UNIT-1 :-

ASCII encoding System:-

ASCII - American Standard Code for Information Interchange.

Each character in the program written by programmer is encoded into Binary format using ASCII encoding system.

0 - 48

A - 65

a - 97

Compilation process :-

1. Preprocessing step:-

```
cpp  file.c  >  file.i
```

In preprocessing step, the contents of system header file are directly inserted into program text. and creates a .i file.

Eg:- #include <stdio.h>

2. Compilation step:-

```
gcc  -S  file.i
```

In compilation step, compiler converts source code is converted into Assembly code and creates a .S file.

3. Assembly phase :-

as file.s -o file.o

Disassembly → | objdump -d file.o |

In Assembly phase, assembler converts assembly language into machine language and creates a .o file.

4. Linking phase :-

ld (locations of Runtime libraries) -o filename

In linking phase, the linker links required Run-time libraries with .o file and forms an executable file.

Hardware Organisation of a System :-

i. PSU :- (Power Supply Unit)

It is a hardware component of computer supplies optimal power to all internal components of computer.

ii. FAN :-

Basically, fan is used in computer for active cooling. Fans expels ~~hot~~ heat ~~air~~ from inside ~~and increase the~~ so that every component is efficient enough.

iii. Hard drive :-

Hard drive is non-volatile data storage device.

iv. CPU :- (Central-Processing unit).

CPU performs all arithmetic, logical and input/output (i/o) operations.

Both are storage devices

v.

RAM (Random access memory)	ROM (Read only memory)
→ RAM is volatile memory and its speed is quite high.	→ ROM is non-volatile memory and speed is slower than RAM.
→ Data is present till power is supplied.	→ Data remains even after power is not supplied.

vi. Buses :-

Buses are designed to transfer a chunk of bytes of data known as words b/w i/o devices.

Size of word ⇒ 4 bytes (32-bit) (or) 8 bytes (64-bit).

vii. Mother board:-

→ The main printed circuit board where all components are plug in is called as 'mother board'.

→ It defines the computer's limits of speed, expandability and memory.

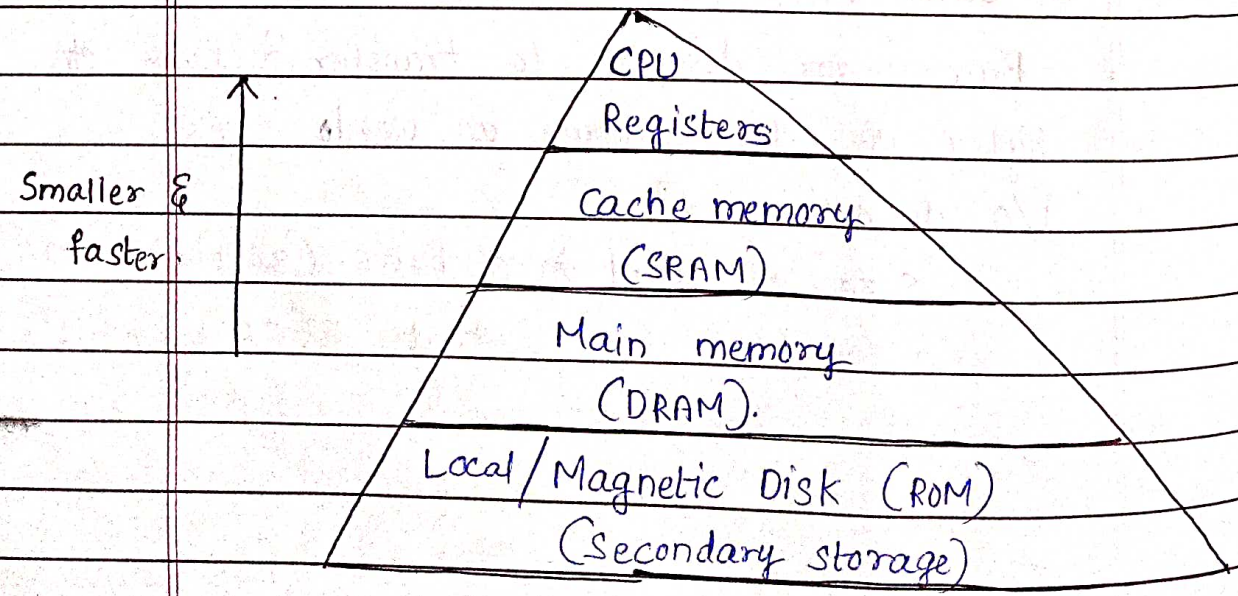
Cache memory:-

→ Cache memory is located in b/w main memory and CPU.

→ The contents of main memory that are frequently used by CPU are stored in cache memory so that processor can access the data in shorter time.

→ Cache memory is the fastest component in memory hierarchy.

Memory hierarchy:-



Threads :-

Threads are virtual components which divides physical core of CPU into virtual multiple cores.

Multi-threading is one way to make programs run faster when multi-core processor is available.

Virtual memory :-

→ Virtual memory creates an illusion having a very big memory. This is done by treating some part of secondary memory as the main memory.

→ Due to virtual memory, the degree of multi-programming will be increased so that CPU utilization will be increased.

Network :-

Two or more computers are connected each other for the purpose of communication is known as network.

Network adapter connects a system to a network.

MEASURING PERFORMANCE :-

Computer performance : Time :-

- **Response / Elapsed Time :-**
  - → Individual user concern.
  - → It is time taken to run an application written by user.

- **Throughput :-**
  - → System manager concern.
  - → It is time taken that how many applications can the machine run at once in parallel.

Defination of Performance :-

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

Suppose, 'X' is n-times faster than 'Y'.

$$(\text{Performance})_X = n \times (\text{Performance})_Y$$

Clock cycle :- 1/(Cycle time).

Clock cycle is the time interval b/w two clock ticks in a processor.

Cycle time ⇒ sec / cycle.

Clock rate (freq.):- cycles / sec (or) Hz.

Performance Equation 1:-

CPU time ↓ = CPU clock cycles ↓ × Clock cycle ↓ time.
(secs/program) (cycles/program) (secs/cycles)

Performance Equation 2:-

CPU time = IC × CPI × (Clock cycle time)

IC ⇒ Instruction count.

CPI ⇒ Clock cycles per instruction. (Avg.)

AMDAHL'S LAW:- (Overall performance gained)

It states that performance gained from using some faster mode of execution is limited by the fraction of time.

$$\text{Speed up} = \frac{\text{improved performance}}{\text{old performance.}}$$

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected.}$$

Eg:- 75% ⇒ integer operations

25% ⇒ floating pt. operations. ⇒ 2x faster.

Overall Speed up $= \frac{(\text{Execution time})_{old}}{(\text{Execution time})_{new}}$

$$= \frac{1}{(1 - \text{fraction}_{enhanced}) + \frac{(\text{fraction})_{enhanced}}{(\text{Speed up})_{enhanced}}}$$

UNIT-2 :-

Encoding byte values :-

1 Byte = 8 bits

Binary : 0000 0000 to 1111 1111

Hex : 00 to FF

Dec : 0 to 255.

Byte-Oriented Memory Organisation :-

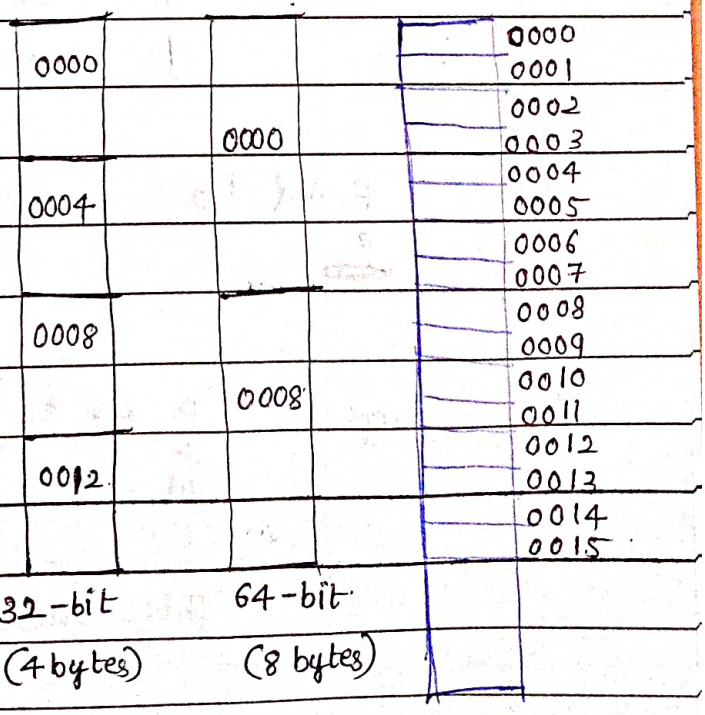
In memory each block is a byte and has a unique address.

Word-size :-

4 bytes for 32-bit system.

8 bytes for 64-bit system.

Word-Oriented Memory Organisation :-



Byte Ordering :-

Eg:- 0x 01 23 45 67

Big-Endian

01234567

Little-Endian (x86)

67452301.

→ Strings are not effected by endian forms.

Eg:- char s[6] = "18243";

For this, both endian forms are same i.e., Big endian.

Bit-level Operators :-

& - Bitwise AND

| - Bitwise OR

~ - Bitwise NOT

^ - Bitwise XOR.

Logical Operators :-

&& - AND

|| - OR

! - NOT

p && *p ⇒ avoid Null pointer access. (Segmentation fault is avoided.

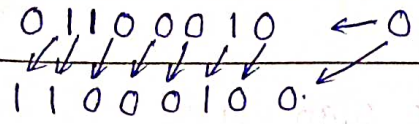
Suppose, (p && *p) ⇒ F.

p → null (F)

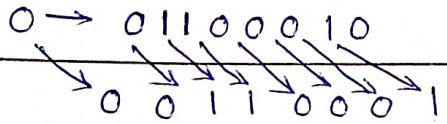
This statement is avoided.

Shift Operations:-

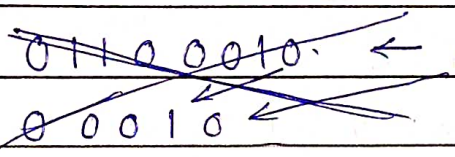
- **Logical left shift :- <<1**



Logical right shift :- >>1.



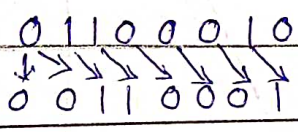
Arithmetic left shift :- <<1



Arithmetic left shift = Logical left shift

Eg:- $x << 3 \Rightarrow x = x * 2^3$.

Arithmetic right shift :- >>1



Eg:- $x >> 3 \Rightarrow x = x / 2^3$.

Encoding integers :-

Unsigned

$$B2U = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Signed (2's complement)

$$B2S = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

MSB ⇒ sign bit

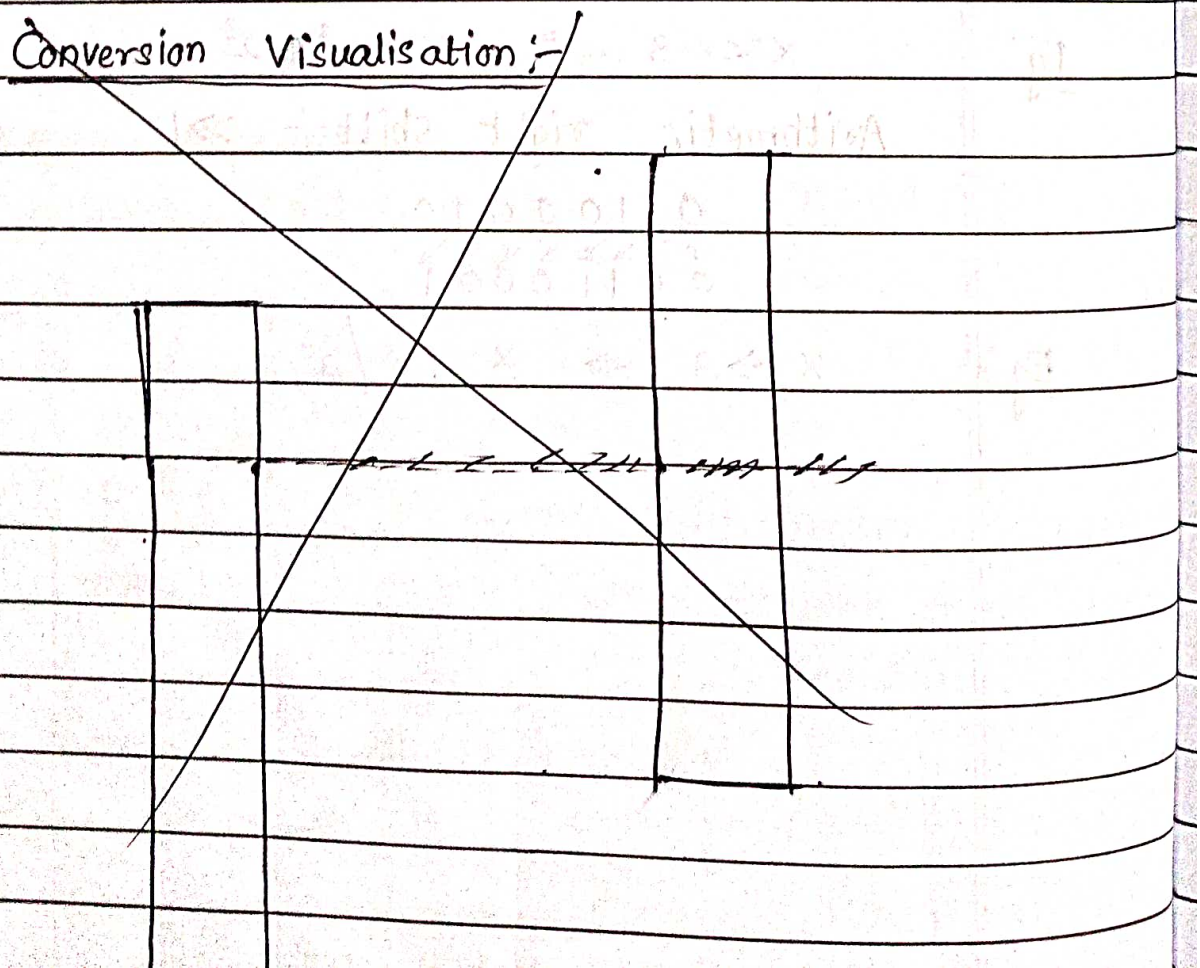
0 → non-negative

1 → negative.

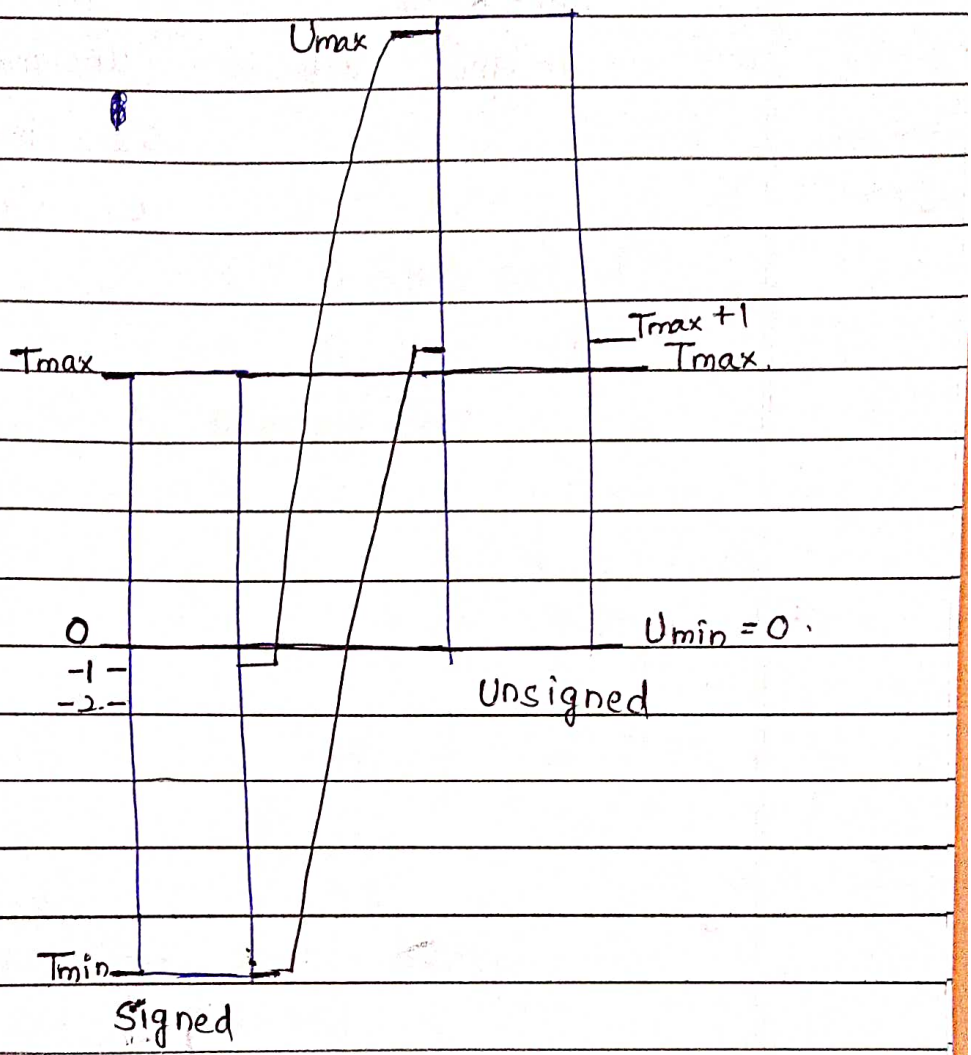
Ranges :-

	Unsigned	Signed
32-bit ⇒	0 to $2^{32}-1$	-2^{31} to $2^{31}-1$
	$U_{min} = 0$	$T_{min} = -2^{31}$
	$U_{max} = 2^{32}-1$	$T_{max} = 2^{31}-1$

Conversion Visualisation :-



Conversion Visualisation :-



Note :- 2's complement Signed integers.

Non-negative signed no.s have same unsigned and signed representation.

Some specific no.s :-

0 : 0000 0000 ----- 0000

-1 : 1111 1111 ----- 1111

Most negative : 1000 0000 ----- 0000

Most positive : 0111 1111 ----- 1111.

2's complement = 1's complement + 1.

Casting :-

~~int x, y;~~
~~unsigned a, b;~~

```
int s1, s2;
unsigned u1, u2;
s1 = (int) u1;
u2 = (unsigned) s2;
```

Floating Point :-

$1011.101_2 \Rightarrow 1\times2^3 + 0\times2^2 + 1\times2 + 1\times2^0 \;.\; 1\times\frac{1}{2} + 0\times\frac{1}{2^2} + 1\times\frac{1}{2^3}$

$\Rightarrow 8 + 2 + 1 \;.\; \frac{1}{2} + \frac{1}{8}$

$\Rightarrow 11.$

Binary to Decimal :-

$1011.101_2 \Rightarrow 1\times2^3 + 0\times2^2 + 1\times2 + 1\times2^0 + 1\times\frac{1}{2} + 0\times\frac{1}{2^2} + 1\times\frac{1}{2^3}$

$\Rightarrow 8 + 2 + 1 + \frac{1}{2} + \frac{1}{8}$

$\Rightarrow 11 + \frac{5}{8} \Rightarrow \frac{88+5}{8} \Rightarrow \frac{93}{8}$

$\Rightarrow 11.625$

Decimal to Binary :-

$11.625 \Rightarrow 1011.101$

$11.625 \rightarrow 1011 \;.\; 101$

$0.625 \times 2 = 1.25 \Rightarrow 1$

$0.25 \times 2 = 0.5 \Rightarrow 0$

$0.5 \times 2 = 1.0 \Rightarrow 1$

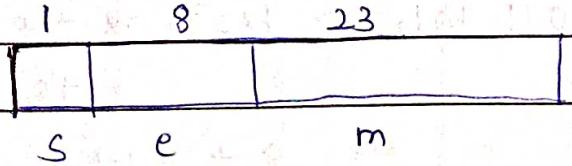
IEEE Floating point :- (Institute of Electrical and Electronic Engineers)

Normalised form :-

→ Integer part should be zero.

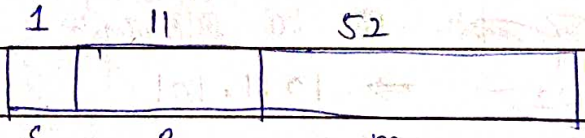
→ $1.d_1 d_2 d_3 \ldots\ldots$

Single Precision :- (32-bit)



$$X_{FP32} = (-1)^S \times (2)^{e-127} (1.m).$$

Double Precision :- (64-bit).



$$X_{FP64} = (-1)^S \times (2)^{e-1023} (1.m).$$

e	m	
255	0	NaN
255	≠0	Infinite no.
0	0	$X = (-1)^S . (2)^{-126} (0.m)$
0	≠0	+0, -0

Q> IEEE format to decimal equivalent

Decimal to IEEE format

Eg:- 40400000H ⇒ Decimal equalent of given floating pt. no.

~~IEEE 754 format~~

sol:

0	100 0000	0100 0000 0000
s	e	m.

$1 \times 2^{-1} = 0.5$

$X_{FP32} = (-1)^s \times (2)^{e-127} \times (1.m)$

$= (-1)^0 \times (2)^{128-127} \times (1.5)$

$= 1.5 \times 2$

$= 3.$

Eg:- Represent following 4 floating point no.s in single precision format?

	s	e	m.
$0.110111 \times 2^{100101}$	= 0	1010 0101	10 1110 0...
$0.110111 \times 2^{-100101}$	= 0	0101 1001	1011100...
$-0.110111 \times 2^{100101}$	= 1	1010 0101	1011100...
$-0.110111 \times 2^{-100101}$	= 1	0101 1001	1011100...

0010 0101

1000 0000

1010 0101

001 00101

1101 1000

+ 1

1101 1001

1000 0000

1 0101 1001

x

Eg:- Represent given number in Single precision format.

263.3

Sol 263.3

2	263	1
2	131	1
2	65	1
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
	1	

$0.3 \times 2 = 0.6 \Rightarrow 0$

$0.6 \times 2 = 1.2 \Rightarrow 1$

$0.2 \times 2 = 0.4 \Rightarrow 0$

$0.4 \times 2 = 0.8 \Rightarrow 0$

$0.8 \times 2 = 1.6 \Rightarrow 1$

$0.6 \times 2 = 1.2 \Rightarrow 1$

0
0
1
0

$\Rightarrow 1\ 0000\ 0111 . 01\ 0011\ 001\ldots$

$\Rightarrow 0.10000\ 0111\ 0100\ 1100\ldots$

s	e	m
0	1000 0111	0000 0111 0100 1100....

$\Rightarrow 1\ 0000\ 0111 . 0100\ 1100\ 1\ldots$

$\Rightarrow 1.0000\ 01110\ 0100\ 1100\ 1\ldots \times 2^8$

s	e	m
0	1000 0111	0000 0111 0100 1100....

8 + 127

Floating point Addition:-

$1.000 \times 2^{-1} + -1.110 \times 2^{-2}$

Convert smaller exp to bigger exp →

$\rightarrow 1.000 \times 2^{-1} - 0.111 \times 2^{-1}$

$\rightarrow (1.000 - 0.111) \times 2^{-1}$

$\rightarrow 0.001 \times 2^{-1}$

$\rightarrow 1.000 \times 2^{-4}$.

```
1.000
0.111
0.001
```

Floating point multiplication:-

$1.000 \times 2^{-1} \times -1.110 \times 2^{-2}$.

$\rightarrow 1.000 \times (-1.110) \times 2^{-3}$.

$\rightarrow -1.110 \times 2^{-3}$.

```
1.11
 x1
1.11
```

UNIT-3 :- Machine Language :-

Q) What is x86 ?

→ x86 is a processor developed based on 8086 microprocessor and its 8088 variant.

→ The term "x86" came because several processors like 8086, 80186, 80286, 80386, 80486 has "86" in end.

Q) CISC, RISC : ?

Complex Instruction Set Computers :
It has a collection of complex instructions and takes long time to execute those instructions.

Reduced Instruction Set Computers :
It has a collection of simple instructions and takes less time to execute those instructions.

Q) Moore's law :-

No. of transistors per chip are doubled every ~~year~~ 18 months.

Q) Instruction Set Architecture :-

It is a very important abstraction b/w hardware and low-level software. (machine language (or) Binary)

Eg :- x86, Pentium, MIPS, ...

Q) Program Counter (PC):

Stores the address of next instruction.

%rip ⇒ PC.

ASSEMBLY LANGUAGE:-

Q) Datatypes:-

char	⇒ Byte (b)	-	1 byte.
short	⇒ Word (w)	-	2 bytes.
int	⇒ ~~double word~~ (l)	-	4 bytes.
char*, long	⇒ ~~quad word~~ (q)	-	8 bytes.
float	⇒ s	-	4 bytes
double	⇒ l	-	8 bytes.

Q) Registers:- Registers are used to store the data.

	8 bytes (l,q)	4 bytes (int)	2 bytes (w)	1 byte (b)
return value ⇒	%rax	%eax	%ax	%al
	%rbx	%ebx	%bx	%bl
	%rcx	%ecx	%cx	%cl
	%rdx	%edx	%dx	%dl
	%rsi	%esi	%si	%sil
	%rdi	%edi	%di	%dil
	%rbp	%ebp	%bp	%bpl
Stack pointer ⇒	%rsp	%esp	%sp	%spl

%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

a) Addressing modes :-

Addressing modes are used to access data stored in registers.

1. Immediate Addressing mode :-

Eg :- $Imm ; $108

2. Register Addressing mode :-

Eg :- %rax , %rbx.

3. Absolute Addressing mode :-

~~Eg :- (%rax) / (%rax)~~

Eg :- 0x104 , 0x108.

4. Indirect Addressing mode :-

Eg :- (%rax).

5. Base + displacement Addressing mode :-

Eg :- 4(%rax) ⇒ 4 + (%rax)

6. Indexed Addressing mode :-

Eg :- (%rax, %rbx). ⇒ (%rax) + (%rbx)

7. Indexed Addressing mode :-

Eg :- 9(%rax, %rbx) ⇒ 9 + (%rax) + (%rbx)

8. Scale index Addressing mode:-

Eg:- (, %rax, 4). ⇒ 4(%rax)

9. Scale index Addressing mode:-

Eg:- 0xFC (, %rcx, 4). ⇒ FCh + 4(%rcx)

10. Scale index Addressing mode:-

Eg:- (%rbx, %rax, 5). ⇒ (%rbx) + 5(%rax)

11. Scale index Addressing mode:-

Eg:- 0x5 (%rbx, %rax, 6). ⇒ 5h + (%rbx) + 6(%rax).

Q) Moving data:-

movq src, dest.

Eg:-

movq $0x4, (%rax) ✓

movq $0x4, %rax. ✓

movq %rax, %rbx ✓

movq %rax, (%rbx) ✓

movq (%rax), %rbx ✓

movq (%rax), (%rbx) ✗

Note:-

movabsq ⇒ unsigned values.

movq ⇒ signed (or) unsigned values.

movzbw ⇒ zero extension from byte to word.

(1 byte to 2 bytes).

AF ⇒ 00AF.

0000 0000 AF

Zero extension.

movsbw ⇒ sign extension from byte to word.

Eg:- ~~move~~ $

~~movsbw~~ $1, 1 ⟹ 00 01

~~movsbw~~ $-1, -1 ⟹ FF FF.

cltq ⇒ sign extension from %eax to %rax.

Q) Stack:-

pushq S

popq D

Q) Address Computation Instruction:-

leaq S, D ⇒ D = &S.

D ⇒ register.

S ⇒ address.

Q) One Operand Operations:-

incq D ⇒ D++

decq D ⇒ D--.

negq D ⇒ 2's complement

notq D ⇒ 1's complement.

Q) Arithmetic Operations :-

addq S,D ⇒ D = S + C.

subq S,D ⇒ D = D − S.

imulq S,D ⇒ D = D * S.

Q) Shift Operation :-

salq S,D. ⇒ D = D << S

sarq S,D ⇒ D = D >> S

shlq S,D ⇒ D = D << S

shrq S,D ⇒ D = D >> S.

Q) Bitwise Operations :-

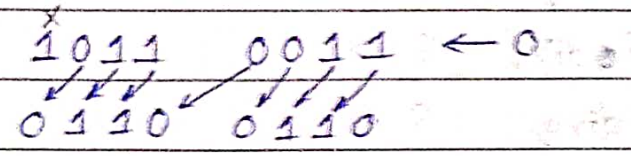
xorq S,D ⇒ D = D ^ S.

andq S,D ⇒ D = D & S

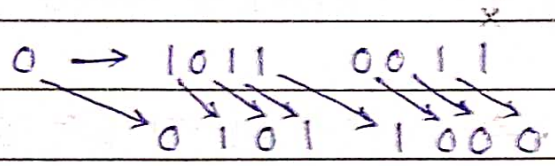
orq S,D ⇒ D = D | S.

Q) Logical shifts :-

left logical shift :- $<<1$

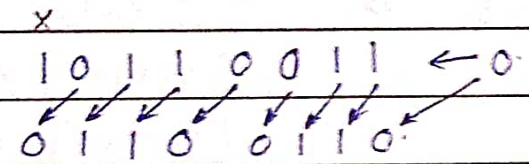


Right logical shift :- $>>1$

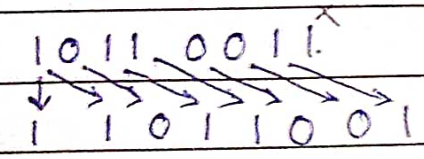


Q) Arithmetic shifts :-

Arithmetic left shift :- $<<1$



Arithmetic right shift :- $>>1$



Eg :-

1) $x << 2 \Rightarrow x = x * 2^2$.

2) $x >> 2 \Rightarrow x = x / 2^2$.

Note :- Arithmetic left shieft and logical left shift operations are same.

Q) Special Arithmetic Operations:-

~~imulq S ⇒~~

~~mulq S ⇒~~

~~idivq S ⇒~~

~~divq S ⇒~~

64-bit mul ⇒ imulq S ⇒ [%rdx]:[%rax] ← S * [%rax]

mulq S ⇒ [%rdx]:[%rax] ← S * [%rax]

idivq S ⇒ [%rdx]:[%rax] / S

[%rax] ← Quotient

[%rdx] ← Remainder

divq S ⇒ [%rdx]:[%rax] / S

[%rax] ← Quotient

[%rdx] ← Remainder