4/5/22 Computer Organization & Architecture

Introduction:

DLD + OS = COA
- DLD → Hardware 70%
- OS → 30%
  - ⇒ Number System
  - ⇒ Binary, decimal, Hexa ...

Languages
- High level language
- Assembly language
- machine language

Languages: ① High level language:
⇒ which is understood by user, easy to understand
⇒ It is closer to human languages.
∴ But this language is converted into machine language
∴ machine can't understand the high level language
Ex: C, C++, Java etc.,

2) Assembly (Middle level languages:-
⇒ It is both high level and machine language.
⇒ It is not a completly highlevel and not a completely machine language.
⇒ Uses mnemonic to represent

Ex: Add
DIV
SUB
MUL

∴ It Uses microprocess Architecture - 8000, 8085, 8086 } lab

∴ It requires Software called an assembler to convert into machine code.

3) low level/ machine languages:
⇒ which is understood by machine. It is also known as binary language
⇒ It contains 0, 1's only
⇒ It can be directly executed by a computer.

Compiler / Interpreter:

These two are used to translate the high level language into machine language.

Compiler:

⇒ Compiler Scans the entire program and translates it as a whole into machine code.

⇒ compilers usually takes a large amount of time to analyze the Source (So) code.

Interpreter:

⇒ It Scans / converts line by line

⇒ Interpreta usually takes less amount of time to analyze the Source code

⇒ Overall execution time is comparatively Slowes than compiler.

Object Oriented and procedural language

Object Oriented programming:

It is based upon the concepts of objects

Ex: fun c ( )
id1, id2 id3 ...
{ object

∴ Suppose we want object 7.
directly we check the object like
id 7
for id 700 ---- 7 id 700

procedural programming:-

which is derived from Structured programming.

Ex: fun
{
id
}

∴ 7 → it is possible
for 700 it is not possible.

which is best Object Oriented / procedural ?

** we can't Say

⇒ It is based on the Situation.

⇒ If we want to check condition at 8. It is betles at procedural.

⇒ If we want to check Condition at 700. It is betles at Object Oriented ⇒ for the Big data we Can Say Object. Oriented

1 bit - either 0 (or) 1

1 Nibble - 4 bits (Half of the byte)

1 byte - 8 bits

1 kilobyte - 1024 bytes $= 2^{10}$

1 mega byte - 1024 KB $= 2^{20} \Rightarrow 1024 \times 1024$ bytes

1 Giga byte - 1024 MB

1 Tera byte - 1024 GB

ASCII

American Standard Code for Information Interchange

ASCII - 256

$\Rightarrow$ we pass information by the help of ASCII

A - 65

a - 97

It contains Small alphabets - 26
Capital alphabets - 26
numbers 0-9
Special letters
} 256

Why char have 1byte:

Characters is nothing but letters

1byte $\rightarrow$ 8 bits, So

2^8 bits/character $= 256$

memory:

Internal Storage

- Register KB - cost
- cache
- primary - GB (or) Virtual memory
- Secondary (Bigger) TB

Main memory: / primary :-

which stores the active instructions and data for the program being executed on the processor

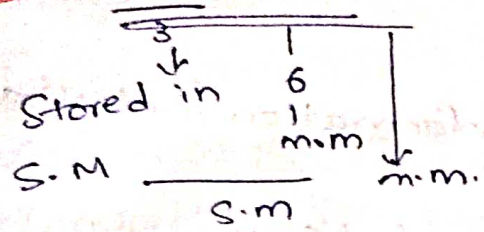
$\Rightarrow$ which is used to store data (or) Information temporarily

$\therefore$ It is Volatile.

Secondary memory:

- external Storage devices that are used to store data (or) Information permanently.
- It is non volatile.
- which is used to backup and stores all active and inactive programs and data, typically as files.

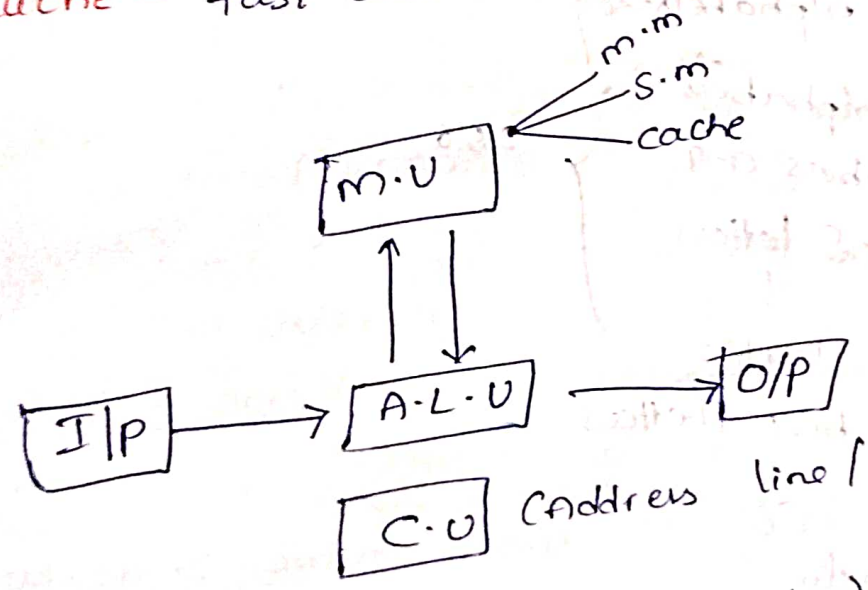
Ex: 1+2+3+4+5+6+7+8 - ..



Register - Small type of memory

KB - cost

Cache - fast access



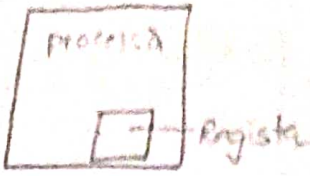
MU — memory un
CU — control uni
ALU — Arithmetic logic unit

∴ CPU (Heart of the computer)

Central processing Unit (CPU)

9/5/22

Register: It is a small type of memory.

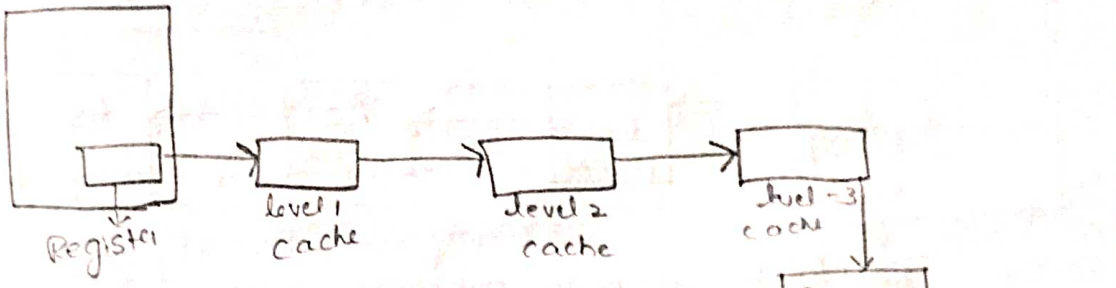


Register is in the processor.

Cache: 3 level of cache is there.

→ level 1 cache
→ level 2 cache
→ level 3 cache

cache → fast access.



Hdd: (Hard disk drive)

to access first compiler sees HDD. If it is not present then It goes to the RAM → then, it is not present in the RAM It goes to the cache.

first time accessing:

first time accessing is from the Registers.

Registers – Small type of memory

Small - MBS, Word

Registers means little one, we take only 1. But we access as a group.

we access by instructions.

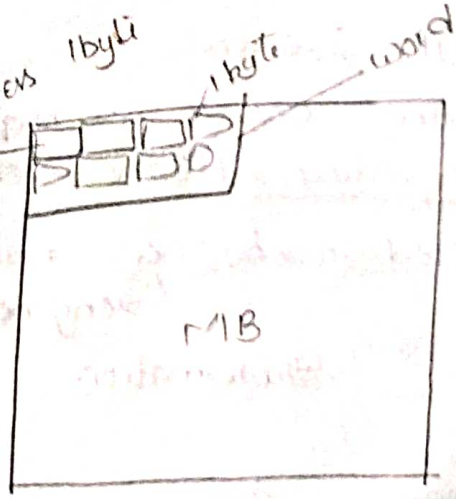
we access by 1 word

1 word length = 16-64 bits

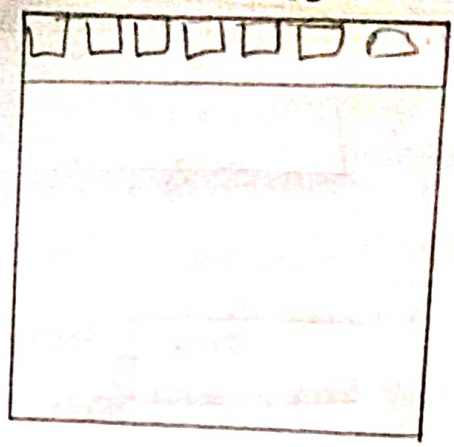
8 bits = 1 word

1 byte = 1 word

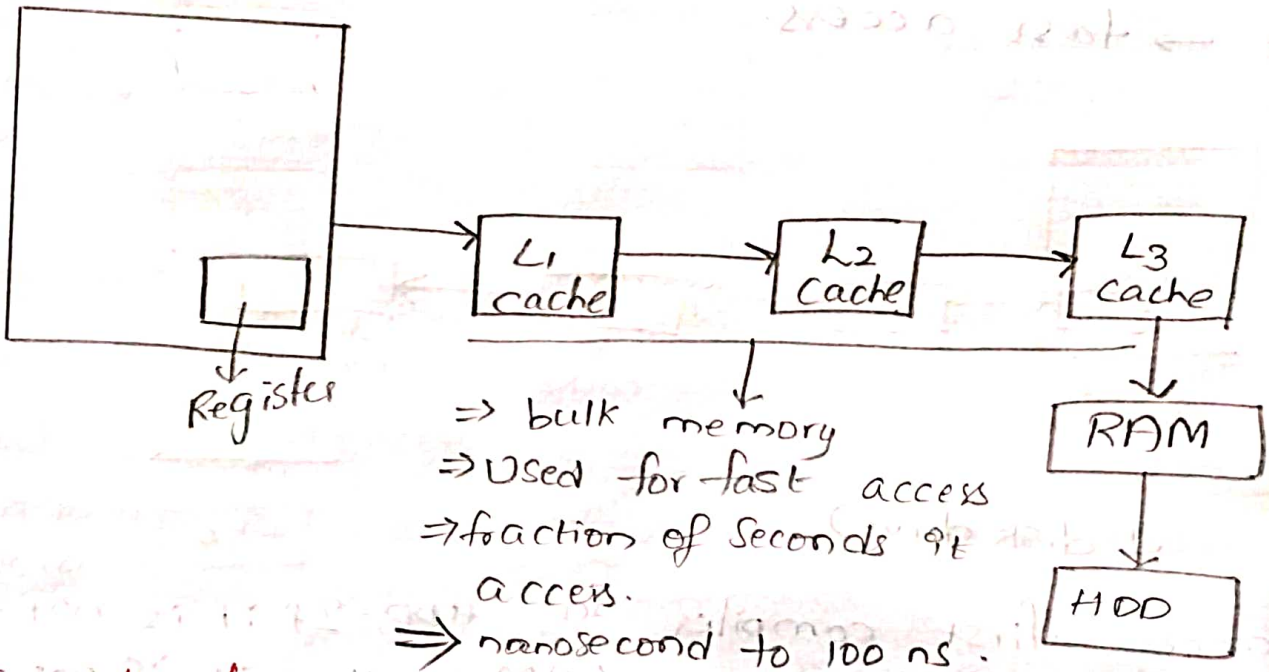
all space = 16-64 bits



16-64.



It is actually in 1 bit
word = 8 bit
word length = 16-64 bits
⇒ cpu access by 1 word
1 byte of data access.



⇒ bulk memory
⇒ Used for fast access
⇒ fraction of seconds of access.
⇒ nanosecond to 100 ns.

Computer types

* Super Computer
* mainframe / Server / enterpriser
* personal computer
* Workstation Computer

① Super Computer:
⇒ It is used in weather forecasting
⇒ bulk size
⇒ more cache's are required.

② mainframe / Server / enterprise
⇒ Used in ⇒ business purpose
↳ engineering
⇒ Animation

③ personal Computer :-

cpu process

Ex: Systems, desktop, mobiles, tab.

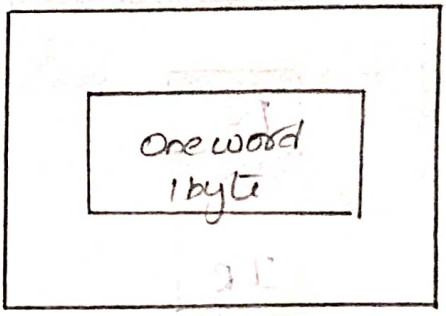
④ Workstation computer:

⇒ It is a big computer

⇒ Used in atm, Stock markets;

*** Memory access time

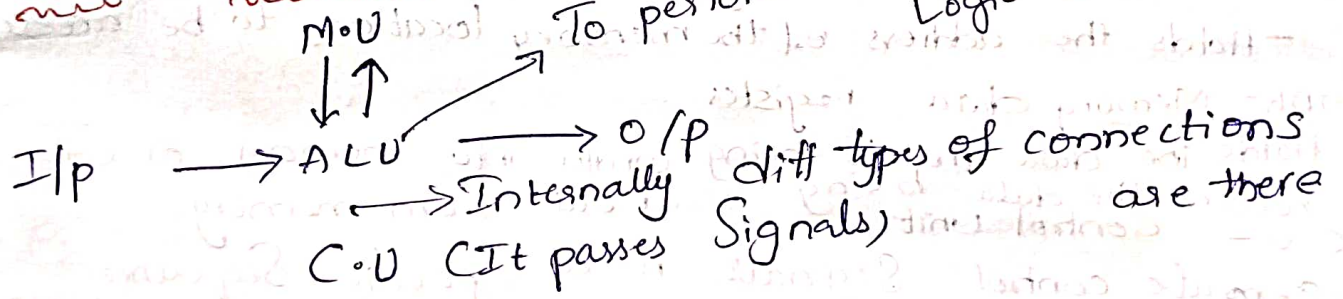
⇒ accessing the word from main memory is called RAM



⇒ time → nanoseconds to 100ns

⇒ How long It takes for a character in RAM to be transferred to (or) from the CPU.

Basic functional Units



CPU

It follows Some notations

1) postfix notation
2) prefix notation
3) Infix notation.

} we do this notation from Bod-mass rule

postfix

Operator is in Back

Ex: a+b

ab+

prefix

⇒ Operators is in front

Ex: a+b

(a+b) +ab

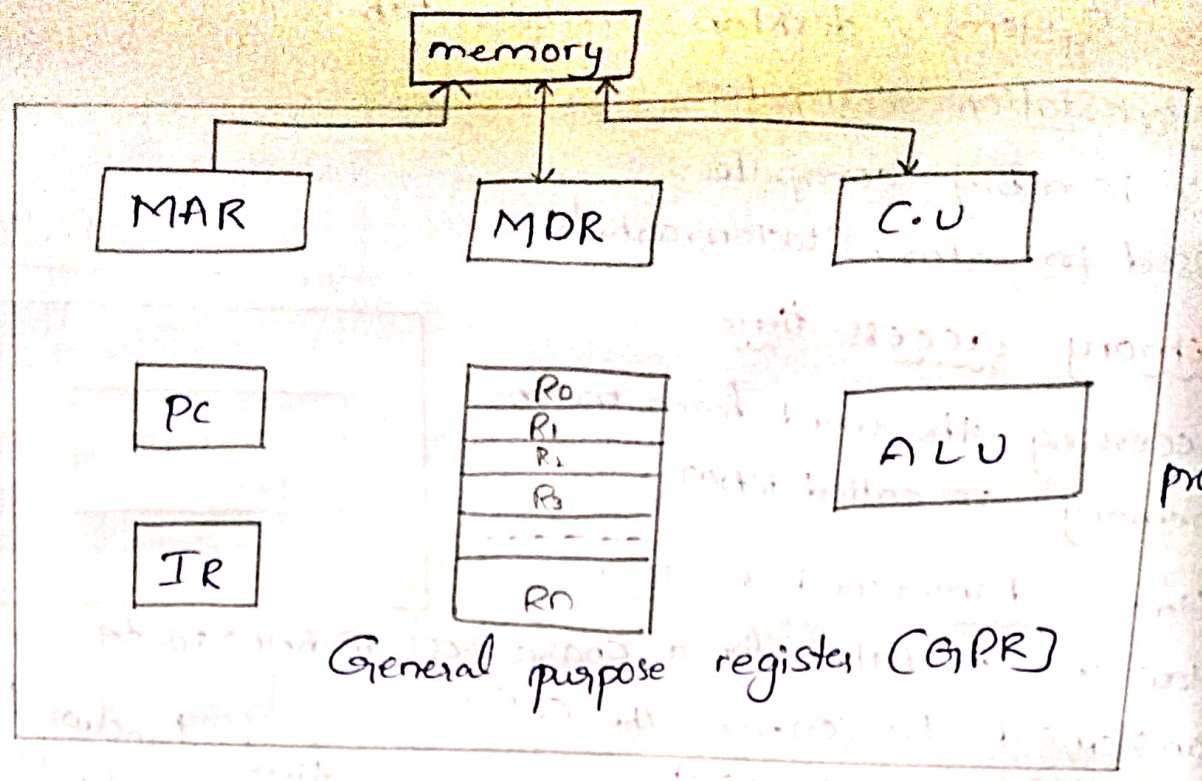
Infix

⇒ Operators is in middle

Ex: a+b

⇒ a+b

memory & processing connections (Internally)



MAR - Memory address register
- Holds the address of the memory location to be accessed

MDR - Memory data register
- Holds the data that is being written into memory, or will receive the data being read out from memory.

C.U - Control Unit
- generate control signals in a specific sequence

PC - program counter ; Holds the memory address of the next instruction to be executed (Next Instruction).

I.R - Instruction register ; Temporarily holds an instruction that has been fetched from memory. (present Instruction)

ALU - Arithmetic logic unit
- It contains some operations
To perform mathematical & logical functions

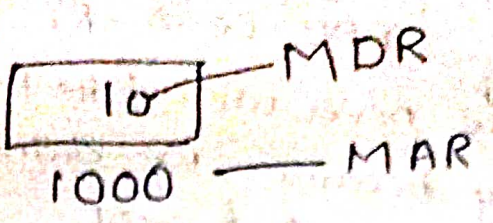
C.U → data lines / Signals
→ Address lines
→ control lines

∴ It pass signalls

ADD LOC A, R1

↓

Address + Data

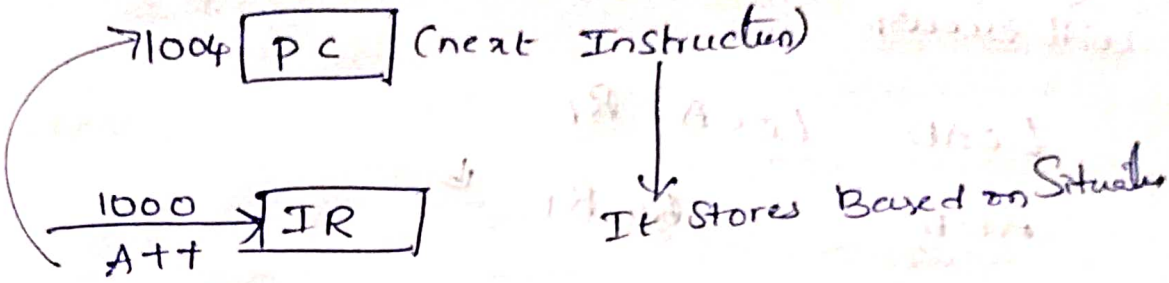


A + B ;

↳ ALU

A = 10

B = 100 0



A + B

10 + 20

= 30

a) A = 10

b) B = 20 + = 30

c) R_1 => 30

Now

R0 = 10

R1 = 30

Add LocA, R1

↓

Address numbers helps us to get the

MAR

+

MDR → Data } with the help of Signals (C.U)

=> we get the data and Save the data in LocA

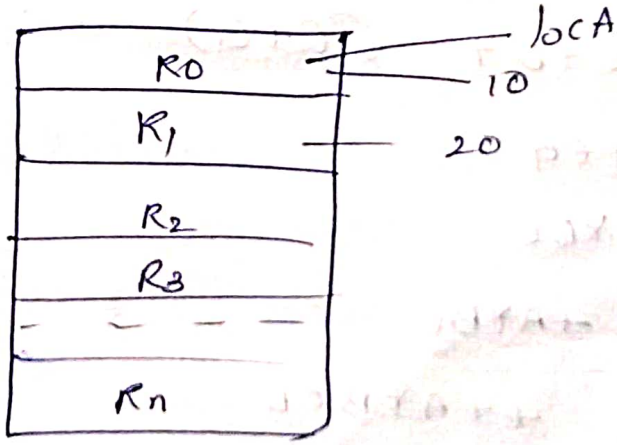
=> If the value is in R_1 → It's OK

=> If the value is not in R_1 we get the data by using

MAR

+

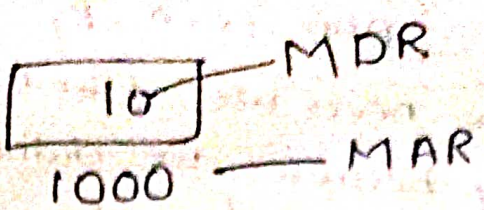
MDR



ADD LOC A, R1

↓

Address + Data

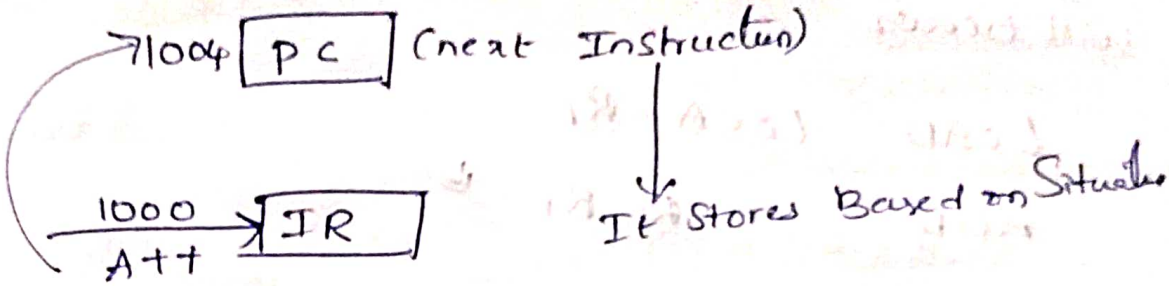


A + B ;

ALU

A = 10

B = 100 0



A + B

10 + 20

= 30

R0) A = 10

R1) B = 20 + = 30

∴ R2 ⇒ 30

NOW

R0 = 10

R1 = 30

Add LocA, R1

↓

Address numbers

helps us to get the

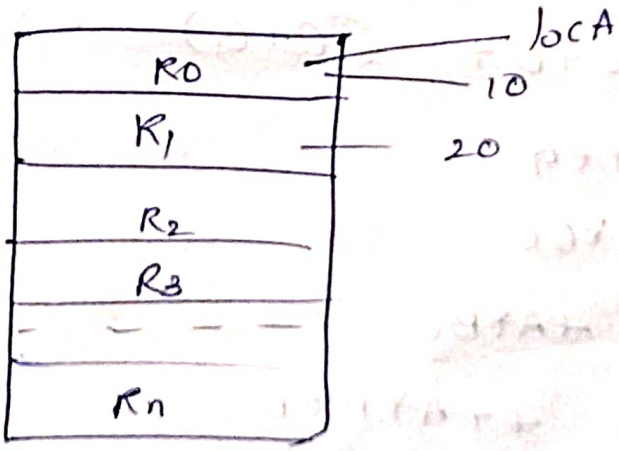
MAR

+

MDR → Data } with the help of Signals (C.U)

⇒ we get the data and

Save the data in LocA



⇒ If the value is in

R1 → It's OK

⇒ If the value is

not in R1 we get

the data by using

MAR

+

MDR

ADD LOCA, R1

⇒ we never use like this Instruction

⇒ why because we didnot represent in Single Statement.

⇒ we won't allow one way Instructions.

⇒ why because when we use like this Hazards will occur.

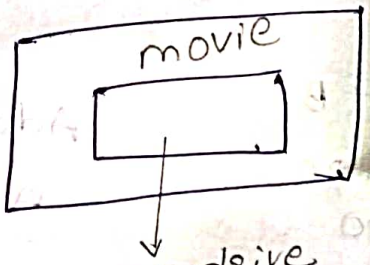
LOAD LOCA, R1

ADD R0, R1 ✓

Interept:

pendrive → movie → display.

1st priority 2nd priority



ISR → Interept Service routine.

prefix, postfix

⇒ we take two operands in between operator

⇒ based on BODMAS rule.

A+B* C+D+ (E+G) → prefix

E+G = +EG

B*C = *BC

A+*BC = +A*BC

+A*BC+D = ++A*BCD

++A*BCD ++EG = +++A*BCD+EG //

(A+B) + (E+G) * E+F*G — prefix

A+B = +AB

E+G = +EG

(E+G)

+EG*E = *+EGE

F*G = *FG

[+AB++EG = ++AB+EG]

++

+AB +(*+EGE) = ++AB*+EGE

(++AB*+EGE)+(*FG)

+++AB*+EGE*FG //

postfix

(A+B) + (E+G) * E+F*G

(AB+) + (EG+) *E + (FG*)

(AB+) + (EG+E*) + (FG*)

(AB+ EG+E*+) + FG*

AB+EG+E*+FG*+ //

Assignment-1

Different types of Computer

Computers can be broadly classified into four categories based on their speed, amount of data that they can process, and price..

These Categories as follows

* Super computers
* Mainframe computers
* Minicomputers
* Microcomputers.

Super Computers:

Among the four categories,

⇒the Supercomputer is the fastest, most powerful, and most powerful, and most expensive computer.

⇒It is first developed in the 1980's to process large amount of data & to solve complex Scientific problems

⇒It perform more than one trillion calculations in a Second.

⇒A Single Super computer can Support thousands of users at the same time.

⇒ Used in weather forecasting, nuclear energy research, aircraft design, automotive design, Online banking. etc..

Mainframe Computers

⇒It is large Scale computers. (Smaller than Supercomp

⇒These are very expensive and need a very large room with air conditioning,

⇒ mainframe Support 50,000 users at the same time.

⇒ Users can access by using terminals or via Pc

⇒used in Organizations Such as banks, Universities....

MINI computers

⇒It is smaller, cheapes, and Slowes than remaining 2.

⇒also knowon as midrange computers.

⇒ Used in busines, education, hospitals etc..

⇒Some mini computers used by only a Single user.

Micro computers:

⇒micro computers are commonly known as pcs,

⇒ Very Small & cheap.

⇒pcs can be classified into the following categories.

Desktop pc

⇒ widely used in homes & office

Laptops

⇒are Small micro computers

Workstations

⇒ Workstations are Single Uses computes

⇒ widely used as poweiful Single-uses computes by Scientists

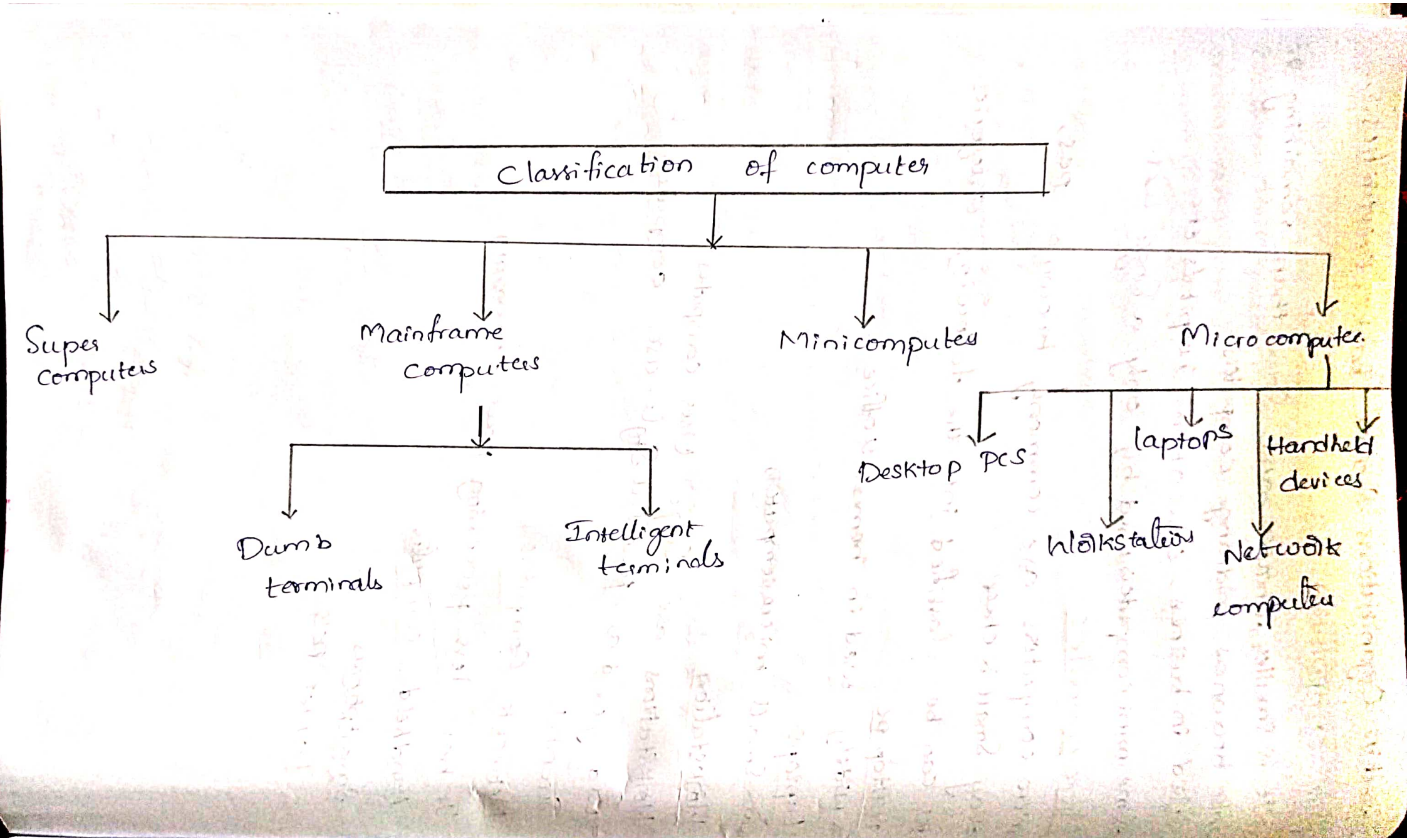
Network Computers

It have less processing power, memory

Handheld computers

Smartphones

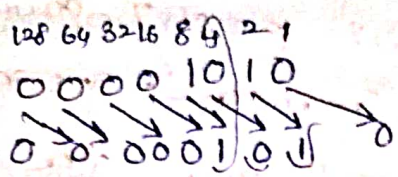
Tablet PC's


Classification of computer
Super computers
Mainframe computers
Minicomputers
Micro computers
Dumb terminals
Intelligent terminals
Desktop PCs
laptops
Handheld devices
Workstations
Network computers

11/5/22

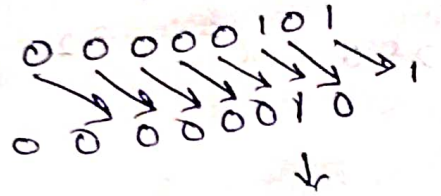
Logical right Shift Operator

Ex: 10



5// ⇒ Shifting one

Shifting 2//



formula:

$$\frac{N}{2^n}$$

N = given numbers.

n = binary shifting numbes.

Ex:

12 >> 2

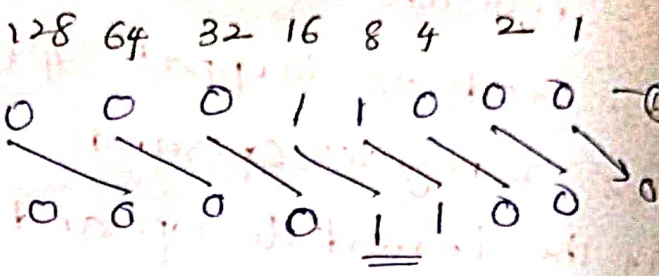
$$\frac{12}{2^2} = \frac{12}{4} = 3//$$

12 >> 1

$$\frac{12}{2^1} = \frac{12}{2} = 6//$$

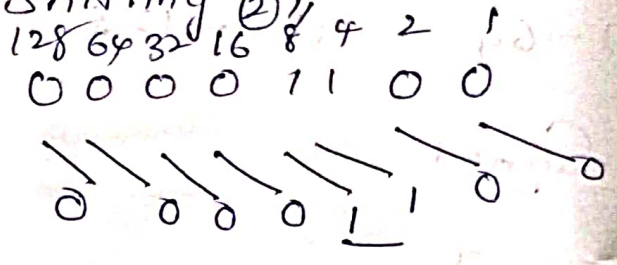
∴ In this logical right shift values will be decreased.

#24



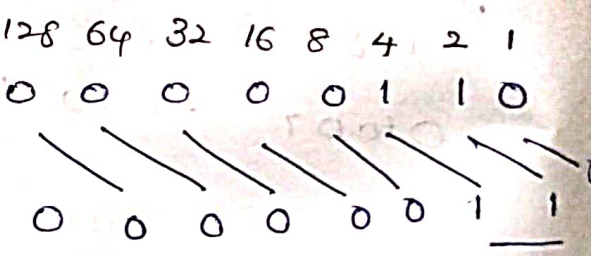
12//

Shifting 2//



6//

Shifting 3



3//

14 >> 1

$$\frac{14}{2^1} = \frac{14}{2} = 7//$$

14 >> 2

$$\frac{14}{4} = 3.5$$

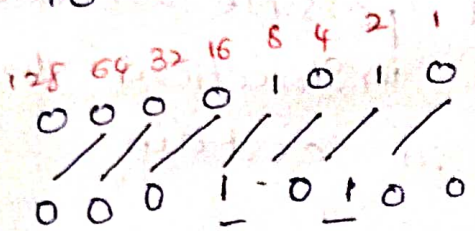
$= (3)$

Logical left shift Operator:-

10

=

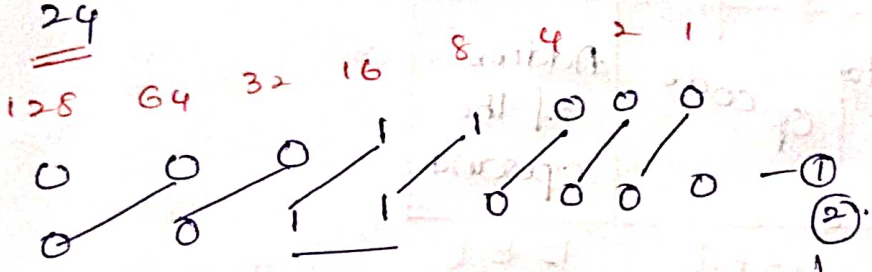
10 10



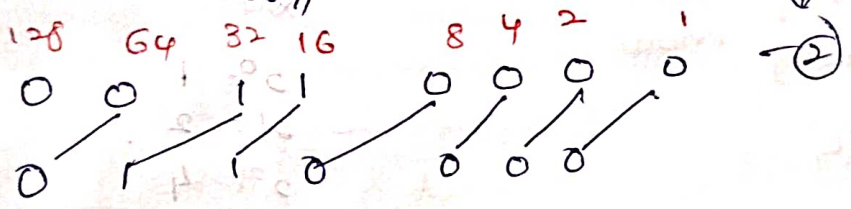
= 20 //

24

=



48 //



96 //

formula:

=

$N * 2^n$

N = given number

n = no of shifting

Ex: 24 << 2

=

$24 * 2^2$

24 * 4

= 96 //

Ex: 24 << 1

=

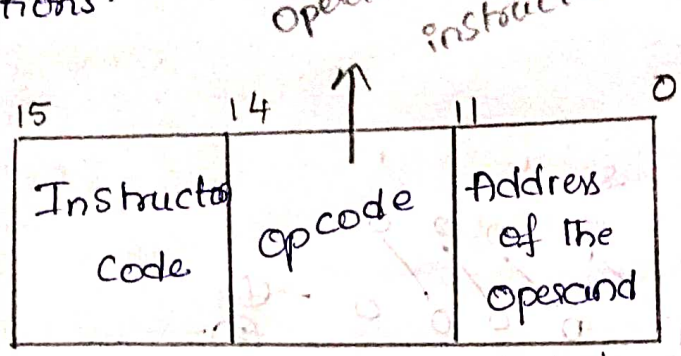
$24 * 2^1$

48 //

Instruction Set:

⇒ An Instruction Set is a group of commands for a CPU in machine language.

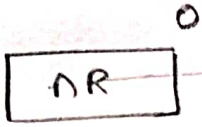
⇒ The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations.



↳ indirect addressing mode

1 → direct addressing mode

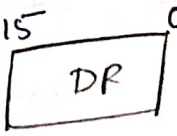
Memory 16 bits



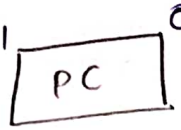
(address register)

2^{12} bits

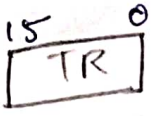
4096 × 16 "



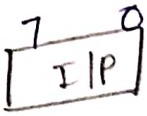
(Data register)



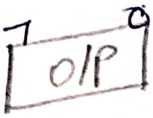
(program counter)



(Temporary register)



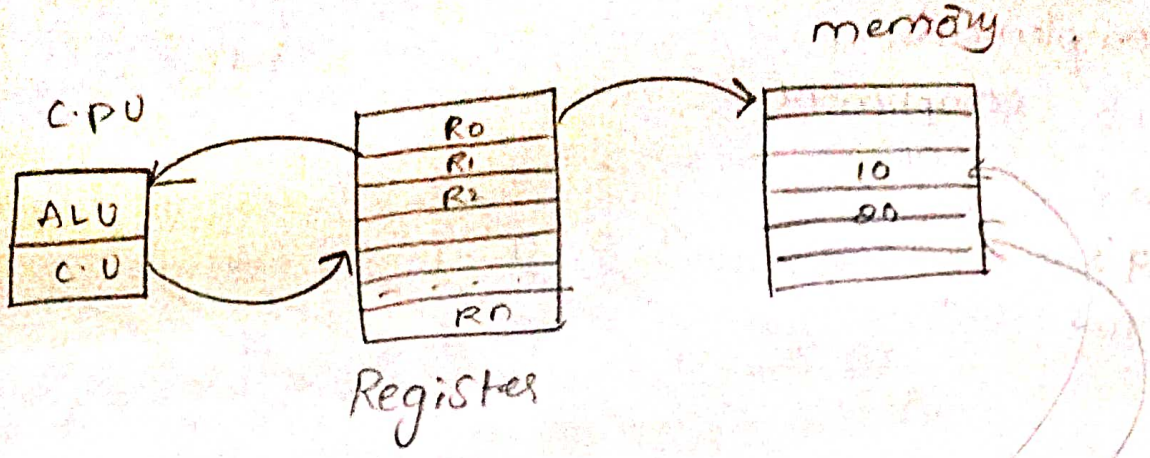
– 8 bits



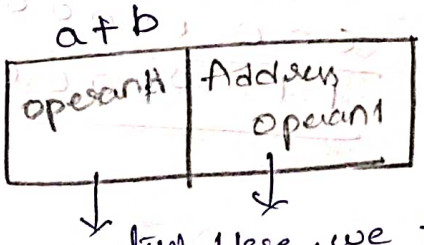
– 8 bits

(Instruction regist

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$
$2^{11} = 2048$
$2^{12} = 4096$
$2^{13} = 8192$




```
#include <Stdio.h>
Void main()
{
   int a=10;
   int b=20, c;
   c=a+b;
   printf("%d", c);
}
```



Instruction Here, we take a+B value.

16MB × 32

How many data lines & address line ?

$2^4 * 2^{20}$

$a^m a^n = a^{m+n}$

$2^4 \times 2^{20} = 2^{4+20}$

$= 2^{24}$

MB = 1024 × 1024

$= 2^{20}$

16 = 4 bits.

24 data lins & 24 address lins

Complements

1) 1's complement
2) 2's "
3) 9's "
4) 10's "

1's

11100111 } changing → 1 as 0 : 0 as 1 :
00011000

2's

adding 1 to the 1's complement.

```
  00011000
         1
 ----------
  00011001
```

trick

00011000

upto 0's same then chang complement

```
  00011000
 ----------
  11101000
```

Ex:

```
  00011000
  11100111   → 1's
      1111   2's
 ----------
  11101000
```

```
  00010100
  11101011   → 1's
      1100   2's
 ----------
  11101100
```

=>

```
  00010100
 ----------
  11101100
```

9's

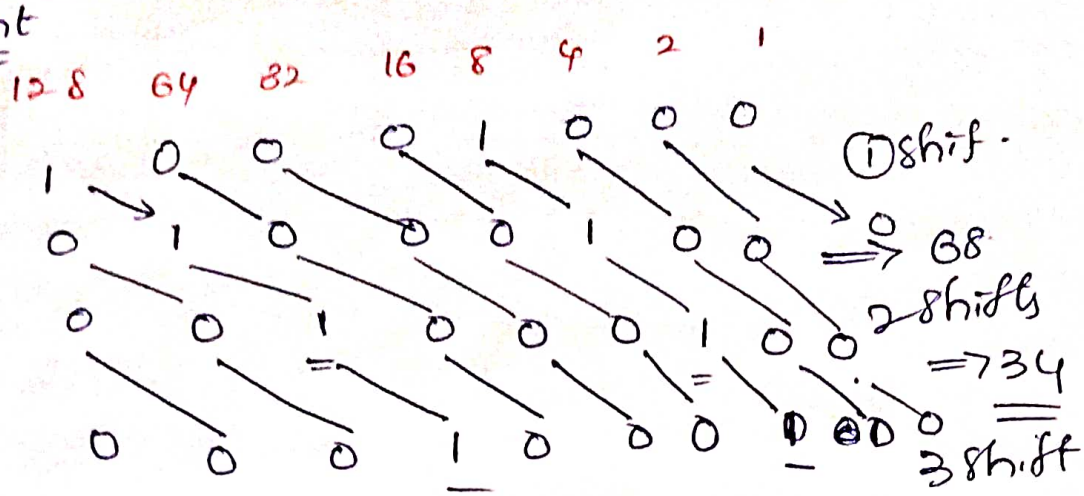
876 => 999 − 876 = 123

adding 1 → 10's: 123 + 1 = 124

11.40

136 - 3 Shift

right



16 + 1 = 17 //

left.