C

What is the benefit of having 32 and 16 bit platform?

The number of bits in a processor refers to the size of the data types that it handles and the size of its registry. A 64-bit processor is capable of storing 264 computational values, including memory addresses, which means it's able to access over four billion times as much physical memory than a 32-bit processor

---

Storage Class

A storage class defines the **storage, default value, scope (visibility) and life-time** of variables or functions within a C Program. They precede the type that they modify

- **Auto**-used by all other programs
- **Register**—when we want frequent operations and fast operations
- **static** --- when we want that value should to alive during functions call (local and global declarations)
- **Extern**- --used is required by the all functions, Global variable, extern **declaration does not allocate storage for variables.**

Why Register storage class is used

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size.

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions

---

Inline Function

Inline functions are a lot like a placeholder. Once you define an **inline function**, using the '**inline**' keyword, whenever you call that **function** the compiler will replace the **function** call with the actual code from the **function**. Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality

Inline int Max (int x, int y)

An inline function is a combination of macro & function. At the time of declaration or definition, function name is preceded by word inline.

When inline functions are used, the overhead of function call is eliminated. Instead, the executable statements of the function are copied at the place of each function call. This is done by the compiler.

Consider the following example :

```
#include <iostream>
inline int sqr(int x)
{
  int y;
  y = x * x;
  return y;
}
int main()
{
  int a =3, b;
  b = sqr(a);
  return 0;
}
```

Here, the statement b = sqr(a) is a function call to sqr(). But since we have declared it as inline, the **compiler replaces the statement with the executable statement of the function (b = a *a)**

What is the difference between inline functions and macros?
- **A macro is a fragment of code which has been given a name**. Whenever the name is used, it is replaced by the contents of the macro. There are two kinds of macros: Object-like macros and function-like macros.
- Inline function is a function that is expanded in line when the function is called. That is the compiler replaces the function call with the function code (similar to macros).

- The disadvantage of using macros is that the usual error checking does not occur during compilation

Advantages of using macro
Macros substitute a function call by the definition of that function. This saves execution time to a great extent

---

Difference between macro and function

No	Macro	Function
1	Macro is **Preprocessed**	Function is **Compiled**
2	Speed of Execution is **Faster**	Speed of Execution is **Slower**
3	Before Compilation macro name is replaced by macro value	During function call , Transfer of Control takes place
4	**Useful where small code appears many time**	**Useful where large code appears many time**
5	**Generally Macros do not extend beyond one line**	**Function can be of any number of lines**
6	**Macro does not Check Compile Errors**	**Function Checks Compile Errors**

Dynamic Memory Allocation (Malloc, calloc, realloc, free)

Function	Use of Function
malloc()	• **memory allocation --ptr=(int*)malloc(100*sizeof(int));** • **Allocates single block of memory** • **Return a pointer of type void, so type casting is done** • Does not initialize the memory allocated. • Takes one argument that is, **number of bytes/ (size of ().)** • malloc is faster than calloc
calloc()	• **contiguous allocation** -- **ptr=(cast-type*)calloc(n,element-size);** • **Calloc() allocates multiple blocks of memory each of same size** • **returns a pointer to memory** • **Initializes the allocated memory to ZERO.** • **Take two arguments :** number of blocks and size of each block (Depends on data type) • calloc takes little longer than malloc because of the extra step of initializing the allocated memory by zero
free()	dellocate the previously allocated space
realloc()	**Reallocation--ptr=realloc(ptr,newsize)** Change the size of previously allocated space If the previously allocated memory is insufficient or more than sufficient. Then, you can change memory size previously allocated using realloc().

Static functions in C
In C, functions are global by default. The "*static*" keyword before a function name makes it static. For example, below function *fun()* is static.
static int fun(void)
{

```
  printf("I am a static function ");
}
```

Unlike global functions in C**, access to static functions is restricted to the file** where they are declared.

Therefore, when we want to restrict access to functions, we make them static.
Another reason for making functions static can be **reuse of the same function name in other files.**

What is pointer and why accessing through the pointer is fast
Pointer is the variable that holds the address of the other variable or whose value is the address of another variable

Pointers have several uses, including:
- Creating fast and efficient code
- Providing a convenient means for **addressing** many types of problems
- Supporting **dynamic memory allocation**
- Making **expressions compact** and succinct
- Providing the ability **to pass data structures by pointer** without incurring a large overhead

Faster and more efficient code can be written because pointers are closer to the hardware. That is, the compiler can more easily translate the operation into machine code. There is not as much overhead associated with pointers as might be present with other operators**.**

Different types of Pointers in C Language

Wild pointer/Bad Pointer:
A pointer in c which has not been initialized with any variable address is known as wild pointer.
Wild pointer is also called Bad pointer, because without assigning any variable address, it points to the memory location

```
# include<stdio.h>
int main()
{
int *ptr;
printf("%u\n",ptr);
printf("%d",*ptr);
return 0;
}
```
Output:
Any address
Garbage value

Null Pointer:

Which pointer variable is initialized with null, it is called Null pointer
NULL pointer is a pointer which is pointing to nothing
When we are working with null pointer by default it doesn't points to any memory location
Datatype*ptr = NULL;------ **int*ptr = NULL;**
Datatype * ptr = (Datatype*)0;------- **int *ptr=(int *)0**;

Dangling pointer:

Pointer whose pointing object has been deleted is called dangling pointer.

If any pointer is pointing the memory address of any variable **but after some operations, variable has deleted from that memory location while pointer is still pointing such memory location**. Such pointer is known as dangling pointer and this problem is known as dangling pointer problem.

How to solve the problem of Dangling Pointer?
Solution of this problem: Make the variable x is as static variable

---

Generic Pointer/Void Pointer

Void pointer in c is known as generic pointer. Literal meaning of generic pointer is a pointer which can point, type of data. Generic pointer means it can access and manipulate any kind of variable data properly. when we are working with void pointer, **at the time of accessing the data, Mandatory to use type casting process** or else it gives errors, because type specification need to be decided at the time of execution

- The size of the void pointer is 2 bytes only
- When we are working with void pointers, **we can't apply airthematic operations**. i.e incrementation/decrementation of the pointer is restricted.

Example:
void *ptr;
Here ptr is generic pointer.

Important points about generic pointer in c? **We cannot dereference generic pointer**.

Near pointer :-
Which pointer variable can handle only one segment of 1MB data. It is called near pointer

- Near pointer will handles only 1 segment i.e data segment only
- The size of the near pointer is "**2**" bytes
- When we are increasing the near pointer value, then it increase offset address only.
- When we are applying the relational operations on near pointer, then it compare half set address only.

Far pointer :-
Which pointer variable can handle any segment of 1MB data, it is called far pointer.

- **When we are working with the far pointer, it can handle only one segment from the range of OXO to OXF.**
- The size of the far pointer is **4** bytes.
- When we are increasing the far pointer value, then it increases offset address only.
- When we are applying relational operators on for pointer, then it compare segment address along with offset address also.

Huge pointer:-
Which pointer variable can handle any segment of 1MB data it is called huge pointer

- When we are working with huge pointer, it can handle any segment of 1MB data from the range of 0X0 to 0XF
- The size of huge pointer is "**4**" bytes.
- When we are increasing the huge pointer value, then it increases offset address along with segment address also.
- When we are applying the relational operators on huge pointers, then it compares normalization value.
- Normalisation is a process of converting 32 bit physical address into 20 bit hexadecimal format

---

Pointer expressions: ptr++,*ptr++, ++*ptr and *++ptr (*ptr is pointing to value 10)

ptr++ means
Evaluate ptr, and then Increment ptr by 1 * base_type_size

***ptr++ means**
*ptr p jo value h phle usko print krega phr bad m pointer agle address p chla jayega--- 10(will print)

++*ptr means (value increment)
Ptr jo value point krega usko increment krega phle phr print krega ---11(will print)

***++ptr**
As we know that ++ has high priority than the *, so first pointer address will increment then it will point to the value at that point.

```
char *ptr = "sachin tendulkar";
printf("%c ",*ptr);          //it prints s
printf("%c ",*ptr++);        //it also prints s, after printing, the ptr variable is pointing to the next location,
printf("%c ",*ptr);          //it prints a, because in the previous statement we have incremented its address.
printf("%u ",ptr);           //prints the address of the letter 's' i.e the base address.
printf("%u ",ptr+1);         //prints the address of the letter 'a',in this we can clearly see that it jumps 1byte.
printf("%u ",ptr++);         //prints the base address and then only it gets incremented.
printf("%u ",ptr);           //previous line we have incremented the address so it will print the next address
```

Array
int a[10]; Declares and allocates an array of int(s)
Arrays are always pointers, but must be dereferenced (an extra level of indirection) if allocated dynamically.

Arrays of Pointers ---int *a[10]; -a ia an array of 10 integer pointer.
Declares and allocates an array of pointers to int. **Each element must be dereferenced individually**.

Pointer to an Array ---int (*a)[10]; -a is a pointer pointing to the array of 10 int.
Declares (without allocating) a pointer to an array of int(s). **The pointer to the array must be dereferenced to access the value of each element.**

int *(arr[8]);
arr is an array of 8 pointers to integers

Constant Pointers-- int * const ptr;
(pointer constant h ,ki vo ek hi address ko point krega so increment possible nhi hoga bhale hi value change ho jaye)---pointer will not vary i.e p=p+1----will not work and *p=M----is possible
A constant pointer is a pointer that **cannot change the address** its holding. In other words, **we can say that once a constant pointer points to a variable then it cannot point to any other variable.**

Pointer to Constant-- const int* ptr;
(pointer jo h vary kr skta h but jo ye value point kr rha h vo constant nature ka h so *p=M nhi chalega)
Pointer will vary i.e p=p+1----will work and *p=M----is not possible
As evident from the name, a pointer through which **one cannot change the value** of variable it points is known as a pointer to constant. **These types of pointers can change the address they point to but cannot change the value kept at those address.**

Constant Pointer to a Constant -- const int* const ptr;
(**esme pointer and value dono constant nature k h so koe v vary nhi krega**)
Pointer will not vary i.e p=p+1----will not work and *p=M----is not possible
A constant pointer to constant is a pointer that can neither change the address it's pointing to nor it can change the value kept at that address

Passing pointers to functions in C (Is that call by reference) - C programming allows passing a pointer to a function.

int *function_pointer(int*,int*) --function pointer/Function to a pointer- Function Returning pointers.
This is a function that goes by the name 'function_pointer'. **The function return value is a pointer that points to an integer (or the other data type which we want).** Functions that return a pointer are quite common. As an example, the function **malloc() allocates a block of memory and returns a pointer to that block of memory**.
Ex: int *add(int num1,int num2) {}
Function pointers in C is to call a function defined at run-time.

int (*f)(int,int) --pointer to a function
Pointer to a function is a function which in turn points to another function. The name of the pointer is 'f'. **But the function it points to could be any function that takes no parameters and returns an int.** The only place they are common is in API functions that take a call-back function
Ex: int add();
int (*addi)();
addi =&add;
then (*addi)() represents add();

How to work with the double and triple pointer
Single pointer is that having the address of the other variable.
Double pointer is that having the address of the first pointer. (in this we have to use ** to get the value)
Triple pointer is that having the address of the double pointer.

Pointer Size depends on?
Since a pointer is just a memory address its always the same size regardless of what the memory it points to contains. So a pointer to a float, a char or an int, are all the same size. Pointer is a memory address - and hence should be the same on a specific machine. 16 bit machine => 2 Bytes, 32 bit => 4 Bytes, 64 bit => 8 Bytes

Remember
Array of structure **struct employee r[3]**
Pointer to an structure **struct employee *r**
Pointer to an array of structure **struct employee (*r)[3]**
Array of pointer to an structure **struct employee *r[3]**
Pointer as Function parameter-- **void sorting (int *x, int y);**
Complicated Function Pointer -- void *(*foo) (int*);

How to access the elements of an array
(a+i)=(i+a)=i(a)=a(i)
((*(*(a+i)+j)+k)+l);
int *p;
p = arr;
or p = &arr[0]; //both the statements are equivalent.

How to store data using the address and the pointer
Int var=20;
Int *ip;
Ip=&var; ------- both will give the address
*ip----------will give the value at that address

---

a+1 is that we are going to increase the address
*(a+1)---phle address ++ then value
*a--1st value
*a+1----vo phli value hoga usme +1
**a

```
int main()
{
int ***r, **q, *p, i=8;
p = &i;
q = &p;
r = &q;
printf("%d, %d, %d\n", *p, **q, ***r);
return 0;
}
```

Define #pragma statements.

The #pragma Directives are used to turn ON or OFF certain features. They vary from compiler to compiler.
When we are going to compile the code, the part of the code written under the #pragma will not compile until unless we will define that
Examples of pragmas are:
#pragma startup // you can use this to execute a function at startup of a program
#pragma exit // you can use this to execute a function at exiting of a program

---

What is Memory Leak? How can we avoid?
Memory leak occurs when programmers **create a memory in heap and forget to delete it**.
Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate
To avoid memory leaks, memory allocated on heap should always be freed when no longer needed

---

Commomd line arguments
Maximum arguments count== it may vary from one operating system to another
Because command prompt depend on the Operating System (OS) and when OS change, then the value of the command line argument changes.

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments.** The command line arguments are handled using main() function arguments where **argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program**.

```
#include <stdio.h>
int main( int argc, char *argv[] )  {
  if( argc == 2 ) {
    printf("The argument supplied is %s\n", argv[1]);
  }
  else if( argc > 2 ) {
    printf("Too many arguments supplied.\n");
  }
  else {
    printf("One argument expected.\n");
```

}}

It should be noted that **argv[0] holds the name of the program itself** and **argv[1]** is a pointer to the first command line argument supplied, and *argv[n] is the last argument**. If no arguments are supplied, argc will be one, and if you pass one argument then argc is set at 2**

Difference between const & const volatile

Const:

When any variable has qualified with const keyword in declaration statement then it is not possible to assign any value or modify it by the program. But indirectly with the help of pointer its value can be changed. Its meaning is that value of variable can be changed after the declaration statement by the program.

```
#include<stdio.h>
int main(){
        const int a=5;
        a++;
        printf("%d",a);
        return 0;
}
```

Output: Compiler error, we cannot modify const variable

Volatile:-

The declaration of the variable as volatile tells the compiler that the variable can be modifies at any time externally to the implementation and every time the fresh value is updated. Complier can't perform the optimization on the variable .it doesn't go into the cache memory.

Proper Use of C's volatile Keyword

1. Memory-mapped peripheral registers
2. Global variables modified by an interrupt service routine
3. Global variables accessed by multiple tasks within a multi-threaded application

Const Volatile Qualifier:-

Const Volatile means that the program cannot modify the variable's value, but the value can be modified from the outside, thus no optimisations will be performed on the variable

Not volatile means --value of variable cannot be changed by any external device or hardware interrupt.

What is meaning of the declaration:
const volatile int a=6;
Answer:
Value of variable cannot be changed by program (due to const) but its value can be changed by external device or hardware interrupt (due to volatile)

Enum, Typedef, Typecasting, Bit operations

ENUM

An enumeration is a user-defined data type consists of integral constants and each integral constant is given a name. Keyword enum is used to define enumerated data type.

enum type_name{ value1, value2,...,valueN };
Here, type_name is the name of enumerated data type or tag. And value1, value2,....,valueN are values of type type_name.
By default, value1 will be equal to 0, value2 will be 1 and so on but, the programmer can change the default value

Typedef

typedef is a keyword used in C language to assign alternative names to existing types. It's mostly used with user defined data types, when names of data types get slightly complicated.
typedef existing_name alias_name
typedef unsigned long ulong;

The above statement defines a term **ulong** for an unsigned long type. Now this **ulong** identifier can be used to define unsigned long type variables.
ulong i, j

typedef vs #define

- **typedef** is limited to giving symbolic names to types only where as **#define** can be used to define alias for values as well, you can define 1 as ONE etc.
- **typedef** interpretation is performed by the **compiler** whereas **#define** statements are processed by the pre-**processor**.

Typecasting
Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.
(type_name) expression

Bit operations
C Bitwise Operators Examples – OR, AND, XOR, NOT, Left/Right **Shift**.
Bitwise operators are used to manipulate one or more **bits** from integral operands like char, int, short, long.
&&--logical and-----give that certain feature is on or off
&--Anding operations

Stack
A Stack is a **data structure** which is used to store data in a particular order and **uses LIFO operations**
Collections of plates

Queue
A queue is a data collection in which the items are kept in the order in which they were inserted, and the primary operations are enqueue (insert an item at the end) and dequeue (remove the item at the front).
Line

Union
A **union** is a special data type available in **C** that allows to store different data types in the same memory location.
You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

Structure
Structure is a user-defined data type in C which allows you to combine different data types to store a particular type of record.

- Union and structure in C are same in concepts, except allocating memory for their members.
- Structure allocates storage space for all its members separately.
- Union allocates one common storage space for all its members
- We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because; Union allocates one common storage space for all its members. Where**as** Structure allocates storage space for all its members separately.
- Many union variables can be created in a program and memory will be allocated for each union variable separately.

What is difference between . and -> in structure (both are the technique to access the data in structure)
. is the member of a structure-- The dot operator is applied to the actual object.

dot_access() only displays information of a Student and **not manipulates this**. So we used 'const' keyword with function parameter to prevent dot_access() function from modifying any information in 'stu' Student

-> is the member of a POINTED TO structure--The arrow operator is used with a pointer to an object
Just pointer to Student 'a' i.e. &a is copied to the pointer-to-Student '*stu' at called function arrow_access(). Just 8 bytes copied. And using this pointer, we accessed the original contents of Student 'a' and displayed information

---

Call back functions
Call back function technique is useful where you need to write a function that's able to perform several different functions at a point. This technique requires to pass a pointer-to-function to another routine, which calls back the user function to perform some task. **Function calls back user's function is called callback function---QSORT**

Data structure alignment
Data structure alignment is the way; data is arranged and accessed in computer memory. It consists of two separate but related issues: **data alignment and data structure padding**.
When a modern computer reads from or writes to a memory address, it will do this in word sized chunks (e.g. 4 byte chunks on a 32-bit system) or larger.

Data alignment means putting the data at a memory address equal to some multiple of the word size, which increases the system's performance due to the way the CPU handles memory

Structure padding
In order to align the data in memory, one or more meaningless bytes are inserted between memory addresses which are allocated for other structure members while memory allocation. This concept is called structure padding.

So structure alignments are necessary so minimize the padding

---

Declare
```
int func();
```

Defintion
```
int func();

int main()
{
   int x = func();
}

int func()
{
   return 2;
}
```

Union Inside a structure is possible, how ?
```
struct info1
{
char hobby[10];
int cardno;
};
struct info2
{
char vehno[10];
int dist;
```

```
};
union info
{
struct info1 a;
struct info2 b;
};
struct emp
{
char n[20];
char grade[4];
int age;
union info f;
};
struct emp e;
```

We can use a single structure for it but then it would lead to wastage of memory coz either hobby name & credit card no. or vehicle no. & distance from com. is used at a time. Both of them are never used simultaneously. So, here union inside structure can be used effectively

---

Segmentation Fault or access violation
A **segmentation fault** occurs when a program attempts to access a memory location that it is not allowed to access. For example attempting to write to a read-only location, or to overwrite part of the operating system.
A dangling pointer points to memory that has already been freed. The storage is no longer allocated. Trying to access it might cause a Segmentation fault

What is page fault and when does it occur?
When the page (data) requested by a program is not available in the memory, it is called as a page fault. This usually results in the application being shut down.
A page is a fixed length memory block used as a transferring unit between physical memory and an external storage.
A page fault occurs when a program accesses a page that has been mapped in address space, but has not been loaded in the physical memory

---

Buid Procedure ----Text editor(.c)->preprossesor(.i)->compiler(.asm)->assembler(.obj)->linker(.exe)->loader

How to pass globle variable to other file----Using **extern** Function

If static variable will be declared at golcal and local at a time with the same value then it will through an error---no

If main function will not be there then there will b an error or not----Yes it will throw an error at the execution time,because the execution starts from the main function.

Which is the another API except the free to free the memory
The ldap_memfree() API is used to free storage that is allocated by some of the LDAP APIs

---

Self-referential structure
A self-referential structure is one of the data structures which refer to the pointer to points to another structure of the same type. For example, a linked list is supposed to be a self-referential data structure. The next node of a node is being pointed, which is of the same struct type
Union can b a referential, really the same way as struct:

```
union toto {
  union toto* a;
  unsigned b;
};
```

as soon as the tag identifier toto is known to be a union type,union toto* is a pointer to an incomplete type

---

Little Endian, Big Endian

Endian-ness
It describes how a multi-byte data is represented by a computer system and is dictated by CPU architecture of the system.
Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory.
Big Endian: MSB stored at lowest address
Little Endian: LSB stored at lowest address

File operations

Opening a file -2 arguments(r,w,a,rb,wb,ab)—file name and the mod in which you want to open
Reading data from a file--2 arguments
Writing data to a file--2 arguments
Closing a file-- arguments

FILE *fp;
'FILE' is a kind of structure that holds information about the file such as file in use, current size and its location in memory etc.
'fp' is a pointer variable that holds the address of the structure FILE.

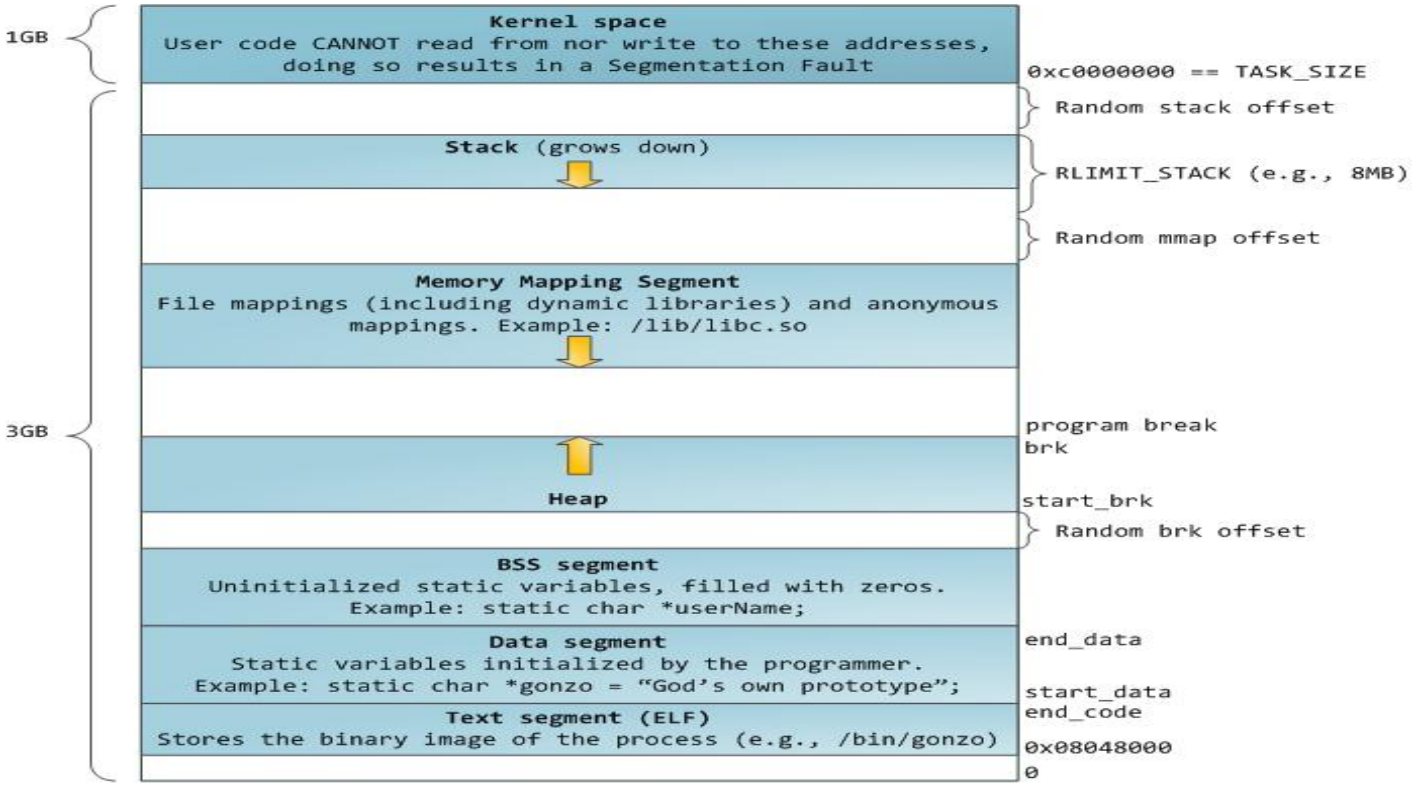
Difference between if, #ifdef
If will look for the condition that if that data is 1 then it will work.
And #ifdef will run if the Name or variable is defined using the MACRO

What is Difference between C and Embedded C?
Embedded C is a device in which programs are running but they are not computer. Example Washing machine

---

Why Unexpected Process Termination takes place and after then what we have to do.? What will happen if code will crash?
Memory leak is the reason

Memory Layout



Code (Text segment)
Data segment)

- Static & global data **initialized with nonzero.**
- Each running process has own data segment

Zero-initialized data

- Block Started by Symbol(BSS)
- Statically & global data **initialized with zero**
- When running, these are placed in the data segment

Heap dynamic memory comes; obtained by malloc ()
Stack segment is where local variables are allocated

Software Development Life Cycle (6 phase)

1. **Requirement analysis, leading to a specification of the problem**
   This is often done through pre-conditions and post-conditions.
2. **Design of a solution**
   Formulation of a method, that is, of a sequence of steps, to solve the problem.
   The design "language" can **be pseudo-code, flowcharts, natural language, any combinations** of those, etc.
   A design so expressed is called an algorithm(s).
3. **Implementation of the solution (coding)**
   Code Re-use
4. **Analysis of the solution**
   Estimation of how much **time and memory** an algorithm takes.
5. **Testing, debugging and integration**
   Syntactical correctness (no compiler errors)
   Testing a program for semantic correctness
   Gluing all the modules together to create a cohesive whole system
6. **Maintenance and evolution of the system.**
   Ongoing, on-the-job modifications and updates of the programs

---

DATA STRUCTURE:

Complexity:

Time complexity of an algorithm signifies the total time required by the program to run to completion. The time complexity of algorithms is most commonly expressed using the big O notation

What is hashing-

Hash table is a DS which store data in associate manner, in this data is stored in array format where each data values has its own unique index value and access of the data becomes very fast if we know the index of desired data.

Hashing is a technique to convert a range of key values into ranges of index of an array

Basic operation is---- Search, Insert, Delete

How does a linked list differ from stack and queue?

Linked list otherwise known as a one way list is a linear collection of data elements called nodes, where the linear order is given by a means of a pointer. Each node is divided into two parts, the first part contains the info about the element and the second part contains the address of the next node in the list and this is called link-field

We can delete and insert anywhere.

Queue is a linear list of element in which deletion can take place at one end called front and insertion can take place only at the other end called rear. It has a phenomenon called FIFO (first in first out)

Stack is a linear structure in which data items maybe added or removed only at one end. The phenomenon is called LIFO (last in first out). It has 2 major operations **push**, which is used to insert an element into a stack, **pop** used to delete an element from a stack

---

Searching Algorithms on Array

An **algorithm** is a step-by-step procedure or method for solving a problem by a computer in a given number of steps. The steps of an algorithm may include repetition depending upon the problem for which the algorithm is being developed. The algorithm is written in human readable and understandable form. To search an element in a given array, it can be done in two ways linear search and Binary search.

Linear Search

A linear search is the basic and simple search algorithm. A linear search searches an element or value from an array till the desired element or value is not found and it searches in a sequence order. **It compares the element with all the other elements given in the list and if the element is matched it returns the value index else it return -1. Linear Search is applied on the unsorted list when there are fewer elements in a list.**

Example with Implementation

To search the element 5 it will go step by step in a sequence order.

```
Function findIndex(values, target)
 {
  for(var i = 0; i < values.length; ++i)
    {
      if (values[i] == target)
        {
          return i;
        }
    }
  return -1;
 }
//call the function findIndex with array and number to be searched
findIndex([ 8 , 2 , 6 , 3 , 5 ] , 5) ;
```

Binary Search

Binary Search is applied on the sorted array or list and is useful when there are large numbers of elements in an array. In binary search, we first compare the value with the elements in the middle position of the array. If the value

is matched, then we return the value. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array

---

Example with Implementation

```
Function findIndex(values, target)
{
  return binarySearch(values, target, 0, values.length - 1);
};

function binarySearch(values, target, start, end) {
  if (start > end) { return -1; } //does not exist

  var middle = Math.floor((start + end) / 2);
  var value = values[middle];

  if (value > target) { return binarySearch(values, target, start, middle-1); }
  if (value < target) { return binarySearch(values, target, middle+1, end); }
  return middle; //found!
}
```

Remember about the binary tree traversal, insert, delete node techniques and the infix to postfix and vice-versa
LNR (infix), NLR (prefix), LRN (Postfix)

What is tree and Type of tree?
A tree is a widely used abstract data type (ADT)—or data structure implementing this ADT—that simulates a hierarchical tree structure, with a root value and sub-trees of children with a parent node, represented as a set of linked nodes

---

MICRO CONTROLLER:

What is microcontroller and how to select the right microcontroller?
Microcontroller is a small computer build on single integrated circuit having RAM and flash (code burn)
Has memory and input output devices----solid state technology (semiconductor)-satellite missile radar
So mostly used as a embedded system

Read input devices, Process data or information, Control output devices, Refrigerator, home automations
Integration level, Lower cast, Increased reliability—don't require external circuit, Space saving

Selections of the microcontroller is depends on the RAM, ROOM and the selection of the GPIO and other like power consumption, lower cast, easily availability

Which is better language?
Assembly language is preferred for simple application but for complex c language are preferred.
Microcontroller understands the machine level language so we have to convert it to machine level language to work with the microcontrollers.

```
#include<avr/io.h>
#include<util/delay.h>
#include<avr/interrupt.h>
#define F_CPU 16000000UL
```

Borad specifivction and details which I had used

AVR-Atmega 64
High-performance, Low-power,8bit micro-controller, RISC Architecture, 32 x 8 General Purpose Working Registers + Peripheral Control Registers, 128K Bytes of In-System Reprogrammable Flash, 4K Bytes EEPROM, JTAG,7Ports, 2.7 - 5.5V operating Voltage

ARM-STM32F437VGTx
Based on high-performance ARM Cortex®-M4 32-bit RISC core operating at a frequency of up to 168 MHz
Best utilization with peripherals and can connect more than 3 at a time (I2C/SPI/UART)
Full compatibility, DMA and SVC features, less power and GPIO register are there, NVIC and arm features
RTC and Watch dog Timer, ADC and DAC

Project-CC3200
1.8 volt operating voltage
Inbuilt ADC and ARM features.

Ardunio vs AVR
Ardunio is better for the beginner and this has already IDE in itself and having boot loader inbuilt
Disadvantage is that we can't use at the high level
AVR GCC is some specific to the point and different from that, it is more specific to the high level

What is Difference between RAM and ROM and what is Flash?
Flash is a kind of ROM used for the fasted access of the data
Where erase can be taken in a very fast manner and same the things

What is difference between Flash and EEPROM
Both of them are storage media
Flash user NAND gate which is cheaper than **NOR (used by the EEPROM)** and having lower cost and slower processing and this is portable and mostly used in phones
Flash only do the block wise erase or access the data while **EEPROM can access and erase the data byte-wise or a byte at a time**

Difference between Processor and Controller

• **Microprocessor is an IC which has only the CPU inside them** i.e. only the processing powers such as Intel's Pentium 1,2,3,4, core 2 duo, i3, i5 etc. These microprocessors don't have RAM, ROM, and other peripheral on the chip. **A system designer has to add them externally to make them functional.** • Application of microprocessor includes Desktop PC's, Laptops, notepads etc • **Microprocessor find applications where tasks are unspecific like developing software, games, websites, photo editing, creating documents etc. In such cases the relationship between input and output is not defined**	**Microcontroller = Processor + Peripherals** • **Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip**. **At times it is also termed as a mini computer or a computer on a single chip** • ATMEL, Microchip, TI, Freescale, Philips, Motorola etc • **Microcontrollers are designed to perform specific tasks. Specific means applications where the relationship of input and output is defined**

8051/AT89s52	PIC	AVR	ARM

Hardvard and CICS GPMC-8bit ALU 8 bit Little Indian 2 stage pipeline execute and fetch Accumulator-register and register-memory 16 bit address bus and 8 bit data bus 4kbyte of program memory 128byte of chip RAM 32 bidirectional i/o lines full duplex uart six interrupt resource **AT89C52-8kbyte of flash,256 byte internal RAM,0-24 MHZ operation**	Programmable/peripheral interface controller Harvard and risc and orthogonal Separate address and data lines easily interface with others **W register and stack memory** has its own space than others microcontrollers They don't have stack pointer register like others Used pipeline technique as others and 16 and 32 bit are used **FDE** **FDE** **FDE**	• 8-bit architecture. • **Better than 8051 and PIC** • Cheap, large number of library files, used in many robotic applications. • Best for the beginners. • 8bit-32 • 1clock/instruction cycle • **RISC** • **Average cost** • **Cheap and effective** • **Atmega 64 128** • 8 is start early • Narrow data pipe line • Timer can process upto 255 for every cycle	• ARM has a 16 and/or 32 bit architecture. • ARM- if u need fast computing, large number of timer and ADC's then ARM will be suitable. • 32 bit-64 • 1clock/instruction cycle • **RISC** • **Low cost** • **High speed** • **Cortex arm7tdmi** • 16 is developed on 8 bit • wide data pipeline • It can process upto 65535

Princeton (VonNeumann) Vs Harvard

• **Single Main Memory holding both program and data** • **Same address & data line** • **Many addressing modes and long instructions** • **High code density** • **Comparatively slower in execution (pipelining not possible)** • **Example- PC , 8085,68K series,ARM7**	• **Contains 2 separate memory holding both program and data** • **Different address & data lines - code & data** • **Compact, uniform instructions** • **More lines of code- large memory footprint** • **Faster in Execution-Facilitate pipelining** • **Example - Microcontroller ,ARM9,PIC,DSP processors**

Difference between CISC, RISC

• Variable length instruction • Several formats of instructions • Memory values can be used as operands in instructions • Small Register Bank • Cannot pipeline instructions • Multi cycle execution on instruction • Better code density	• Fixed width instructions • Few formats of instructions • Load/Store Architecture • Large Register bank • Instructions can be pipelined • Single cycle execution of instruction • Poor code density

---

Simulator vs Emulator

Simulator	Emulator
• Simulator is a host based symbolic debugger • It is not a real time debugging as the target is not connected • Very useful in checking the functional correctness of the programs	• An emulator based debugging is perfect debugging. • It is real time debugging and there is a separate emulation hardware • Emulator based debugging is Non Intrusive and transparent debugging • None of the microcontrollers resources are lost due to emulation

---

IO Mapped IO Technique and Memory Mapped IO Technique

IO Mapped IO Technique	Memory Mapped IO Technique
• IO devices are directly interfaced to the I/O space of the processor • Separate instruction set available for accessing the I/O (Like IN & OUT) • This does not use memory related instruction • IO mapped IO is one where the processor and the IO device have different memory located to each other.	• IO devices are interfaced in the memory space of the processor • The memory space is the RAM space of the processor • The memory space could be internal RAM or External RAM • Internal RAM memory mapping of I/O devices is defined by the processor • Memory mapped IO is one where the processor and the IO device share the same memory location

---

In-System Programming and In-Application Programming

In-System Programming	In-Application Programming
• **In-System Programming** means that the device can be programmed in the circuit by using a utility such as the ULINK USB-JTAG Adapter. • **The Flash device can be reprogrammed via UART/Parallel interface without being removed from the target** • **In this case the processor is put in a special boot or programming mode** • **The application needs to stopped during the reprogramming process**	• **In-Application Programming** means that the application itself can re-program the on-chip Flash ROM. • **User can program the Flash from an application running in RAM** • **Done by calling the functions in the boot loader area by the application code** • **It does the required operation on the flash memory**

AVR Architecture (Atmel 128)

AVR 8-Bit RISC High Performance

- True single cycle execution
  - single-clock-cycle-per-instruction execution
  - PIC microcontrollers take 4 clock cycles per instruction
- One MIPS (million instructions per second) per MHz
  - up to 20 MHz clock
- 32 general purpose registers
  - provide flexibility and performance when using high level languages
  - prevents access to RAM
- Harvard architecture

  Separate bus for program and data memory

1.8 to 5.5v
Program counter and status register (is going to affect if the any operation is going to happen)
Tiny, mega (8 bit mostly) and x mega (USB Ethernet etc)

- **8 status registers (SREG/Flags)**
  - C, Carry flag
  - Z, Zero flag
  - N, Negative Flag
  - V, Overflow flag
  - S, Sign bit
  - H, half carry flag
  - T, transfer bit
  - Global interrupt enable

Addressing Modes
Direct Single Register ----LDI R19, 0x25
Direct Program Addressing Mode---LDS R19, 0x560
Register Indirect Addressing Mode-- LD R24, X
Program Memory Constant Addressing Mode—LDI ZH, 0x00
Direct Program Addressing Mode--JMP and CALL
Indirect Program Addressing Mode--IJMP, ICALL
Relative Program Addressing Mode--RJMP, RCALL

DDRX- Sets whether a pin is Input or Output of PORTX
PORTX- Sets the Output Value of PORTX
PINX- Reads the Value of PORTX

Cross Compiler

- Runs on a machine based on one type of CPU and produces machine instructions for a different kind of CPU
- Allows execution at development time
- Possible to target multiple instruction sets

Host - Target Development Environment

- The distinguishing feature of embedded software development is host-target development environment
- All the development tools like Editors, compilers and linkers are available on the host machine
- Typical host machines are Windows 95/98, NT and Unix workstations where the above development tools are available
- Application programs will be written on the host, get compiled, linked and get the executable file
- **Target systems are the ones where compiled and linked code is executed**
- **Target systems being a microprocessor based boards does not offer any development environment themselves, so an host machine is required**

Watchdog Timer

- High end applications require multiple or critical calculations to be done on the microcontroller
- This may lead to cases when the controller enters into wrong or infinite loops resulting in system crash
- The solution to overcome these situations is to automatically reset the system whenever such a situation arises
- Watchdog Timer is a hardware or software generated timer interrupt which reboots/resets the system in the situations mentioned above
- **Is a special timer which can be enabled in any section of the code and when enabled it ensures that a certain number of instructions execute within a pre-defined time frame**

What is Timer and Counter and we can generate the one sec delay?

Timer	Counter
• **A timer is a specialized type of clock which is used to measure time intervals**. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer • The register incremented for every machine cycle. • Maximum count rate is 1/12 of the oscillator frequency. • A timer uses the frequency of the internal clock and generates delay.	• **A counter is a device that stores (sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal**. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop • The register is incremented considering 1 to 0 transitions at its corresponding to an external input pin (T0, T1). • Maximum count rate is 1/24 of the oscillator frequency. • A counter uses an external signal to count pulses.

Different type of Timer is there.
Timer 0 and Timer 2 (TOV, TCNT, OCR, TCCR, OCF)—8 bit
Timer 1 and Timer 3--- 16 bit

One sec Delay:
For auto reload: (FF-initial value)*64 microsec--- TCNT i.e initial values will be zero always (FF- OCR value)
Non auto reload: [(FF-initial value) +1]*64 microsec---TCNT i.e initial value will be zero after one cycle

---

What does the stack pointer and program counter?

SP, the Stack Pointer
Register R13 is used as a pointer to the active stack.
In Thumb code, most instructions cannot access SP. The only instructions that can access SP are those designed to use SP as a stack pointer. The use of SP for any purpose other than as a stack pointer is deprecated. Note Using SP for any purpose other than as a stack pointer is likely to break the requirements of operating systems, debuggers, and other software systems, causing them to malfunction.

LR, the Link Register

Register R14 is used to store the return address from a subroutine. At other times, LR can be used for other purposes.
When a **BL or BLX** instruction performs a subroutine call, LR is set to the subroutine return address. **To perform a subroutine return, copy LR back to the program counter**. This is typically done in one of two ways, after entering the subroutine with a BL or BLX instruction:

• Return with a BX LR instruction.
• On subroutine entry, store LR to the stack with an instruction of the form: PUSH {,LR} and use a matching instruction to return: POP {,PC} ...

What is Difference between Interrupt and Exception?
Interrupts and Exceptions both alter program flow.

Interrupts are used to handle external events (serial ports, keyboard) Interrupts are **handled by the processor after finishing the current instruction** **Interrupt is an as asynchronous event that is normally(not always) generated by hardware** When an interrupt signal is present, a kernel routine is called to handle the interrupt **US→Interrupt→K S→IDT→IVT→ISR→Execution→U S**	**Exceptions are used to handle instruction faults** (division by zero, Segmentation Fault) Exceptions on the other hand are divided into three kinds. **Faults, Traps and Abort** Exceptions are **synchronous** events generated when processor detects any predefined condition while executing instructions. **Atomic operation play an imp role during the execution of the interrupt**

What is going to happen if any interrupt is going to occur?

1. Save process status
Copies the CPSR into the SPSR_<mode>
Stores the return address in LR_<mode>
2. Change process status for the exception
Core switch to exception mode and disable the further interrupt and then uses the vector table
3. Execution of the exception handler
4. Return to main application
Restore the CPSR from the SPSR_<mode>
Restore PC from the LR_<mode>

UART, SPI, I2C, CAN

UART	USART
UART requires only data signal	In USART, **Synchronous mode requires both data and a clock.**
In UART, the data does not have to be transmitted at a fixed rate.	In USART's synchronous mode**, the data is transmitted at a fixed rate**.
In UART, data is normally transmitted one byte at a time.	In USART, Synchronous data is normally transmitted **in the form of blocks**
In UART, data transfer speed is set around specific values like 4800, 9600, 38400 bps ,etc.	**Synchronous mode allows for a higher DTR** (data transfer rate) than asynchronous mode does, if all other factors are held constant.
UART speed is limited around 115200 bps	**USART is faster than 115kb**
Full duplex	**Half duplex**

UART
Serial Protocol. Universal Asynchronous Receiver Transmitter.

It is very simple and old one from all the serial protocols. It is the famous one and most used serial protocol.
Lines : **3 lines are more than enough [TX , RX & GND]**
2 pins for operation in asynchronous and 3 in synchronous
Communication between 2 devices only
It **not suitable** for long distance transmission
Translate data from parallel to serial and vice versa
Tx and Rx must operate at the same baud rate
Support full duplex

- **Synchronous mode-** Clocks are synchronized thus no need of start and stop bits and can send a block of data
- **Asynchronous mode-** Uses start and stop bit and transfers a single byte at a time
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Start and Stop Bits
- **Odd or Even Parity Generation and Parity Check**
- Data Over Run Detection and Framing Error Detection
- **Baud rate =Fosc/(16(X+1))—X is the value we load in the UBRR**

SPI
Motorola-1970
The SPI (Serial Peripheral Interface) is a peripheral used to communicate between the controller and other devices, like others Controller, external EEPROMs, DACs, ADCs, etc.
With this interface, you have one Master device which initiates and controls the communication, and one or more slaves who receive and transmit to the Master
Serial Peripheral Interface is a very low power serial protocol.
SPI bus is **full-duplex** bus, which allows communication to flow **to and from the master device** simultaneously.
Communication may be between 2 or more device, one is master and the others are slave
Synchronous
Data line from the master to the slave named MOSI
R232 are used for invention of the SPI
Clock is controlled by master
Disadvantage is that we have to increase the pin as the slave is going to increase, over come in I2c
Lines: 4 lines: MISO, MOSI, SCL and SS' (SS' means SS complemented).
Speed: **up to 10Mbps**

I2C
Philips Semiconductor-1980
Inter Integrated Circuit is an official standard serial communication protocol that only requires two signal lines that was designed for communication between chips **on a PCB or between 2 device.**
Synchronous
Half duplex
Support **multi master** mode-advantage and master – slave also.
Master is microcontroller which starts the data transfer and generate clock signal
128 device with 7 bit addressing and 1024 with 10 bit addressing
Using of pull-up resistance during the use of i2C
Frequency range 100 kHz-400 kHz
Start and stop condition should be there
Format- Start –8 bit addressing for the slave—ack—8 bit internal register address (for x and y two type of data)—ack—8 bit data--ack--stop
Lines: 2 lines
-- Serial Data Line (SDA)
-- Serial Clock Line (SCL)—Should be high during the data Transfer
Speed: **100kbps to 3.4Mbps**

CAN
Robert bosch-1983--1st chip in 1987--Officially—CAR Automotive/ion application---Embedded industry
Controller Area Network is the secure and fastest serial communication protocol.

CAN is vehicle bus standard designed to allow microcontroller and device to communicate with each other in applications without host computer.
CAN is generally used for different device at different location i.e., **usually outside of our PCB.**
CAN is a multi-master serial bus standard for connecting Electronic circuit units
It is a **message based protocol and Message** is communicate within a system along with message identifier
The length of the message identifier is 11 bit to 29 bit
Node with the highest priority get the bus to transfer the data which is decide by CAN Arbitration
Arbitration (based on CSMA/CA) **id is used during the transmission of the frame**
Lines: 2 Lines [TX and RX]
Speed: **200kbit to 1 Mbit/sec**

Format-
0 is Dominant bit and 1 is recessive (start of frame) --- CAN Identifier either 11 (standard) or 29 bit ---RTR(remote transmission request)—control filed –DLC (decide how many byte of data field are valid)-data field(0-8 bit)—CRC-ACK-EOF

CAN Arbitration:-
It is process which is done on the message identifier to decide that who will get the bus.
If there are 3 node each having 3 digit data and the last digit of the 3rd node is different from the others then that node will win the CAN bus to send the data.

CAN Stuffing:-
To ensure enough transitions to maintain synchronization, a bit of opposite polarity is inserted after the five consecutive bits of the same polarity. This is called bit stuffing and is necessary due to the non-return to zero coding used with the CAN. The stuffed data frames are destuffed by the receiver.
The fields where bit stuffing is used, 6 consecutive bits of the same type are considered an error and an active flag is transmitted by the node when an error has been detected.

In CAN network there are Central bus and CAN node with CAN high and CAN low with some impedance
3 Major parts are
CAN transreceiver--CAN controller—Microcontroller

Error handling
Transmit Error counter and receive error counter

1. Error active--- every node will be in error active by default and can go in any of the other (when REC and TEC≪127)
2. Error passive≫127
3. Bus off≪255

LIN
Local Interconnect Network is a serial network protocol used for communication between components **outside the PCB/BOARD**. **LIN is a broadcast serial network comprising one master and typically up to 16 slaves**.
Lines: Single wire communication
Speed: **up to 20 kbit/s**

What is RS232

It is serial wire cable like UART used for communicating with external devices.
There is start and stop bit and we can set the no. of data which we want to send either 5, 6 and 7.
Parity is also there.
It works as Full duplex also

ARM

ARM Features

- Performance
- Code density
- Low power
- Memory system
- Memory protection unit
- Interrupt handling
- OS support and system level features
- Cortex-M4 specific features
- Ease of use
- Debug support
- Scalability
- Compatibility

Thumb State

ARM architecture defines a 16-bit instruction set called the Thumb instruction set.
A processor that is executing Thumb instructions is said to be operating in *Thumb state*.

ARM state

A Thumb-capable processor that is executing ARM instructions is said to be operating in *ARM state*.
ARM processors always start in ARM state. **You must explicitly change to Thumb state using a BX (Branch and exchange instruction set) instruction**

How to go in ARM state or thumb state write assembly program (Assignment Q)

A processor that is executing ARM instructions is operating in *ARM state*. A processor that is executing Thumb instructions is operating in *Thumb state*. A processor that is executing ThumbEE instructions is operating in *ThumbEE state*. A processor can also operate in another state called the *Jazelle® state*. **These are called *instruction set states***.

A processor in one instruction set state cannot execute instructions from another instruction set. For example, a processor in ARM state cannot execute Thumb instructions, and a processor in Thumb state cannot execute ARM instructions. You must ensure that the processor never receives instructions of the wrong instruction set for the current state.

To do this, you must use an appropriate instruction, for example BX or BLX to change between ARM and Thumb states when performing a branch.

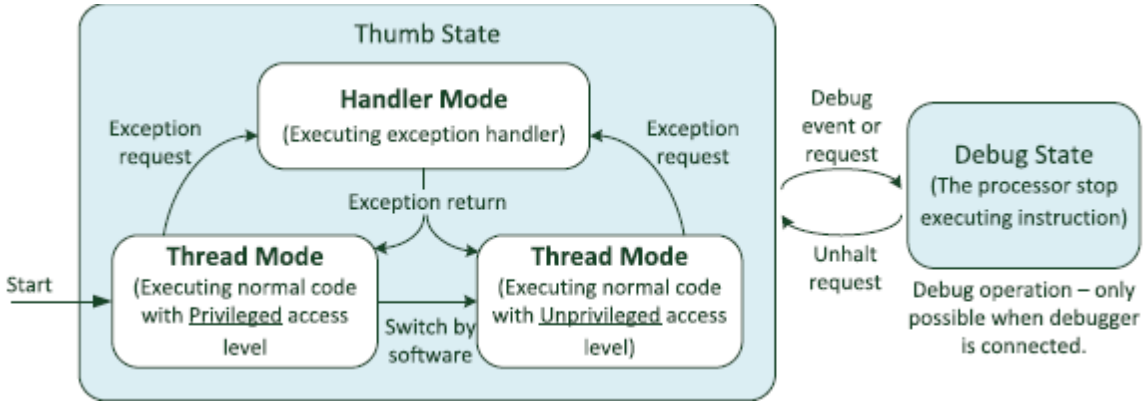
Handler and thread Mode

Handler mode

Executing an exception handler such as an Interrupt Service Routine (ISR).
Processor always has privileged access level.

Thread mode

Executing normal application code
Processor can be either in privileged access level or unprivileged access level.
This is controlled by a special register called “CONTROL.”



ARM Architecture (STM32/ ARM 7TDMI**)**

Architecture refers to the programmer's view of the processor, Includes instruction sets, visible registers, memory management table structures and exceptional handling model

The ARM architecture is based on the ARM7 core **and** similar to Reduced Instruction Set Computer (RISC) architecture, with features including:

- A uniform **register file load/store architecture**, where data processing operates only on register contents, not directly on memory contents.
- Enhancements to a basic RISC architecture enable ARM processors to achieve a good balance of high performance, small code size**, low power consumption and small silicon area and CPU design, Less CPU execution time ~ 1.9 CPI**
- **Bits** :32-bit ,64-bit
- **Endiness**: big
- Pipeline: - ARM is having 3 (**Cortex a8**) to 13 (**Cortex a9**) stage pipeline ...by which more time saving to use Code density is improved by ~30%, saving program memory space

But some difference from RISC and Enhanced instructions for:

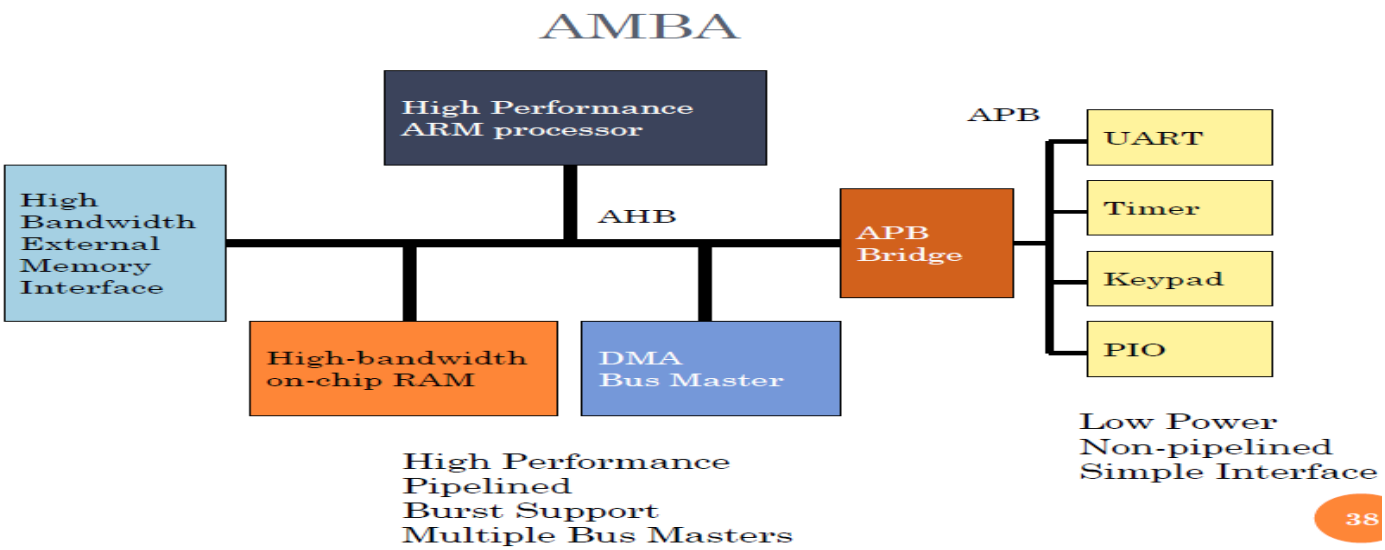
T=THUMB state-- thumb instruction set

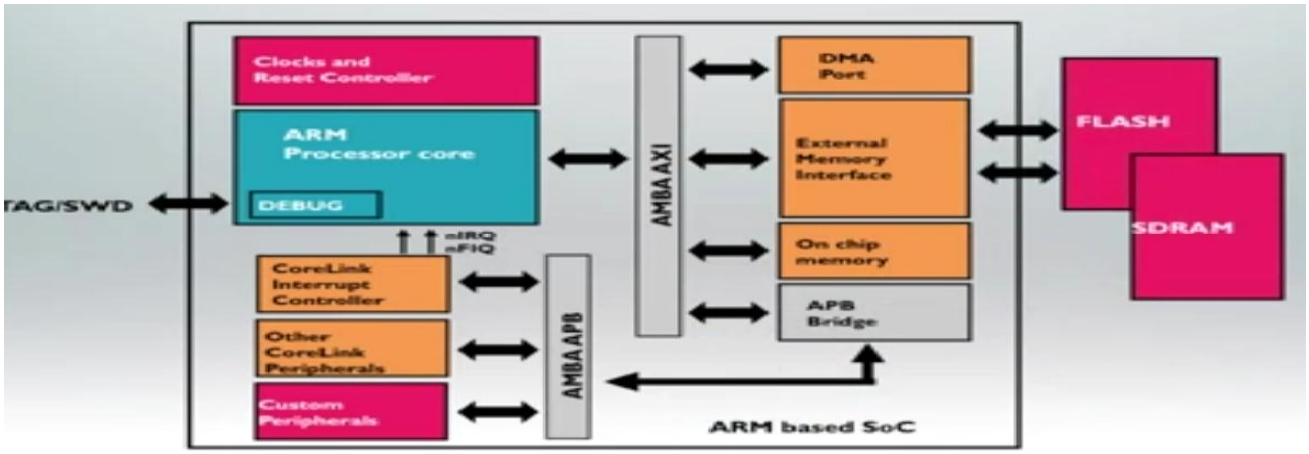
- 32 bit ARM instruction set
- 16/32 bit THUMB instruction set
- The latest technology is uses the thumb 2 technology which so either 16/32 bit so better code density is achieved using the mix of the ARM and THUMB Both

D=including debugs extension
M-Enhance multiplier with instructions for the 64 bit results
I-Core has embedded ICE logic extensions
S-fully synthesisable
DSP instructions
Conditional Execution Instructions
32-bit Barrel shifters

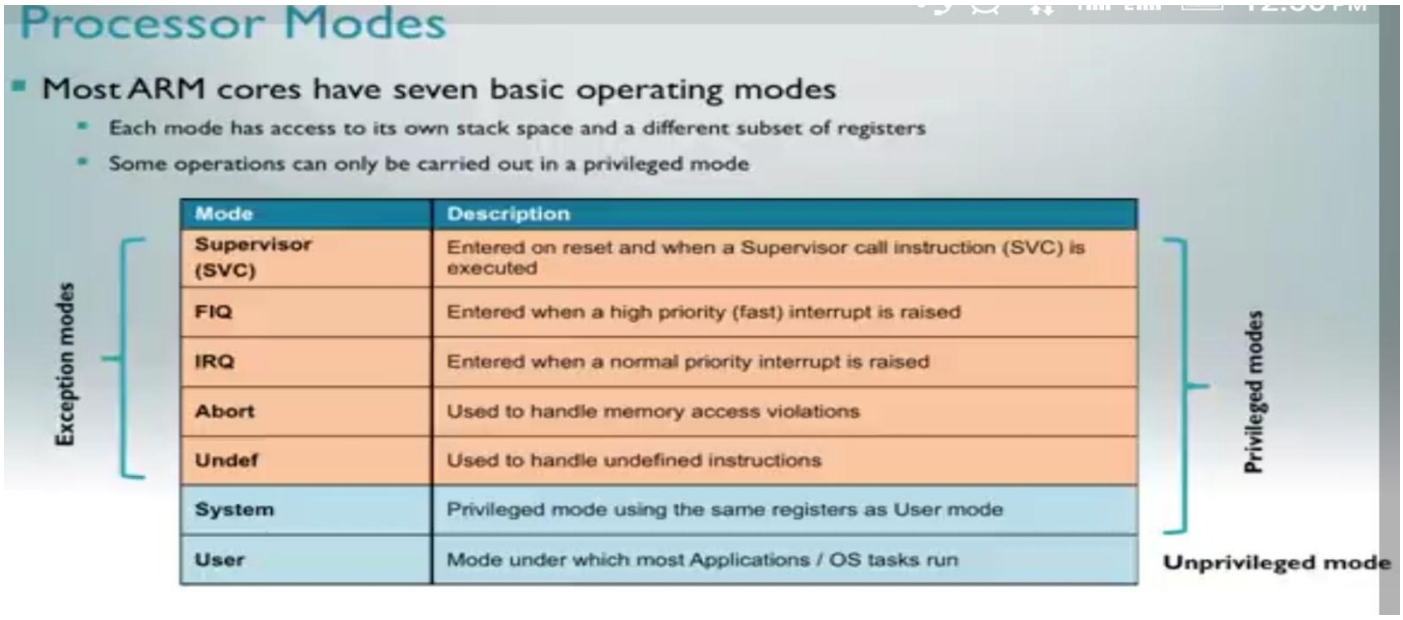
Arm v7-A-Application profiles-support linux-MMU-higher performance at low cast
Arm-v7-R-high performance real time-low latency and predictability-hardware divide-**support memory mapping**
Arm- v7-M-microcontroller and thumb only Embedded use-gate count entry points-low power and
Best feature is exception handling and fixed memory map.....

AMBA peripheral bus is used to explain the arm architecture
APB and AHB
For waste majority of the use AMBHA bus is interconnected
High performance system bus called AXI and low power system bus called APB
APB for peripherals and AXI for memory and other high speed device





7 Basic Operating Modes
NUMBER OF MODES IN ARM (FIQ, IRQ, ABORT,UNDEF, SYSTEM, USER)



What is SVC?

Supervisor calls are normally used to request privileged operations or access to system resources from an operating system
Supervisor call (SVC) is a processor instruction that directs the processor to pass control of the computer to the operating system's supervisor program.Most SVCs are requests for a specific operating system service from an application program
Each service has a preassigned SVC number. When the computer's processor executes the instruction that contains the SVC, the code representing "SVC" causes a program interrupt to occur, which means that control of the processor is immediately passed to the operating system supervisor program. The supervisor then passes control to programming that performs the service that goes with the specified SVC number.

What is DMA?
Direct Memory Access (DMA) is a capability provided by some computer bus architectures that allows data to be sent directly from an attached device (such as a disk drive) to the memory on the computer's motherboard. The microprocessor is freed from involvement with the data transfer, thus speeding up overall computer operation.
Usually a specified portion of memory is designated as an area to be used for direct memory access. In the ISA bus standard, up to 16 megabytes of memory can be addressed for DMA.
An alternative to DMA is the Programmed Input/Output (PIO) interface in which all data transmitted between devices goes through the processor.

The devices feature two general-purpose dual-port DMAs (DMA1 and DMA2) with 8 streams each. They are able to manage memory-to-memory, **peripheral**-to-memory and memory-to-peripheral transfers. They feature dedicated FIFOs for APB/AHB peripherals, support burst transfer and are designed to provide the maximum peripheral band width (AHB/APB).

Bit Banding
Bit Banding is a method of performing atomic bitwise modifications to memory.
Allias region and SRAM region

Prescaling
A prescaler is an electronic counting circuit used to reduce a high frequency electrical signal to a lower frequency by integer division

Maskable Interrupts- interrupt whose request can be denied by microprocessor. eg- RST 1, RST2, RST 5, RST 6.5 etc.
Nonmaskable Interupts - Interrupts whose request cannot be denied.eg -RST 4.5 or TRAP, RESET

Operating Systems

Boot loader
A boot loader is a small program which is started from the Master Boot Record (MBR) of a hard disk
Boot loader could be more aptly called the kernel loader. The task at this stage is to load the Linux kernel
A boot loader, also called a boot manager, is a small program that places the operating system (OS) of a computer into memory. When a computer is powered-up, the basic input/output system (BIOS) performs some initial tests, and then transfers control to the master boot record (MBR) where the boot loader resides
The role of a boot loader is to load an operating system from a storage device, set up a minimal environment in which the OS can run, and run the operating system's start-up procedure.

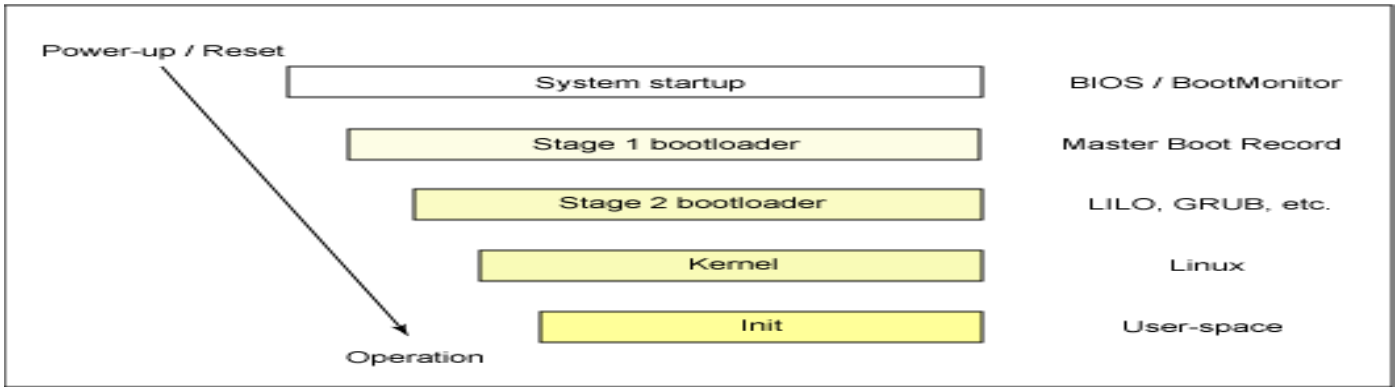
Boot strapping
Bootstrap is the process of loading a set of instructions when a computer is first turned on. During the start-up process, diagnostic tests are performed, such as the power-on self-test (POST) that check configurations for devices and implement routine testing for the connection of peripherals, hardware and external memory devices. The boot-loader or bootstrap program is then loaded to initialize the OS.

Boot sequence

A boot sequence is the set of operations the computer performs when it is switched on that load an operating system

- **Booting sequence**
  - **Tern on**
  - **CPU jump to address of BIOS (0xFFFF0)**
  - **BIOS runs POST (Power-On Self Test)**
  - **Find bootable devices**
  - **Loads and execute boot sector form MBR**
  - **Load OS**

The primary function of BIOS is code program embedded on a chip that recognizes and controls various devices that make up the computer



KERNAL

- Portion of operating system that is in main memory
- **The kernel is the central part in most computer operating systems because of its task, which is the management of the system's resources and the communication between hardware and software components**
- Different from BIOS which is hardware dependent.
- **Kernel is always store on memory until computer is turn off**
- Process requests services of kernel through system calls

KERNA L TASK:

- Process management
- Memory management
- Device management
- System call

Hard ware interrupt: reset, ctrl c, ctrl+alt+dlt
Soft: trap, fault, abort, **SVC (Supervisor control ...is that thumb and ARM state are there)**

Process
– is created by OS to execute a program
– OS puts it in the main memory
– Creates a data structure

Diagram of process state
New, Ready, running, waiting, terminated

Process Control block

Process Control Block (PCB, also called Task Controlling Block, process table, Task Struct, or Switchframe) is a **data structure in the operating system kernel**, containing the information needed to manage a particular process.
The PCB is "the manifestation of a process in an operating system".
Process id, resource, state, register, no.etc contains

Critical Section Problem and how it solved ----Solution is Mutex, Semaphore

Critical section is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution.

Deadlock (Detection, Prevention and avoidance)
Starvation occurs when one or more threads /process in your program are blocked from gaining access to a resource

Deadlock, the ultimate form of starvation, occurs when two or more process is waiting on a condition that cannot be satisfied.

Problem of this solved by the synchronisation using semaphore and mutex (mainly used in thread)

Semaphore is an integer value and it is signaling mechanism ("I am done, you can carry on" kind of signal).

Mutex- Mutex is locking mechanism used to synchronize access to a resource
Mutex is an abbreviation for "mutual exclusion". Mutex variables are one of the primary means of implementing **thread synchronization** and for **protecting shared data when multiple writes occur.**
Mutexes can be used to prevent **"race conditions"**.

Thread:
It is an independent stream of control that can execute its instructions independently and can use the process resource
Dividing the problems
Concurrency and Efficiency

What is the initial value of the semaphore?
The initial value of a semaphore signifies the number of concurrent accesses your locked object can handle. This can be an arbitrary design specific value.

When it is specifically desired to provide only one thread to access your resource, you must initialize the semaphore to 1, so that the call semwait () decrements it to zero. Zero signifies that no further thread can lock the resource, and this type of semaphore is known as a binary semaphore

If you are careful, you will see that the value of the counter is either 1 or 0, and never has any other value. Therefore, it is referred to as a **binary semaphore**. If we replace the counter with a Boolean variable and interpret 1 and 0 as true (i.e., lock is open) and false (i.e., lock is closed) respectively, then a binary semaphore becomes a Mutex lock.Therefore, you can use mutex lock or binary semaphore interchangeably.

What is difference between counting and binary semaphore
Binary semaphores are binary, they can have two values only; one to represent that a process/thread is in the critical section(code that access the shared resource) and others should wait, the other indicating the critical section is free.

On the other hand, counting semaphores take more than two value, they can have any value you want. The max value X they take allows X process/threads to access the shared resource simultaneously.

Binary semaphores are easier to implement comparing with the counting semaphore.
Binary semaphore allows only one thread to access the resource at a time. But counting semaphore allows N accesses at a time.

The 2 operations that are defined for binary semaphores are **take and release**.
The 2 operations that are defined for counting semaphores are **wait and signal**

Inter Process Communications (Socket, Shared-Memory Mapping, Message Queue and pipe)

A mechanism that allow co-operating processes to exchange data and information

Reasons for cooperating (not independent) processes:

- Information sharing
- Computation speedup
- Modularity
- Convenience
- Privilege separation

Address translation is to protect one process from other (heap, stack, text, data, etc.,)

Message Queue
Two (or more) processes can exchange information via access to a common system called as Message Queue
Each message & Message Queue is uniquely identified by an IPC identifier
A common key is shared between processes accessing the queue

Message queue is better than pipe. This is because, in pipe, you have to decide the protocol used for communication

---

Difference between paging, segmentation and fragmentations

Virtual memory
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory

Swapping
Swapping is mechanisms in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Fragmentation
As processes are loaded and removed from memory, the free memory space is broken into little pieces**. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused.** This problem is known as Fragmentation

External fragmentation
Total memory space is enough to satisfy a request or to reside a process in it, **but it is not contiguous**, so it cannot be used.

Internal fragmentation
Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process

Segmentation
Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program

Paging

Paging is a memory management technique in which process address space (memory) is **broken into blocks of the same size called pages**

Paging is used for faster access to data. When a program needs a page, it is available in the main memory as the OS copies a certain number of pages from your storage device to main memory. Paging allows the physical address space of a process to be non-contiguous.

Advantages and Disadvantages of Paging

- **Paging reduces external fragmentation, but still suffers from internal fragmentation**.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM

Links

A link is an association between a filename and an inode.

UNIX has two types of links

Hard link----

Multiple names can point to same inode

The inode keeps track of how many links exists for the file

Symbolic link (soft link)

Soft link is a pointer to the another file

A reference to the name of a symbolic link causes the operating system to use the name stored in the file, rather than the name itself

COPY-ON-WRITE

Synchronisation Technique

Round robin—

esme yad kya rakhna h ki ek proper time k liye sbko cPU diya jayega jise TQ khte h agr pura hua to shi h wrna jitna time bacha h .....utne time bad hi vo dubra aayega tb tk cpu dusra process utha lega

aur sb complet ho jane k bad dubara bacha hua process reqest queue m chla jayega then same process till the completion of the all task

primitive and easy to implement

startvation free

time quantum

request queue-the process which we have to complete again

BSP

A **board support package** (**BSP**) is essential code for a given computer hardware device that will make that device work with the computer's operating system. **The BSP contains a small program called a boot loader or boot manager that places the OS and device drivers into memory**

Porting

In software engineering, **porting** is the process of adapting software so that an executable program can be created for a computing environment that is different from the one for which it was originally designed

Porting is the process of adopting software in an environment for which it was not originally written or intended to execute in. (requirement is Cross complier, Boot loader, Kernel, Root file system**)**

LINUX DEVICE DRIVER

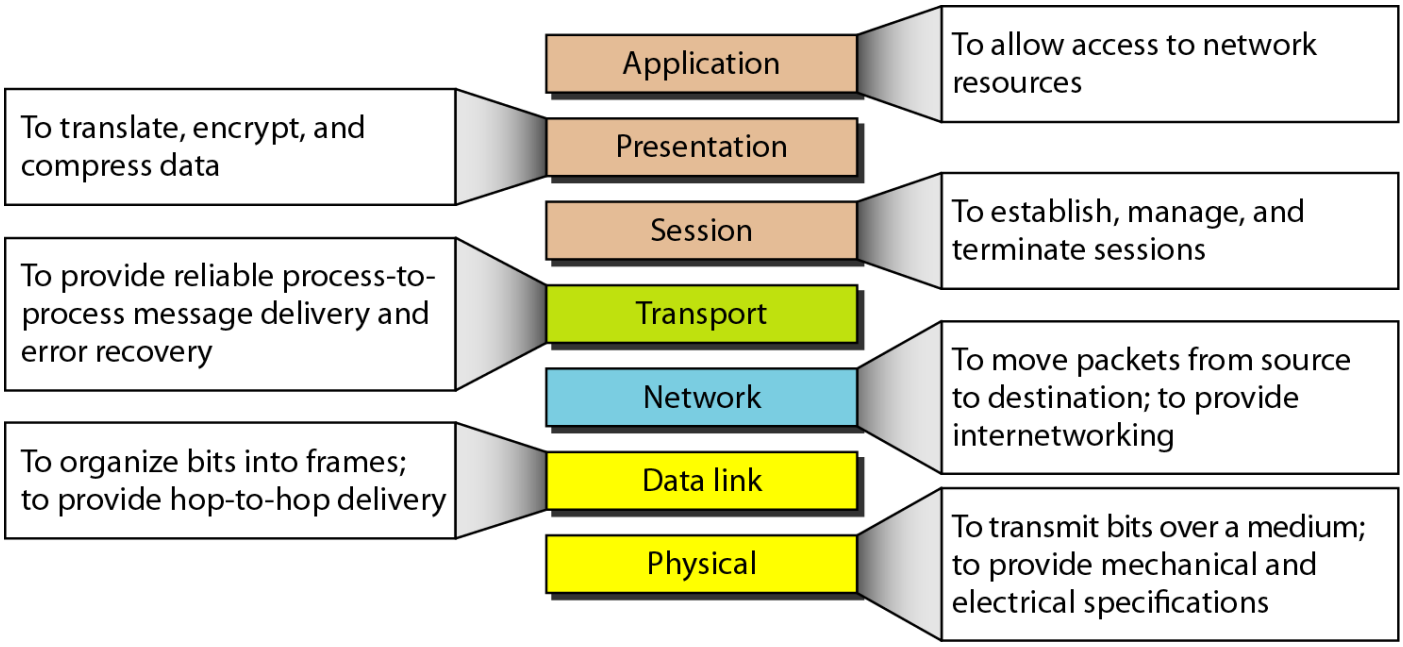
What do you mean by character and block driver

A device, such as a terminal or printer that conveys data character by **character**

A device, such as a magnetic tape drive or **disk drive** that conveys data in blocks through the buffer management code

---

DATA COMMUNICATION NETWORK



HUB—

A hub is a common connection point for devices in a network. Hubs are commonly used to connect segments of a LAN. A hub contains multiple ports. When a packet arrives at one port, it is copied to the other ports so that all segments of the LAN can see all packets.

Bridge

In telecommunication networks, a bridge is a product that connects a local area network (LAN) to another local area network that uses the same protocol (for example, Ethernet or token ring).

Router-

A **router** is a device that forwards data packets along **networks**. A **router** is connected to at least two **networks**, commonly two LANs or WANs or a LAN and its ISP's **network**. **Routers** are located at gateways, the places where two or more **networks** connect

Switch

A network switch is a computer networking device that connects devices together on a computer network, by using packet switching to receive, process and forward data to the destination device