Table of Contents

Prepare for the interview.

Mock Interview

Consider doing a practice interview with friends or family members (especially if they have experience interviewing employees themselves). The practice will be helpful and give you more confidence. Ask for feedback on your answers, your body language, and your preparedness. **Have the mock interviewer ask both common questions as well as offbeat ones to see how well you can think on your feet.**

Taking the time to review typical interview questions will help give you a framework for your responses. It will also **reduce stress because you won't be scrambling for an answer during the interview**. **Practice interviewing with a friend or family member ahead of time, and it will be much easier when you're actually in a job interview.**

Interview Platform/Syllabus

S.NO	Subject / Topic / Chapter	
1	C Language	
2	Data Structure – C	
3	LDD3	
4	ARM – Processor - Information	
5	Operating System Concepts	
6	Device Tree	
7	U-boot Source - Compilation / Configuration / CODE FLOW	
8	Linux Source – Compilation / Configuration / CODE FLOW	
9	SOC – Main Controllers – Schematics	
10	Board Bring up Procedure – BOOT Sequence	

Confusing / Tricky: C language Questions:

1. **What is the output?**

```
#include<stdio.h>
int main()
{
        int i=5;
        i = i+++i;
        printf("Value of i=%d\n",i);
        return 0;
}
```

2. **What is the output?**

```
#include<stdio.h>
int main()
{
        char *ptr1="hello",*ptr2;
        ptr2=ptr1;
        ptr1="world";
        printf("ptr1=%s ptr2=%s\n",ptr1,ptr2);
        return 0;
}
```

3. What is the output?

```
#include<stdio.h>
int main()
{
	int i=2;
	i=++2;
	printf("value of i=%d\n",i);
	return 0;
}
```

4. What is the output?

```
#include<stdio.h>
int main()
{
	int i=10;
	i*=10+1;  //110
	printf("value of i=%d\n",i);
	return 0;
}
```

5. What is the output?

```
#include<stdio.h>
int main()
{
	int i=(1,2,3,4,5,6,7,8,9,10);
	int j;
	j=1,2,3,4,5,6,7,8,9,10;
	printf("value of i=%d\n",i);
```

```
        printf("value of j=%d\n",j);
        return 0;
}
```

6. What is the output?

```
#include<stdio.h>
int main()
{
        int i=10;
        while(i<=15,i<=20) {
                printf("value of i=%d\n",i);
                i++;
        }
        return 0;
}
```

7. What is the output?

```
#include<stdio.h>

int main()
{
        int i=10;
        while(i++<15);
        {
                printf("value of i=%d\n",i);
        }
        return 0;
}
```

8. What is the output?

```
#include<stdio.h>
int main()
{
	int i=10,j=20,k=30;
	printf("%d%d\b\b\b\b%d",i,j,k);
	return 0;
}
```

9. What is the output?

```
#include<stdio.h>
int main()
{
	int i=10;
	int *ptr=&i;
	scanf("%d",ptr);
	printf("the value of i=%d\n",i);
	return 0;
}
```

10. What is the output?

```
#include<stdio.h>

int main()
{
        printf("value of exp=%d\n",5%2);

        printf("value of exp=%d\n",-5%2);

        printf("value of exp=%d\n",5%-2);

        printf("value of exp=%d\n",-5%-2);

        return 0;

}
```

11. What is the output?

```
#include<stdio.h>
void f()
{
        printf("Hello\n");
}

void main()
{
         ;
}
```

12. What is the output?

```
#include<stdio.h>

void main()
{
   int x = 5;

   if(x=5)
   {
       if(x=5) break;
       printf("Hello");
   }
   printf("Hi");
}
```

13. What is the output?

```
#include<stdio.h>
int main()
```

```
{
   const int x = 5;

   const int *ptrx;
   ptrx = &x;
   *ptrx = 10;
   printf("%d\n", x);
   return 0;
}
```

14. What is the output?

```
#include<stdio.h>

void main()
{
   int x;
   float y;

   y = x = 7.5;
   printf("x=%d y=%f", x, y);
}
```

15. What is the output?

```
#include<stdio.h>

void main()
{
   int a[3] = {2,,1};

   printf("%d", a[a[0]]);
}
```

16. What is the output?

```
#include<stdio.h>

void main()
{
   int a = 5, b = 3, c = 4;

   printf("a = %d, b = %d\n", a, b, c);
}
```

17. What is the output?

```
#include<stdio.h>

void main()
{
   int x = 3;

   x += 2;
   printf("%d\n", x);
   x =+ 2;
   printf("%d\n", x);
}
```

18. what is the output?

```
#include<stdio.h>
#include<string.h>
#include<stdarg.h>

int main()
{
   struct abc{
               int len;
               char *str;
        } ;

   struct abc student_name = {10,"Naveen"};
   struct abc *ptr=&student_name;

   ++ptr->len;

   printf("name=%s,len=%d\n",student_name.str,student_name.len);
   return 0;
}
```

19. what is the output?

```
#include<stdio.h>
#include<stdarg.h>
#include<string.h>
```

```
int main()
{
         int a=3;
        switch(a) {
                printf("I will be executed allways\n");
                default:
                        printf("I am a default case , I am executed every time\n");
                        break;
                case 1:
                        printf("I am :case 1\n");
                        break;
                case 2:
                        printf("I am :case 2\n");
                        break;
        }
}
```

20. What is the output?

```
#include<stdio.h>
#include<stdarg.h>
#include<string.h>

int main()
{
        int a[10]={0,2,3,4,5,6,7,8,9};
        int *ptr1=&a;
        int *ptr2=&a;
        int *ptr3=&a;

         printf("Values=%d,%d,%d\n",*++ptr1,*ptr2++,++*ptr3);
}
```

21. What is the output ?

```
#include<stdio.h>
#include<stdarg.h>
#include<string.h>

int add(int a,int b)
{
         printf("i am in add function\n");
         return (a+b);
}
int main()
{
	void *ptr=add;
	printf("Sum of two variables=%d\n", ((int(*)(int,int))ptr)(10,20));
}
```

22. what is the output?

```
#include<stdio.h>
#include<stdarg.h>
#include<string.h>

int main()
{
	int num1,num2,total=0;

	printf("Enter two number :\n");
	scanf("%d%d",&num1,&num2);

	while(num1 > 0)
	{
		if(num1 & 1) {
			total+=num2;
		}
		num1=num1>>1;
		num2=num2<<1;
	}
	printf("output=%d\n",total);
}
```

23. what is the output ?

```
#include<stdio.h>

int main()
{
	printf("hi") , printf("Hello");
	return 0;
}
```

24. what is the output?

```
int main()
```

```
{
	int i=5;
	i = ++i + ++i;
	printf("value=%d\n",i);
	return 0;
}
```

25. what is the output?

```
#include<stdio.h>
int main()
{
	int a=5,b=3;
	printf("value=%x\n",-1<<4);
	return 0;
}
```

26. what is the output?

```
#include<stdio.h>
#include<limits.h>
int main()
{
	char x=120;
	while(++x > 0) {
		printf("value x=%d\n",x);
	}
	return 0;
}
```

27. what is the output?

```
#include<stdio.h>
int main()
{
	int a=5,b=3;
	printf("value=%d\n",++(a*b+1));
	return 0;
}
```

28. what is the output?

```
#include<stdio.h>
int main()
{
	float f=3.14;
	printf("value=%f\n",f%2);
```

```
        return 0;
}
```

29. what is the output?

```
#include<stdio.h>
#include<limits.h>
int main()
{
        int a=4,b=5;
        b= a++ + a--;
        a= ++b + b--;
        printf("a=%d\n",a);
        printf("b=%d\n",b);
        return 0;
}
```

30. what is the output?

```
#include<stdio.h>
void fun(int );
int main()
{
        int x=2;
        fun(x++);
        return 0;
}
void fun(int x)
{
        printf("value=%d\n",x++);
}
```

31. what is the output?

```
#include<stdio.h>
int main()
{
        int i=5,j;
        j=(i=i+1,i=i+2,i+3);
        printf("values i and j = %d %d \n",i,j);
        return 0;
}
```

32. what is the output?

```
int main()
```

```
{
        int i=0;
        switch(i) {
                case 2+2: printf("four"); break;
                case 2/2: printf("one"); break;
                case 2-2: printf("zero"); break;
                case 2+1: printf("three"); break;
                default : printf("default\n"); break;

        }
        return 0;
}
```

33. what is the output?

```
#include<stdio.h>
int main()
{
        int i=-5;
        i=~i;
        printf("value i = %d \n",i);
        return 0;
}
```

34. what is the output?

```
#include<stdio.h>
int main()
{
        int x,y,z;
        x=0; y=1;
        z= x && ++y;
        printf("values x y z = %d %d %d\n",x,y,z);
        return 0;
}
```

35. what is the output?

```
#include<stdio.h>
int main()
{
        char ch=124;
        while(0 < ++ch) {
                printf("ch value= %d\n",++ch);
        }
```

```
	return 0;
}
```

36. what is the output?

```
#include<stdio.h>
int fun(int,int);
int main()
{
	int i=5;
	fun(--i,i++);
	fun(++i,i--);
	return 0;
}
int fun(int x, int y)
{
	printf("values x=%d y=%d\n",x++,y--);
	return 0;
}
```

Implementation : C language Questions:

S.NO	**Question**
1	Write a Program to say little endian or big endian
2	Write a Program to set bit,clear bit ,toggle bit and print bits...using the macro
3	d
4	Write a Program to print “string reverse” recursively
5	Write a Program to show the difference between int (*p)[3]; vs int *p[3]
6	Write a Program Given string is rotation of original string or not?
7	Write a Program : I/P string: aaabbbcccddd o/p: print: a3b3c3d3
8	Write a Program: initial string =He is good Reverse string =good is He
9.	Write a function[most effective] to count the numbers 1 in a given number
10.	Write a Program: number conversion:from decimal to hexa, decimal to oct, decimal to binary
11.	write a program to do the strcat operation

12.	wrnite a program to do strtok operation
13.	write a program to do strstr operation
14.	write a program to do strdup operation
15.	write a program to do strchr operation
16.	How do you swap nos without using third variable
18.	write a program to do strcmp, strncmp, memcmp operations
19.	write a program to do strcpy,memcpy on string and char array
20.	How to find the offset of variable in a given structure? (container_of) implementation How to find the address of structure base, if you are given with variable/element address,which is inside of struct?

Hacker Ranker : C language Questions:

S.NO	**Question**	**Link**
1	AND Product	https://www.hackerrank.com/challenges/and-product/problem
2	Flips the bits	https://www.hackerrank.com/challenges/flipping-bits/problem
3	Lonely Integer	https://www.hackerrank.com/challenges/lonely-integer/problem
4	MAX of xor	https://www.hackerrank.com/challenges/maximizing-xor/problem
5	Sum vs xor	https://www.hackerrank.com/challenges/sum-vs-xor/problem
6	Greatest xor	https://www.hackerrank.com/challenges/the-great-xor/problem
7	recursive-digit-sum	https://www.hackerrank.com/challenges/recursive-digit-sum/problem
8	jumping-on-the-clouds	https://www.hackerrank.com/challenges/jumping-on-the-clouds/problem
9	jumping-on-	https://www.hackerrank.com/challenges/jumping-on-

	the-clouds-revisited	the-clouds-revisited/problem
10	kangaroo	https://www.hackerrank.com/challenges/kangaroo/problem
11	lisa-workbook	https://www.hackerrank.com/challenges/lisa-workbook/problem
12	migratory-birds	https://www.hackerrank.com/challenges/migratory-birds/problem
13	strange-advertising	https://www.hackerrank.com/challenges/strange-advertising/problem
14	breaking-best-and-worst-records	https://www.hackerrank.com/challenges/breaking-best-and-worst-records/problem
15	beautiful-days-at-the-movies	https://www.hackerrank.com/challenges/beautiful-days-at-the-movies/problem
16	utopian-tree	https://www.hackerrank.com/challenges/utopian-tree/problem
17	circular-array-rotation	https://www.hackerrank.com/challenges/circular-array-rotation/problem
18	strange-code	https://www.hackerrank.com/challenges/strange-code/problem
19	funny-string/	https://www.hackerrank.com/challenges/funny-string/problem
20	greedy-florist	https://www.hackerrank.com/challenges/greedy-florist/problem

C language: Oral Questions:

1)prototype of memory allocation functions

Information:

1 void * malloc (size_t) : malloc (number *sizeof(int));

2 void * calloc (size_t,size_t) : calloc (number, sizeof(int));

3 void * realloc (void*,size_t) : realloc (pointer_name, number * sizeof(int));

4 void free (void (*) : free (pointer_name);

2) what is Callback Functions:

A callback function is a function that is called through a function pointer.

When you pass the pointer of a function as a parameter to another function, and when that pointer is used to call the function it points to a call back is made.

3) what is constant pointer [it will hold only one assignment[address]]

A constant pointer is declared as follows :

<type of pointer> * const <name of pointer>

An example declaration would look like :

int * const ptr;

4) what is pointer to constant [we can assign the new address to pointer, but should not change the value]

const <type of pointer>* <name of pointer>

An example of definition could be :

const int* ptr;

5) what is constant pointer to constant [it will hold the address and value, both can not be altered]

const <type of pointer>* const <name of pointer>

for example :

const int* const ptr;

5)What is the difference between int (*p)[3]; vs int *p[3];

        p=p+1==> int * 4

6) *ptr++ , (*ptr)++ what is the difference?

*ptr++ means ,Increment the Pointer not Value Pointed by It

++*ptr means ,Increment the Value being Pointed to by ptr (*ptr)++

7) what is wild & dangling pointer?,how to avoid it in program?

Dangling pointer: Pointer assigned some memory and then free that memory. After freeing when not assigned to NULL

it still points to same memory address, such pointer **are dangling pointer.**

Wild pointer: Pointer which are not initialized during its definition holding some junk value(a valid address) are **Wild pointer.**

8) Size of data type depends on which compiler we use?

A) yes, for gcc in 64bit machine gcc 4.9.2 version

char	- 1Byte [Independent of Compiler]
short int	- 2Byte
int	- 4Byte [Independent of Compiler , min 2, max 4]
float	- 4Byte [Independent of Compiler]
double	- 8Byte [Independent of Compiler]
Long double	- 16Byte

char*   - 8Byte (for 64bit machine)

9)Representation of positive and negative numbers in C language/compiler?

There are three well known methods for representing negative values in binary:

Signed magnitude. This is the easiest to understand, because it works the same as we are used to when dealing with negative decimal values: The first position (bit) represents the sign (0 for positive, 1 for negative), and the other bits represent the number. Although it is easy for us to understand, it is hard for computers to work with, especially when doing arithmetic with negative numbers.

In 8-bit signed magnitude, the value 8 is represented as 0 0001000 and -8 as 1 0001000.

One's complement. In this representation, negative numbers are created from the corresponding positive number by flipping all the bits and not just the sign bit. This makes it easier to work with negative numbers for a computer, but has the complication that there are two distinct representations for +0 and -0.

The flipping of all the bits makes this harder to understand for humans.

In 8-bit one's complement, the value 8 is represented as 00001000 and -8 as 11110111.

Two's complement. This is the most common representation used nowadays for negative integers because it is the easiest to work with for computers, but it is also the hardest to understand for humans.

When comparing the bit patterns used for negative values between one's complement and two's complement, it can be observed that the same bit pattern in two's complement encodes for the next lower number.

For example 11111111 stands for -0 in one's complement and for -1 in two's complement, and similarly for 10000000 (-127 vs -128).

In 8-bit two's complement, the value 8 is represented as 00001000 and -8 as 11111000.

10)typedef vs #define

#define is a C-directive which is also used to define the aliases for various data types similar to typedef but with the following differences −

typedef is limited to giving symbolic names to types only where as #define can be used to define alias

for values as well, q., you can define 1 as ONE etc.

typedef interpretation is performed by the compiler whereas #define statements are processed by the pre-processor.

11)stack underflow -

Computer Definition. An error condition that occurs when an item is called for from the stack, but the stack is empty. Contrast with stack overflow.

Now that we have a basic picture in mind of what a stack conceptually looks like, we can define what underflow and overflow are.

Stack underflow happens when we try to pop (remove) an item from the stack, when nothing is actually there to remove.

This will raise an alarm of sorts in the computer, because we told it to do something that cannot be done.

Stack overflow happens when we try to push one more item onto our stack than it can actually hold. You see, the stack usually can only hold so much stuff. Typically, we allocate (set aside) where the stack is going to

be in memory and how big it can get. So, when we stick too much stuff there or try to remove nothing, we will generate a stack overflow condition or stack underflow condition, respectively.

12)Enumerated Types are a special way of creating your own Type in C. The type is a "list of key words".

Enumerated types are used to make a program clearer to the reader/maintainer of the program.

```
enum week{ sunday, monday, tuesday, wednesday, thursday, friday, saturday};

enum week today;

enum boolean{    false;    true;};

enum boolean check;

enum month {JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,DEC};

enum month rmonth;

enum compass_direction  {north,    east,    south,    west  };

enum compass_direction my_direction;

enum planets  {Mercury,    Venus,    Earth,    Mars,    Jupiter,    Saturn,
Uranus,    Neptune,    Pluto  };

enum planets planet1, planet2;
```

13)pointer dereference

The operator * is used to do this, and is called the dereferencing operator.

```
int a = 10; int* ptr = &a; printf("%d", *ptr);

// With *ptr I'm dereferencing the pointer. // Which means, I am asking the
value pointed at by the pointer.
```

// ptr is pointing to the location in memory of the variable a.

Dereferencing Operation is performed to access or manipulate data contained in memory location pointed to by a pointer

14) inline vs macro

15) typedef vs macro

16) sections of code

17) storage types, application use cases

auto, register,static,extern

18) for, do-while,while ===> decrementing loop

19) optimization levels

20) volatile key word, const attribute

volatile is key word, tell to compiler do not do any optimization on this variable, whose value can vary dynamic, not in the program ,code control.

21)difference between declaration vs definition

definition: type and memory allocation

declaration: say somebody, some portion of code, it is defined, it says its data type.

Data Structures : Questions:

S.NO	**LIST**	**Operations**
1	Single Linked List	Insert Node, Delete Node, Search Node. Print the List, Print Reverse of the List.
2	Stack [Single Linked List]	Push,pop,empty,top
3	Queue [Single Linked List]	Queue,dequeue, empty,front
4	Double Linked List	Insert Node, Delete Node, Search Node. Print the List, Print Reverse of the List.
5	Single Linked List	Find the middle node
6	Single Linked List	N th node from the end of the list
7	Single Linked List	Reverse of the List
8	Single Linked List	Rotation after given position
9	Circular Linked List	Insert Node, Delete Node, Search Node. Print the List, Print Reverse of the List.
10.	Postfix Evaluation using stack	
11	Find Largest Rectangle using stack	
12	Balanced Brackets using stack	
13	Conversion: Infix to Postfix, Postfix to Infix	
14	Conversion:Prefix to Postfix,Postfix to Prefix	
15	Conversion:Infix to Prefix, Prefix to Infix	
16	Stack , min element	
17	Stack , max element	
18	Single Linked List	Loop detection.
19	BT	Insert, Delete , Count, Find, Pre/In/Post Order, Level Order, Height.
20	BST	Insert, Delete , Count, Find, Pre/In/Post Order, Level Order, Height.

Data Structures : Oral Questions

1) What is data structure?

Data structure refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with the data structure, we not only focus on one piece of data but the different set of data and how they can relate to one another in an organized manner

2) When is a binary search best applied?

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is searched starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

3) What is a linked list?

A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link for data storage.

4) How do you reference all the elements in a one-dimension array?

To reference all the elements in a one -dimension array, you need to use an indexed loop, So that, the counter runs from 0 to the array size minus one. In this manner, You can reference all the elements in sequence by using the loop counter as the array subscript.

5) In what areas do data structures are applied?

Data structures are essential in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

6) What is LIFO?

LIFO is a short form of Last In First Out. It refers how data is accessed, stored and retrieved. Using this scheme, data that was stored last should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

7) What is a queue?

A queue is a data structure that can simulate a list or stream of data. In this structure, new elements are inserted at one end, and existing elements are removed from the other end.

8) What are binary trees?

A binary tree is one type of data structure that has two nodes, a left node, and a right node. In programming, binary trees are an extension of the linked list structures.

9) Which data structures are applied when dealing with a recursive function?

Recursion, is a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

10) What is a stack?

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

11) Explain Binary Search Tree

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

12) What is algorithm

Algorithm is a step by step procedure, which defines a set of instructions to be executed in certain order to get the desired output.

13)What is an AVL tree?

An AVL tree is a type of binary search tree that is always in a state of partially balanced. The balance is measured as a difference between the heights of the subtrees from the root. This self-balancing tree was known to be the first data structure to be designed as such.

14)What are linear and non linear data Structures?

- **Linear:** A data structure is said to be linear if its elements form a sequence or a linear list. Examples: Array. Linked List, Stacks and Queues
- **Non-Linear:** A data structure is said to be non-linear if traversal of nodes is nonlinear in nature. Example: Graph and Trees

15)What are the various operations that can be performed on different Data Structures?

- **Insertion** ? Add a new data item in the given collection of data items.
- **Deletion** ? Delete an existing data item from the given collection of data items.
- **Traversal** ? Access each data item exactly once so that it can be processed.
- **Searching** ? Find out the location of the data item if it exists in the given collection of data items.
- **Sorting** ? Arranging the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

16) What is a Binary Tree ?

A Binary Tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child and the topmost node in the tree is called the root.

The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is:

2^(h+1) – 1

17) What is full binary tree

A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children.

18) What is complete binary tree

A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

LDD3 : Driver Implementation Questions

S.NO	**Questions**
1	Module insmod, rmmod, lsmd, modprobe, depmod, compilation Makefile
2	Char Module(static device number) with file operations: open,release,read,write,lseek
3	Char Module(dynamic device number) with file operations: open,release,read,write,lseek
4	Char Module(dynamic device number) with file operations: open,release,read,write,lseek,ioctl
5	Char Module(dynamic device number) with file operations: open,release,read,write,lseek,ioctl,poll

6	Module with irq handler, compilation Makefile
7	Module with bottom half handler(tasklet, work queue), compilation Makefile
8	Module with timer, compilation Makefile
9	Char Module(dynamic device number) with file operations: open,release,read,write,lseek,ioctl: locking semaphore,complete, wait queue
10	Char Module(dynamic device number) with file operations: open,release,read,write,lseek,ioctl: locking mutex,complete, wait queue
11	Char Module(dynamic device number) with file operations: open,release,read,write,lseek,ioctl: locking spinlock , complete, wait queue
12	Module with bus, class driver, device attributes , compilation Makefile ,
13	Module resource / io resource, compilation Makefile
14	Module : PCI Client driver: Information
15	Module : USB Client driver: Information
16	Module : I2C,SPI Client driver: Information
17	Char Module(dynamic device number) with file operations: open,release,read,write,lseek,ioctl,**mmap**: locking semaphore,complete, wait queue
18	Module: memory allocation methods: kmalloc, vmalloc, free page
19	Module: dma memory allocation methods

KERNEL : DRIVERS DIRECTORY:

kernel directory structure:	arch , block , crypto, Documentation , drivers, firmware, fs, include, init,ipc, kernel, mm, net, samples, security, sound, scipts, tools, user,virt
irqchip drivers(controller)	drivers/irqchip
pinctrl drivers directory(controller)	drivers/pinctrl/
i2c drivers directory(controller)	drivers/i2c
spi drivers directory(controller)	drivers/spi
gpio drivers directory(controller)	drivers/gpio
Video drivers core directory(controller)	drivers/media/v4l2-core
media controller directory	drivers/media/ files: media-device.c media-devnode.c media-entity.c
input subsystem directory	drivers/input drivers/input/touchscreen
rtc subsystem(controller)	drivers/rtc
mmc susbsytem(controller)	drivers/mmc
serial driver (controller)	drivers/tty

LDD3 : Kernel Export Functions

S.NO	EXPORT FUNCTION	FILE NAME
1	register_chrdev_region alloc_chrdev_region unregister_chrdev_region cdev_alloc cdev_init cdev_add cdev_del	fs/char_dev.c
2	Insmod , rmmod	kernel/module.c
3	request_threaded_irq, free_irq	kernel/irq/manage.c
4	up,down,down_interruptible,down_killable,down_trylock, down_timeout	kernel/locking/semaphore.c
5	mutex_unlock,mutex_trylock, mutex_lock	kernel/locking/mutex.c
6	fork,vfork,clone	kernel/fork.c
7	tasklet_init, tasklet_kill, __tasklet_schedule, __tasklet_hi_schedule, irq_exit (invoke_softirq)	kernel/softirq.c
8	__init_work, destroy_work_on_stack, queue_work_on, queue_delayed_work_on, flush_workqueue, flush_work	kernel/workqueue.c
9	Kmalloc, kfress, kzfree	mm/slab_common.c
10	Vmalloc, vfree	mm/vmalloc.c
11	init_completion, wait_for_completion, complete, complete_all	kernel/sched/completion.c
12		
13		
14		
15		

16		

Module:

Module state
[include/linux/module.h]
	enum module_state {
	  MODULE_STATE_LIVE, /* Normal state. */
	  MODULE_STATE_COMING, /* Full formed, running module_init. */
	  MODULE_STATE_GOING, /* Going away. */
	  MODULE_STATE_UNFORMED, /* Still setting it up. */
	};

struct module
[include/linux/module.h]
	struct module {
	        enum module_state state;
	
	        /* Member of list of modules */
	        struct list_head list;
	
	        /* Unique handle for this module */
	        char name[MODULE_NAME_LEN];
	
	        /* Startup function. */
	        int (*init)(void);
	
	        /* Destruction function. */
	        void (*exit)(void);
	}

module_init and module_exit	/* Each module must use one module_init(). */
	#define module_init(initfn) \\
	        static inline initcall_t __maybe_unused __inittest(void) \\
	        { return initfn; } \\
	        int init_module(void) __attribute__((alias(#initfn)));
	
	/* This is only required if you want to be unloadable. */
	#define module_exit(exitfn) \\
	        static inline exitcall_t __maybe_unused __exittest(void) \\
	        { return exitfn; } \\
	        void cleanup_module(void) __attribute__((alias(#exitfn)));
	
	#endif

	/* Generic info of form tag = "info" */ #define MODULE_INFO(tag, info) __MODULE_INFO(tag, tag, info) /* For userspace: you can also call me... */ #define MODULE_ALIAS(_alias) MODULE_INFO(alias, _alias) #define MODULE_LICENSE(_license) MODUINFO(license, _license) #define MODULE_AUTHOR(_author) MODULE_INFO(author, _author) #define MODULE_DESCRIPTION(_description) MODULE_INFO(description, _description)

insmod
kernel/module.c
	finit_module,load_module,do_init_module,
	  /* Start the module */
	   if (mod->init != NULL)
	                ret = do_one_initcall(mod->init);

rmmod
kernel/module.c
	delete_module, find_module
	 if (mod->exit != NULL)
	                mod->exit();
	
	free_module

	static struct module_attribute *modinfo_attrs[] = {
	        &module_uevent,
	        &modinfo_version,
	        &modinfo_srcversion,
	        &modinfo_initstate,
	        &modinfo_coresize,
	        &modinfo_initsize,
	        &modinfo_taint,
	#ifdef CONFIG_MODULE_UNLOAD
	        &modinfo_refcnt,
	#endif
	        NULL,
	};

Char driver

fs/char_dev.c
include/linux/cdev.h
	static struct char_device_struct {
	        struct char_device_struct *next;
	        unsigned int major;
	        unsigned int baseminor;
	        int minorct;
	        char name[64];
	        struct cdev *cdev; /* will die */
	} *chrdevs[CHRDEV_MAJOR_HASH_SIZE];
	
	EXPORT_SYMBOL(register_chrdev_region);
	EXPORT_SYMBOL(unregister_chrdev_region);
	EXPORT_SYMBOL(alloc_chrdev_region);
	EXPORT_SYMBOL(cdev_init);
	EXPORT_SYMBOL(cdev_alloc);
	EXPORT_SYMBOL(cdev_del);
	EXPORT_SYMBOL(cdev_add);

Open	/* * Called every time a character special file is opened */ static int chrdev_open(struct inode *inode, struct file *filp)

copy_to_user, copy_from_user	check_copy_size, _copy_to_user, _copy_from_user, access_ok,raw_copy_to_user, raw_copy_from_user, arm_copy_from_user, arm_copy_to_user

Semaphore: locking mechanisam

struct semaphore
include/linux/semaphore.h
	struct semaphore {
	    raw_spinlock_t lock;
	    unsigned int count;
	    struct list_head wait_list;
	};

extern void down(struct semaphore *sem);
extern int down_interruptible(struct semaphore *sem);
extern int down_killable(struct semaphore *sem);
extern int down_trylock(struct semaphore *sem);
extern int down_timeout(struct semaphore *sem, long jiffies);
extern void up(struct semaphore *sem);
kernel/locking/semaphore.c
	#define __SEMAPHORE_INITIALIZER(name, n) \\
	{ \\
	    .lock = __RAW_SPIN_LOCK_UNLOCKED((name).lock), \\
	    .count = n, \\
	    .wait_list = LIST_HEAD_INIT((name).wait_list), \\
	}
	
	#define DEFINE_SEMAPHORE(name) \\
	    struct semaphore name = __SEMAPHORE_INITIALIZER(name, 1)
	
	static inline void sema_init(struct semaphore *sem, int val)
	{
	    static struct lock_class_key __key;
	    *sem = (struct semaphore) __SEMAPHORE_INITIALIZER(*sem, val);
	    lockdep_init_map(&sem->lock.dep_map, "semaphore->lock", &__key, 0);
	}

	__down_common(sem, TASK_UNINTERRUPTIBLE, MAX_SCHEDULE_TIMEOUT); __down_common(sem, TASK_INTERRUPTIBLE,

	MAX_SCHEDULE_TIMEOUT);
	
	__down_common(sem, TASK_KILLABLE, MAX_SCHEDULE_TIMEOUT);
	
	__down_common(sem, TASK_UNINTERRUPTIBLE, timeout);

	struct semaphore_waiter {
	        struct list_head list;
	        struct task_struct *task;
	        bool up;
	};

	static noinline void __sched __up(struct semaphore *sem)
	{
	        struct semaphore_waiter *waiter = list_first_entry(&sem->wait_list,
	                                        struct semaphore_waiter, list);
	        list_del(&waiter->list);
	        waiter->up = true;
	        wake_up_process(waiter->task);
	}

	* down_trylock - try to acquire the semaphore, without waiting * @sem: the semaphore to be acquired * * Try to acquire the semaphore atomically. Returns 0 if the semaphore has * been acquired successfully or 1 if it it cannot be acquired.

include/linux/spinlock.h -	generic spinlock/rwlock declarations

include/linux/spinlock_types.h	typedef struct spinlock {
	        union {
	                struct raw_spinlock rlock;
	        };
	} spinlock_t;
	
	#define __SPIN_LOCK_INITIALIZER(lockname) \\
	        { { .rlock = __RAW_SPIN_LOCK_INITIALIZER(lockname) } }

	#define __SPIN_LOCK_UNLOCKED(lockname) \\ (spinlock_t) __SPIN_LOCK_INITIALIZER(lockname) #define DEFINE_SPINLOCK(x) spinlock_t x = __SPIN_LOCK_UNLOCKED(x)

Completion:

Waiting for completion is a typically sync point, but not an exclusion point

struct completion
kernel/sched/completion.c
	struct completion {
	        unsigned int done;
	        wait_queue_head_t wait;
	};

#define init_completion(x) __init_completion(x)	static inline void __init_completion(struct completion *x)
	{
	        x->done = 0;
	        init_waitqueue_head(&x->wait);
	}

	void complete(struct completion *x)
	{
	        unsigned long flags;
	
	        spin_lock_irqsave(&x->wait.lock, flags);
	
	        if (x->done != UINT_MAX)
	                x->done++;
	        __wake_up_locked(&x->wait, TASK_NORMAL, 1);
	        spin_unlock_irqrestore(&x->wait.lock, flags);
	}
	EXPORT_SYMBOL(complete);
	
	__wake_up_common ====>

	wait_for_common,__wait_for_common,do_wait_for_common void __sched wait_for_completion(struct

	completion *x)
	{
	        wait_for_common(x, MAX_SCHEDULE_TIMEOUT, TASK_UNINTERRUPTIBLE);
	}
	EXPORT_SYMBOL(wait_for_completion);

Wait queue:

kernel/sched/wait.c	struct wait_queue_head {
	        spinlock_t lock;
	        struct list_head head;
	};
	typedef struct wait_queue_head wait_queue_head_t;

	#define init_waitqueue_head(wq_head) \\
	        do { \\
	                static struct lock_class_key __key; \\
	                                                                                \\
	                __init_waitqueue_head((wq_head), #wq_head, &__key); \\
	        } while (0)

	void __init_waitqueue_head(struct wait_queue_head *wq_head, const char *name, struct lock_class_key *key)
	{
	        spin_lock_init(&wq_head->lock);
	        lockdep_set_class_and_name(&wq_head->lock, key, name);
	        INIT_LIST_HEAD(&wq_head->head);
	}

	#define wake_up(x) __wake_up(x, TASK_NORMAL, 1, NULL) __wake_up_common_lock __wake_up_common

Function call	**located**	**Code flow / used**
insmod		finit_module, copy_module, mod->init
rmmod	kernel/module.c	delete_module, find_module, mod->exit, free_module.

request_threaded_irq	kernel/irq/manage.c	irq_to_desc, __setup_irq
		struct irqaction *action;
		struct irq_desc *desc;

free_irq	kernel/irq/manage.c	irq_to_desc, __free_irq,
		free(action);
		return devname;

alloc_chrdev_region, register_chrdev_region, cdev_alloc,cdev_init, cdev_add	fs/char_dev.c	Static struct **char_device_struct** * __register_chrdev_region
open	fs/char_dev.c	chrdev_open
Read,write	fs/read_write.c	
static DEFINE_SPINLOCK(cdev_lock)	fs/char_dev.c	chrdev_open, cd_forget, cdev_purge
static DEFINE_MUTEX(chrdevs_lock);	fs/char_dev.c	chrdev_show, __unregister_chrdev_region, __register_chrdev_region

* %GFP_USER -	Allocate memory on behalf of **user**. **May sleep.**

* %GFP_KERNEL -	Allocate **normal kernel ram**. **May sleep.**
* %GFP_ATOMIC - *	**Allocation will not sleep.** May use **emergency pools**. *For example, use this inside interrupt handlers.
* %GFP_HIGHUSER - *	Allocate **pages from high memory.**
* %GFP_NOIO - *	Do not do any I/O at all while trying to get memory.
* %GFP_NOFS - *	Do not make any fs calls while trying to get memory.
* %GFP_NOWAIT - *	**Allocation will not sleep.**
* * %GFP_DMA - *	**Allocation suitable for DMA.** * **Should only be used for kmalloc() caches**. Otherwise, use a * slab created with **SLAB_DMA.** * * Also it is possible to set different flags by OR'ing * in one or more of the following additional @flags:
* %__GFP_HIGH - *	This allocation has high priority and may use emergency pools.
* %__GFP_NOFAIL - *	Indicate that this allocation is in no way allowed to fail * (think twice before using).
* %__GFP_NORETRY -	If memory is not immediately available, * then give up at once.

* * %__GFP_NOWARN - * *%__GFP_RETRY_MAYFAIL -	If allocation fails, don't issue any warnings. Try really hard to succeed the allocation but fail * eventually.

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        __poll_t (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*release) (struct inode *, struct file *);
};
```

LDD3:   Device Driver Oral Questions

1) What is device driver and what is the need of it?

Device Driver is a program which interacts and control your hardware devices present in a system. For newly developed devices or enhancing of some feature for existing device , we need to write device driver.

2) What are different kind of devices?

There are three kind of devices available in market

1. character devices
2. block devices
3. network devices

3) What is module in Linux.

Kernel module is the piece of code which can be loaded and unloaded into kernel upon demand. it means you can load your code or module into kernel whenever it required and you can unload your code and module from kernel whenever you do not required.

4) How modules are loaded in Linux.

modules can be loaded dynamically by following command

- insmod
- modprobe

5) Difference between insmod and modprobe.

Both commands are used to insert kernel module dynamically . modprobe first loads the modules on which new module has dependency but insmod doesn’t check any dependency .

6) How parameters are shared between driver modules.

different parameters can be shared by exporting parameters in global space which can be done by

EXPORT_SYMBOL(sym)

7) What are IOCTLS.

Linux provide us a generic call which can be used to control any device which is called **IOCTL**or input output control. As the name suggest by using of ioctl we can write to the device as well as we can read from the device also.

8) What is syscalls

System Calls provide an interface between kernel service and user space. System calls are required when we have to use any service of kernel. Suppose we want to change the date and time of system in that case we need to request to kernel through Syscalls.

9) What are the benefits of syscalls.

It serves mainly three purpose :-
1) Provide abstract hardware interface.
2) Security and stability of our system

3)It provide a ritualized system.

10) How character driver is registered in Linux.

We can register driver by following function

1. register_chrdev_region / alloc_chrdev_region
2. cdev_alloc
3. cdev_init
4. cdev_add

11) What is init and exit function of a driver

our driver needs to interact with kernel. So driver has to notify to kernel that it also exist in the system. So to notify to the kernel driver has init calls which is used for hooking our driver routines into kernel. Besides notifying or registering with kernel it also allocates any kernel memory required for device and initialize hardware. This function is called at boot time.

In the same way we need to detach our driver from the system when our system is shutting down . To serve this purpose driver use Exit call which is called by kernel at the time of shutting down of system. This call unregister driver or free all the resources it allocated at the time of init call.

12) How and when init and exit function of driver get called.

init function is called at the time of module load and **exit is called** at the time of unloading of module.

13) What is probe function?

probe function is used for bus based devices . **Probe function pointer is initialized in the init function** . Probe function is used to initialize hardware, allocating resources and registering the devices with the kernel.

14) When probe is get called.

Whenever device compatible name match with driver name (of match pointer) initialized at boot time , probe function is called.

15) What is platform devices.

Platform are the devices which are non discoverable at the hardware level and connected to a virtual bus i.e platform bus.

16) What is device tree.

Linux use a data structure to represent connected devices , this data structure is called device tree.

17) What are the benefits of device tree over board files.

following are the advantages of device tree

- Simple to change the configuration without compiling any source code.
- can easily add support for new hardware
- provide easy to use and easy to read description of hardware.

18) What is sysfs and procfs?

procf is a special file system which is used to avail information about processes , and to change kernel parameter at run time. it is mounted as /proc at the boot time. Its an old interface between kernel space and user space.

Sysfs is a virtual file system provided by Linux kenel to expose information about Hardware devices, kernel subsystems and driver at run time. It also provide capability to pass some controlling commands to some hardware devices.

19) How logs are printed in Linux kernel and what are the logs level available in Linux.

In Linux kernel logs are printed by special function printk() ,which is just like our printf function . There are different types of log level available in Linux kernel , below is the list of Log levels

0 (KERN_EMERG)	system is unusable
1 (KERN_ALERT)	action must be taken immediately
2 (KERN_CRIT)	critical conditions
3 (KERN_ERR)	error conditions
4 (KERN_WARNING)	warning conditions
5 (KERN_NOTICE)	normal but significant condition
6 (KERN_INFO)	informational
7 (KERN_DEBUG)	debug-level messages

20) What is copy_to_user and copy_from_user.

copy_to_user is a function which is used to copy some data from kernel space to user space.

copy_from_user is a function which is used to copy data from user space to kernel space.

21) What do you mean by kernel configuration and what are the various way of configuring kernel.

Kernel which we get from Kernel.org is generic kernel which we can build and install on different hardware platform . But before building Linux kernel we need to customize or configure the Linux kernel or we can say we need to do some platform specific changes to port our linux kernel on our hardware platform.

For configuring the kernel go into root directory of Kernel uncompressed folder and use one of the following way of Kernel Configuration.

Linux kernel provides us various way of configuring the Linux Kernel. Following are some method to configure the Linux kernel

1. make config
2. make menuconfig
3. make defconfig

22) What is menuconfig

Linux provides a graphical view of configuring kernel , menuconfig. when we run “make menuconfig” command we get a GUI on which we can see multiple kernel settings.

23) What is ioremap

ioremap is the function provided by Linux kernel to map physical memory into kernel virtual address space . To access any hardware device first we need to map that device into kernel virtual address then only we can access that device.

24) What is segmentation fault.

segmentation fault is an error generated by Linux kernel , whenever any process tries to access memory location for which it doesn’t have permission to do so.

25) What are the various ways of debugging Linux kernel.

There are various way of debugging linux kernel . GDB, KGDB, printk statements etc are the method of software debugging while JTAG debugging is one of the hardware debugging.

26) What is zimage and bzimage.

zImage :- Its a compressed image format of Linux kernel which is self extracting. We can build it by giving following command while building kernel

make zImage

bzImage:- Its also a compressed format of Linux kernel image but this image is much compressed as compare to zImage.

27) How parameter are passed from boot loader to kernel?

Parameters are passed to linux kernel from bootloader by ATAG.

28) What is ATAGS

ATAGS is way of sending different command at boot time to send different parameters from boot loader to Linux kernel.

29) From which file kernel execution starts.

Linux kernel execution starts from head.s file.

30) What is bootloader

Bootloader is a program or code which is responsible for early stage initialization during booting and loading linux kernel into memory.

33) What is primary and secondary bootloader

Primary bootloader is small piece of code which executes from ROM and does core initialization and pass control to secondary stage boot loader which is a program which executes from RAM and does further initialization and load linux kernel to memory.

34) Why we need two bootloader

Because primary bootloader executes from ROM and the size of ROM is too small to accommodate all the code required for initialization , hence we need secondary boot loader to perform rest of the initialization.

35) Difference between poll and select

In select you have limit on fds on which you can have a watch but in case of poll no limit on number on fds.

36) What is priority inheritance and priority inversion.

Priority inversion is a situation in which a resource is acquired by low priority process, which high priority task wants to acquire and in between some medium priority task preempts low priority task which was holding the resources.

To solve this priority inversion problem , the solution is to increase the priority of process which is holding resource to the max priority. which is called priority inheritance.

37) What are different type of kernel?

There are three types of Linux kernel

1. Micro kernel
2. Monolithic Kernel
3. Hybrid Kernel

38) What is DMA

DMA stands for Direct memory access . Its a method which allows different input output device to send and receive data directly to or from main memory by bypassing CPU to speed up the data transaction .

39) What is cache coherency

cache coherency is a way to maintain consistency of data stored or shared in different local cache.It ensures that changes in the values of shared data are propagated throughout the system in a timely fashion.

40) What is copy on write

Usually when we call the fork a new address space is created and parents data is copied into that but it’s a wastage of huge memory . To avoid this Linux implement fork() in such a way that same process data is shared between process and child until child write something new or updates . If child tries to write something then the copy of parent process data is created.

41) What is highmem and lowmem

Kernel virtual memory is divided into two part Highmem and lowmem.

Highmem is the memory which contains memory which is not permanently mapped to physical memory. It generally contains Process related stuff or kernel module specific stuff.

Lowmem is the memory section which is permanently mapped and never swaps out. It mostly contains kernel image with other core things.

42) What happens if we pass invalid address from userspace by

using ioctls

If you pass invalid address from user space to kernel then it might lead to severe problems like crash , data overriding and loosing of some data etc.

43) What are different ipc mechanism in Linux

sockets , Pipes , Shared memory ,signal, message queue, semaphore , mail box

44) what are sockets?

Socket is in IPC mechanism supported by linux kernel to provide communication between different process. Its basically a logical endpoint for communication between different process.

45) How page fault is handled in Linux?

Whenever the kernel tries to access an address that is currently not

accessible or that particular page is not available in memory, the CPU generates a page fault exception and calls the

page fault handler

void do_page_fault(struct pt_regs *regs, unsigned long error_code)

Page fault handler then first load the required page from some secondary memory to main memory and modifies the page tables accordingly. After modifying page tables it notifies to CPU to continue .

47) What is I2c and SPI.

The Inter-integrated Circuit (I2C) Protocol is a protocol intended to allow multiple “slave” digital integrated circuits (“chips”) to communicate with one or more “master” chips.

read more from

https://learn.sparkfun.com/tutorials/i2c

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between CPU and peripherals such as registers, sensors etc. It uses clock and data lines, along with a select line to choose the device you wish to communicate.

48) How physical to virtual translations works in Linux.

phys_to_virt(phys_addr) api can be used for this purpose.

49) What is thrashing, segmentation and fragmentation.

Thrashing is a state when process of swap in and swap out of page to or from main memory is consuming most of the CPU time. This state occurs whenever successive page fault occurs in very small duration.

As a result of dynamic memory allocation , memory is divided into chunks of free memory and there might be scenario that we are not able to satisfy memory allocation request because size of free memory chunks is too small but overall total free space is large than memory required is called fragmentation.

segmentation is a process of dividing memory into different sections or segments or we can say a logic unit such as main program , procedure , function , method etc.

50) What is preempt_count and what is the need of that.

Preempt_count is a variable associated with each process , which maintain count of number of lock hold by particular process . Default value of preempt_count is zero. It increase by one when process acquire a lock or decrease by one when process release the lock. Linux kernel check its value before it preempt a process to make ensure process is not holding any lock

51) What is synchronization

Process of limiting simultaneous access to shared resources by more than one process is called kernel synchronization. synchronization ensure that only one process will get access to shared data at a time.

52) What is critical section

it’s a part of code which cannot be accessed by more than one process at a time to avoid any inconsistency of shared data or we can say this part of code access and manipulate shared data.

53) What is race condition

When more than one process access the same critical at same time , its called race condition.

54) Why we need to take care of synchronization

If we avoid the synchronization then there are chances that shared data might be manipulated or modified by more than one process . Which can leads to inconsistency of data.

55)What is various synchronization techniques in Linux.

Linux kernel provide various synchronization techniques

1. Semaphore
2. Mutex
3. Spinlock
4. read write spin lock
5. read write semaphore

56) What is deadlocks.

A deadlock is a condition involving one or more threads of execution and one or more resources, such that each thread waits for one of the resources, but all the resources are already held. The threads all wait for each other, but they never make any progress toward releasing the resources that they already hold. Therefore, none of the threads can continue, which results in a deadlock.

57) What is atomic operations.

Atomic operations provide which executes atomically , without interruption. It means atomic operations cannot be interrupted in between by some other process.

58) What is spin locks

Spin lock is one of the synchronization method in which a process has to acquire a lock before accessing critical section . If lock is already held by some other process then our process which is requesting to enter in critical section ,has to wait for releasing of lock. Waiting process will keep on executing on the processor and will be waiting for lock release.

59) What is reader-writer spin lock.

Reader writer spin lock is an advancement to spinlock. In reader writer spin lock more than process can acquire lock at the same time if they just want to read the shared data. But Only one process will get lock if they wants to write or modify the data.

60) What is semaphore.

Semaphore is one of the synchronization technique supported by linux kernel. In semaphore only one process can get lock to access the same critical section at a time. If a process tries to acquire a lock for a critical section which is already held by some other process then our process which is requesting has to wait and it will go for sleep till the lock is released by some other process.

61) What is binary semaphore

Binary semaphores can take only 2 values (0/1). They are used to acquire locks. When a resource is available, the process in charge set the semaphore to 1 else 0.

62) What is difference between semaphore and spin lock

In semaphore , if lock is already held by some other process then process goes into sleep state till the time lock is available. But in case of spinlock process who made the request of lock will be keep on executing on processor in busy loop and will be waiting for lock to be released.

63) When to choose what among spin lock and semaphore.

If you are sure that your process will get lock in very short period of time then spinlock is recommended to avoid context switching overhead. But if your process has to wait for long to get a lock then semaphore is the preferred way of implementing synchronization.

64) What is difference between semaphore and mutex.

Both are the techniques for maintaining synchronization . Mutex ensure mutual exclusion . It means a process who acquired the lock that only can release the lock. No other high priority task can release the lock acquired by some other process in mutex. But in case of semaphore any high priority process can release the lock acquired by some low priority process.

65) What is preemption disabling and what is the use of this.

As Now Linux kernel is preemptive . Any task can be preempted by some high priority task. Preemption disabling is used to disable the preemption of low priority task by other high priority task. Preemption disabling is used for implementing spinlock on uniprocessor system.

66) How memory is managed in Linux.

Memory in Linux , managed by memory management subsystem in Linux kernel which takes care of all the memory related stuff like memory allocation, page allocation, virtual memory management , memory mapping etc.

67) What are pages

basic unit of memory management in Linux kernel is pages. Physical memory is divided into pages which is of size 4kb in case of 32 bit system .

68) What are different memory zones in Linux.

Linux kernel divides memory into four zones

ZONE_DMA:	This zone contain pages that is used for DMA operations
ZONE_DMA32:	This zone is just like ZONE_DMA but this can be accessed only by 32-bit devices.
ZONE_NORMAL:	This Zones contain normal regularly mapped page.
ZONE_HIGHMEM:	This zone contains pages that are not permanently mapped into kernel's address space

69) How to allocated pages.

below are the APIs provided by Linux kernel for allocating pages

- alloc_pages()
- get_free_page()
- get_zeroed_page()
- get_free_pages()

70) How to freeing page

Page can be freed by below APIs in linux kernel

- __free_pages()
- free_pages()
- free_page()

71) What is kmalloc and what are action modifier we can pass while using kmalloc?

kmalloc is an api in Linux kernel which is used to allocate memory in Linux kernel. Action modifiers is one of the argument need to passed while allocating memory by using of kmalloc. Action modifiers specifies how the Linux kernel is supposed to allocate the requested memory. below are some of the action modifiers

- __GFP_WAIT
- __GFP_HIGH
- __GFP_IO
- __GFP_KERNEL
- __GFP_DMA

72) What is zone modifier in Linux.

Zone Modifier specifies from where to allocate memory. Is tells from which zone memory is to be allocated.

73) What is vmalloc

vmalloc is also an API just like kmalloc which is used to allocate memory. Vmalloc ensure memory allocated will be virtually contiguous but physical memory non contiguous

74) What is interrupt ?

Interrupts enable hardware to signal to processor. When CPU is busy in doing or processing something at that time any hardware can request to

CPU for some service. CPU saves it current state and serves the hardware request. After full filling the request of hardware CPU restore its previous state and start processing previous that it was doing before receiving from hardware.

75) What is interrupt handler or ISR ?

This is the function which kernel runs in response to a specific interrupt is called interrupt handler or ISR. This function actually handle and process the request of interrupts. Interrupt handler are mainly divided into two parts

76) What is top halves and bottom halves ?

Interrupt handling is divided into two parts

1) Top Halves:- Its executed as immediate response to interrupt.

2)Bottom Halves:- Its executed some time later when CPU get free time.

77) How interrupt is registered ?

It is the responsibility of device driver to write the interrupt handler and take care of interrupt request generated from its device. Kernel provide below function to register an interrupt.int request_irq()

78) What are different interrupt handler flags ?

Below are the different Interrupt handler flags and their usageIRQF_DISABLED:- when this flag is set. It indicates the kernel to disable all interrupts when executing this interrupt handler. when unset , interrupt handler run with all interrupts except their own enabled.IRQF_TIMER:- this flag specify that this handler processes interrupts for the system timer.IRQF_SHARED:- This flag specifies that interrupt line can be shared among multiple interrupt handler.

79) How interrupt are freed ?

Below is the function availed by kernel to free the interrupt handler.

void free_irq(unsigned int irq, void *dev)

80) What are the considerations needs to taken care while writing interrupt handler ?

In writing interrupt handler we need to think about what are the services our device may request. kernel provide below syntax of interrupt handler. While writing interrupt handler we should not use any system calls which may lead to call sleep. As we are not allowed to call sleep while handling interrupt.

81) What is interrupt context ?

Linux Kernel execution on behalf of any interrupt is called interrupt context.

82) How to disable and enable interrupts ?

Below are the functions provided by Linux kernel for disabling and enabling of interrupts.

83) What are different bottom halves techniques in linux ?

1) Original bottom Half

2)Task Queues

3) Softirq

4)Tasklets

5)workqueues

84) When to choose which bottom halves ?

There are many considerations to be taken while deciding which bottom halves to be usedIf you don’t need to maintain any synchronization between execution of code on different processor then go for softirq but

If you need to maintain sync , go for tasklets.

If your code required to go into sleep state then workqueue is the available option.

85) How to schedule tasklet ?

Tasklets are scheduled via tasklet_schedule() which takes pointer to tasklet_struct.after tasklet is scheduled , it runs once at some time in the near future.

86) How to disable bottom halves ?

Local_bh_disable() and local_bh_enable() are the APIs provided by kernel to disable and enable bottom halves respectively.

87) what is process scheduling ?

selecting which process to give chance to run is called Process scheduling. It is responsibility of processor to decide which process to run, when and for how long. For multiprogramming system scheduler is base . Scheduler has to decide which process to run and which process has to wait. Scheduler has to allot the time to each process efficiently and make the efficient use of Processor so that user can feel that all the processes is executing at the same time.

88) What is cooperative multitasking and pre-emptive multitasking?

Co-operative multitasking:-In this method process does not stop running until it voluntarily stops. The act of process voluntarily stopping itself called yielding.

Preemptive multitasking:-In this kind of multitasking scheduler decides which process to start running and which process to stop. Good thing in this multitasking is that scheduler can take the global decision. All the linux kind OS comes in preemptive scheduling flavor.

89) What is yielding?

The act of process voluntarily stopping itself called yielding.

90) What is limitation of cooperative multitasking?

- Scheduler cannot take globalize?? decision.
- Any process can monopolize the system.

91) I/O bound versus Processor bound process?

I/O bound processes spend most of the time waiting for some input/output event to occur and take very less time of processor. Processor bound process takes most of the time executing the code . These kind of process runs continuously without waiting for any input/output event and stopped while printed by compiler.

92) What is process priority?

Process priority is a way of specifying the process importance in the system.

93) What kind of priority is maintained in Linux?

Linux implements two types of process priority value.

1) Nice value:-A number from -20 to +19 with a default value of 0. Large nice value corresponds to less priority.
2) Real Time Priority:-A value from 0 to 99. Higher the real time value means higher priority.

94) What is virtual run time?

The Vruntime variable stores the virtual runtime of a process is the actual runtime (the amount of time spent running) normalized by the number of runnable processor. CFS uses Vruntime to account for how long a process has run and thus how longer it ought to run.

95) What are the available scheduling classes in linux?

- Completely Fair Schedule class
- RT schedule class

96) How next task is picked for scheduling ?

CFS (complete fair scheduler)search the leftmost child of RB tree which has smallest Vruntime. It is performed by pick_next_entity(). If it returns NULL , it means there is no task to run so schedules the idle task.

97) what is scheduler entry point in Linux?

The main entry point in the process schedule is the function schedule() defined in **kernel/sched.c** .This function is called by rest of kernel to schedule another process.schedule() is a generic call and it internally calls each scheduler class and check highest priority task from highest priority scheduler class. It?? is done by pick_next_task().

98) what is waitqueus?

Its a simple list of process waiting for an event to occur. Waitqueue are represented in the kernel by wake_queue_head_t. Waitqueue are created statically via DECLARE_WAITQUEUE() or dynamically via init_waitqueue_head() .

99) How context switching is handled in Linux?

Switching from one runnable process to another runnable process is called Context switching. In Linux context switching is handled by context_switch() function. This function defined in **/kernel/sched.c** . When some process terminates then context_switch() is called by schedule() which does two basic things:-

1) switch_mm() :-This call is to switch the virtual memory mapping of currently running process to upcoming process.
2)switch_to():-This call is to switch the processor state from previous process to current process. It save the current info of current processor state register.

100) What is user preemption and kernel preemption?

User Preemption

When the kernel is about to return to user-space , need-reschedule is set and therefore the scheduler is invoked.

Kernel Preemption

Any task in the kernel which is not holding any lock can be preempted. In Linux we maintain a variable preempt_count in the thread_info structure. When process acquired a lock the preempt_value is increased by 1 and vice versa. whenever this value is 0 we can preempt the kernel.

1) How to manipulate the current process states?

Linux kernel provide set_task_state(task,state) by which you can manipulate the state of given task. By using of set_task_state you can change to state of given task.

2) What are kernel thread?

Kernel perform some background operation by using of some thread that is called kernel thread. kernel thread are the special thread which does not have the process address space .They just run in the kernel space . We can reschedule kernel thread like a normal process .Kernel delegates several task to kernel thread like flush etc.

3) How threads are implemented in Linux kernel?

In Linux there is no special provision for thread . In Linux thread is a process which actual share some resources with other process . Every thread has a unique task_struct. **creating thread**Thread also created by using the clone sys call except some flags for sharing the data like file system , signal handler,open files also set. Flags are used to specify that which resources are shared between parent and child process or thread.

clone(CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0);

4) What are different states of a process in Linux?

In Linux a process can be in one of the following state.

1)TASK_RUNNING :- This state specify that process is either running or in ready state and waiting in runqueue .
2)TASK_INTRUPTTIBLE:- This condition states that the task is in sleeping state and waiting for some event to occur. When this condition occurs task comes in TASK_RUNNING state.
3) TASK_UNINTRRUPTIBLE:- This state is similar as previous state but in this state process does not wait for any event to occur.
4)_TASK_TRACED:-The process is getting traced by another process , such as a debugger.

5) What is difference between process and thread?

Linux does not differentiate between process and thread. Linux kernel understand a thread as a process and implements process and thread in same way. In Linux thread is a process which actually share some resources with other process .

6) Generally what resources are shared between threads?

Threads can share lots of things . Some of the most shared resources are file system information , open files , signal handlers , Address space etc.

7) What is process descriptor?

In Linux kernel a process is represented by Process descriptor which is a structure of type **task_struct** . Kernel stores all the process by maintaining a doubly linked list in which each node is type of **task_struct**. In other words we can say kernel maintained a doubly linked list of process descriptors.

8) What is task_struct?

The task_struct is a large data structure around 1.7 Kb on a 32 bit machine. This structure maintain all the information about a process. This structure contains info like signals, files ,stack ,state and much more.

10) What was the need of thread_info structure?

The task_struct is a large data structure around 1.7 Kb on a 32 bit machine and kernel stack is either 4KB or 8KB . Hence the task of storing structure of 1.7 kb is very much difficult . So kernel introduced concept of thread_info ,which is very much slimmer than task_thread and just points to task_struct structure.

11) Difference between fork() and vfork() ?

The vfork() system calls has the same effect as fork() except that the page table entries of the parent process are not copied . Instead the child executes as a sole in the parents address space, and the parent is blocked until the child either call exec() or exits.

12) What is process context ?

Normally process runs in user space but when a program calls a syscall or triggers an exception then it enter into the kernel-space at that time kernel is said to be executed on behalf of process and in other word kernel is in process context.

13) What is zombie process?

Process which is dead and completed its execution but still it had an entry in process table is called zombie process.

14) How parent less process is handles in Linux ?

If a parents exist before its children then child process are re-parented to another process in same thread group or , if that fails , then **init process.**

ARM : Oral Questions

1. ARM Vector Table

0x00- Reset Handler

0x04 -Undefined Instruction

0x08 - SWI

0x0C - Prefetch Abort

0X0F -Data Abort

0X14- Reserved

0X18-IRQ Handler

0X1C-FIQ Handler

2. ARM Modes

Privileged Modes (six)

1. system

2. supervisor

3. IRQ

4. FIQ

5. ABORT

6. Undefined

non-privileged Mode

1. user

3. CPSR

N Z C V Q → Flags

I F ---> Interrupt Flags

T --- ARM / THUMB Mode

MODES

reset:

```
	/*	 * set the cpu to SVC32 mode */
	mrs	r0,cpsr
	bic	r0,r0,#0x1f
	orr	r0,r0,#0xd3
	msr	cpsr,r0
```

4. Registers (R0- R15)

R15-PC

R14-LR

R13-SP

R11-FP

R0-R3 = Arguments

5. ARM - RISC

1. Single Cycle Instructions
2. Register Based operations , not memory
3. Performance Based

6. ARM -Pipeline

ARM7: FETCH - DECODE - EXECUTE

ARM9: FETCH - DECODE - EXECUTE - MEMORY - WRITE

ARM10: FETCH - ISSUE - DECODE - EXECUTE - MEMORY - WRITE

7. ARM -CP15: CO-PROCESSOR 15

1. MMU
2. CACHE
3. TLB / TCM
4. VECTOR TABLE ADDRESS

8. LOAD, STORE

LDR ==> LOAD TO REGISTER

STR ==> STORE TO MEMORY

9. ADDRESSING MODE

PRE INDEX	MEM[REG + BASE]	REG=REG+BASE	LDR R0,[R1,#4]!

WITH WRITE BACK			
PRE INDEX	MEM[REG + BASE]	NOT UPDATED	LDR R0,[R1,#4]
POST INDEX	MEM[REG]	REG=REG+BASE	LDR R0,[R1],#4

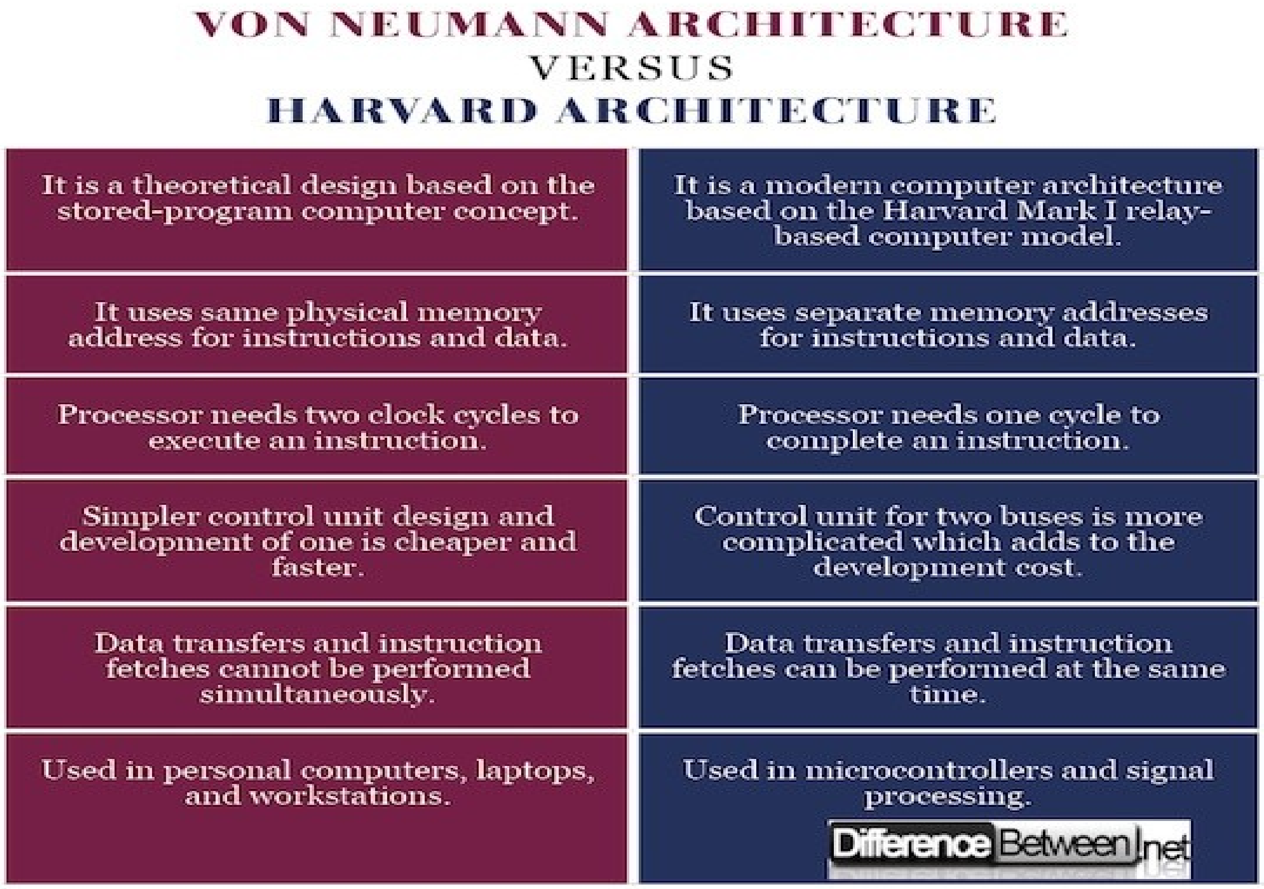
10. LDM,STM [IA,IB,DA,DB]

LDMIA	LDMIA R0!,{R2-R4}
STDIA	STMIA R0!,{R2-R4}

11. FULL STACK, EMPTY STACK

FULL STACK	SP pointer points to the last used or full location
EMPTY STACK	SP pointer points to the first unused or empty location

12 .What is harward and vonumen architecture

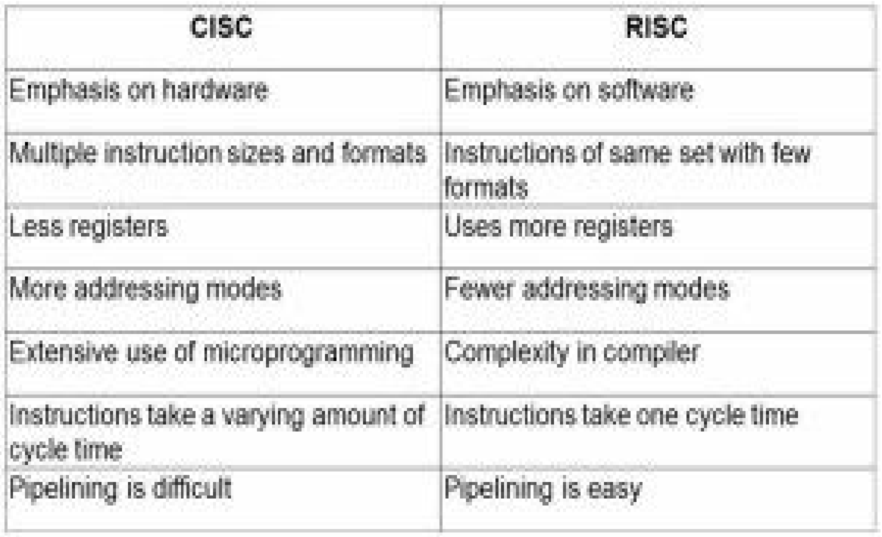


VON NEUMANN ARCHITECTURE
VERSUS
HARVARD ARCHITECTURE

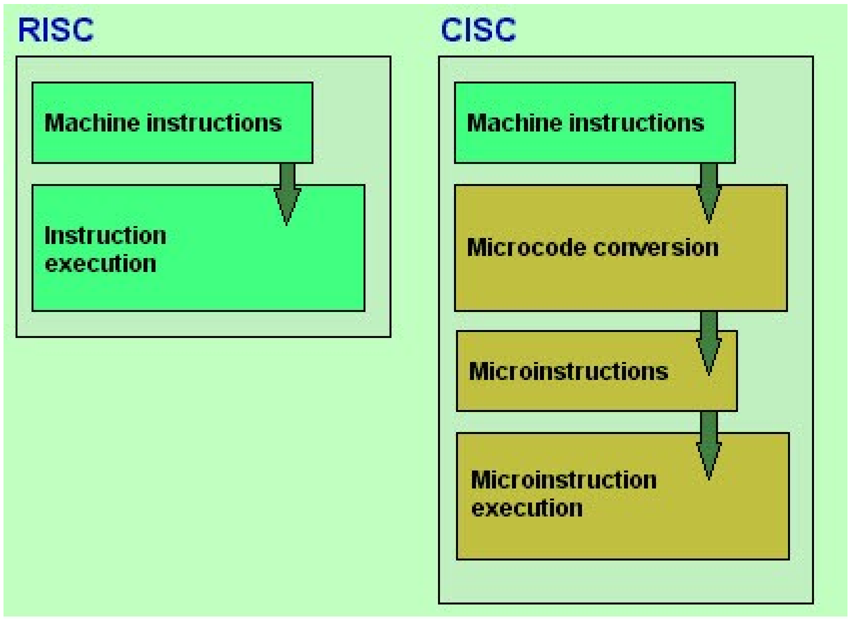
It is a theoretical design based on the stored-program computer concept.	It is a modern computer architecture based on the Harvard Mark I relay-based computer model.
It uses same physical memory address for instructions and data.	It uses separate memory addresses for instructions and data.
Processor needs two clock cycles to execute an instruction.	Processor needs one cycle to complete an instruction.
Simpler control unit design and development of one is cheaper and faster.	Control unit for two buses is more complicated which adds to the development cost.
Data transfers and instruction fetches cannot be performed simultaneously.	Data transfers and instruction fetches can be performed at the same time.
Used in personal computers, laptops, and workstations.	Used in microcontrollers and signal processing.

DifferenceBetween.net

13.What is RISC and CISC instruction set?



CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy



Memory Unit: RISC has no memory unit and uses a separate hardware to implement instructions. CISC has a memory unit to implement complex instructions

Program : RISC has a hard-wired unit of programming. CISC has a microprogramming unit

Design : RISC is a complex compiler design. CISC is an easy compiler design

Calculations : RISC calculations are faster and more precise. CISC calculations are slow and precise

Decoding: RISC decoding of instructions is simple. CISC decoding of instructions is complex

Time: Execution time is very less in RISC. Execution time is very high in CISC.

External memory: RISC does not require external memory for calculations. CISC requires external memory for calculations.

Pipelining: RISC Pipelining does function correctly. CISC Pipelining does not function correctly.

Stalling:RISC stalling is mostly reduced in processors. CISC processors often stall.

Code Expansion : Code expansion can be a problem in RISC whereas, in

CISC, Code expansion is not a problem.

14.What is THUMB, THUMB-2 and ARM mode?

ARM and Thumb are two **different** instruction sets supported by **ARM** cores **with a** “T” in their name. For instance, ARM7 TDMI supports **Thumb mode**. **ARM** instructions are 32 bits wide, and **Thumb** instructions are 16 wide. **Thumb mode** allows for code to be smaller, and can potentially be faster if the target has slow memory.

There are no Thumb coprocessor instructions, no Thumb semaphore instructions, and no Thumb instructions to access the CPSR or SPSR

Thumb-2 technology was introduced in the *ARM1156 core*, announced in 2003. Thumb-2 extends the limited 16-bit instruction set of Thumb with additional 32-bit instructions to give the instruction set more breadth, thus producing a **variable-length instruction set**. A stated aim for Thumb-2 was to **achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory**.

15.what is difference between FIQ/IRQ in arm?

ARM treats FIQ(Fast interrupt request) at a higher priority as compared to IRQ(interrupt request). When an IRQ is executing an FIQ can interrupt it while vice versa is not true.

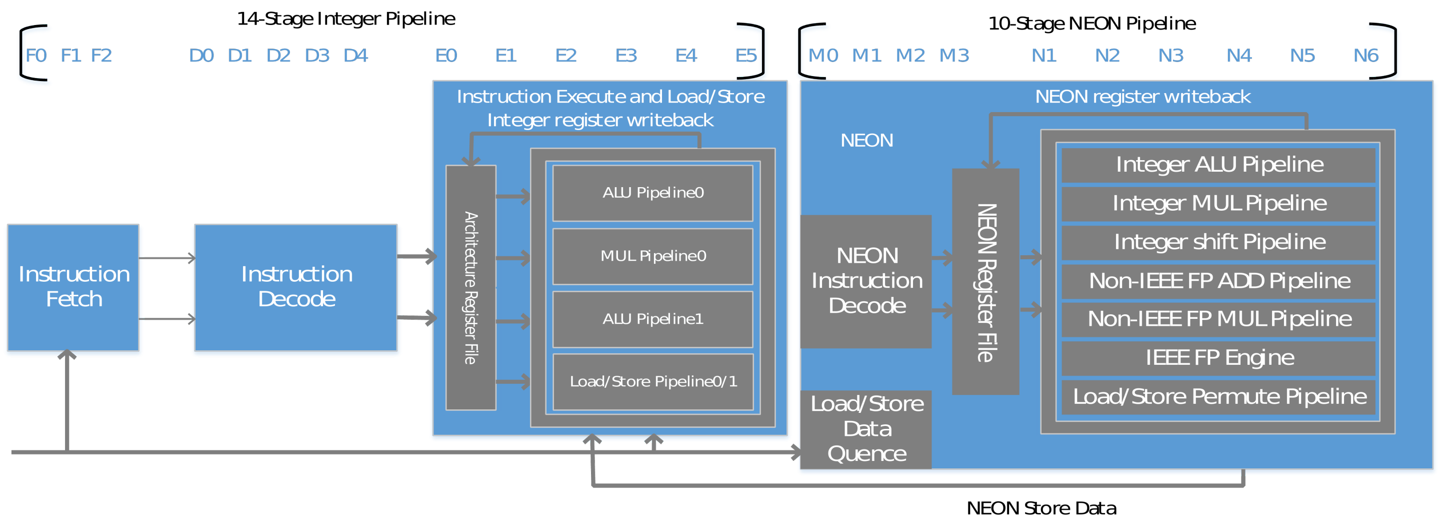
16.what is mean of ARM CORTEX architecture?

ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17

1. Single-issue, in-order micro-architecture with an 8-stage pipeline

ARM Cortex-A9

Designed by	ARM Holdings
Max. CPU clock rate	0.8 GHz to 2 GHz
Microarchitecture	ARMv7-A
Cores	1–4
L1 cache	32 KB I, 32 KB D
L2 cache	128 KB–8 MB (configurable with L2 cache controller)



17.Difference of CORTEX A8,A9,A15?

18. What is the use of 'SWI' in ARM assembly?

The `SWI` instruction causes a SWI exception. This means that the processor state changes to **ARM, the processor mode changes to Supervisor**, the CPSR is saved to the Supervisor Mode SPSR, and execution branches to the SWI vector.

19. What is called 'pipeline bubble' in ARM?

A pipelined processor may deal with hazards by stalling and creating a bubble in the pipeline, resulting in one or more cycles in which nothing useful happens.

Because of the bubble (the blue ovals in the illustration), the processor's Decode circuitry is idle during cycle 3. Its Execute circuitry is idle during cycle 4 and its Write-back circuitry is idle during cycle 5.

When the bubble moves out of the pipeline (at cycle 6), normal execution resumes. But everything now is one cycle late. It will take 8 cycles (cycle 1 through 8) rather than 7 to completely execute the four instructions shown in colors.

20. Tell the 2 software methods available to remove interlocks following load instructions.

21. Tell about 'Load Scheduling by Preloading' and 'Load Scheduling by unrolling'?

22. List the issues when porting C code to the ARM processor?

23. If the pipeline is wider, the instruction throughput is high – True/False?

24. What is out of order execution? Have you considered it in selection of processor?

out-of-order execution, OoOE, is a technique used in most high-performance microprocessors to make use of cycles that would otherwise be wasted by a certain type of costly delay. Most modern CPU designs include support for out of order execution.

The key concept of OoO processing is to allow the processor to avoid a class of delays (termed: *"stalls "*) that occur when the data needed to perform an operation are unavailable.

In earlier **In-order processors**, the processing of instructions is normally done in these steps:

1. Instruction fetch.
2. If input operands are available (in registers for instance), the instruction is dispatched to the appropriate functional unit. If one or more operands is unavailable during the current clock cycle (generally because they are being fetched from memory), however, **the processor stalls until they are available.**
3. The instruction is executed by the appropriate functional unit.
4. The functional unit writes the results back to the register.

Out-of-order processors breaks up the processing of instructions into these steps:

1. Instruction fetch.
2. Instruction dispatch to an instruction queue (also called **instruction buffer** or reservation stations).
3. The instruction waits in the queue until its input operands are available. The instruction is then allowed to leave the queue before earlier, older instructions.
4. The instruction is issued to the appropriate functional unit and executed by that unit.
5. The results are queued (Re-order Buffer).
6. Only after all older instructions have their results written back to the register file, then this result is written back to the register. This is called the **graduation** or **retire** stage.

In the examples outlined above, the **OoO** processor avoids the stall that occurs in step (2) of the i**n-order processor** when the instruction is not completely ready to be processed due to missing data.

25. What is branch prediction?

Branch prediction is a technique used in CPU design that attempts to guess the outcome of a conditional operation and prepare for the most likely result. A digital circuit that performs this operation is known as a branch predictor. It is an important component of modern CPU architectures, such as the x86

Branch prediction is required in the design to reduce the core CPU loss that arises from the longer pipeline. To improve the branch prediction accuracy, a combination of static and dynamic techniques is employed.

26. What is pipeline shutdown

27. What is interrupt pipelining?

28.**What are 1st/ 2nd/3rd/4th generation processors?**

The processors made of PMOS, NMOS, HMOS, HCMOS technology are called 1st/ 2nd/3rd/4th generation processor’s and are made up of 4, 8, 16, 32-bits

29. **What is machine cycle?**

The steps performed by computer processor for each machine language instruction received. The machine cycle is 4 process cycle.

- **Fetch:** Retrieve an instruction from memory
- **Decode:** Translate the retrieved instruction into series of computer commands.
- **Execute:** Execute the computer command.
- **Store:** Send and write the result back into memory.

30. **What is called scratch pad of computer?**

Cache memory is scratch pad of computer.

31. **What is instruction set?**

The set of instructions that the microprocessor can execute.

32. ARM Family attributes

	ARM7	ARM9	ARM10	ARM11
Pipe line -depth	3	5	6	8
Typical Clock MHZ	80	150	260	335
Architecture	Von Neumann	Harvard	Harvard	Harvard

33. What is physical cache vs Virtual Cache ?

Physical Cache is located between MMU and Main Memory

Virtual Cache is located between CPU and MMU

34. What is write-through and write-back policy on cache?

write-through: It writes to both cache and main memory,when there is a cache hit on write, so that cache and main memory are coherent

Write-back: It writes to valid cache data memory and not to main memory, hence cache line and main memory are not sync, it sets the dirty bit. Once it will update the main memory and clear the dirty bit.

35. Page tables used by the ARM: MMU

	TYPE	**Memory Consumed(kb)**	**Page size(kb)**	**Number of entries**
Master/Section	LEVEL-1	16	1024	4096
Fine	LEVEL-2	4	1(tiny),4(smal	1024

			l),64(large)	
Coarse	LEVEL-2	1	4,64	256

36. what is Arm CoreSight ?

Arm CoreSight technology is a set of tools that can be used to debug and trace software that runs on Arm-based SoCs.

CoreSight, ARM’s debug and trace product family, is the most complete on-chip debug and trace solution for the entire System-On-Chip (SoC), making ARM-based SoCs the easiest to debug and optimize.

CoreSight DAP
The CoreSight Debug Access Port provides software debugger access to the debug and trace subsystem within a SoC. The DAP supports access through JTAG or, for pin limited applications, Serial Wire Debug (SWD) offers the same functionality over two pins.

Sources : Generate trace data for output through the *Advanced Trace Bus* (ATB). Examples include:

- *Program Trace Macrocell* (PTM).
- *System Trace Macrocell* (STM),
- CoreSight *Embedded Trace Macrocells* (ETMs),

Links: Provide connection, triggering, and flow of trace data. Examples include:

- Synchronous 1:1 ATB bridge.
- Replicator.
- Trace funnel.

Sinks: End points for trace data on the SoC. Examples include:

- *Trace Port Interface Unit* (TPIU) for output of trace data off-chip.
- ETB for on-chip storage of trace data in RAM.

Instruction fetch

The instruction fetch unit predicts the instruction stream, fetches instructions from the L1 instruction cache, and places the fetched instructions into a **buffer for consumption by the decode pipeline**. The instruction fetch unit also includes the L1 instruction cache

Instruction decode

The instruction decode unit decodes and sequences all **ARM and Thumb-2 instructions** including debug control coprocessor, CP14, instructions and system control coprocessor, CP15, instructions.

Instruction execute

The instruction execute unit consists of **two symmetric Arithmetic Logical Unit (ALU) pipelines**, an address generator for load and store instructions, and the multiply pipeline. The execute pipelines also perform register write back.

Load/store

The load/store unit encompasses the entire L1 data side memory system and the integer load/store pipeline.

Processor operating states

The processor has the following operating states controlled by the T bit and J bit in the CPSR.

ARM state : 32-bit, word-aligned ARM instructions are executed in this state.

T bit is 0 and J bit is 0.

Thumb state : 16-bit and 32-bit, halfword-aligned Thumb-2 instructions.

T bit is 1 and J bit is 0.

ThumbEE state :16-bit and 32-bit, halfword-aligned variant of the Thumb-2 instruction set designed as a target for dynamically generated code. This is code compiled on the device either shortly before or during execution from a portable bytecode or other intermediate or native representation.

T bit is 1 and J bit is 1

Switching state

You can switch the operating state of the processor between:

ARM state and Thumb state using the **BX and BLX instructions**, and loads to the PC.

Thumb state and ThumbEE state using the **ENTERX and LEAVEX instructions**.

DEVICE TREE

1) what is a device tree

device tree is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals.

The device tree was derived from SPARC-based workstations and servers via the Open Firmware project

2) device tree format

A device tree can hold any kind of data as internally it is a tree of named **nodes and properties. Nodes contain properties and child nodes, while properties are name–value pairs.**

Device trees have both a binary format for operating systems to use and a textual format for convenient editing and management

3) what is the device tree

The primary purpose of Device Tree in Linux is to provide a way to describe non-discoverable hardware. This information was previously hard coded in source code(board file)

4) device tree tools

dtc (Device Tree Compiler) - converts between the human editable device tree source "dts" format and the compact device tree blob "dtb"

The Linux version of dtc is located in **`scripts/dtc/`** in the kernel source directory. New versions are periodically pulled from the upstream project

fdtdump

fdtdump is a tool to convert an FDT (flattened device tree, aka device tree blob) to source.

5) device tree file convention

- .dtb - File name suffix, by convention, for compiled device tree.
- .dts - File name suffix, by convention, for device tree source.
- .dtsi - File name suffix, by convention, for device tree source to be included by a .dts or .dtsi file.

6)Debugging the device tree

For newer kernels where the **CONFIG_PROC_DEVICETREE option does not exist**, /proc/device-tree will be created if CONFIG_PROC_FS is set to 'Y'

7) generated script in build directory

After a .dtb has been built by the Linux kernel make system, files containing the precise build commands will remain in the build directory:

- arch/arm/boot/dts/.*.dtb.cmd
- arch/${ARCH}/boot/dts/.*.dtb.cmd

make dtbs

8) what is address-cells and size-cells?

Devices that are addressable use the following properties to encode address information into the device tree:

- reg
- **#address-cells**

- **#size-cells**

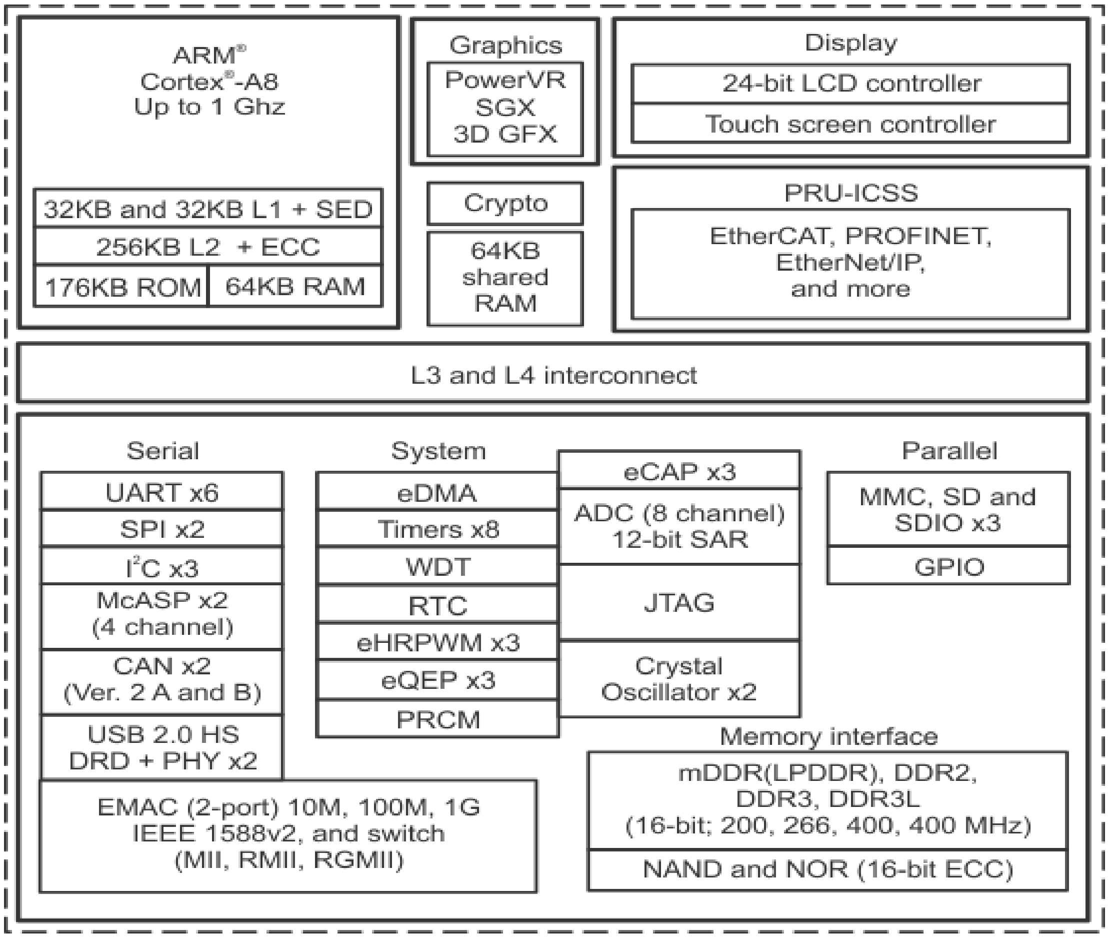
Each addressable device gets a reg which is a list of tuples in the form

reg = <address1 length1 [address2 length2] [address3 length3] ... >. Each tuple represents an address range used by the device. Each address value is a list of one or more 32 bit integers called *cells*. Similarly, the length value can either be a list of cells, or empty.

9) sample device tree

```
/dts-v1/;
/ {
    compatible = "acme,coyotes-revenge";
    cpus {
        cpu@0 {
            compatible = "arm,cortex-a9";
        };
        cpu@1 {
            compatible = "arm,cortex-a9";
        };
    };
    chosen {
        bootargs = "root=/dev/nfs rw nfsroot=192.168.1.1 console=ttyS0,115200";
    };
     memory {
                device_type=”memory”;
                reg=<0x80000000 0x20000000>;
         };
};
```

10) write AM335X – Bone Black Device tree



S.NO	IP	Details
1	CPU	ARM CORTEX A8, 1GHZ, Single Core, RISC, 32 Bit Processor
2	Cache (L1)	32 KB Instruction + 32 KB Data Cache
3	Cache (L2)	256 KB
4	Serial	UART x6
5	SPI	SPI X2
6	I2C	I2C X3
7	GPIO	
8	Timers	Timer X8
9	RTC	RTC
10	WDT	WDT
11	EDMA	EDMA

12	MMC, SD	MMC,SD X3
13	USB 2.0	USB 2.0 HOST,DRD
14	LCD CONTROLLER	
15	TUCH CONTROLLER	
16	16 BIT DDR : DDR2,DDR3,DDR3L	200,266,400 MHZ

```
/ {
        model = "TI AM335x BeagleBone Black";
        compatible="ti,am335x-bone-black", "ti,am335x-bone",
"ti,am33xx";
        interrupt-parent = <&intc>;
        #address-cells = <1>;
        #size-cells = <1>;

        memory@80000000 {
                device_type = "memory";
                reg = <0x80000000 0x10000000>; /* 256 MB */
        };

 ocp {
        intc: interrupt-controller@48200000 {
                        compatible = "ti,am33xx-intc";
                        interrupt-controller;
                        #interrupt-cells = <1>;
                        reg = <0x48200000 0x1000>;
                };
        gpio0: gpio@44e07000 {
                        compatible = "ti,omap4-gpio";
```

```
            ti,hwmods = "gpio1";
            gpio-controller;
            #gpio-cells = <2>;
            interrupt-controller;
            #interrupt-cells = <2>;
            reg = <0x44e07000 0x1000>;
            interrupts = <96>;
        };
    uart0: serial@44e09000 {
            compatible = "ti,am3352-uart", "ti,omap3-uart";
            ti,hwmods = "uart1";
            clock-frequency = <48000000>;
            reg = <0x44e09000 0x2000>;
            interrupts = <72>;
            status = "disabled";
            dmas = <&edma 26 0>, <&edma 27 0>;
            dma-names = "tx", "rx";
        };
    i2c0: i2c@44e0b000 {
            compatible = "ti,omap4-i2c";
            #address-cells = <1>;
            #size-cells = <0>;
            ti,hwmods = "i2c1";
            reg = <0x44e0b000 0x1000>;
            interrupts = <70>;
            status = "disabled";
        };
```

```
mmc1: mmc@48060000 {
        compatible = "ti,omap4-hsmmc";
        ti,hwmods = "mmc1";
        ti,dual-volt;
        ti,needs-special-reset;
        ti,needs-special-hs-handling;
        dmas = <&edma_xbar 24 0 0
                &edma_xbar 25 0 0>;
        dma-names = "tx", "rx";
        interrupts = <64>;
        reg = <0x48060000 0x1000>;
        status = "disabled";
    };
timer1: timer@44e31000 {
        compatible = "ti,am335x-timer-1ms";
        reg = <0x44e31000 0x400>;
        interrupts = <67>;
        ti,hwmods = "timer1";
        ti,timer-alwon;
        clocks = <&timer1_fck>;
        clock-names = "fck";
    };
rtc: rtc@44e3e000 {
        compatible = "ti,am3352-rtc", "ti,da830-rtc";
        reg = <0x44e3e000 0x1000>;
        interrupts = <75    76>;
        ti,hwmods = "rtc";
```

```
                clocks = <&l4_per_clkctrl AM3_CLKDIV32K_CLKCTRL 0>;
                clock-names = "int-clk";
        };
spi0: spi@48030000 {
                compatible = "ti,omap4-mcspi";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0x48030000 0x400>;
                interrupts = <65>;
                ti,spi-num-cs = <2>;
                ti,hwmods = "spi0";
                dmas = <&edma 16 0
                        &edma 17 0
                        &edma 18 0
                        &edma 19 0>;
                dma-names = "tx0", "rx0", "tx1", "rx1";
                status = "disabled";
        };
};
        am33xx_pinmux: pinmux@800 {
                                compatible = "pinctrl-single";
                                reg = <0x800 0x238>;
                                #address-cells = <1>;
                                #size-cells = <0>;
                                #pinctrl-cells = <1>;
                                pinctrl-single,register-width = <32>;
                                pinctrl-single,function-mask = <0x7f>;
                        };
};
```

dsti

```
&am33xx_pinmux {
       pinctrl-names = "default";
       pinctrl-0 = <&clkout2_pin>;

       user_leds_s0: user_leds_s0 {
              pinctrl-single,pins = <
          AM33XX_IOPAD(0x854, PIN_OUTPUT_PULLDOWN | MUX_MODE7)    /*
gpmc_a5.gpio1_21 */
           AM33XX_IOPAD(0x858, PIN_OUTPUT_PULLUP | MUX_MODE7)       /*
gpmc_a6.gpio1_22 */
         AM33XX_IOPAD(0x85c, PIN_OUTPUT_PULLDOWN | MUX_MODE7)     /*
gpmc_a7.gpio1_23 */
              AM33XX_IOPAD(0x860, PIN_OUTPUT_PULLUP | MUX_MODE7)     /*
gpmc_a8.gpio1_24 */
              >;
       };

       i2c0_pins: pinmux_i2c0_pins {
              pinctrl-single,pins = <
                        AM33XX_IOPAD(0x988, PIN_INPUT_PULLUP | MUX_MODE0)
/* i2c0_sda.i2c0_sda */
                        AM33XX_IOPAD(0x98c, PIN_INPUT_PULLUP | MUX_MODE0)
/* i2c0_scl.i2c0_scl */
              >;
       };
```

```
};

&i2c0 {
	pinctrl-names = "default";
	pinctrl-0 = <&i2c0_pins>;
	status = "okay";
	clock-frequency = <400000>;
	tps: tps@24 {
		reg = <0x24>;
	};

	baseboard_eeprom: baseboard_eeprom@50 {
		compatible = "atmel,24c256";
		reg = <0x50>;

		#address-cells = <1>;
		#size-cells = <1>;
		baseboard_data: baseboard_data@0 {
			reg = <0 0x100>;
		};
	};
};
&mmc1 {
	status = "okay";
	bus-width = <0x4>;
	pinctrl-names = "default";
	pinctrl-0 = <&mmc1_pins>;
```

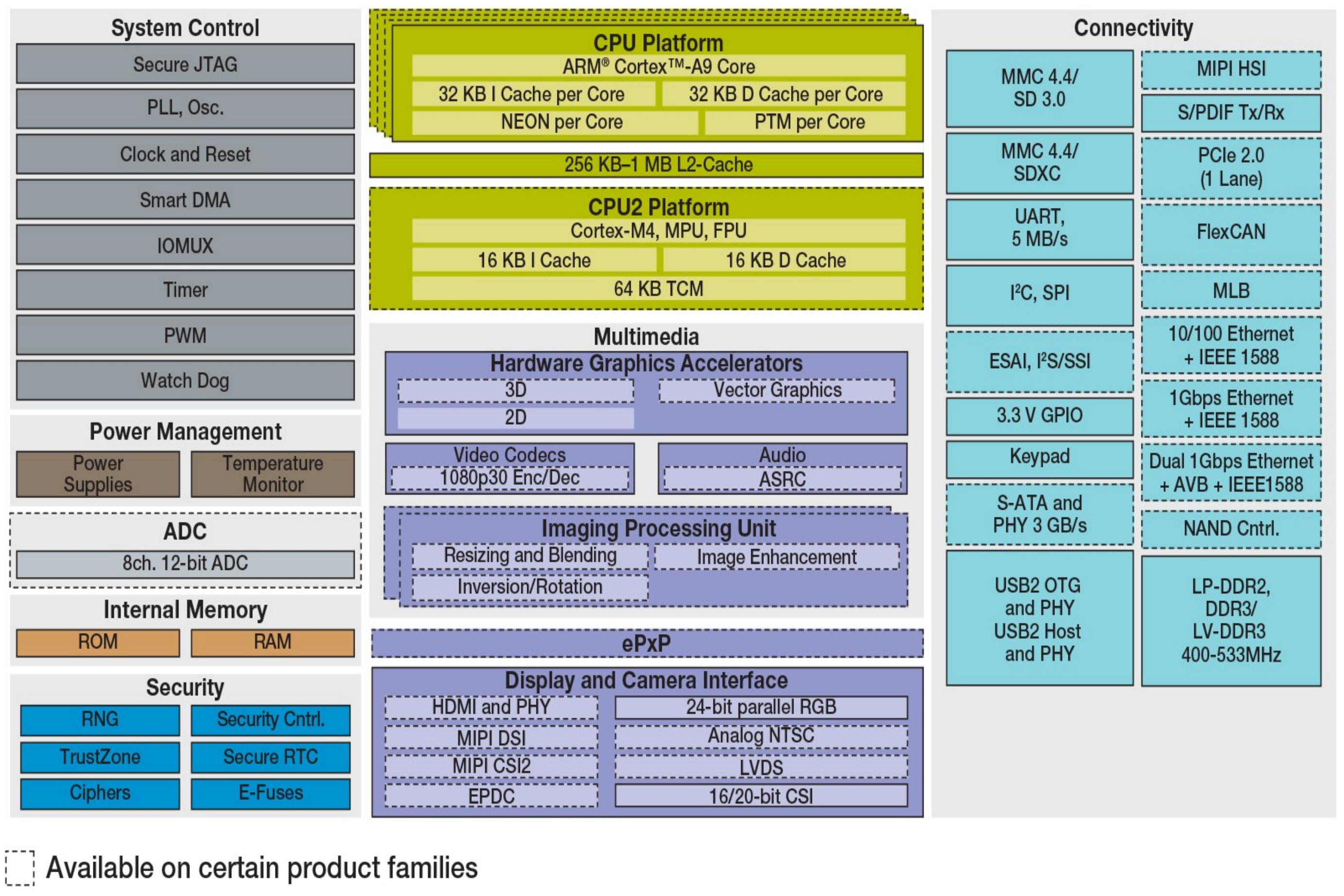
```
        cd-gpios = <&gpio0 6 GPIO_ACTIVE_LOW>;
};
```

10) write I.MX6 – (SOLO/DUAL/DUALLITE/QUARD) Device tree

The i.MX 6 series are based on the **ARM Cortex A9 solo, dual or quad cores n CMOS 40 nm process**. **i.MX 6 Solo, Dual and Quad were announced in January 2011, during Consumer Electronics Show in Las Vegas.**

Name	Clock speed	CPU cores	L2 cache in kB	Embedded SRAM in kB	3D GPU / shaders / shader clock in MHz	2D GPU	Vector GPU	VPU	other graphics cores	other cores
i.MX 6 Solo	1.0 GHz	1	512	128	Vivante GC880 / 1 / 528	Vivante GC320		Full HD (1080p decode)	1× IPUv3, 1× PXP	security
i.MX 6 DualLite	1.0 GHz	2	512	128	Vivante GC880 / 1 / 528	Vivante GC320		Full HD (1080p decode)	1× IPUv3, 1× PXP	security
i.MX 6 Dual	1.2 GHz	2	1024	256	Vivante GC2000 / 4 / 594	Vivante GC320	Vivante GC355	Full HD (1080p decode)	2× IPUv3	security
i.MX 6 Quad	1.2 GHz	4	1024	256	Vivante GC2000 / 4 / 594	Vivante GC320	Vivante GC355	dual Full HD (1080p decode)	2× IPUv3	security

i.MX 6 Series Applications Processor Block Diagram



```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        cpu@0 {
            compatible = "arm,cortex-a9";
            device_type = "cpu";
            reg = <0x0>;
            next-level-cache = <&L2>;
            operating-points = <
                /* kHz    uV */
                996000  1275000
                792000  1175000
                396000  975000
            >;
      };
};

    intc: interrupt-controller@a01000 {
        compatible = "arm,cortex-a9-gic";
        #interrupt-cells = <3>;
        interrupt-controller;
        reg = <0x00a01000 0x1000>,
              <0x00a00100 0x100>;
        interrupt-parent = <&intc>;
    };
 soc {
```

```
#address-cells = <1>;
#size-cells = <1>;
compatible = "simple-bus";
interrupt-parent = <&gpc>;
ranges;
        uart1: serial@2020000 {
                    compatible = "fsl,imx6sl-uart",
                             "fsl,imx6q-uart", "fsl,imx21-uart";
                    reg = <0x02020000 0x4000>;
                    interrupts = <0 26 IRQ_TYPE_LEVEL_HIGH>;
                    clocks = <&clks IMX6SL_CLK_UART>,
                           <&clks IMX6SL_CLK_UART_SERIAL>;
                    clock-names = "ipg", "per";
                    dmas = <&sdma 25 4 0>, <&sdma 26 4 0>;
                    dma-names = "rx", "tx";
                    status = "disabled";
             };
        ssi1: ssi@2028000 {
                    #sound-dai-cells = <0>;
                    compatible = "fsl,imx6sl-ssi",
                                 "fsl,imx51-ssi";
                    reg = <0x02028000 0x4000>;
                    interrupts = <0 46 IRQ_TYPE_LEVEL_HIGH>;
                    clocks = <&clks IMX6SL_CLK_SSI1_IPG>,
                           <&clks IMX6SL_CLK_SSI1>;
                    clock-names = "ipg", "baud";
                    dmas = <&sdma 37 1 0>,
                        <&sdma 38 1 0>;
                    dma-names = "rx", "tx";
                    fsl,fifo-depth = <15>;
                    status = "disabled";
                };
            pwm1: pwm@2080000 {
                #pwm-cells = <2>;
                compatible = "fsl,imx6sl-pwm", "fsl,imx27-pwm";
                reg = <0x02080000 0x4000>;
                interrupts = <0 83 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks IMX6SL_CLK_PWM1>,
                         <&clks IMX6SL_CLK_PWM1>;
                clock-names = "ipg", "per";
            };
        gpio1: gpio@209c000 {
                compatible = "fsl,imx6sl-gpio", "fsl,imx35-gpio";
                reg = <0x0209c000 0x4000>;
                interrupts = <0 66 IRQ_TYPE_LEVEL_HIGH>,
                             <0 67 IRQ_TYPE_LEVEL_HIGH>;
                gpio-controller;
                #gpio-cells = <2>;
                interrupt-controller;
                #interrupt-cells = <2>;
                gpio-ranges = <&iomuxc  0 22 1>, <&iomuxc  1 20
2>,
                              <&iomuxc  3 23 1>, <&iomuxc  4 25 1>,
                              <&iomuxc  5 24 1>, <&iomuxc  6 19 1>,
                        <&iomuxc  7 36 2>, <&iomuxc  9 44 8>,
                        <&iomuxc 17 38 6>, <&iomuxc 23 68 4>,
                        <&iomuxc 27 64 4>, <&iomuxc 31 52 1>;
        };
        kpp: kpp@20b8000 {
                compatible = "fsl,imx6sl-kpp", "fsl,imx21-kpp";
                reg = <0x020b8000 0x4000>;
                interrupts = <0 82 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks IMX6SL_CLK_DUMMY>;
                status = "disabled";
        };

        wdog1: wdog@20bc000 {
                compatible = "fsl,imx6sl-wdt", "fsl,imx21-wdt";
                reg = <0x020bc000 0x4000>;
                interrupts = <0 80 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks IMX6SL_CLK_DUMMY>;
        };
         i2c1: i2c@21a0000 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "fsl,imx6sl-i2c", "fsl,imx21-i2c";
                reg = <0x021a0000 0x4000>;
                interrupts = <0 36 IRQ_TYPE_LEVEL_HIGH>;
                clocks = <&clks IMX6SL_CLK_I2C1>;
                status = "disabled";
        };
```

```
                usdhc1: usdhc@2190000 {
                        compatible = "fsl,imx6sl-usdhc", "fsl,imx6q-usdhc";
                        reg = <0x02190000 0x4000>;
                        interrupts = <0 22 IRQ_TYPE_LEVEL_HIGH>;
                        clocks = <&clks IMX6SL_CLK_USDHC1>,
                                 <&clks IMX6SL_CLK_USDHC1>,
                                 <&clks IMX6SL_CLK_USDHC1>;
                        clock-names = "ipg", "ahb", "per";
                        bus-width = <4>;
                        status = "disabled";
                };
                iomuxc: iomuxc@20e0000 {
                        compatible = "fsl,imx6sl-iomuxc";
                        reg = <0x020e0000 0x4000>;
                };

                csi: csi@20e4000 {
                        reg = <0x020e4000 0x4000>;
                        interrupts = <0 7 IRQ_TYPE_LEVEL_HIGH>;
                };
```

dtsi:

```
&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog>;

        imx6sl-evk {
```

```
pinctrl_hog: hoggrp {
        fsl,pins = <
                MX6SL_PAD_KEY_ROW7__GPIO4_IO07    0x17059
                MX6SL_PAD_KEY_COL7__GPIO4_IO06    0x17059
                MX6SL_PAD_SD2_DAT7__GPIO5_IO00    0x17059
                MX6SL_PAD_SD2_DAT6__GPIO4_IO29    0x17059
                MX6SL_PAD_REF_CLK_32K__GPIO3_IO22 0x17059
                MX6SL_PAD_KEY_COL4__GPIO4_IO00  0x80000000
                MX6SL_PAD_KEY_COL5__GPIO4_IO02  0x80000000
                MX6SL_PAD_AUD_MCLK__AUDIO_CLK_OUT 0x4130b0
        >;
};

pinctrl_audmux3: audmux3grp {
        fsl,pins = <
                MX6SL_PAD_AUD_RXD__AUD3_RXD       0x4130b0
                MX6SL_PAD_AUD_TXC__AUD3_TXC       0x4130b0
                MX6SL_PAD_AUD_TXD__AUD3_TXD       0x4110b0
                MX6SL_PAD_AUD_TXFS__AUD3_TXFS     0x4130b0
        >;
};

pinctrl_ecspi1: ecspi1grp {
        fsl,pins = <
                MX6SL_PAD_ECSPI1_MISO__ECSPI1_MISO      0x100b1
                MX6SL_PAD_ECSPI1_MOSI__ECSPI1_MOSI      0x100b1
                MX6SL_PAD_ECSPI1_SCLK__ECSPI1_SCLK      0x100b1
                MX6SL_PAD_ECSPI1_SS0__GPIO4_IO11        0x80000000
        >;
    };
```

11) ov5640 : device tree node in sabresd (I.mx6) (imx6qdl-sabresd.dtsi)

```
ov5640: camera@3c {
    compatible = "ovti,ov5640";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_ov5640>;
    reg = <0x3c>;
    clocks = <&clks IMX6QDL_CLK_CKO>;
    clock-names = "xclk";
    DOVDD-supply = <&vgen4_reg>; /* 1.8v */
    AVDD-supply = <&vgen3_reg>;  /* 2.8v, rev C board is VGEN3
                                    rev B board is VGEN5 */
    DVDD-supply = <&vgen2_reg>;  /* 1.5v*/
    powerdown-gpios = <&gpio1 19 GPIO_ACTIVE_HIGH>;
    reset-gpios = <&gpio1 20 GPIO_ACTIVE_LOW>;

    port {
        ov5640_to_mipi_csi2: endpoint {
            remote-endpoint = <&mipi_csi2_in>;
            clock-lanes = <0>;
            data-lanes = <1 2>;
        };
    };
```

```
        };

&mipi_csi {
        status = "okay";

        port@0 {
                reg = <0>;

                mipi_csi2_in: endpoint {
                        remote-endpoint = <&ov5640_to_mipi_csi2>;
                        clock-lanes = <0>;
                        data-lanes = <1 2>;
                };
        };
};

mipi_csi: mipi@21dc000 {
                                compatible = "fsl,imx6-mipi-csi2";
                                reg = <0x021dc000 0x4000>;
                                #address-cells = <1>;
                                #size-cells = <0>;
                                interrupts = <0 100 0x04>, <0 101 0x04>;
                                clocks = <&clks IMX6QDL_CLK_HSI_TX>,
                                         <&clks IMX6QDL_CLK_VIDEO_27M>,
                                         <&clks IMX6QDL_CLK_EIM_PODF>;
                                clock-names = "dphy", "ref", "pix";
                                status = "disabled";
```

```
};
```

12) device tree node for xilinx sdirx source ?

SDI Rx SS ==> Framebuffer Write ==> Memory

```
v_smpte_uhdsdi_rx_ss: v_smpte_uhdsdi_rx_ss@80000000 {
        compatible = "xlnx,v-smpte-uhdsdi-rx-ss";
        interrupt-parent = <&gic>;
        interrupts = <0 89 4>;
        reg = <0x0 0x80000000 0x0 0x10000>;
        xlnx,include-axilite = "true";
        xlnx,include-edh = "true";
        xlnx,include-vid-over-axi = "true";
        xlnx,line-rate = "12G_SDI_8DS";
        clocks = <&clk_1>, <&si570_1>, <&clk_2>;
        clock-names = "s_axi_aclk", "sdi_rx_clk", "video_out_clk";

        ports {
                #address-cells = <1>;
                #size-cells = <0>;

                port@0 {
                        reg = <0>;

                        xlnx,video-format = <XVIP_VF_YUV_422>;
                        xlnx,video-width = <10>;

                        sdirx_out: endpoint {
                                remote-endpoint = <&vcap_in>;
                        };
                };
        };
};



v_frmbuf_wr_0: v_frmbuf_wr@80000000 {
        #dma-cells = <1>;
        compatible = "xlnx,axi-frmbuf-wr-v2.1";
        interrupt-parent = <&gic>;
        interrupts = <0 92 4>;
        reset-gpios = <&gpio 80 1>;
        reg = <0x0 0x80000000 0x0 0x10000>;
        xlnx,dma-addr-width = <64>;
        xlnx,vid-formats = "bgr888","yuyv","nv16","nv12";
        xlnx,pixels-per-clock = <2>;
        xlnx,dma-align = <16>;
        clocks = <&vid_stream_clk>;
        clock-names = "ap_clk"
};


video_cap {

        compatible = "xlnx,video";

        dmas = <&v_frmbuf_wr_0 0>;
        dma-names = "port0";

        ports {

                #address-cells = <1>;

                #size-cells = <0>;

                port@0 {

                        reg = <0>;

                        direction = "input";

                        vcap_in: endpoint {

                                remote-endpoint = <&sdirx_out>;

                        };

                };


            };

        };
```