X86 Microprocessor Architecture & Toolchain.

↓

(8086)

Toolchain

X86

Date:

Editor:

File Name : Demo.c

```
#include<stdio.h>
#define MAX 10

int sum(int Num1 ,int Num2)
{

  int Ans;
  Ans = Num1+Num2+MAX
Printf("Addition of two numbers
          is %d", Ans);
  return Ans;
}
```

↓ 1

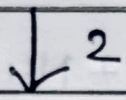
Preprocessor:

File Name : Demo.i

```
int printf(const chat * format,....);
int scanf(const chat *format,....);
```

Date:

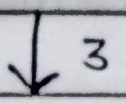
```
int sum(int Num1, int Num2)
{

Int Ans;
Ans = Num1 + Num2 + 10
Printf("Addition of two numbers
        is %d", Ans);
return Ans;
}
```



Compiler:

File Name: Demo.asm

```
Add: PUSH ECX PUSH EDX
     ADD ECX, 10
     MOV EAX, ECX
     RETN
```



Assembler:

File Name : Demo.obj

10101010101010101010
10101010101010101010

Date:

4

Linker:

file Name : Demo.exe

Demo.obj		other.obj
10101010 1010		10110100 101010
10101010 1010		10011110 011010
10 10 10 10 10 10	+	10 10 10 10 10 10 100
10 10 10 10 1010		1100 10 10 10 10 100

Demo.exe

100101101101110010

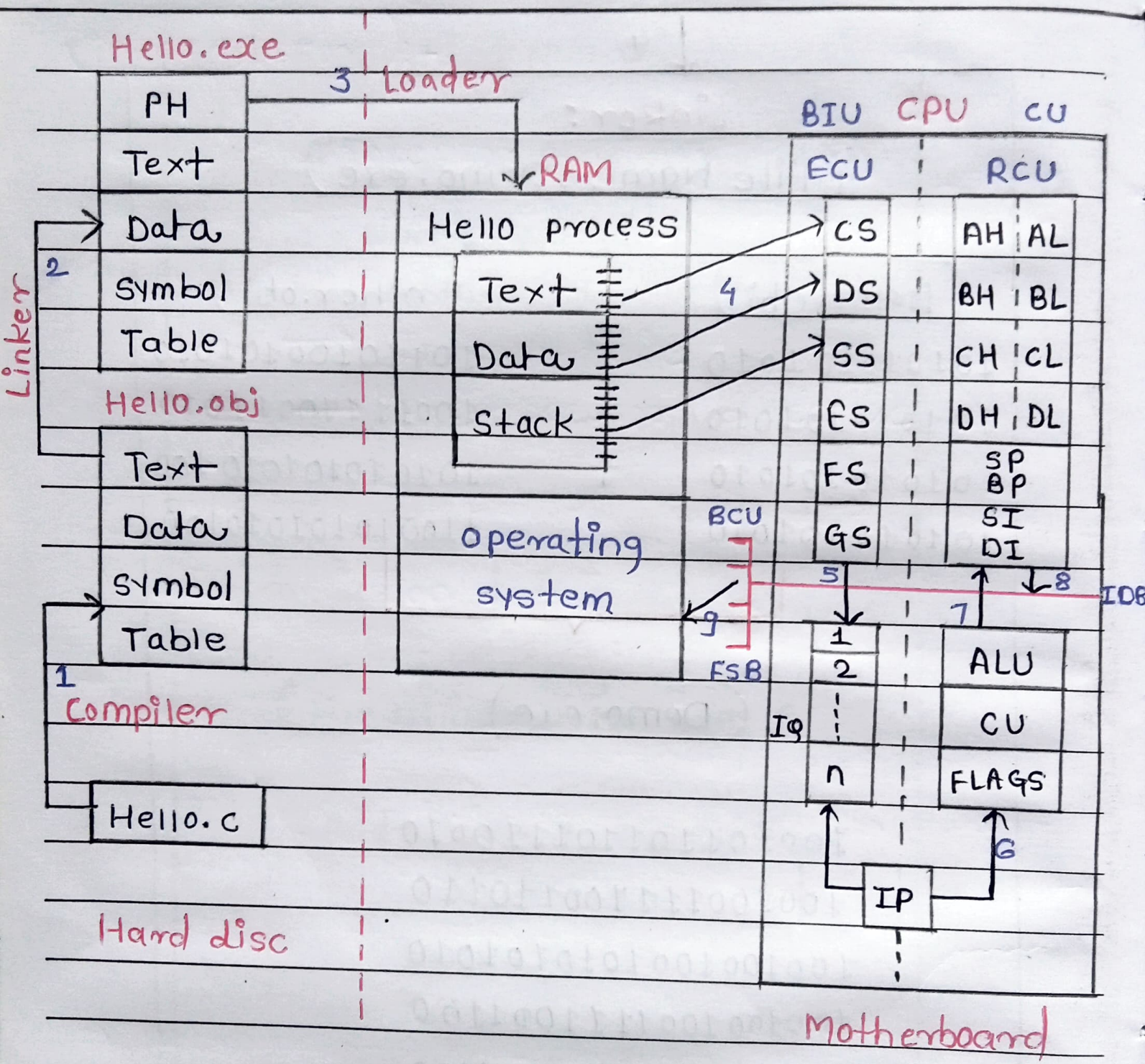
100100111100110110

100100100101010101010

100100100111100110 0

100100101001001001000

Date:

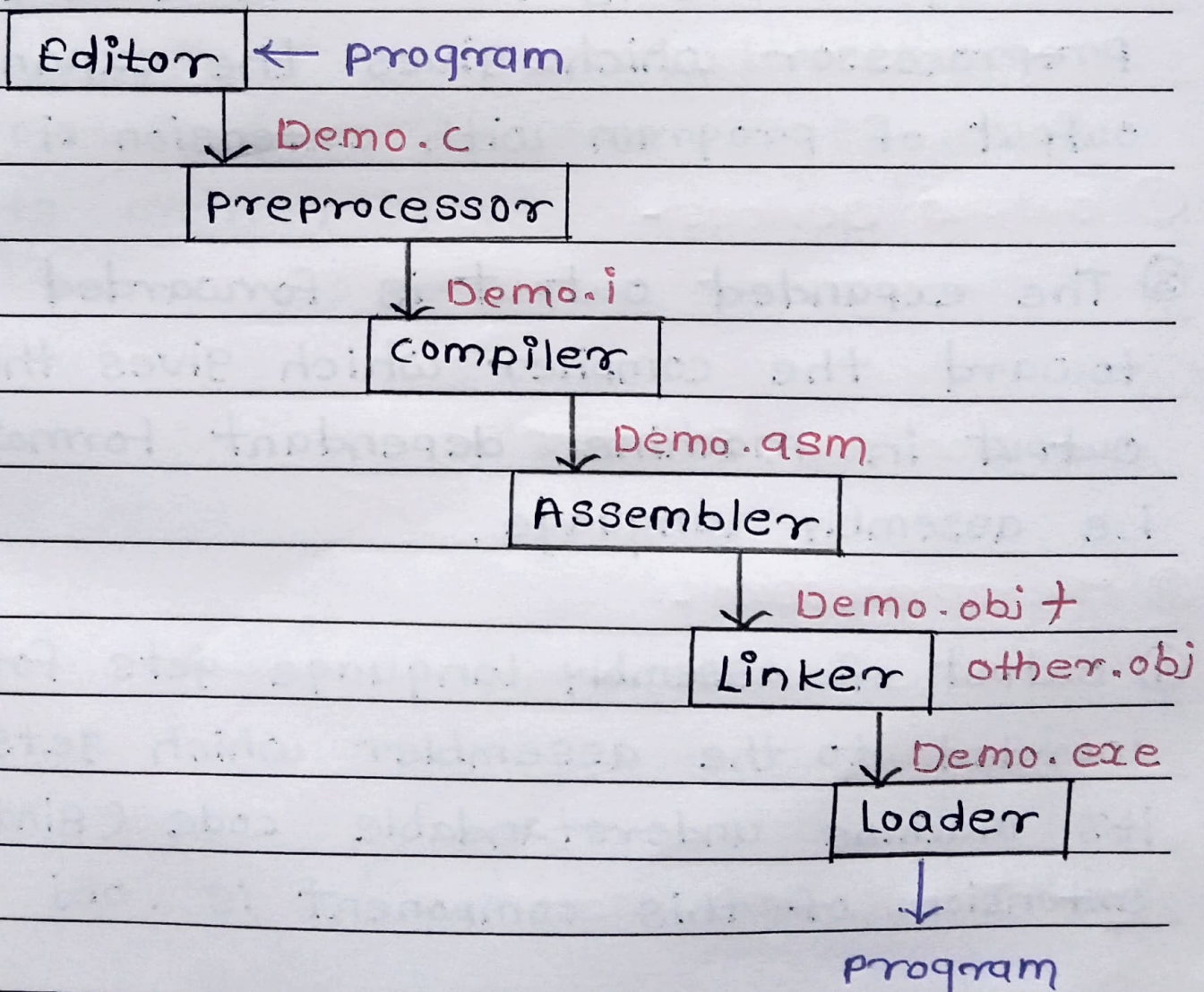

Hello.exe
PH
Text
Data
Symbol
Table
3 Loader
RAM
Hello process
Text
Data
Stack
2
Linker
Hello.obj
Text
Data
Symbol
Table
1
Compiler
Hello.c
Hard disc
operating system
BCU
FSB
BIU CPU CU
ECU
RCU
CS
DS
SS
ES
FS
GS
AH AL
BH BL
CH CL
DH DL
SP
BP
SI
DI
4
5
6
7
8
IDB
1
2
IQ
n
ALU
CU
FLAGS
IP
Motherboard

Date:

X-86 TOOLCHAIN

Toolchain is used to convert our program into an exicutable file. If toolchain output is 1, 2 is considered as input to the next.

Toolchain of c-programming



Date:

* Steps of X86 Toolchain -

① programmer writes a program on any editor (notepad or wordpad) and save that file with extension .c after file some operation it gets stored into the Hard Disk.

② c program file is forwarded to the preprocessor which gives the expanded output of program with extension .i

③ The expanded output is forwarded toward the compiler which gives the output in machine dependant format i.e assembly language.

④ output of assembly language gets forworded to the assembler which gets it's machine understandable code (Binary) extension of this component is .obj

Date:

⑤ The above .obj file is not complete file due to which linker links it with the other obj file

The output of linker is .exe file.

⑥ To any program it should be loaded in RAM. It is the responsibility of loader to load and exicutable file into memory (RAM)

⑦ Now program becomes process which gets written by as.

Date:

* Components of Computer -

Every computer is made up of multiple Hardware component's. Every computer contains some fix Hardware components as

① Input Devices -

It is used to acepts the input from the user

Ex - Keyboard, Mouse.

② output Devices -

It is used to display or genarate the output

Ex - Monitor, Printer.

③ Microprocessor -

The decision making task of computer is performed by processor. we can also called up as CPU

Ex - Intel, AMD, motorolla.

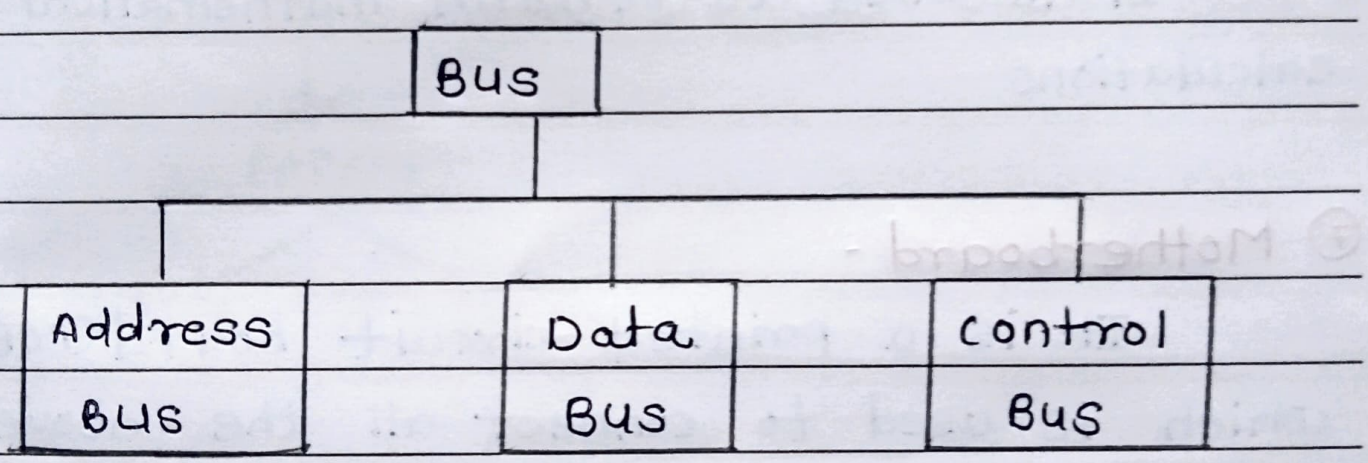
Date:

④ microcontroller -

Every Hardware component has it's own microcontroller which is used to communicate with the up.

⑤ Bus -

It is common medium which is required to connect and communicate between any two components of computer



i) Address Bus -

Address Bus describs the max RAM that we can connect

It also decides the size of processor.

ii) Data Bus -

Date:

Data bus is used to travel the data from storage device to the microprocessor.

iii) control BUS -

control BUS is used to send the control signals from one device to other device.

⑥ Mathcoprocessor - (ALU)

It is used to perform mathematical calculations

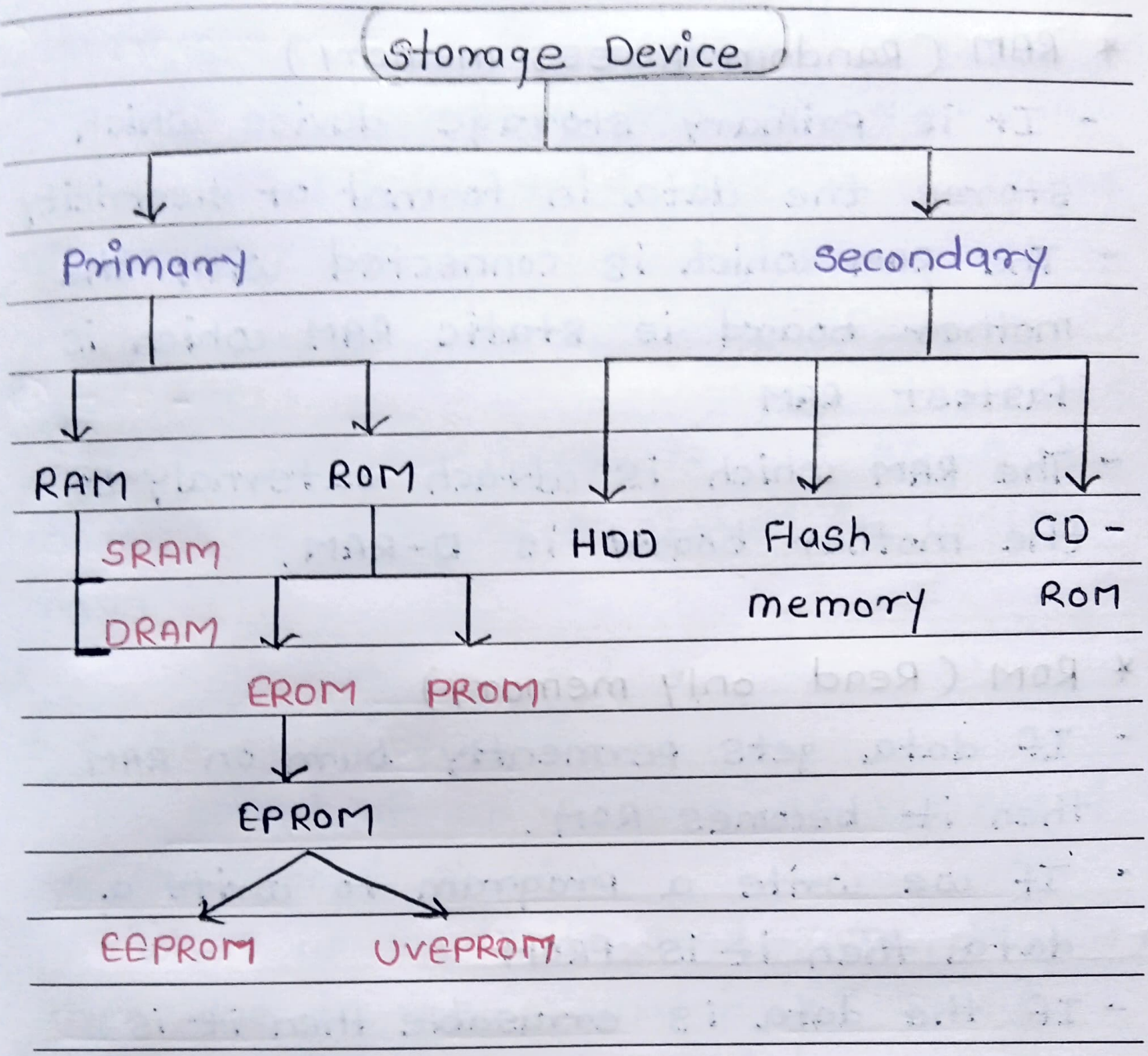
⑦ Motherboard -

It is a printed circuit board (PCB) which is used to connect all the above devices with each other.

⑧ storage devices -

while processing if you want to store any data then we can use the storage device.

Date:



- The storage device directly common with up is primary storage device
- The storage device which accepts data from primary storage device is secondary S.D.

Date:

* RAM (Random Access memory)

- It is primary storage device which stores the data in format of electricity
- The RAM which is connected with the mother board is static RAM which is fastest RAM
- The RAM which is attach externaly to the mother board is D-RAM.

* ROM (Read only memory)

- If data gets permently burn on RAM then it becomes ROM.
- If we write a program to write a data then it is PROM
- If the data is errasable then it is EROM.
- There are two ways in which we can errase the data i.e using electricity or using ultraviolet rays (EEPROM, UVEPROM).

Date:

* Hard Disk

It is secondary storage device in which the data get's stored in terms of magnetism.

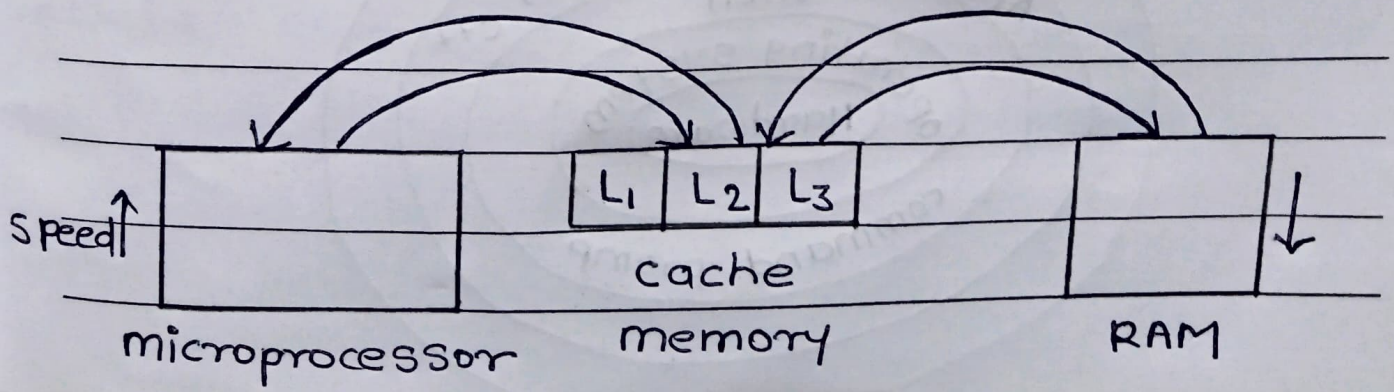
* CD-ROM-

It is secondary storage device where the data gets stored in optical rays.

* Cache memory-

Speed of up is more as compared to the speed of RAM

when up wants to send some data then due to the mismatch of speed there may be chance of data lost to avoid that up and RAM share some intermediate memory i.e cashe memory.



Date:

* operating system -

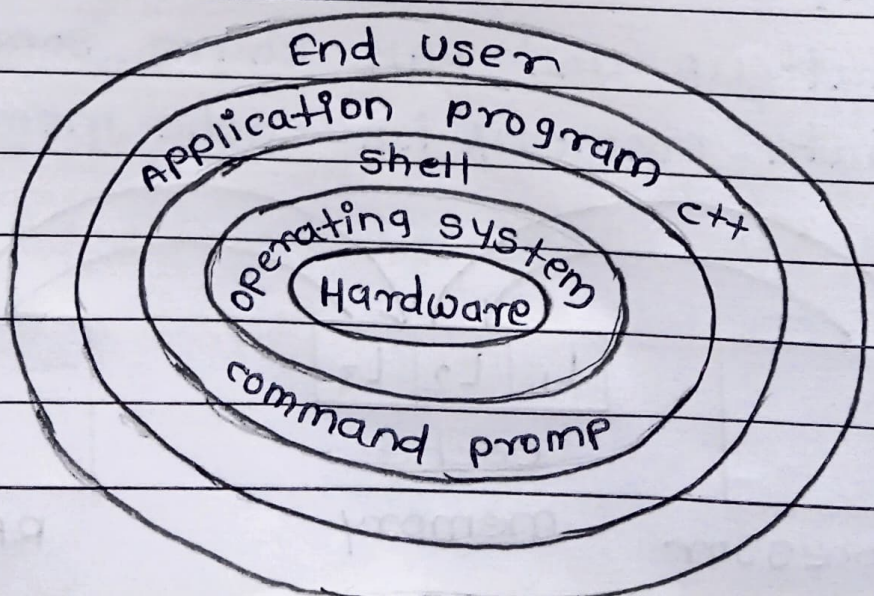
It is set of programming tools which are used to perform below tasks as

① File management
② Process management
③ memory management
④ CPU scheduling
⑤ H/w abstraction.

Any operations performs the above task The component of os which is used to manage the above task is called as kernal.

* Ring architecture of os -

Every os is divided into multiple layred format as.



Date:

- End users uses the application program which is written in any programming language i.e C, C++, Java, PHP.
- That program interacts with shell. Shell interacts with the o/s. That os performs 5 tasks.
- os interacts with Hardware and gives the apporiate results.

* Internal details of x-86 toolchain.

1) .obj file contains three parts as text, data, symbol table.

*Text - Text section is the section which contains the compile instructions of our program in binary format.

*Data - Data section contains memory called global variable used in the program

Data section is divided into two parts -

Date:

① BSS - (Block starting with Symbol)
- It contains memory for non initilize global variable.

② Non BSS - (Block starting with value)
- It contains memory for initilize global variable.

*Symbol Table -

It contains the information about the variable which are used in our program

Symbol Table hold's the metadata.

Symbol Table contains the fallowing entities as.

1) Name of Identities
2) size of variable
3) Address of that variable
4) value initilized with variable
5) Scope of variable.

After the linking activity .exe file gets generated which contains

Date:

primary header, text, data, symbol table.

* primary Header-

It is a header used by os to retrive the information about the executable file.

* Primary header contains -

1. magic number
2. Type of executable (self or Dependant)
3. Address of entry point function (main)
4. Time data staore.

when loader load executable file into RAM new section gets added which is called as stack.

* Stack -

It holds the information about the function that we called stack.

Date:

* CPU registers -

It is considered as storage of CPU which is used to store the temporary information about the instruction.

There are different types of CPU register as

① Segment Registers -

* Code segment - Hold part of text section.
* Data segment
* Stack segment
* Extra segment - If above three segment are full then we can use ES
* FS, GS - It is required if above segments are full.

② General purpose Registers (GPR) -

* EAX - Accumulator Register used for arithmatic operations.
* EBX - Base Register used to store the address.
* ECX - count register used to perform

Date:

counting

* EDX- Data register used to store the output.

③ pointer register-

* IP - Instruction pointer points to the currently running instruction.
* SP - stack pointer which points to end of stack frame.
* BP - Base pointer points to start of Stack frame

④ Index Register -

* SI - Source Index points to the starting point of instruction
* DI - Destination Index point to the end of instruction.

⑤ Flag Register -

This register contain multiple bit's where each bit has it's own meaning

Date:

* carry bit
* auxylary cary bit
* Parity bit
* sign bit
* zero bit

Date:

C - PROGRAMMING

* History -

1) In 1965 some of the member from MIT, GE, AT & T Bell laboretories decided to develope the operating system named as MULTICS

2) In 1968 due to some reasons the development of that os is paused.

3) In 1969 Dennis Ritchi joint AT & T Bell.

4) Inside AT & T Bell Dennis Ritchi, Ken Thomeson, Brian Kernighan to develope new operating system named as Unics

5) while developing this os Dennis Ritchi designed there own language named as C.

6) c language is influenced from the BCPL of ken Thompson (Basic combined

Date:

Programming Language.)

7) In 1972 The development of c language gets completed and unics os gets rewritten in c

* Standardization –

1) c programming is standrise in K & R standard which is developed by Dennis Ritche & Brian Kernighan.
This std. is introdused in 1978 (1982).

2) In 1988 (1989) the new standard gets introduced for c language named as ANSI (American National STD institute)

3) In 1990 ISO standrization is used for c (Internation orgnazation of standrization).

4) At the end c language is standrized by c99 standard in 1999.

Date:

* characteristics of C

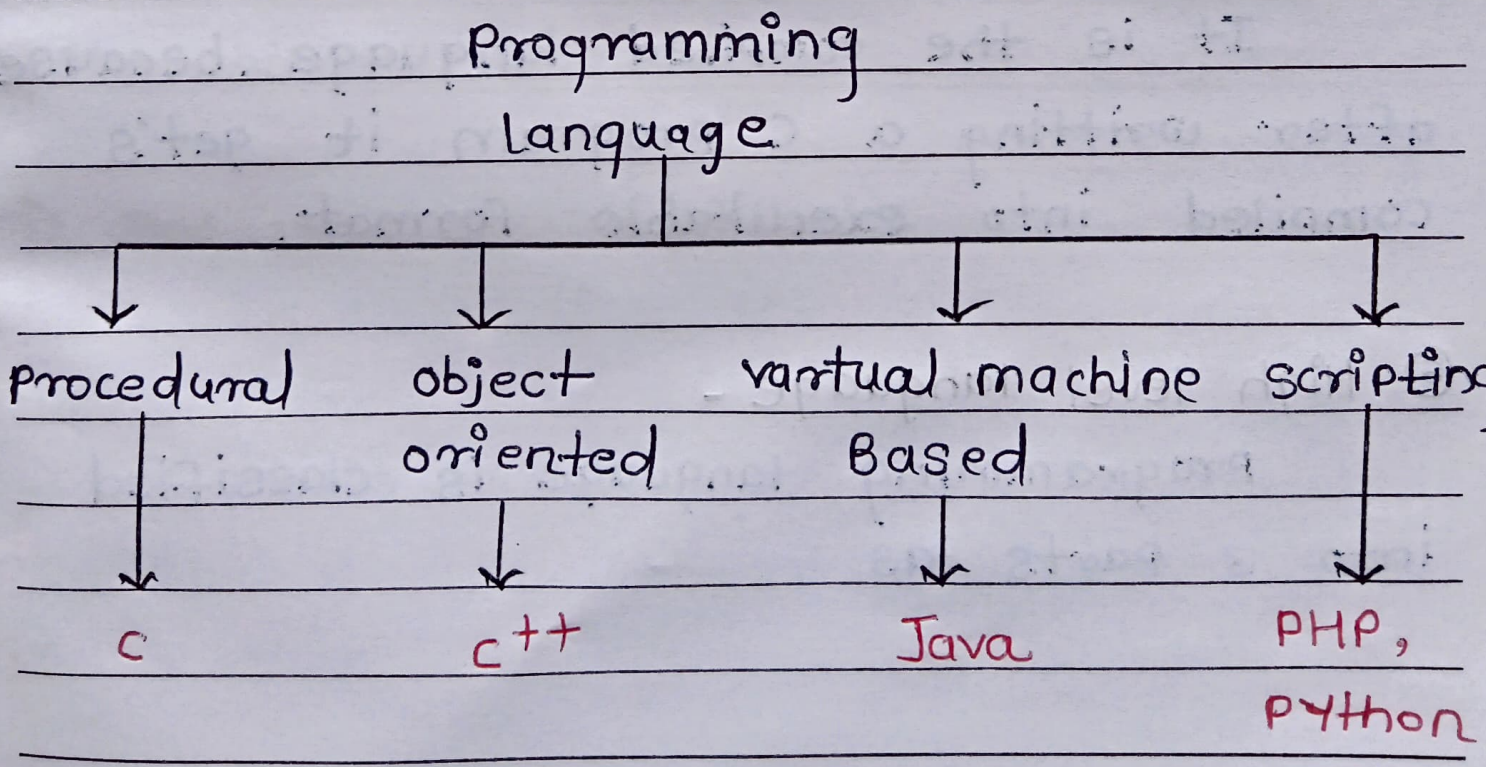
① General purpose language -

In c programming we can write application program, system program, mobile application, web based application.

As we can write any type of application due to which c is considered as general purpose language.

② procedural -

There are four types of language as



Date:

According to above digram . C is considerd as Procedural language.

③ Standardized -

C is standarized by K & R, ANSI, Iso, cgg

④ Case sensitive -

It is an case sensetive language. due to which capital and small case is considered as diffrent case.

⑤ Compiled -

It is the compiled language . because after writting a C program it get's compiled into executable format.

⑥ High level language -

Programming language is classified into 3 parts as

Date:

High level → C

↓

middle level → Assembly

↓

low level → Binary

↓

processor.

According to above digram c is considered as High level language after compilation it get's converted into assembly language after the task of assembler is completed it get's converted into the binary language.

⑦ Platform Dependant -

In case of c the program which is written in operating system dependant means executable file is generated on the x os. can be executed on the x os only.

Date:

⑧ Architecture dependant -

The c program which is written on the specific architecture only executed on the same architecture (processor)

⑨ Block stuctured -

In c language everything should be enclosed in a block. Block starts with { and ends with }

Date:

DATA TYPES

- Data types is a concept which indicates the size of memory that we can allocate and the set of operation's that we can perform on the that memory.

- Every language contains it's own set of data type.

- In c programming data types are classified into three parts.

① Premative Data type -

This are the data types which are provided by language designer that data types can be used directly in our application program.

② Derived Data type -

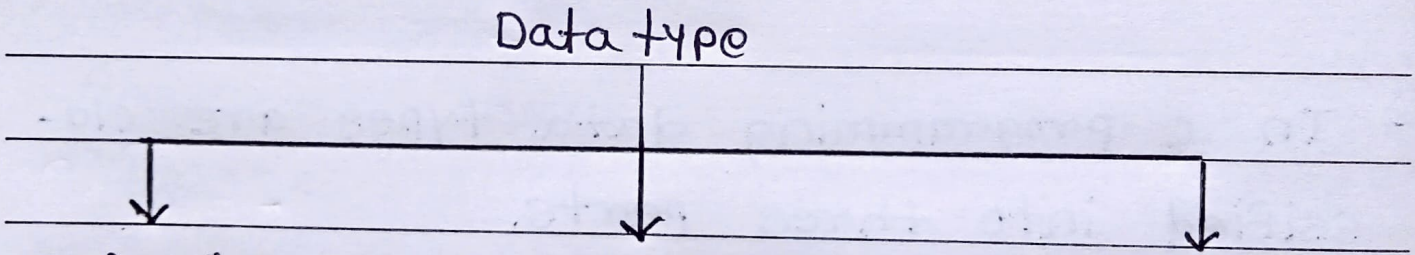
This are the data types which are generated from any existing data types.

Date:

③ User defined Data type -

Programmer is considered as the user of that programming language. depends on the programmer. if we decide the data type then it is user defined data type.

Under the above types of data types there are different inbuilt types as



Primitive	Derived	User defined
1. int [4]	1. Array	1. Structure
2. char [1]	2. Pointer	2. union
3. float [4]	3. function	3. enumaration
4. Double [8]		
5. void [-]		

① Premitive Data type -

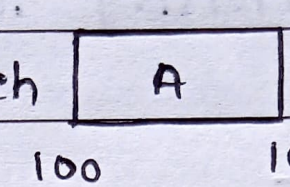
A) character - char

It is used to store single 1 bite

Date:

Value

Ex - char ch = 'A';



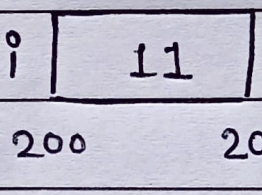
ch is a variable of type character [char] currently initilized with value A.

B) Integer - int

It is used to store integral constant

Ex -

int i = 11;



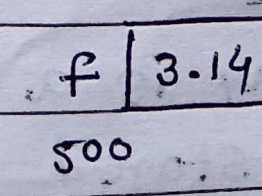
i is a variable of type integer currently initialized with value 11.

C) float -

It is store decimal values with less accuracy.

Ex -

float f = 3.14;



f is a variable which is of type float

Date:

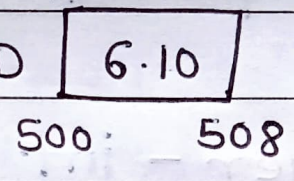
and currently initialized with value 3.14

D) Double -

used to store decimal values with more accuracy

Ex -

```
double D = 6.10;
```



D is a variable of type double initilized with value 6.10

e) void -

we can not create the variable of void but it is used in case of written value of function as well as type of pointer

② Derived Data type

A) Array -

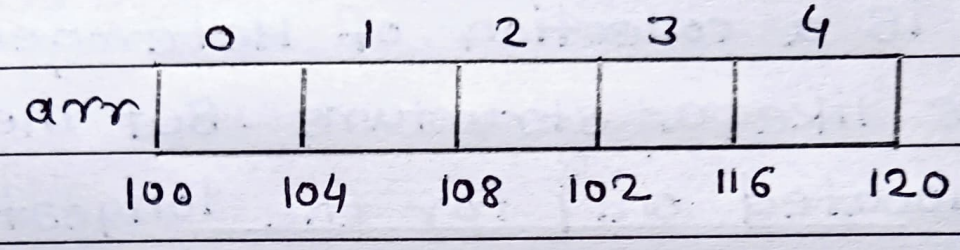
Array is a collection of Homogenious elements stored in indexed format

Ex -

Date:

int arr[5];

Arr is one dimentional array which contains 5 elements, each element is of type integer



B) pointer -

pointer is a variable which store address.

As pointer stores the address and address is integer, size of integer is four byte due to which size of each pointer is four byte.

C) function -

It is a set of related statements which are written under a single name.

③ User Defined Data type -

A) structure -

Date:

It is a collection of Heterogenious elements stored under Single name.

B) union -

It is a collection of Heterogenious elements like a structure. But memory gets allocated only for the longest data type.

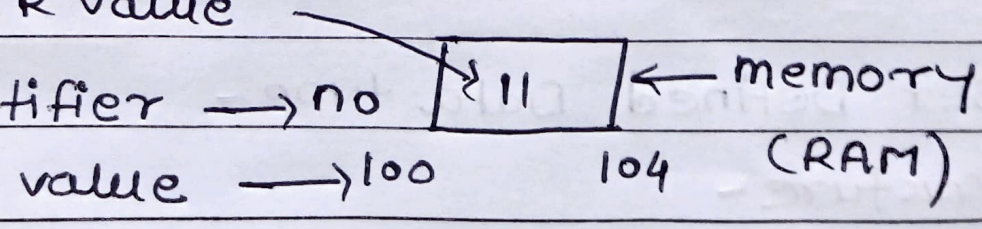
c) enumeration -

It is a collection of symbolic constants

* Data object

In a programming every variable is considered as an data objects for every data object the seprate memory gets allocated.

```
int i=11;
```



Date:

R value indicates the resident at the memory location L value indicates the location of that variable

for every data object there is it's own L value and R value.

* Data type qualifiers and modifiers.

1) Data type qulifier provides new quality to an existing variable

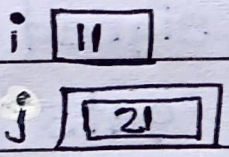
There are two qualifies as const and variable

const -

Due to the const. The value of the variable remains unchanged (constant)

Ex - ① int i = 11;

const int j = 21;



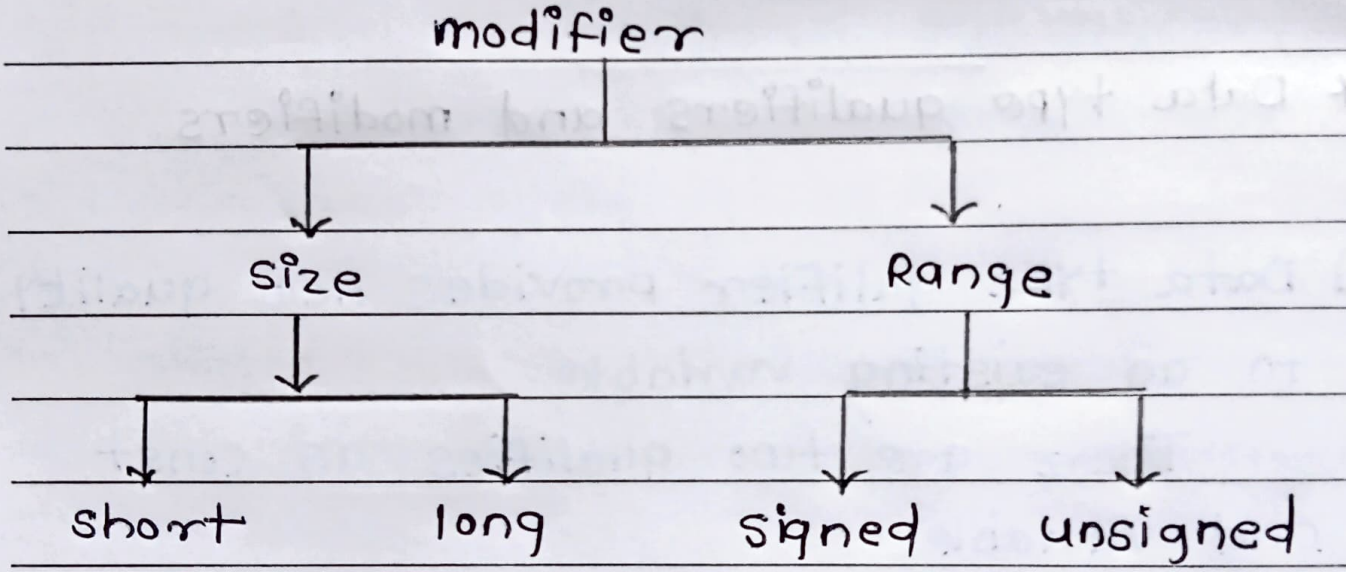
② i = 12;

j = 22; not allowed

Date:

2) Data type modifier-

By using the modifier we can either increase or decrease size, increase or decrease the range



- Due to short and long the size of our variable gets increased or decreased
- Due to signed or unsigned range of variable gets increased or decreased.
- Due to the unsigned variable we can use all the allocated bits where as in case of signed variable we can use all the bits except first bit.

Date:

* storage classes in c and c++

① The concept of storage class in language independant.

② In c, c++, Java the concept of storage class is almost similor.

③ There are four storage classes as
   i) Auto
   ii) register
   iii) Static
   iv) extern

④ Every storage class is classified based on five characteristics as
   ① where the memory gets allocated :-
   - memory gets allocated either in text, Data, stack section
   - In some cases memory gets allocated in CPU registers or Heap section.
   ② what is the default value of the variable
   - There are two default values as
   (A) Garbage (B) zero.

Date:

③ What is the scope of the variable -

- Scope of the variable indicates the region in which the variable is accessible.

There are three types of scope as local scope, global scope (file scope), program scope.

④ lifetime of the variable -

- Lifetime of variable indicate the span which gets calculated from the memory allocation time to memory deallocation time

There are two types of lifetime as i) Throughout the function.

ii) Throughout the program.

⑤ linkage of the variable -

- linkage of variable indicates the linking of variable capability of the variable

There are three types of linkage as

i) no linkage

ii) Internal linkage

iii) external linkage

Date:

- on the above five points classify the storage classes into four parts.
- while writting the program if we consider the storage class then it avoids the problem due to the segmentation fault or any other run time accidents.
- In c and c++ all the above four storage classes are applicable.
- But in Java there are only two storage classes as auto and static.

* first c program -

Demo.c ← program name.

```
line no
↓
1  # include<stdio.h>
2  # include<stdlib.h>
3
4  int add (int, int);
5
6  int main()
7  {
8      int x=0, y=0, ans=0;
```

Date:

```
9
10    printf("Enter first no");
11    scanf("%d", &x);
12
13    Printf("Enter second no");
14    scanf("%d", &y);
15
16    ans = add(x,y);
17
18    printf("Addition of two no is %d", ans);
19
20    return 0;
21    }
22
23    int add(int no1, int no2);
24    {
25      int ret=0;
26
27      ret = no1+no2;
28
29      return ret;
30    }
```

Date:

* Explanation of above program -

1, 2 - it is used to include the header files, the task of file inclusion is performed by preprocessor.

4 - it is a declaration of function add is a function which accepts two parameters, both are integers and it returns integer.

6- it is an entry point function of program Entry point function is a function from where the execution of program shorts. Inside primary header address of entry point gets stored.

8- x, y, ans are local variables of a function

10 - we can printf function to display Statement on monitor.

11 - scanf function is used to accept

Date:

The input from user. User can enter the input through the keyboard.

16- from this line we call the function named as add.
while calling the function we pass two arguments as x and y.

23- At this line we define the function. This function accepts the input into it's input argument named as no, and no2.

25 - ret is considered as the local variable of add function.

27- at this line we perform the addition of values from the variable no1 & no2 The result of addition gets stored into the variable ret

29- return keyword is used to return

Date:

the value from callee to caller (from add to main)

16 - The return value of add function gets stored into the local variable of main i.e ans.

20 - The return zero as the exit status of our program.
zero indicates success.

Date:

STORAGE CLASS

① auto storage class -

- It is a default storage class for the local variables of a function.

- If the storage class is not specified then it is by default considered as auto

A) memory Allocation - Stack
B) Default value - garbage
C) Scope - function scope
D) lifetime - throughout the function.
E) linkage - no linkage.

example - // Demo.c

```
1  # include < stdio.h >
2  # include <stdlib.h>
3
4  void fun(int);
5
```

Date:

```
6  int main()
7  {
8      int no=10;
9      auto int x=20;
10
11     fun (no);
12
13     return 0;
14 }
15
16 void fun(int value)
17 {
18 int i=11;
19 printf("%d", value);
20
21 }
```

② register storage class -

- for the fastest execution we can use the register storage class.
- In case of register storage class memory for the variable gets allocated

Date:

in CPU registers directly. Due to which it increase the speed of execution.

- register storage class is the request.
- If the CPU registers are available then it allocates the register otherwise it is bidefault considered as auto

① memory allocated at - CPU register
② Default value - garbage
③ Scope - fun^n scope
④ lifetime - throughout the function.
⑤ linkage - no linkage

Ex - // Demo.c

```
1  #include<stdio.h>
2    void fun()
3
4   int main()
5   {
6     int i=10;
```

Date:

```
7    auto int j=20;
8    register int k=30;
9
10   fun();
11   return 0;
12 }
```

- In above program there are two register storage classes as k and no.

- To build the above program we have to use below command.

gcc Demo.c -o myexe ↵

After this command executable file gets created named as myexe, Now we can run that executable file as

./myexe ↵

Internaly gcc command performs preprocessing, compilation, assembly,

Date:

linkage.

When we specify ./myexe loader loads the executable file and it becomes a process.

* Limitations of register storage class -

① register storage class is the request
② we can not store float, double into the register storage class.
③ we cannot creat array of the CPU registers.
④ If our CPU register is of 32 bit's then we can store the maximum 32 bit value in it.
⑤ we cannot fetch the address of CPU register.
⑥ we cannot apply pointer to register variable.

③ Static storage class -

① there are two types of static variables

Date:

as static local and static global

② generaly if you want to presure the value of local variable across the fun^n calls then we can use the concept of static variable.

consider the below program which demonstrates the problem due to the lack of static storage class.

Example - // Demo.c

```
1  #include<stdio.h>          // preprocessor
2
3  void fun();               // function declaration
4
5  int main()
6  {
7      printf("Inside main");
8      fun();                 // function call
9      fun();                 // function call
10     return 0;
11 }
```

Date:

```
12
13  void fun()                    // function defination
14  {
15
16      int i=10;                 // variable defination
17      i++;
18      printf("%d",i);
19  }
```

gcc Demo.c -o myexe

output :- Inside main
11
11

Explanation -

- when we run the above program the entry point function i.e main gets called implicity.
- from the main function we call fun function.
- Inside fun function there is a local variable

Date:

named as i which is initilized with 10.
- we increment the value of i by 1.
- when we call the fun function second time the seprate stack frame gets created which contains it's own local variable named as i.
- Again the value of i variable get initilize to 10.
- According to the auto storage class we cannot preserve the value of local variable
- To avoid this we can use static storage class.

④ extern storage class -
- extern storage class is used if we divide our program into multiple files.
- Due to extern storage class we declare the variable (declaration point of variable only indicates it's name and it's data type)
- But at the point of declaration there is no memory allocation.

Date:

Consider the below program which gets divided into two files named as Demo.c and Hello.c.

// Hello.c

```
1  # include <stdio.h>
2
3  int i = 10;                    // Defination
4  void fun()                     // Declaration
5  {
6      printf ("insidefun");
7  }
```

// Demo.c

```
1  #include <stdio.h>
2  extern int i;                  // Declaration
3  extern void fun();             // Declaration
4
5  int main ()                    // entry point function
6  {
```

Date:

```
7       printf("Inside main");
8       printf("%d", i);
9       fun();                    // function call
10      return 0;
11  }
```

```
gcc Demo.c Hell.c -o myexe
./myexe
```

output :- Inside main

10

Inside fun

① memory allocated - Data section

② Default value - 0, 0.0, \0

③ scope - Throughout the program

④ lifetime - Throughout the program

⑤ linkage - external.

In above program extern keyword indicates the declaration of variable

Date:

as well as function.

In Demo.c we declare the i variable and in Hell.c we define the i variable

* Declaration Vs Defination of variable -

① Declaration of variable indicates it's name, data type.

② At the point of declaration there is no memory allocation.

③ At point of defination name of variable is identified by compiler, it's data type is also decided as well as the memory gets allocated.

④ extern keyword is required to declare the variable.

* extern continued

① By using extern keyword we can declare variable as well as function. At the point of declaration there is no memory allocation but it just indicates about the

Date:

variable or function.

② If variable is defined globaly then its default storage class is extern.

③ If the variable is defined localy then it's default storage class is auto.

④ we cannot use two storage class as at a time. eg- auto register int i=10; //error

⑤ when we declare the variable using extern keyword then we cannot initilize it immediatly because at that point there is no memory allocatation

eg- extern int =10; // error

⑥ after declaration of extern we can initilize it seperatly.

eg- extern int i;

i=20; // allowed.

* Global Vs static Global -

consider the below program which demonstrate the concept of global and static global.

Date:

// Hello.c

```
int i=10;                    // non BSS Data
static int j=20;             // static segment
int k;                       // BSS Data
void fun()
{
  printf("Inside fun");
}
```

// Demo.c

```
#include<stdio.h>

extern int i;
extern int j;
extern int k;

extern void fun();

int main()
{
```

Date:

```
printf("%d", i);        // 10
printf("%d", j);        // error
printf("%d", k);        // 0
fun();                  // Inside fun
return 0;
}
```

```
gcc Demo.c Hello.C -o myexe
./myexe
```

- In above program there are three global variables as i, j and k. i is non-static initilized variable due to which it's memory gets allocated into non-BSS section (Data)

- j variable is global static variable due to which it's memory gets allocated in static segment.

- As our variable is static it's scope is throughout the file.

Date:

- Due to which we cannot acess that variable outside the file by using extern keyword.

- K variable is non-static global variable It is non-initilized due to which its memory gets allocated in BSS section (data)

- From the above program we calculate that non static global variable is accesible outside the file using extern keyword. But we cannot access static global variable outside the file.

* Symbol Table

① When we complile the program the symbol table gets created which holds the information about the identifiers used in program.

② When our executable file gets loaded

Date:

into the RAM then the symbol table gets removed but the new symbol table get's formed by the os which holds the metadata about the data, of our program.

③ symbol table contains below entities as

i) Name of identifier

ii) Its address

iii) It's size

iv) It'S Data type

v) It's another name

④ starting point of scope.

⑤ ending point of scope.