (+4) \end{array}$$

perform

2's complement of (-7)

is 1001

1 1 1 1 1's compliment

0 1 1 0

+1

0 1 1 1

(f)
$$\begin{array}{rl} 0\,0\,1\,0 & (+2) \\ -0\,1\,0\,0 & (+4) \end{array} \Longrightarrow \begin{array}{rl} 0\,0\,1\,0 & (+2) \\ +1\,1\,0\,0 & (-4) \\ \hline 1\,1\,1\,0 & (-2) \end{array}$$

2's complement of (+4)

0 1 0 0

1 1

1 0 1 1

+1

1 1 0 0

g) 0 1 1 0 (+6)
− 0 0 1 1 (+3)

⟹ 0 1 1 0 (+6)
+ 1 1 0 1 (−3)
1 0 0 1 1 (+3)

2's complement of (+3)

0 0 1 1
↓ ↓ ↓ ↓
1 1 0 0
+ 1
1 1 0 1

h) 1 0 0 1 (−7)
− 1 0 1 1 (−5)

⟹ 1 0 0 1 (−7)
+ 0 1 0 1 (+5)
1 1 1 0 (−2)

2's complement of (−5)

1 0 1 1
0 1 0 0
+ 1
0 1 0 1

i) 1 0 0 1 (−7)
− 0 0 0 1 (+1)

⟹ 1 0 0 1 (−7)
+ 1 1 1 1 (−1)
1 1 0 0 0 (−8)

2's complement of (+1)

0 0 0 1
↓ ↓ ↓ ↓
1 1 1 0
+ 1
1 1 1 1

j) 0 0 1 0 (+2)
− 1 1 0 1 (−3)

⟹ 0 0 1 0 (+2)
+ 0 0 1 1 (−3)
0 1 0 1 (+5)

2's complement of (−3)

1 1 0 1
↓ ↓ ↓ ↓
0 0 1 0
+ 1
0 0 1 1

(1.94) overflow in integer Arithmetic

In the 2's complement number representation system, n bits can represent values in the range

n-bits ⟹ -2^{n-1} to $+2^{n-1}-1$

Ex 4-bits ⟹ -2^{4-1} to $+2^{4-1}-1$
⟹ −8 through +7

range is 0 to 2^4-1 i.e 0 to 15

* when the results of an arithmetic operation is outside the representable range, an arithmetic overflow has occured
* when adding unsigned numbers, the carry out c_n from the most significant bit (MSB) position serves as the overflow indicator

This doesn't work for adding signed number's

1 1
1 1 1 1 (15)
+ 1 0 1 0 (10)
1 1 0 0 1 (25)

overflow indicator ↑ → actual result

Ex - 4-bit signed no's

adding +7 and +4 (Addition of 2 +ve numbers)

carry out signal from the most position is '0'

0 1 1 1
0 1 0 0
0 1 0 1 1 → 5, sum vector

which is the code for -5, an incorrect result

addition of 2 -ve numbers

adding -4 and -6

1 1 0 0 (2's Complement of -4)
+ 1 0 1 0 (2's Complement of -6)
1 0 1 1 0 (+6) another incorrect result

In this case, the carry out signal is 1

Thus overflow may occur if both summands have the same sign

clearly, the addition of numbers with different signs cannot cause overflow.

conclusions

* overflow can occur only when adding two numbers that have the same sign
* The carry-out signal from the sign-bit position is not a sufficient indicator of overflow when adding signed numbers.
* A simple way to detect overflow is to examine the signs of the two summands x and y and the sign of the result when both operands x and y have the same sign, an overflow occurs when the sign of s is not the same as the signs of x and y

x - 4
y + -6
sign of s 1 0 → overflow occurs

* characters

* In addition to numbers, Computers must be able to handle nonnumeric text information consisting of characters.. characters can be letters of the alphabet, decimal digits, punctuation marks and so on
* They are represented by codes that are usually eight bits long.
one of the most widely used such codes is the American standards Commitee on Information Interchange (ASCII).

(Institute of Electrical and electronics Engineers)

1.10 * IEEE STANDARD FOR FLOATING POINT NUMBERS

* In the 2's Complement system, the signed value F, represented by the n-bit binary fraction

$B = b_0 . b_{-1} b_{-2} \cdots b_{-(n-1)}$

is given by

$F(B) = -b_0 \times 2^0 + b_{-1} \times 2^{-1} + \cdots + b_{-(n-1)} \times 2^{-(n-1)}$

where the range of F is

$-1 \leq F \leq 1 - 2^{-(n-1)}$

* Consider the range of values represented in a 32-bit signed fixed point format Interpreted as integers, the value range is approximately 0 to $\pm 2.15 \times 10^9$

* If we consider them to be fractions, the range is approximately
$\pm 4.55 \times 10^{-10}$ to ± 1.
Neither of these ranges is sufficient for scientific calculations like Avag-adro's number (6.0247×10^{23} mode^{-1}) or plank's constant (6.6254×10^{-27} erg.s)etc

* Hence we need to easily accomdate both very large integers and very small fractions.
In such a case, computer must be able to represent numbers and operate on them in such a way that the position of the binary point is said to float and the numbers are called floating point numbers

* Because the position of the binary point in a floating point number is variable, it must be given explicitly in the floating point representation.

* we start with a general form and size for floating point numbers in the decimal system and then relate this form to a comparable binary represen-tation.

* A useful form is
$\pm X_1 . X_2 ------ X_7 \times 10^{\pm Y_1 Y_2}$
where X_i and Y_i are decimal digits

* Both the numbers of significant digits (7) and the exponent range (± 99) are sufficient for a wide range of calculations.

* It is possible to approximate this mantissa precision and scale factor range in a binary representation that occupies 52 bits

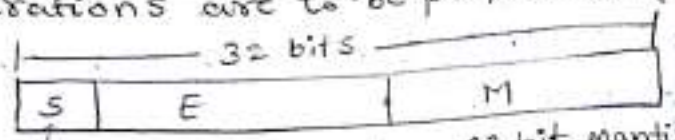
* It is possible to approximate this mantissa precision and scale factor range in a binary representation that occupies 32 bits, which is a standard computer word length.

* A 24-bit mantissa can approximately represent a 7-digit decimal number and an 8-bit exponent to an implied base of 2 provides a scale factor with a reasonable range. one bit is needed for the sign of the number

* This Standard representingboth the representatio floating point numbers in 32-bits has been developed and specified in detail by the IEEE (Institute of Electrical and Electronics Engineers)

* The Standard represents both the representation (① Single precision and ② Double precision) and the

way in which the four basic arithmetic operations are to be performed



Sign of no
0 = +ve
1 = -ve

8 bit signed Exponent

23 bit mantissa fraction

fig (a)

Value = $\pm 1.M \times 2^{E'-127}$

In the figure 32-bit single precision representation is given.

* Sign of the number is given in the first bit, followed by a representation for the exponent of the scale factor.

* Instead of the signed Exponent E, the value actually stored in the exponent field is an unsigned integer $E' = E + 127$. this is called Excess-127 format.

Thus E is in the range $0 \leq E' \leq 255$

The end values of this range 0 and 255 are used to represent special values.

The range of E for normal value is $-1 \leq E \leq 254$

This means the actual exponent E, is in the range $-126 \leq E \leq 127$

The single precision representation occupies 32-bit word

* The scale factor has a range of 2^{-126} to 2^{127} which is approximately equal to $10^{\pm 38}$

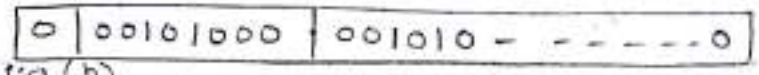
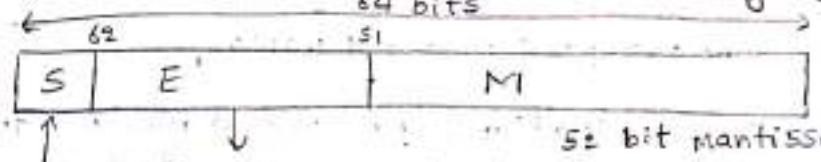


fig (b)

Value represented = $+1.001010\text{---}0 \times 2^{-87}$

$E' = 40 \Rightarrow E = E' - 127 = 40 - 127 = -87$

* To provide more precision and range for floating point numbers, the IEEE Standard also specifies a double precision format as shown in fig (b)



Value represented = $\pm 1.M \times 2^{E'-1023}$

fig (b) – Double precision

* Two ways we can operate on floating point numbers

i) If a number is not normalised it can always be front in normalized form by shifting the fraction and adjusting the exponent

fig (c) : unnormalized value

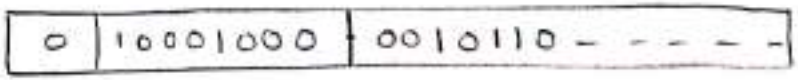


Fig shows an unnormalized value $+0.0010110---\times 2^9$ and its normalized value is $1.0110---\times 2^6$

- Since the scale factor is in the form 2^1, shifting the mantissa right or left by one-bit position is compensated by an increase or a decrease of 1 in the exponent respectively

0	10000101	0110 - - - - - - -

fig (d) : Normalized value

Value represented = $+1.0110---\times 2^6$

2) As compitations proceed, a number that does not fall in the representable range of normal numbers might be generated. In single precision, normalization representation requires an exponent less than -126 or greater than $+127$

In the first case, we say that under flow has occured and in the second case we say that ones flow has occured.

Memory Location and Addresses

* number and character operands (value), as well as instructions are stored in a memory of a computer.
* Memory consists of many millions of storage cells, each of which can store a bit (small amount of information) of information having the value 0 or 1
* The memory is organised so that a group of n-bits can be stored (or) retrieved in a single basic operation.

Each group of n-bits is reffered to as a word (result of processing data) of information and n is called "word length" (number of bits).

* modern computers have word length that typically range from 16 to 64 bits if the word length of a computer is 32 bits a single word can store a 32-bit 2's complement number (or) four ASCII characters (8-bits) (↳ byte) is shown in figure 1.8
* Accessing (to find information stored in computer) the memory to store or retrieve a single item (permission) of information, either a word (or) byte, requires distinct names (or) addresses for each item location.

Example – 24-bit address generates an address space of 2^{24} (16,777,216) locations = 16 M (16 mega), where 1M is the number 2^{20} (1,048,576)

→ if we take 16 megabytes of RAM
$2^{20} \times 2^4 = 2^{24} = (16,777,216)$ locations
The number of address bit requires is 24 to identify each location.

* A 32-bit address creates an address space of 2^{32} or 4 G (4 giga) locations, where 1G is 2^{30}

* other notational conventions that are commonly used are K (Kilo) for the number 2^{10} (1,024) and T (tera) for the number 2^{40}

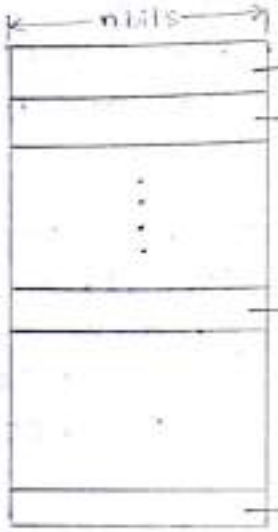
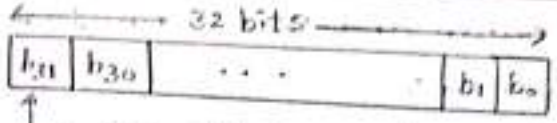


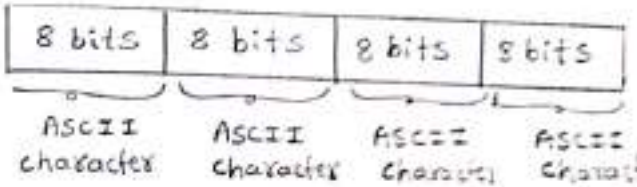
Figure - 1.7 – Memory words

* The memory of a computer can be schematically represented as a collection of words. as shown in figure 1.7



$b_{31} = 0$ for positive numbers
$b_{31} = 1$ for negative numbers

(a) A signed integer



(b) Four characters

Figure - 1.8 – Examples of encoded information in a 32-bit word

Byte addressability

* 3 basic information quantities to deal with
1) bit → is the abbreviation for binary digit or a unit of stored information that can be 0 or 1
2) byte → always 8-bits
3) word → ranges from 16 to 64 bits

* Term 'Byte-addressable memory is used for assignment of successive memory location for address in memory.

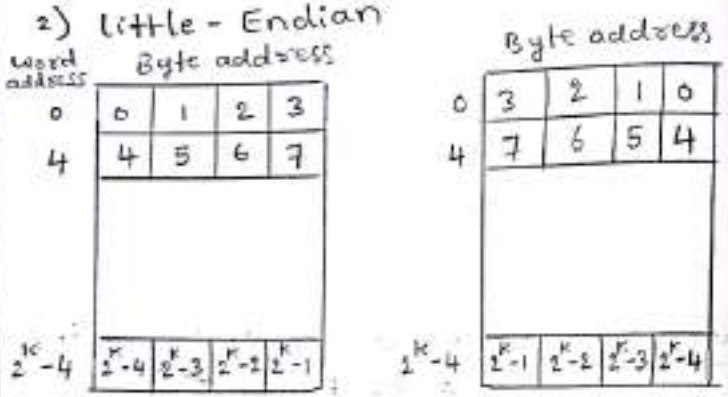
* Byte locations have addresses 0, 1, 2 ---- Thus if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8 - - - - , with each word consisting of four bytes.

1.11.2 Big - Endian and little - Endian Assignments

- 2 ways that byte addresses can be assigned across words

1) Big - Endian

2) Little - Endian



(a) Big - endian assignment

(b) Little - endian assignment

Figure - 1.9 - Byte and word addressing

1) Big - Endian - It is used when lower byte addresses are used for the most significant bytes (the leftmost bytes) of the word.

2) little - Endian - It is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.

- The words "more significant" and "less significant are used in relation to the weights (powers of 2) assigned to bits when the word represents a number.

- Both little - endian and big - endian assignments are used in commercial machines. (heavy-duty machinery; ex- like biscuit production lines)

- In both cases, byte addresses 0,4,8,--- are taken as the addresses of successive words in the memory and are the addresses used when specifying memory read and write operations for words.

- In addition to specifying the address ordering of bytes within a word, it is also necessary to specify the labeling of bits within a byte or a word.

1.11.3 Word Allignment

- In case of 32-bit word length, natural word boundaries occur at addresses 0, 4, 8 - - - etc
the number of bytes in a word is a power of 2 Hence
→ if the word length is 16 (2 bytes), aligned words begin at byte addresses 0,2,4 - - -
→ word length of 64 (2^3 bytes), aligned words begin at byte addresses
0, 8, 16 - - - - -

(1.11.4) ## Accessing numbers, characters and character strings

* A number usually occupies one ward it can be accessed in the memory by specifying its word address.
* Indivisual characters can be accessed by their byte addresses.
* It is necessary to handle character strings of variable length. the begining of the string is variable length. the indicated by giving the address of the byte containing its first character. succesive byte location contain successive characters of the string.

2 ways to indicate the length of string

* A special control character with "end of String" can be used as last character in the string.

(or)

* Seperate memory word location (or) processor register can contain a number indicating the length of the strings in bytes.

(1.12) ## Memory operations.

* Two basic operations involving the memory are needed.
  1) Load : (Read or fetch)
  2) Store : (write)

1) Load (Read or fetch)

* Transfers a copy of contents of a specific memory location to the processor
* memory contants remain unchanged.

To start load operation

* The processor sends the address of the desired location to the memory and requests that its contents be read.
* memory reads the data stored at that address and sends them to the processor.

2) Store operation (write)

* Transfers an item of information from the processor to a specific memory location, destroying the farmer contents of that location. (specific = place)
* The processor sends the address of desired location to the memory, together with the data to be written into that location.
* An information item of either one word or one byte can be transferred between the processor and the memory in a single operation. (an action or task that can be completed on its own without further steps)

Instructions and instruction sequencing

1.13

A computer must have instructions capable of performing four types of operations.

1) data transfers between the memory and the processor registers.
2) Arithmetic and logic operations on data.
3) program sequencing and control
4) Input/output transfers.

Register transfer notation

1.13.1

Transfer of information from one location in the computer to another possible locations that may be involved in such transfers are memory locations, processor registers (or) registers in the Input/output subsystem.

we identify a location by a symbolic name standing for its hardware binary address.

names for the addresses of memory locations may be LOC, PLACE, A, VAR2;

processor register names may be R0, R5; and Input/output Register names may be DATAIN, OUTSTATUS and so on.

Contents of a location are denoted by placing square brackets around the name of the location.

$$R1 \leftarrow [LOC]$$

means that the contents of memory location LOC are transferred into processor register R1

Example 2 — $R3 \leftarrow [R1] + [R2]$

R3: name of a location where the value is placed

[R1] + [R2]: value

This type of notation known as Register Transfer Notation (RTN)

Add the contents of registers R1 and R2 and then places their sum into Register R3

Assembly Language notation

1.13.2

Ex - Move LOC, R1

transfer from memory location LOC to processor register R1

The contents of LOC are unchanged by the execution of this instruction but the old contents of R1 are overwritten

Ex-2 Add R1, R2, R3

Adding two numbers contained in processor registers R1 and R2 and placing their sum in R3

Basic Instruction types

1.13.3

There are 4 types of instructions

1) Three-address instructions
2) Two-address instructions
3) one-address instructions
4) zero-address instructions.

1) Three-address instruction

It can be represented symbollically as

Add A, B, C

operand A and B are source operand
C is called destination operand

A general instruction of this type has the format

operation Source 1, Source 2, destination

if k bits are needed to specify the memory address of each operand, the encoded form of the above instructions must contain 3k bits for addressing purpose in addition to the bits needed to denote the add operation.

disadvantage

* For a modern processor with 32-bit address space, a 3-address instructions is too large to fit it one-word for a reasonable word length so that two-address instructions used.

2) Two-address instructions

Format : operation Source, Destination

Ex - Add A, B

which performs the operation B ← [A]+[B] when Sum is calculated, the result is sent to the memory and stored in location B, replacing the original contents of this location. this means that operand B is both a source and destination.

* The operation C ← [A] + [B] can be performed by the two-address instruction sequence

```
Move B, C
Add  A, C
```

3) one-address instruction

* Add A

means add the contents of memory location A to the contents of the accumulator register and place the sum back to the accumulator.

* Load A

The load instruction copies the contents of the memory location A into the ACCumulator.

* Store A

Store instruction copies the contents of ACCumulator into memory location A

* using only one-address instruction, the operation

C ← [A] + [B]

can be performed by using / executing sequence of instructions

```
Load A
Add  B
Store C
```

if more number of registers present inside the cpu then operations are allowed only on operands that are in processor registers.

The C = A+B can be performed in instruction sequence

```
MoV A, Ri
MoV B, Rj
Add Ri, Rj
Move Rj, C
```

4) zero - address instruction

* In this instruction location of all operands are defined implicitly.
* This type of instruction are found in machine that store operands in a structure called "pushdown stack"

```
PUSH A
PUSH B
ADD A+B
PUSH C
PUSH D
ADD C+D.
```

Example - $X = (A+B) * (C+D)$

Three - address instruction —
```
Add A, B, R1
Add C, D, R2
Mul R1, R2, X
```

Two - address instruction —
```
MOV A, R1
add B, R1
Move C, R2
add D, R2
Mul R1, R2
Move R2, X
```

one - address instruction —
```
Load A
Add B
Store T
Load C
Add D
Mul T
Store X
```

Zero address instruction

PUSH A	TOS ← A
PUSH B	TOS ← B
Add	TOS ← (A+B)
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← (C+D)
MUL	TOS ← (A+B) * (C+D)
pop X	M[X] ← TOS

Instruction Execution and straight-line sequencing (13.4)

* program Stored in memory consist of instruction So program execution as whole nothing but a series of instruction cycle.
* Instruction cycle consists of 2 phases

1) Instruction Fetch phase
2) Instruction Execute phase

] IC = FI + EI

1) Instruction Fetch phase - (First phase) (FI)

* The instruction is fetched from the memory location whose address is in the PC (program Counter) to CPU register and then it will be decoded

2) Instruction execution phase - (Second phase) - (EI)

* The instruction in IR (Instruction Register) is examined to determine which operation is to be performed.
The Specified operation is then performed by the processor.

- during program execution, the instructions are executed one after another, until a branch instruction is encounter. this is known as "straight-line sequencing (or) instruction sequency"

Example - let us consider a program add two number.

$$C \leftarrow [A] + [B]$$

Let us Assume that computer allows one memory operand per instruction and word length is 32-bits and memory is byte addressible.

- The processor contains a register called program counter (pc), which holds the address of the instruction to be executed next.
- To begin executing a program, the address of its first instruction must be placed into the program Counter. then the cpu control circuits use the information in the pc to fetch and execute instructions one at a time, in order of increasing addresses this is called "straight line sequencing"
- during the execution of each instruction the pc is incremented by 4 to point to the next instruction. thus after the move instruction at location i+8 is executed, the pc contains the value i+12, which is the address of the first instruction of the next program segment.

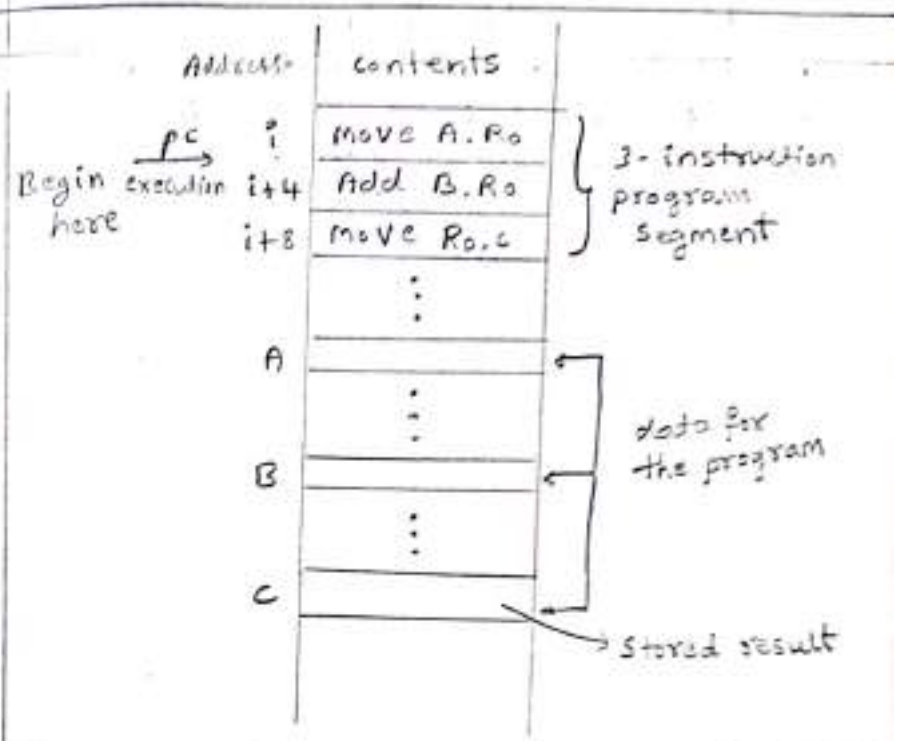


figure 1.10 - A program for $C \leftarrow [A] + [B]$

1.12.5 ## Branching

- This type of instruction loads a new value into the program counter. As a result, the processor fetches and executes the instruction at this new address called the 'branch target' instead of instruction at the location that follows the branch instruction in sequential address order.
- A conditional branch instruction causes a branch only if a specified condition is satisfied.

* If the condition is not satisfied, the pc is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

figure 1.11 – A straight line program for adding n numbers

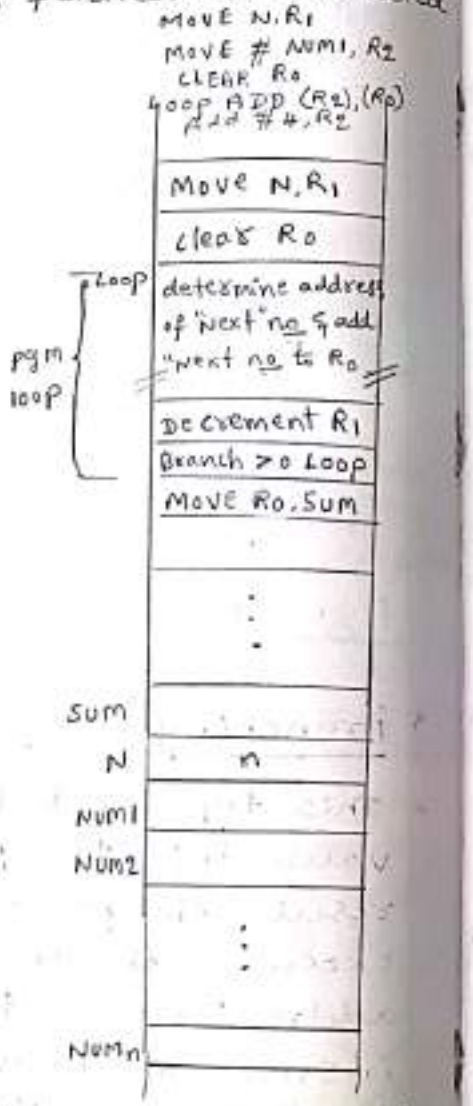


figure - 1.12 – using a loop to add n numbers

Figure Explanation.

* Instead of using a long list of add instruction it is possible to place single add instruction in a program loop.
* In sequential figure 1.11 instruction to be executed as many times needed.
* using loop figure 1.12 it starts at location loop and ends at the location branch > 0
* during each pass through this loop the address of next list entry is determined and that entry is fetched and added to R0.
* Register R1 is used as a counter to determine the number of times the loop is executed. Hence the contents of location N are loaded into register R1 at the begining of the program.

Then, within the body of the loop, the instruction Decrement R1 reduces the contents of R1 by 1 each time through the loop, Execution of the loop is repeated as long as the result of the decrement operation is > 0.

Condition codes

* processor keeps track of information about results of various operations for use by subsequent conditional branch instructions. this is accomplished by recording the required information in individual bits = condition code flags
* These flags are usually grouped together in a special processor register called condition codes register (or) Status registers
  ↓
  are set to 1 (or) cleared to 0, depending on the outcome of the operation performed
* Commonly used flags are

N (Negative) → Set to 1, if the result is -ve otherwise cleared to 0

Z (zero) → Set to 1, if the result is 0 otherwise cleared to 0

V (overflow) → Set to 1 if Arithmetic overflow occurs, otherwise cleared to 0

C (carry) → Set to 1 if a carry-out results from the operation. otherwise cleared to 0

Module - 1

Important questions

1. with a neat diagram, describe the functional units of a computer.
2. Explain Big-endian and little-endian byte address assignment
3. with a neat diagram, discuss the operation concepts in a computer highlighting the role of PC, MAR, MDR and IR.
4. Illustrate single bus structure of a computer
5. Explain the following with an example
   i) Three-address instruction
   ii) Two-address instruction
   iii) one-address instruction
   iv) zero-address instruction
6. Discuss IEEE standard for single precision and double precision floating point numbers with standard notations.
7. write a program to evaluate the arithmetic statement $Y=(A+B)*(C+D)$ using three address, two address and one address instruction.

8 write a short note on
i) Basic performance equation
ii) clock rate

9 Explain the following
i) Byte addressability

10 Explain the memory operation.

11 Difference between system Software and Application Software.

12 perform the subtraction on the following pairs of numbers using 2's Complement format.
i) −3 and −7
ii) +2 and +4
iii) −7 and −5
iv) +6 and +3