Embedded System Concepts

1. What is an Embedded System? Give some examples of embedded system in daily life. ★★

Ans. Embedded System is a system which hides a small computation unit inside. This computer is not general purpose computer like our Desktop PC but a small microcontroller or microprocessor. Embedded System is designed to perform specific tasks. Best example of an Embedded System is our mobile phone which includes a processing unit or no. of processing units along with peripherals like keypad/touchpad, touch screen, LCD, Communication modules etc.

There are other examples like Electronic System in washing machine, ATM machine, front panel of Microwave oven, printers, TV, set top boxes, Remote control, Dashboard of a 4-wheeler and many more. If you observe carefully we are surrounded by a no. of embedded system and use them very frequently in our day to day life.

2. What are basic features of an Embedded System? ★★

Ans. A good embedded system must possess following features…

- Microcontroller based
- Software driven
- Reliable
- Real time control system
- Autonomous and human interactive
- Operating on diverse physical variables
- Operating in diverse environments

3. What are various categories of an Embedded System based on complexity? ★★

Ans. Embedded Systems can be broadly classified in three categories…

a. Small Scale – 8/16 bit CPU, little hardware, less software complexity, no RTOS, battery operated.
b. Medium scale – 16/32 bit uC or DSPs, Complex hardware and software design, RTOS source code engg tool, IDE.
c. Sophisticated – Configurable processors, Cutting edge applications, highly complex hardware and software design

4. Why PC can't be treated as an Embedded System? ★★

Ans. Of course PC is also embedded with a processing unit but still it can not be called as an Embedded System. The reason is very clear that PC is not supposed to do a specific task. For a bank employee PC is accountings add for his work while for an engineer it is designs add. For a doctor PC helps to analyze patient reports and for children it's more like video game or a communication system to stay connected with their friends through social networking websites. Hence PC has different applications for different people; hence it cannot be treated as an embedded system.

5. What is difference between a PC based System and an Embedded System? ★★★

Ans. A PC based system is bulky hence difficult to be deployed specially in remote areas. For example if one wants a data logger system to monitor all the processes and events in an aircraft or on a hilly area, PC is never a suitable option. In this case an Embedded System can do the job well consuming very small space. Also PC based system are not so cost efficient and as reliable as embedded systems.

6. Ok if embedded systems are so important and popular than why PCs still exist, why can't we use embedded systems instead of PCs? ★★

Ans. Well PC works with operating systems and they support comprehensive cross compilers for generating executable files for embedded systems. Hence we can never ignore the importance of PCs

7. What are the main components of an Embedded System? ★★

Ans. Components of an embedded system depends on the application it is intended to perform. Generally an embedded system contains sensors, processor(s) and actuators. Main components of an embedded system are Processor, Memory, Peripherals like Keypad, LCD etc.

We can classify these components in three categories

a. **Hardware** includes sensors, actuators, Processor, RAM, ROM, ADC, DAC, Timers, Ports etc.
b. **Software** includes System software and Application software. A system software can be a scheduler or an RTOS (Real Time Operating System).
c. **Firmware** responsible for running hardware components like disk drivers, protocol gateways etc.

8. What's difference between a microcontroller and microprocessor? ★★★★

Ans. Microcontroller has a CPU along with various peripherals like timers, I/Os, Serial communication modules, Memory etc integrated on a single chip while microprocessor doesn't. It has to be connected will all these externally as Microprocessor just has a CPU core on a chip.

9. Ok if microcontroller is so powerful then why don't they use a microcontroller instead of microprocessor for desktops and laptops? ★★

Ans. Actually in Desktop PCs and Laptops space efficiency is not as important as in mobile phones or other handy gadgets. Also much is expected from a Desktop or Laptop hence it's better to integrate highly efficient processor in terms of speed and data handling capability with peripherals outside the chip i.e. on a mother board.

10. What's difference between RISC and CISC? ★★★

Ans.

RISC	CISC
1. Simple Instruction taking 1 cycle	Complex instructions taking multiple cycles
2. Only LOAD/STORE instructions reference memory	Any instruction may reference memory
3. Highly pipelined	Not pipelined or less pipelined
4. Instructions executed by the hardware	Instructions interpreted by the micro program
5. Fixed format instructions	Variable format instructions
6. Few instructions and addressing modes	Many instructions and no. of addressing modes
7. Complexity is in the compiler	Complexity is in the micro program
8. Multiple register sets	Single register set

11. What's difference between Harvard and Von Neumann architecture? ★★★

Ans. In Von Neumann architecture instructions and data are stored together in a common memory space while in Harvard they are stored separately.
Harvard architecture has separate data and instruction busses, allowing transfers to be performed simultaneously on both busses. **Von Neumann architecture or Princeton architecture** has only one bus which is used for both data transfers and instruction fetches, and therefore data transfers and instruction fetches must be scheduled - they cannot be performed at the same time.

12. What do you mean by clock or the operating frequency of an Embedded System? ★★★

Ans. Embedded System is a digital system hence needs clock for its operation. Clock only decides the data rate for communication and timing of operations. Most of the processes in embedded system are synchronous i.e. they are performed at clock edges. Usual clock speeds of embedded systems are few kHz to several hundred MHz.

13. What are the different methods of clocking a microcontroller? ★★★

Ans. An embedded system can be clocked externally by using crystal oscillator or simpler RC oscillators. Some microcontrollers come with internal clocking mechanisms.

14. If clock frequency is higher does that mean the embedded system is better? ★★★

Ans. No, it is not always true. Higher clock speed tend to higher power consumption hence the designer must be very careful while selecting an operating frequency for his system. There must be optimization between power requirement and speed requirement. Power consumption concern is very important especially in battery operated systems.

15. What are different types of memories involved in an Embedded System? ★★

Ans. A typical embedded system includes RAM for temporary storage of program data and an EEPROM for storing the code. Now a day's instead of EEPROM, microcontrollers come with FLASH memory in order to store the programs and also for permanent data storage.

16. How code memory of the system can be optimized? ★★★

Ans. Code memory optimization is very important for making the code compact and fitting into a small memory space without affecting code performance. There are various ways of code optimization…

- Use of unsigned byte
- Avoiding stdlib functions, when simple coding is possible
- Usage of Assy. Code when target processor is well known
- Usage of global variable, when no chance of shared data problem
- Reduce usage of frequent function calls
- Usage of delete/free, to release the used memory

17. What is difference between System Software and Application Software? ★★★

Ans. System Software are generally Application independent like OS, Compiler, Assembler, Linker, BIOS etc. but application software is responsible for performing particular application like printing-software in a printer.

18. What is difference between Software and Firmware?★★★

Ans. Firmware is very essential software and a must for an embedded system to run its hardware. A firmware is something that is inside a chip. Like BIOS is called a firmware without which system can never run. While software is application program that is not very essential for the device to work basic functionalities.

Example: firmware makes the CD drive work, but the software makes CDs burn.

19. What is difference between Microcontroller, FPGA, ASIC and SoC?

★★★

Ans. The biggest difference between is that the microcontroller is a software based solution while the FPGA or CPLD are a hardware based solutions. An FPGA (Field Programmable Gate Array) design is programmed the hardware

way and is suited for very fast applications. A microcontroller is programmed the software way and is better suited for medium performance and control.
Hence FPGA is used to realize digital hardware circuits inside a chip while a microcontroller is used to realize a software application.
On the other hand ASIC or Application Specific Integrated Circuit is application specific device made of FPGA or CPLD. Hence it is also hardwired solution for the problem and faster than microcontrollers.
SoC or system on chip. refers to chip that includes so many features on single integrated circuit like memory, timers, interrupts, I/O interfaces etc. Microncontroller can also be called as a SoC but usually the term is used for sophisticated systems. Like a microcontroller inside a mobile phone can be called as SoC.

20. What are the different criteria to select a microcontroller? ★★
Ans. There are criteria like... Data size it can handle (8bit/16bit/32bit), Code size (FLASH), Clock frequency, Power Requirement, Cost and peripherals required for specific application.

21. What's difference between 8bit and 16bit microcontroller? ★★★
Ans. A CPU can be classified on the basis of the data it can access in a single operation. An 8-bit processor can access 8 bits of data in a single operation, as opposed to a 16-bit processor, which can access 16 bits of data in a single operation...." By this definition, 32-bit microcontroller is one that able to access 32bit data in a single operation.

22. What factor of an architecture defines no. of bytes in data types? ★★★
Ans.
23. What's difference between a simulator and an emulator, give examples? ★★★
Ans. The *Simulator* tries to duplicate the *behavior* of the device. The *Emulator* tries to duplicate the *inner workings* of the device.
Simulation -- if you do not have the thermometer yet, but you want to test that this message rate will not overload you system, you can simulate the sensor by attaching a unit that sends a random number 8 times a second. You can run any test that does not rely on the actual value the sensor sends.
Emulation -- suppose you have a very expensive thermometer that measures to 0.001 C, and you want to see if you can get by with a cheaper thermometer that only measures to the nearest 0.5 C. You can emulate the cheaper thermometer using an expensive thermometer by rounding the reading to the nearest 0.5 C and running tests that rely on the temperature values.

24. What do you mean by an in-circuit emulator? ★★★
Ans. An in-circuit emulator is a hardware used to debug software of an embedded system. It provides a window into the embedded system. The

programmer uses the emulator to load programs into the embedded system, run them, step through them slowly, and view and change data used by the system's software.

25. What's difference between memory-mapped I/O and I/O-mapped I/O? ★★★

Ans. In memory mapped I/O, a chunk of the CPU's address space is reserved for accessing I/O devices. In I/O mapped I/O, I/O devices are handled distinctly by the CPU and hence occupy a separate chunk of addresses predetermined by the CPU for I/O.
In case of Memory mapped I/O the same address BUS is used for accessing both Memory (RAM) and the Registers of I/O devices.
For I/O Mapped I/O, separate address BUS is used.
As Address space is generally larger for Memory than I/O registers, the length of I/O address is larger in case of Memory Mapped I/O.
for a system which uses I/O Mapped I/O, there is a requirement for an extra hardware Circuitry.

26. What's difference between little endian and big endian representations? ★★★★

Ans. Endianness refers to the way data is stored in memory.
So, suppose a data containing 4 bytes: 90, AB, 12, CD where each byte requires 2 hex digits.
It turns out there are two ways to store this in memory.

Big Endian

In big endian, you store the most significant byte in the smallest address. Here's how it would look:

Address	Value
1000	90
1001	AB
1002	12
1003	CD

Little Endian

In little endian, you store the *least* significant byte in the smallest address.
Here's how it would look:

Address	Value
1000	CD
1001	12
1002	AB

1003	90

27. Give some examples of processors which use little endian and big endian representations. Which of the two representations is more popular and why? ★★★

Ans. Well-known processor architectures that use the little-endian format include x86 (including x86-64), 6502 (including 65802, 65C816), Z80 (including Z180, eZ80 etc.), MCS-48, 8051,DEC Alpha, Altera Nios, Atmel AVR, SuperH, VAX, and, largely, PDP-11.

Well-known processors that use the big-endian format include Motorola 6800 and 68k, Xilinx Microblaze, IBM POWER, and System/360 and its successors such as System/370,ESA/390, and z/Architecture. The PDP-10 also used big-endian addressing for byte-oriented instructions. SPARC historically used big-endian until version 9, which is bi-endian, similarly the ARM architecture was little-endian before version 3 when it became bi-endian, and the PowerPC and Power Architecture descendants of IBM POWER are also bi-endian. There is no such popularity dispute between the two representations both are used as per application requirement or the project specifications.

28. What is SDLC? What are different models involved? ★★★

Ans. A **software development process**, also known as a **software development life cycle (SDLC)**, is a structure imposed on the development of a software product. It represents various stages involved in software development process right from the scratch to the deployment.

Software development models

Several models exist to streamline the development process. Each one has its pros and cons, and it's up to the development team to adopt the most appropriate one for the project. Sometimes a combination of the models may be more suitable.

Waterfall model

The waterfall model shows a process, where developers are to follow these phases in order:

1. Requirements specification (Requirements analysis)
2. Software design
3. Implementation and Integration
4. Testing (or Validation)
5. Deployment (or Installation)
6. Maintenance

In a strict Waterfall model, after each phase is finished, it proceeds to the next one. Reviews may occur before moving to the next phase which allows for the possibility of changes (which may involve a formal change control process). Reviews may also be employed to ensure that the phase is indeed complete; the phase completion criteria are often referred to as a "gate" that the project must

pass through to move to the next phase. Waterfall discourages revisiting and revising any prior phase once it's complete. This "inflexibility" in a pure Waterfall model has been a source of criticism by supporters of other more "flexible" models.

Spiral model

The key characteristic of a Spiral model is risk management at regular stages in the development cycle.

The Spiral is visualized as a process passing through some number of iterations, with the four quadrant diagram representative of the following activities:

1. formulate plans to: identify software targets, selected to implement the program, clarify the project development restrictions;
2. Risk analysis: an analytical assessment of selected programs, to consider how to identify and eliminate risk;
3. the implementation of the project: the implementation of software development and verification;

Risk-driven spiral model, emphasizing the conditions of options and constraints in order to support software reuse, software quality can help as a special goal of integration into the product development. However, the spiral model has some restrictive conditions, as follows:

1. The spiral model emphasizes risk analysis, and thus requires customers to accept this analysis and act on it. This requires both trust in the developer as well as the willingness to spend more to fix the issues, which is the reason why this model is often used for large-scale internal software development.
2. If the implementation of risk analysis will greatly affect the profits of the project, the spiral model should not be used.
3. Software developers have to actively look for possible risks, and analyze it accurately for the spiral model to work.

The first stage is to formulate a plan to achieve the objectives with these constraints, and then strive to find and remove all potential risks through careful analysis and, if necessary, by constructing a prototype. If some risks can not be ruled out, the customer has to decide whether to terminate the project or to ignore the risks and continue anyway. Finally, the results are evaluated and the design of the next phase begins.

Iterative and incremental development

Iterative development prescribes the construction of initially small but ever-larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster.

Agile development

Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software. There are many variations of agile processes:

- In Extreme Programming (XP), the phases are carried out in extremely small (or "continuous") steps compared to the older, "batch" processes. The (intentionally incomplete) first pass through the steps might take a day or a week, rather than the months or years of each complete step in the Waterfall model. First, one writes automated tests, to provide concrete goals for development. Next is coding (by a pair of programmers), which is complete when all the tests pass, and the programmers can't think of any more tests that are needed. Design and architecture emerge out of refactoring, and come after coding. The same people who do the coding do design. (Only the last feature — merging design and code — is common to *all* the other agile processes.) The incomplete but functional system is deployed or demonstrated for (some subset of) the users (at least one of which is on the development team). At this point, the practitioners start again on writing tests for the next most important part of the system
- Scrum
- Dynamic systems development method

Code and fix

Without much of a design in the way, programmers immediately begin producing code. At some point, testing begins (often late in the development cycle), and the inevitable bugs must then be fixed before the product can be shipped. See also: Continuous integration

29. How to improve reliability of an Embedded System. ★★★★

Ans. Some of the commonly used techniques to ensure reliability of embedded system include use of watchdog timers, redundant sub systems, use of embedded hypervisor etc.

In computing, a **hypervisor** or **virtual machine manager** (**VMM**) is a piece of computer software, firmware or hardware that creates and runs virtual machines.

A computer on which a hypervisor is running one or more virtual machines is a **host machine**. Each of those virtual machines are called a *guest machine*. The hypervisor presents to the guest operating systems a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources.

Embedded C Programming

1. What is difference between Assembly and C programming? ★★

Ans. Assembly and C language can be differentiated as follow…

Assembly	**C, C++**
Full of processor specific instructions	Use of high level syntaxes. Software development cycle is short due to usage of functions, std library functions, modular programming approach, Top Down design etc.
Machine codes are compact	Data types declarations make programming easy
Device driver codes needs only few assembly instructions	Type checking eliminates errors.
	Usage of control structures, like if else, for, while, do-while etc. Portability to different hardware and OS

2. What is difference between Compiler and Interpreter? ★★★

Ans. A compiler converts entire program into executable code before running, when running the program only the executable version of the code is running. An interpreter converts the code during at run time, it converts the code one line at a time.

OCTAVE, MATLAB, MAPLE, R and APL are interpreters while TurboC, Borland, Eclipse, gcc, g++ are the compilers

3. What is need of an infinite loop while programming for an Embedded System? ★★★

Ans. Infinite loops are very common while programming an embedded system. An embedded system is supposed to work forever after programmed once. For example a temperature indicator is supposed to indicate temperature at all times, day and night without interruption. The temperature sensor gives sample to the microcontroller inside after specified period of time (hardware part) and the microcontroller is supposed to perform required calculation and display the temperature output on display panel (software part). This is a continuous process hence the software is written under an infinite loop.

4. Difference between for and while loop? Which one is better? ★★★★

Ans. while loop just has loop terminating condition in its parenthesis but for has initialization, condition and updatation of the loop variable at same place.

For infinite loop, while is preferred over for loop as it represents the infinite loop condition explicitly as while(1). On the other hand for loop just says for(;;) which is little confusing.

5. What do you mean by a cross compiler? ★★★★

Ans. Cross Compiler is system software that runs on general purpose PC but compiles your code and generates executables for some other target machine like ARM, 8051, AVR etc. Keil C51 C compiler, Keil-ARM, AVR gcc are the example of cross compilers which run on PCs but generate executables for different target platforms (microcontrollers).

6. What are the different steps involved in the process of translation of a Source code into an Executable code? ★★★★

Ans. There are basically four steps involved…
Preprocessing – Removal of comment messages. Macro replacement etc.
Compiling – Converting source code into the assembly language.
Assembling – Converting assembly code into the object code.
Linking- Combining object codes to form single executable.

7. What are various files involved in the above mentioned process? ★★★

Ans. Source files – High level language files with extension .c, .cpp or .asm are source files depending upon in which language the program is written. Although a program can be group of two or more languages as well.
Asm files- If program is written in any of the higher level languages like c or c++ it is first converted into assembly language called .asm files.
Object files- Assembler is the software which converts .asm files into machine level code called object files.
Executable or Hex files- These are the files outcome of linking processes which combines (links) all the involved object files to form a single executable unit which can now be deployed into the system.
List files- Some compilers also produce list files (.lst files). These files include the address information of instructions and data.

8. How does combination of functions reduce memory requirement in Embedded System? ★★★

Ans. Combining similar activities by different functions can save considerable amount of memory as using number of function increases the calling overheads on stack and memory requirement increases due to formal and actual arguments of the functions.

9. What's size of integer, float, and char variables? ★★★★★

Ans. They are machine dependent or compiler dependent hence size varies machine to machine and compiler to compiler e.g. for some compilers integer is of 2 bytes and for some compilers it's of 4 bytes.

10. What are various type specifiers and type qualifiers in C? ★★★

Ans. void, int, float, char, double, long double, struct, union etc. are type specifiers in C while static, volatile, const, etc. are type qualifiers

11. Write a Program to identify the no. of binary 1s in an integer number. ★★★★★

Ans. Well there can be many ways but the simplest one is this…

```
unsigned int a = 10, b=0, count;
count = 0;
        while(a != 0)
{
b = a & 1;
                if (1 == b)
                count++;
                a = a >> 1;
        }
```

12. What is static variable and its property? Can you use a static variable outside the file it is declared? ★★★★★

Ans. Static is a qualifier which states that the variable which is declared as static can

- Persist its value between different function calls
- Initialized only once in a file.
- No static variable can't be used outside the file as it has the file scope only.

13. What is register storage class and where it is used. What is CPU register is not free. ★★★

Ans. Register storage class specifies the CPU registers as storage area for the variable. This class is generally used for storing frequently used variables (e.g. loop indices). If CPU registers are not available then the variable is stored in RAM area only (auto).

14. What are different storage classes in C? What's their significance? ★★★★

Ans. Storage classes basically define 4 things about a variable…
Storage location of the variable, its scope, its life and the initial value.
Auto – Storage in Stack memory, uninitialized variables go in .BSS segment. Default value is garbage value, Scope- within the block, Life is, till control remains in block in which variable is defined.
Register – CPU registers if available otherwise on Stack, Default value is garbage value, Scope local to block where variable is defined. Life is till the control remains in block.
Static – Storage in Data segment of RAM if initialized otherwise in BSS, Initial value – Zero, Scope – local to the block in which it is defined, Life – value persist between different function calls. The variable is initialized once only.

Extern – Storage Data Segment of RAM if initialized otherwise BSS, Default value = Zero, Scope Global, Life – As long as the program execution comes to end.

15. What are various C coding standards etc? What are the coding guidelines you use like MISRA (Motor Industry Software Reliability Association) etc.? ★★★

Ans. There are various C coding standards established by different organizations. Some of them are GNU Coding Standards, K&R (Kernighan and Ritchie) Coding Standards, AT&T Bell Coding Standards, MISRA C standards, CERT C programming language secure coding standards etc.

MISRAC stands for "Motor Industry Software Reliability Association" C standards. *MISRA-C: 1998* had 127 rules, of which 93 were required and 34 were advisory; the rules were numbered in sequence from 1 to 127.

The *MISRA-C:2004* document contains 141 rules, of which 121 are "required" and 20 are "advisory"; they are divided into 21 topical categories, from "Environment" to "Run-time failures".

Note that logically the rules can be divided into a number of categories:

- Avoiding possible compiler differences, e.g. the size of a C integer may vary but and INT16 is always 16 bits.
- Avoiding using functions and constructs that are prone to failure, e.g. malloc() may fail.
- Produce Maintainable and Debuggable Code, e.g. naming conventions and commenting.
- Best Practice Rules
- Complexity limits.

16. What's difference between structure and union? ★★★

Ans. When we declare structure, separate memory is allocated to all the members of it but in union memory is allocated to the largest member only, other members just share it. Hence in structure any member can be accessed any time but in union only one member can be accessed at one point of time.

	Structure	**Union**
Access Members	We can access all the members of structure at any time	At a time only one member can be accessed
Memory Allocation	Separate memory allocated for each of the member in structure	Memory allocated to the largest member. Other members share the same space
Initialization	All members can be initialized	Only first member of union can be initialized

17. Can you give some practical examples where structure and union are used? ★★★★

Ans. Structures are generally used to form a Linked list data structure which consists of nodes having data field and address field as single entity.
Structure bit fields can be used to declare data members of variable bit sizes.
Unions are very useful in embedded systems.
It can be used for endianness conversions.
Unions can also be used to define bit addressable memory areas i.e. dividing a datafield into significant sub fields like dividing 32bit data into 4 different bytes.

18. What is enum? How is it different from structure? ★★★

Ans. Enumerated data type variables can only assume values which have been previously declared.

```
enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov,
dec };
enum month this_month;
this_month = feb;
```

In the above declaration, *month* is declared as an enumerated data type. It consists of a set of values, jan to dec. Numerically, jan is given the value 1, feb the value 2, and so on. The variable *this_month* is declared to be of the same type as month, then is assigned the value associated with feb. This_month cannot be assigned any values outside those specified in the initialization list for the declaration of month.

Enumerations are a distinct type consisting of a set of named constants called the enumerator list. Every enumeration type has an underlying type, which can be any integral type except char. Enumerations are integer-types that make code clearer and easier to maintain.

A structure is a value type and the instances or objects of a structure are created in stack. A struct can contain fields, methods, constants, constructors, properties, indexers, operators and even other structure types.

19. What will be the output of following program? ★★★★

```
union Test
{
        int a;
        char b;
} obj;

void main()
{
        obj.a = 5;
        obj.b = 3;
```

```
        printf("%d", obj.a);
    }
```

Ans. As we know all the members of union share common memory space hence when obj.b is assigned some value it replaces the older value of memory space which was assigned as obj.a = 5 hence the **output will be 3.**

20. What do you mean by structure bit fields? How to define 4bit, 5bit and 8 bit variables in C. ★★★★★

Ans. We can define data types of variable no. of bits called bit fields but following restrictions apply to bit fields. You **CANNOT:**

- Define an array of bit fields
- Take the address of a bit field
- Have a pointer to a bit field
- Have a reference to a bit field

The following structure has three bit-field members a, b and c, occupying 12, 6, and 2 **bits** respectively:

```
struct BitFields {
    unsigned int a : 12;
    unsigned int b : 6;
    unsigned int c : 2;
    };
```

21. What will be total size of following structure instance? ★★★

```
struct Test {
                int a;
                char b;
                float c;
        };
```

Ans. Well this one is tricky. If we consider integer to be of 2 bytes, char of one byte and float be of 4 bytes hence the total size seems to be 7 bytes but due to bit packing it is **8 bytes**.
This is due to the fact that int is declared before char hence char also become of 2 bytes.

22. What is difference between typedef and macro definitions? ★★★★

Ans. typedef keyword is used to simplify the datatype definitions they are very useful when you want to change datatypes of no.of variables in your code. So you need to modify only the typedef definition of datatype and it reflects everywhere you used it (sound similar to #define macro!).
Example typedef unsigned int* uptr
Now if you want to declare a long pointer then you just have to write
uptr p;
This saves programming efforts.

Although this purpose can be achieved with macro as well but be careful using macros because macros are just the text replacement hence they may lead you to logical errors.

23. What's difference between gdb and valgrid debuggers? ★★

Ans. gdb is used to debug for compile time and some of run time errors but valgrind is used to check for memory related errors like to identify whether there is any memory leak or a segmentation fault in the code. Hence gdb is a debugger while valgrind is just a memory checker. Gdb let you step interactively in the program while valgrind doesn't. Also Valgrind often shows the cause of fault too.

24. What are Function macros? How they are different from inline functions? ★★★★

Ans. #define SUM(a,b,c) a+b+c
SUM(1, 2, 3) // returns sum of numbers i.e. 6
Function macros are just textual replacement of parameters with functionality to reduce the compiler overhead. Use of function macros improves speed of execution.
Inline function are different from the macros in way that they replace the function call with complete definition of the function hence to reduce the function call and context switching overheads. Condition to use inline functions is that they must be smaller in sizes. Function macros may sometimes produce logical errors.
Inline is just a request to the compiler and it is upto to the compiler whether to substitute the code at the place of invocation or perform a jump based on its performance algorithms.

25. What is disadvantage of using a function macro? ★★★

Ans. Function macros are just the text replacement hence may lead to logical errors.
In last example, what if we define function macro simply like
#define SQR(c) (c * c)
Yes of course if we pass a single variable (like in last example), there is no problem at all but try this…
y = SQR(a+b);
The macro will be replaced as
Y = (a + b * a + b); // which was not desired at all!!
The major disadvantage of macros is that they are not really functions and the usual error checking and stepping through of the code does not occur. Inline functions are expanded whenever it is invoked rather than the control going to the place where the function is defined and avoids all the activities such as saving the return address when a jump is performed. Saves time in case of short codes.

26. Write a generalized function macro to set nth bit of variable. Variable and bit positions are passed by user. ★★★

Ans. #define SET(x, n) x = x | (1 << n)

27. Write a generalized function macro to set and reset nth bit. ★★★★

Ans.

```
#define BITn(n) (0x1 << n)
static int a;
void set_bit_n(void) {
a |= BITn(n);
}
void clear_bit_n(void) {
a &= ~BITn(n);
}
```

28. Write a C function to identify Endianness. ★★★★

Ans. Endianness refers to the arrangement of data bytes inside the memory. Whether the Most Significant Byte (MSB) is placed first (Called Big Endian) or a Least Significant Byte (LSB) is placed first.

```
#define LITTLE_ENDIAN 0
#define BIG_ENDIAN    1

int endian() {
int i = 1;
char *p = (char *)&i;

if (1 == p[0])
    return LITTLE_ENDIAN;
else
    return BIG_ENDIAN;
}
```

29. Write a C function to convert a little endian data to big endian. ★★★★★

Ans.

```
int reverseEndianness(int num)
{
    int byte0, byte1, byte2, byte3;
    byte0 = (num & 0x000000FF) >> 0;
    byte1 = (num & 0x0000FF00) >> 8;
    byte2 = (num & 0x00FF0000) >> 16;
    byte3 = (num & 0xFF000000) >> 24;
    return ( (byte0 << 24) | (byte1 << 16) | (byte2 << 8) | (byte3 << 0) )
}
```

30. What are Function pointers? How they are declared. What do you mean by callback mechanism? ★★★

Ans. When there are various functions having same prototypes but performing different functionality, then we declare a common function pointer pointing to all those functions one by one and call them using that function pointer. Function pointer is a pointer which holds address of a function a normal syntax of declaring a function pointer is
int (*p) (int a, int b); //a pointer which will hold address of functions having two integer parameters and integer return type.
Callback mechanism refers to passing function as parameter to some other function. This mechanism is generally used for event handling.

```
#include <math.h>
#include <stdio.h>

// Function taking a function pointer as an argument
double compute_sum(double (*funcp)(double), double lo,
double hi)
{
   double  sum = 0.0;

   // Add values returned by the pointed-to function '*funcp'
   for (int i = 0;  i <= 100;  i++)
   {
      double  x, y;

      // Use the function pointer 'funcp' to invoke the function
      x = i/100.0 * (hi - lo) + lo;
      y = (*funcp)(x);
      sum += y;
   }
   return sum;
}

int main(void)
{
   double  (*fp)(double);     // Function pointer declaration
   double  sum;

   // Use 'sin()' as the pointed-to function
   fp = &sin;
   sum = compute_sum(fp, 0.0, 1.0);
   printf("sum(sin): %f\n", sum);

   // Use 'cos()' as the pointed-to function
   sum = compute_sum(&cos, 0.0, 1.0);
   printf("sum(cos): %f\n", sum);
   return 0;
}
```

31. What is volatile variable? What's its significance? ★★★★

Ans. Volatile qualifier keyword is used for variables whose values are supposed to be changed by some external device or due to some interrupt hence to avoid the compiler optimization on such variables, we declare them as volatile
In C volatile just tells the compiler - "You don't have enough knowledge to assume the value of this variable hasn't changed". There is no "section" eg BSS, CSS for it.
volatile is a type qualifier not a storage class specifier, so it does not determine storage location at all; it affects the definition of a variable's type, not its storage. Consider it a flag to the compiler to prevent certain types of optimizations. Its very handy in embedded programming, where memory at a certain address may "change" due to a hardware device input.

32. What is difference in following declarations...? ★★★★★

const int a;
int const a;

const int *a;
int * const a;
int const * a const;

Ans. The first two mean the same thing, namely 'a' is a const (read-only) integer. The third means a is a pointer to a const integer (that is, the integer isn't modifiable, but the pointer is). The fourth declares a to be a const pointer to an integer (that is, the integer pointed to by a is modifiable, but the pointer is not). The final declaration declares a to be a const pointer to a const integer (that is, neither the integer pointed to by a, nor the pointer itself may be modified).

33. Why is const keyword so important? ★★★★

Ans. **Const** keyword is very important because…

- The use of const conveys some very useful information to someone reading your code. In effect, declaring a parameter const tells the user about its intended usage. If you spend a lot of time cleaning up the mess left by other people, you'll quickly learn to appreciate this extra piece of information. (Of course, programmers who use const , rarely leave a mess for others to clean up.)
- const has the potential for generating tighter code by giving the optimizer some additional information
- Code that uses const liberally is inherently protected by the compiler against inadvertent coding constructs that result in parameters being changed that should not be. In short, they tend to have fewer bugs

34. Can a parameter be both const and volatile? ★★★

Ans. Yes. An example is a read-only status register. It is volatile because it can change unexpectedly. It is const because the program should not attempt to modify it.

35. Can a pointer be volatile? ★★★

Ans. Yes, although this is not very common. An example is when an interrupt service routine modifies a pointer to a buffer.

36. What's wrong with the following function? ★★★

```
int square(volatile int *ptr)
{
return *ptr * *ptr;
}
```

Ans. This one is wicked. The intent of the code is to return the square of the value pointed to by *ptr . However, since *ptr points to a volatile parameter, the compiler will generate code that looks something like this:

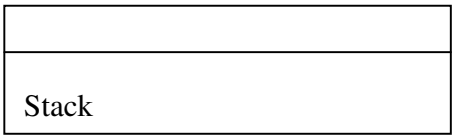
```
int square(volatile int *ptr)
{
        int a, b;
        a = *ptr;
        b = *ptr;
        return a * b;
}
```

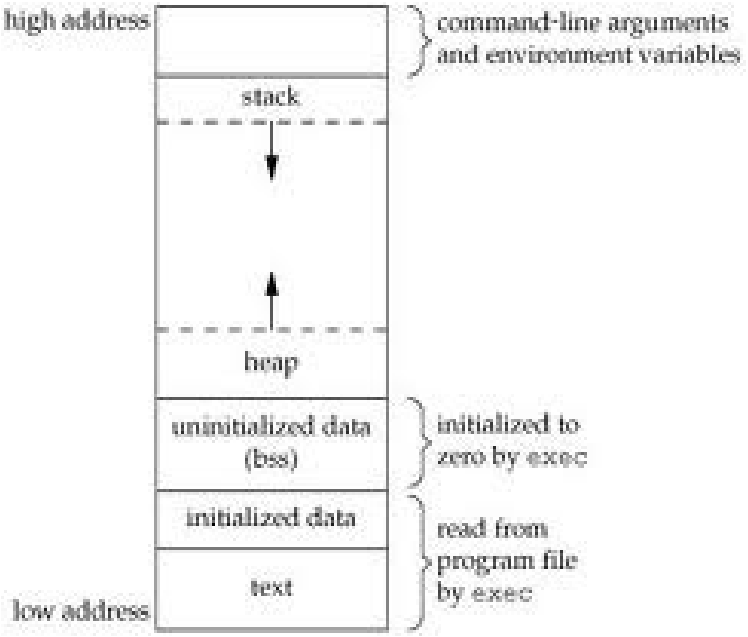
Because it's possible for the value of *ptr to change unexpectedly, it is possible for a and b to be different. Consequently, this code could return a number that is not a square! The correct way to code this is:

```
long square(volatile int *ptr)
{
int a;
a = *ptr;
return a * a;
}
```

37. What are different storage segments of RAM? ★★★

Ans.





The computer program memory is organized into the following:

- Data Segment (Data + BSS + Heap)
- Stack
- Code segment

Data

The data area contains global and static variables used by the program that are explicitly initialized with a value. This segment can be further classified into a read-only area and read-write area. For instance, the string defined by char s[] = "hello world" in C and a C statement like int debug=1 outside the "main" would be stored in initialized read-write area. And a C statement like const char* string = "hello world" makes the string literal "hello world" to be stored in initialized read-only area and the character pointer variable string in initialized read-write area. Ex: **both** static int i = 10 and global int i = 10 will be stored in the data segment.

BSS (Block Started by Symbol)

The **BSS segment**, also known as *uninitialized data*, starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code. For instance a variable declared static int i; would be contained in the BSS segment. Hence, the bss section typically includes all uninitialized variables declared at the file level (i.e., outside of any function) as well as uninitialized local variables **declared with the static keyword.** An implementation may also assign statically-allocated variables initialized with a value consisting solely of zero-valued bits to the bss section.

Heap

The heap area begins at the end of the **BSS segment** and grows to larger addresses from there. The heap area is managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single "heap area" is not required to fulfill the contract of malloc/realloc/free; they may also be implemented using mmap to reserve potentially non-contiguous regions of virtual memory into the process' virtual address space). The heap area is shared by all shared libraries and dynamically loaded modules in a process.

Stack

The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory. A "stack pointer" register tracks the top of the stack; it is adjusted each time a value is "pushed" onto the stack. The set of values pushed for one function call is termed a "stack frame". A stack frame consists at minimum of a return address. Automatic variables are also allocated on the stack.

The stack area traditionally adjoined the heap area and grew in the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. With large address spaces and virtual memory techniques they tend to be placed more freely, but they still typically grow in opposite directions. On the standard PC x86 architecture the stack grows toward address zero, meaning that more recent items, deeper in the call chain, are at numerically lower addresses and closer to the heap. On some other architecture it grows the opposite direction.

38. What's application of pragma keyword? ★★★★

Ans. It is a compiler or OS specific keyword used to send some message to compiler. Messages can be regarding changing function calling sequence or resizing the stack area or deciding the storage location of variables etc.

The **#pragma** directives offer a way for each compiler to offer machine- and operating system-specific features while retaining overall compatibility with the C and C++ languages. Pragmas are machine- or operating system-specific by definition, and are usually different for every compiler.

Pragmas can be used in conditional statements, to provide new preprocessor functionality, or to provide implementation-defined information to the compiler.

```
#pragma interrupt_handler timer_handler:4 //telling the compiler that using interrupt
//handler of priority 4
void timer_handler(void)
{
:
:

}
```

OR

```
// pragma_directive_bss_seg.cpp
int i;                  // stored in .bss
#pragma bss_seg(".my_data1")
int j;                  // stored in "my_data1"

#pragma bss_seg(push, stack1, ".my_data2")
int l;                  // stored in "my_data2"

#pragma bss_seg(pop, stack1)   // pop stack1 from stack
int m;                   // stored in "stack_data1"

int main() {
}

// pragma_directive_data_seg.cpp
int h = 1;                  // stored in .data
int i = 0;                  // stored in .bss
#pragma data_seg(".my_data1")
int j = 1;                  // stored in "my_data1"

#pragma data_seg(push, stack1, ".my_data2")
int l = 2;                  // stored in "my_data2"

#pragma data_seg(pop, stack1)   // pop stack1 off the stack
int m = 3;                   // stored in "stack_data1"

int main() {
}
// pragma_directive_const_seg.cpp
// compile with: /EHsc
#include <iostream>

const int i = 7;              // inlined, not stored in .rdata
const char sz1[]= "test1";     // stored in .rdata

#pragma const_seg(".my_data1")
const char sz2[]= "test2";     // stored in .my_data1

#pragma const_seg(push, stack1, ".my_data2")
const char sz3[]= "test3";     // stored in .my_data2

#pragma const_seg(pop, stack1) // pop stack1 from stack
const char sz4[]= "test4";     // stored in .my_data1

int main() {
   using namespace std;
  // const data must be referenced to be put in .obj
```

```
  cout << sz1 << endl;
  cout << sz2 << endl;
  cout << sz3 << endl;
  cout << sz4 << endl;
}
// pragma_directive_code_seg.cpp
void func1() {              // stored in .text
}

#pragma code_seg(".my_data1")
void func2() {              // stored in my_data1
}

#pragma code_seg(push, r1, ".my_data2")
void func3() {              // stored in my_data2
}

#pragma code_seg(pop, r1)     // stored in my_data1
void func4() {
}

int main() {
}
```

39. Difference between function macro and inline function? ★★★★

Ans. Function macro is handled by preprocessor but inline function is handled by compiler. Macro is just the text replacement and may cause logical errors but inline functions replace the function call with the complete function definition.

40. Difference between function templates and function overloading? ★★★

Ans. Function template operate on same no. of parameters and produce same functionality although works for different types of parameters...while function overloading, we have to write different functions with the same name but variable no. of parameters or return types or different types of parameters.

41. What are static functions? ★★★★

Ans. Static functions are the **global** functions whose scope is restricted to the file in which they are defined. If we try to call that function in other file, we get an error. In C, functions are global by default. The "*static*" keyword before a function name makes it static. For example, below function *fun()* is static.

```
static int fun(void)
{
  printf("I am a static function ");
}
```

Unlike global functions in C, access to static functions is restricted to the file where they are declared. Therefore, when we want to restrict access to functions, we make them static. Another reason for making functions static can be reuse of the same function name in other files.

For example, if we store following program in one file *file1.c*

```
/* Inside file1.c */
static void fun1(void)
{
 puts("fun1 called");
}
```

And store following program in another file *file2.c*

```
/* Iinside file2.c  */
int main(void)
{
 fun1();
 getchar();
 return 0;
}
```

Now, if we compile the above code with command "*gcc file2.c file1.c*", we get the error *"undefined reference to `fun1'"* . This is because *fun1()* is declared *static* in *file1.c* and cannot be used in *file2.c*.

42. What is dangling pointer? ★★★★★

Ans. Dangling pointers arise when an object is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory.

```
{
  char *dp = NULL;
  /* ... */
  {
     char c;
     dp = &c;
  } /* c falls out of scope */
    /* dp is now a dangling pointer */
```

```
}
```

43. What do you mean by a forward reference? ★★★

Ans. int a = 10;
int *p = &a;
*p is called forward reference

44. What's difference between malloc and calloc? What are other dynamic memory allocation functions? ★★★

Ans.

There are two differences.
First, is in the number of arguments. Malloc() takes a single argument (memory required in bytes), while calloc() needs two arguments ().
Secondly, malloc() does not initialize the memory allocated, while calloc() initializes the allocated memory to ZERO.

- calloc() allocates a memory area, the length will be the product of its parameters. calloc fills the memory with ZERO's and returns a pointer to first byte. If it fails to locate enough space it returns a NULL pointer.

Syntax: ptr_var=(cast_type *)calloc(no_of_blocks , size_of_each_block);
i.e. ptr_var=(type *)calloc(n,s);

- malloc() allocates a single block of memory of REQUSTED SIZE and returns a pointer to first byte. If it fails to locate requsted amount of memory it returns a null pointer.

45. What is segmentation fault? What are main reasons? ★★★★

Ans. Segmentation fault is a bus error or access violation when CPU tries to access a memory location when there is no physical address for the same.
A few causes of a **segmentation fault** can be summarized as follows:

- Attempting to execute a program that does not compile correctly. Note that while most compilers will not output a binary given a compile-time error;
- Dereferencing NULL pointers i.e. trying to assign values at unaddressed location
- Attempting to access memory the program does not have rights to (such as kernel structures in process context)
- Attempting to access a nonexistent memory address (outside process's address space)
- Attempting to write read-only memory (such as code segment),
- A buffer overflow/ Stack overflow,
- Using uninitialized pointers.

Here are few example code snippets…

```
int main(void)
{
   char *s = "hello world";
   *s = 'H';
}
```

Or

Trying to assign some value at unaddressed location.

```
int *ptr = NULL;
*ptr = 1;
```

Or

Recursion without condition which may cause stack overflow

```
int main(void)
{
  main();
  return 0;
}
```

46. What is memory leak problem? ★★★★

Ans. A memory leak occurs when a computer program consumes memory but is unable to release it back to the operating system.

Example: *p = (int*)malloc(sizeof(int));

p = &a; //where a is some other integer variable.

Now the memory allocated earlier which was being reference by p now has no reference hence causes the memory leakage.

Memory leak description: Memory is allocated but not released causing an application to consume memory reducing the available memory for other applications and eventually causing the system to page virtual memory to the hard drive slowing the application or crashing the application when than the computer memory resource limits are reached the system may stop working as these limits are approached.

It usually happens when we dynamically allocate memory using malloc or calloc functions but forget to free it.

47. What are linker files and makefiles? ★★★

Ans. When it comes to so many source files need to be compiled together then the command for compiling files are integrated in file called makefile. This file contains target and dependencies and only those files are compiled whose timestamps have been changed i.e. which are modified. The files with no change are not compiled again.

48. Which header file is included for consoled input/output in Linux environment? ★★

Ans.

49. How to create child process in linux? ★★★

Ans. Prototype of the function used to create a child process is pid_t fork(void);
Fork is the system call that is used to create a child process. It takes no arguments and returns a value of type pid_t.
If the function succeeds it returns the pid of the child process created to its parent and child receives a zero value indicating its successful creation.
On failure, a -1 will be returned in the parent's context, no child process will be created, and errno will be set
The child process normally performs all its operations in its parents context but each process independently of one another and also inherits some of the important attributes from it such as UID, current directory, root directory and so on.

Microcontroller Peripherals

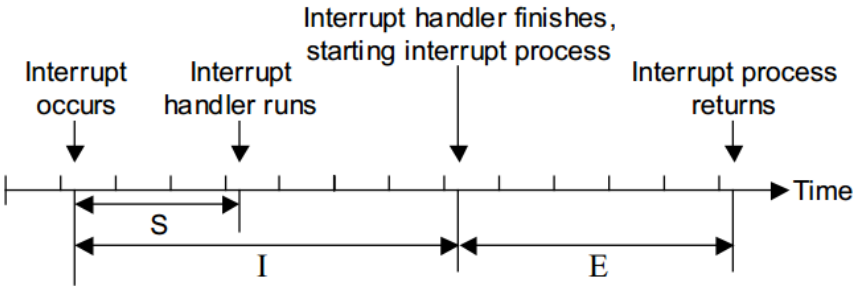
1. What are various peripherals inside a common microcontroller? ★★★

Ans. Usually a microcontroller is equipped with RAM, ROM (EEPROM or FLASH), Timers, Interrupt Controllers, serial communication interfaces (UART, SPI, I2C, CAN etc.), internal oscillator, ADC, DAC etc.

2. What do you mean by an Interrupt and interrupt latency? ★★★

Ans. Interrupts are the signals that disturb normal flow of a program and may lead the flow to switch to some other routine called Interrupt service routine or ISR.

Interrupt latency refers **to** the time that elapses from when an interrupt is generated to when the source of the interrupt is serviced. When an interrupt fires, the microprocessor executes an ISR that has been installed to serve the interrupt. The amount of time that elapses between a device interrupt request and the first instruction of the corresponding ISR is known as interrupt latency.



I = interrupt latency
E = interrupt process execution time
S = time to start the interrupt handler

3. How to optimize interrupt latency? ★★★★

Ans. Interrupt latency can be reduced by writing smaller ISRs, use of cache memories and using FIQ methods of interrupts.

4. Difference between RET and RETI instruction of 8051 assembly program? ★★★

Ans. RET instruction doesn't reset the interrupt flag of IE register which was set when an interrupt was generated while RETI instruction first resets all flags before CPU returns to the main flow of program.

5. What is difference between Polling and Interrupt? ★★★

Ans. Polling is the way in which microcontroller waits for an event to occur (say for a flag to set) and only after the event occurs it takes further action. Until that event occurs microcontroller just halts its current flow of execution for the time

hence polling is not an efficient way of programming because most of the times it consumes lot of CPU time in waiting for an even.
On the other hand Interrupts are the way where CPU continues doing its normal program execution without waiting for such events. But whenever such events occur, they generate an interrupt and then only CPU goes to serve them executing corresponding ISR (Interrupt Service Routine).

6. How Interrupt handling takes place? ★★★

Ans. Whenever an internal or external interrupt takes place, first the microcontroller executes the current instructions completely and stores the address of next instruction to be executed on the stack. After storing the address on stack it looks for the interrupt vector table to find the address of ISR to be executed for the interrupt occurred. Interrupt vector table is the list of addressed where different ISRs are written (or supposed to be written by developer). As soon as it gets the address, it jump to specific ISR and start executing its instructions. After completing the ISR execution, microcontroller again pops up the address of instruction which was stored on stack and start executing onwards.

7. What do you mean by an ISR (Interrupt Service Routine)? ★★

Ans. ISR (Also called an interrupt handler), is a special function which is invoked (not called like normal function) whenever an interrupt occurs. Each and every interrupt has its separate ISR.

8. Can an ISR return a value? If so what's return type? ★★★

Ans. ISRs neither returns a value nor can accept the parameters. They are not the usual functions. Hence ISRs are always defined as void.

9. What do you mean by a reentrant function? ★★★★★

Ans. A computer program or subroutine or a function is called reentrant if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution. The interruption could be caused by an internal action such as a jump or call, or by an external action such as a hardware interrupt or signal. Once the reentered invocation completes, the previous invocations will resume correct execution.
A reentrant function may be shared by several processes at the same time. When a reentrant function is executing, another process can interrupt the execution and then begin to execute that same reentrant function. Reentrant functions are often required in real-time applications or in situations where interrupt code and non-interrupt code must share a function.
The **reentrant** function attribute allows you to declare functions that may be reentrant and, therefore, may be called recursively. For example:

```
int calc (char i, int b) reentrant  {
int  x;
x = table [i];
return (x * b);
```

}

10. Suppose CPU is executing an ISR and meanwhile another interrupt occurs, what steps will be taken? ★★★

Ans. In this case, actions will be taken depending upon priority of the new interrupt. If the newly generated interrupt has higher priority than the current interrupt being served then CPU switches its execution to the ISR of new interrupt and after completing it returns back to the last ISR. If the priority of new interrupt is lower, then it is served after completion of the current ISR.

11. What do you mean by masking an interrupt? ★★★

Ans. Masking means particular interrupt will not be served if occurred. There are respective interrupt flags in CPU register for each of its interrupt. These flags are set by interrupts when they occur and cleared by CPU when the ISR is executed. Similarly there are interrupt enable flags which decide whether to serve particular interrupt or not. These flags can be set or reset by programmer. In general conventions, if flag for particular interrupt is reset then it is said to be masked i.e. CPU will not jump to its ISR if the interrupt occurs.
Masking is important when two ISRs are using and modifying the same information.

12. What is a trap? How will you differentiate between trap and an interrupt? ★★★

Ans. Interrupt is a general term used for exceptions, faults, aborts, traps. Hence trap is also an interrupt but usually a software interrupt which in generated due to some error or to call some system routines. Trap is always of higher priority than the user code.
A *trap* is a special kind of *interrupt* which is commonly referred to as software *interrupt*. An *interrupt* is a more general term which covers both *hardware interrupts* (interrupts from hardware devices) and *software interrupts* (interrupts from software, such as *traps*).

13. What are various types of interrupts involved in an Embedded System? ★★★

Ans. Interrupts can be categorized into **maskable interrupt**, non-**maskable interrupt, interprocessor interrupt, software interrupt and spurious interrupt.** Maskable interrupts can be ignored or turned off by programmer by setting respective bit in interrupt mask register whereas non maskable interrupts are the hardware interrupts that can never be ignored or turned off. Timer interrupts especially watchdog timer interrupts are mostly non-maskable. Interprocessor interrupt is a special case of interrupt generated by one processor to interrupt another in a multiprocessor system.

14. What is difference between interrupts and exceptions? ★★★

Ans. Both interrupts and exceptions alter the normal program flow. The difference is that interrupts are used to handle external events like timer overflow, serial communication etc whereas exceptions are used to handle instruction faults like division by zero, undefined opcode etc.

15. Differentiate between Function and an ISR. ★★★

Ans.

S. No.	Function	ISR
1.	Set of instructions for performing an action	ISR is independent
2.	Called by Proccess/Task/ISR	Invoked by H/W or S/W
3.	Each function has a context	ISR has context as well as priority
4.	Provision for nested functions	Other interrupts can preempt the current running interrupt depending upon priority and type of scheduling

16. What is application of Timers in an Embedded System? ★★

Ans. Timers are very important and the most used peripheral in any embedded systems. They are the hardware peripheral usually integrated on the microcontroller chip only. Timers are used to provide timing constraints to different tasks, programs or to provide delays between executions, generate wait states etc. Timers can also be used as counters to count external events (pulses) hence the great application can be frequency meter.
Timer applications can be broadly provided as

- Implementing real time clock (RTC) for the system
- Initiating an event after preset time delay
- Initiating an event after comparison of preset times
- Capturing count value in timer on an event
- Finding time interval between two events
- Watchdog timer
- Baud rate control for serial communication
- Scheduling of various tasks in RTOS
- Time slicing of various tasks
- Time Division Multiplexing (TDM)

17. Can you generate a square wave using a single timer interrupt? ★★★

Ans. YES. Steps will be…

a. Enable timer interrupt
b. Load calculated timer values in timer registers.

c. Start Timer
d. Complement the pin of microcontroller in Timer ISR.

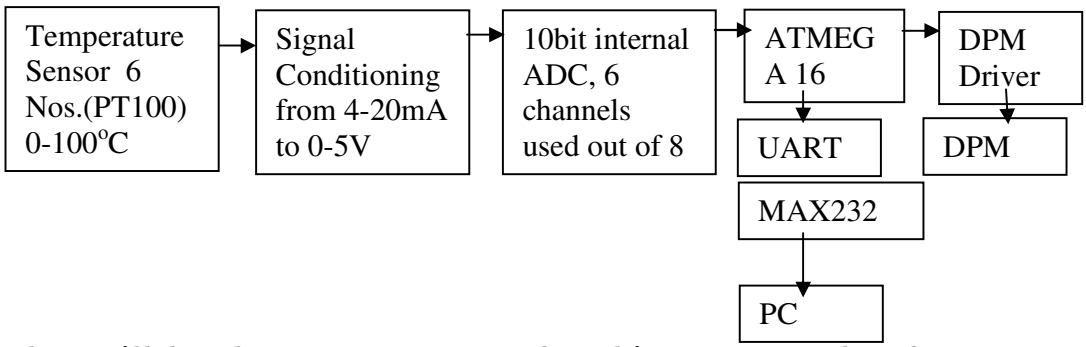
18. What do you mean by a Timer prescaler? What's its significance? ★★

Ans. The prescaler takes the basic timer clock frequency (which may be the CPU clock frequency or may be some higher or lower frequency) and divides it by some value before feeding it to the timer, according to how the prescaler register(s) are configured. The prescaler values that may be configured might be limited to a few fixed values (powers of 2), or they may be any integer value from 1 to 2^P, where P is the number of prescaler bits.

The purpose of the prescaler is to allow the timer to be clocked at the rate you desire. For shorter (8 and 16-bit) timers, there will often be a tradeoff between resolution (high resolution requires a high clock rate) and range (high clock rates cause the timer to overflow more quickly). For example, you cannot (without some tricks) get 1us resolution and a 1sec maximum period using a 16-bit timer. If you want 1us resolution you are limited to about 65ms maximum period. If you want 1sec maximum period, you are limited to about 16us resolution. The prescaler allows you to juggle resolution and maximum period to fit your needs.

As an example of a fixed selection prescaler, many AVR devices allow for fixed prescale values of 1, 8, 64, 256 and 1024. If the system clock frequency is e.g. 8MHz, this results in a timer clock period of 0.125, 1, 8, 32 or 128 microseconds per clock tick. Likewise, the 9S12 prescaler allows fixed prescale values in powers of 2: 1, 2, 4, 8, 16, 32, 64 and 128.

19. Block diagram of any controller based project with at least 3 peripherals (ADC, Timers and UART as shown above) ★★★
Ans.



20. What will be the output count of 10 bit ADC at value for 1 V input if ref is 5V? ★★★
Ans. Output Count = 1x1024/5 = 204

21. What are different parameters need to be considered while selecting an ADC? ★★★

Ans. While selecting an ADC, parameters to be taken care of are - Resolution (depends on data bus width), Chip Size, Cost and Package

22. Can a reference voltage of ADC be configured using software? ★★★★

Ans. Yes, some microcontrollers like AVR ATMEGA16 come with internal ADCs i.e. ADC is integral part of the microcontroller itself hence provide ADC configurations registers. Configuration registers can be used to set reference voltage as well.

23. What do you mean by sampling and quantization? ★★

Ans. The terms are related to Analog/Digital Conversion. Sampling refers to taking some instantaneous values of analog quantity and Quantization refers to assigning some discrete values to it.

24. How 16x2 LCD works in 4 bit mode? What signal to be given for initialization? ★★★

Ans. It requires a special command 0x28 to be sent along with usual commands and then the data can be sent in form of nibbles instead of bytes.

25. What is watchdog timer? State and its application. ★★★

Ans. A watchdog timer is a simple countdown timer which is used to reset a microprocessor after a specific interval of time. In a properly functioning system, software will periodically "pet" or restart the watchdog timer. After being restarted, the watchdog will begin timing another predetermined interval. When software or the device is not functioning correctly, software will not restart the watchdog timer before it times out. When the watchdog timer times out, it will cause a reset of the microcontroller. If the system software has been designed correctly and there has been no hardware failure, the reset will cause the system to operate properly again. The reset condition must be a "safe" state. For instance, it would not be wise to have the reset state of a magnetic stripe card reader enabling the write head. Physical location

- Within a chip external to the processor
- In circuitry included within the CPU chip, as is done in many microcontrollers
- On an expansion card in the computer's chassis
- Clock source for the watchdog
- The CPU clock
- An independent clock, so that a CPU clock failure will cause a watchdog timeout

- How long a timeout must be to trigger the watchdog
- Typical timeouts are from 10 milliseconds to 10 seconds
- What action the watchdog takes on a timeout
- Processor reset
- Non-maskable interrupt

26. What is PLL (Phase Locked Loop)? What's its application? ★★★

Ans. A **phase-locked loop** (PLL) is a closed-loop frequency-control system based on the phase difference between the input clock signal and the feedback clock signal of a controlled oscillator. The main blocks of the PLL are the phase frequency detector, charge pump, loop filter, voltage controlled oscillator (VCO), and counters.

The PFD detects the difference in phase and frequency between the reference clock and feedback clock inputs and generates an "up" or "down" control signal based on whether the feedback frequency is lagging or leading the reference frequency. These "up" or "down" control signals determine whether the VCO needs to operate at a higher or lower frequency, respectively.

The PFD outputs these "up" and "down" signals to a charge pump. If the charge pump receives an up signal, current is driven into the loop filter. Conversely, if it receives a down signal, current is drawn from the loop filter.

The loop filter converts these signals to a control voltage that is used to bias the VCO. Based on the control voltage, the VCO oscillates at a higher or lower frequency, which affects the phase and frequency of the feedback clock. If the PFD produces an up signal, then the VCO frequency increases. A down signal decreases the VCO frequency. The VCO stabilizes once the reference clock and the feedback clock have the same phase and frequency. The loop filter filters out jitter by removing glitches from the charge pump and preventing voltage over-shoot.

27. What is UART? Why it is used for? ★★

Ans. UART stands for Universal Asynchronous Receiver Transmitter. This is either integrated over a microcontroller chip or can be used as separate IC. Job of UART is to perform framing of data, assigning start and stop bits, clock configurations etc.

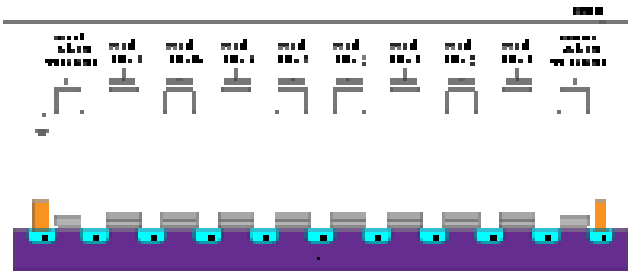
28. What is difference between NAND flash and NOR flash? ★★★

Ans. NOR and NAND flash differs in two important ways:

The connections of the individual memory cells are different the interface provided for reading and writing the memory is different (NOR allows random-access for reading, NAND allows only page access)

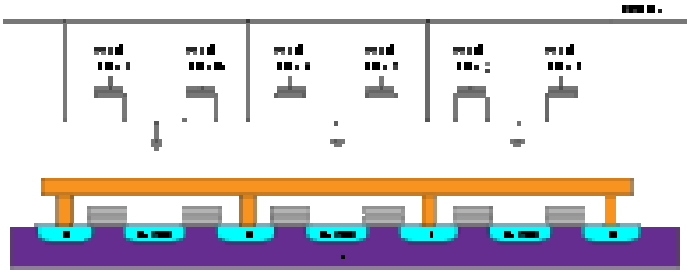
NAND flash uses tunnel injection for writing and tunnel release for erasing. NAND flash memory forms the core of the removable USB storage devices

known as USB flash drives, as well as most memory card formats and solid-state drives available today.



•

NAND flash memory wiring and structure on silicon



NOR flash memory wiring and structure on silicon

The NAND type is primarily used in memory cards, USB flash drives, solid-state drives, and similar products, for general storage and transfer of data. The NOR type, which allows true random access and therefore direct code execution, is used as a replacement for the older EPROM and as an alternative to certain kinds of ROM applications. However, NOR flash memory may emulate ROM primarily at the machine code level; many digital designs need ROM (or PLA) structures for other uses, often at significantly higher speeds than (economical) flash memory may achieve. NAND or NOR flash memory is also often used to

store configuration data in numerous digital products, a task previously made possible by EEPROMs or battery-powered static RAM.

29. What is RAM? ★★

Ans. RAM stands for random access memory. Programs and data currently run by CPU are stored in RAM memory. It is also called primary memory. RAMs are of two types static RAM or SRAM and dynamic RAM or DRAM. SRAMs are made of flip flops and are faster than DRAMs and costlier too. DRAMs are made of capacitors and need to be refreshed periodically in order to save data.

30. What is purpose of a PWM system? State its applications. ★★★

Ans. Purpose of PWM peripheral is to generate Pulse Width Modulated Waveforms at microcontroller output pins. Timers and Compare registers are usually associated with PWM system in order to configure frequency of PWM waveforms. PWM Technology is widely used in motor speed control applications. There are various types of PWM techniques involved. The most basic form of PWM is called Sine wave PWM.

31. What is JTAG, why is it used for? ★★★★

Ans. Boundary scan is a method for testing interconnects (wire lines) on printed circuit boards or sub-blocks inside an integrated circuit. Boundary scan is also widely used as a debugging method to watch integrated circuit pin states, measure voltage, or analyze sub-blocks inside an integrated circuit.
The ***Joint Test Action Group*** **(JTAG)** developed a specification for boundary scan testing that was standardized in 1990 as the IEEE Std. 1149.1-1990.
The boundary scan architecture also provides functionality which helps developers and engineers during development stages of an embedded system. A JTAG Test Access Port (TAP) can be turned into a low-speed logic analyzer.

32. What are various buses involved in an Embedded System? ★★

Ans. The buses are used to pass the messages between different components of the system. There are buses existing as:
• **Memory Bus**: it is related to the processor that is connected to the memory (RAM) using the data bus. This bus includes the collection of wires that are in series and runs parallel to each other to send the data from memory to the processor and vice versa.
• **Multiplexed Address/Data Bus**: Multiplex data bus consists of the bus that can read and write in the memory but it decreases the performance due to the time consumed in reading and writing of the data in the memory.
• **De-multiplexed Bus:** these consists of two wires in the same bus, where one wire consists of the address that need to be passed and the other one consists of the data that need to be passed from one to another. This is a faster method compared to other.

• **Input/Output bus**: it uses the multiplexing techniques to multiplex the same bus input and output signals. This creates the problem of having the deadlock due to slow processing of it.

33. How does a multiplexed address/data bus function? What is advantage of using multiplexed address/data bus? ★★★

Ans. The memory bus is used to carry the address and the data from the processor to the memory so that it can be easily accessed by the devices. These buses carry the value of the data that has to be passed for the proper functioning. The use of the technique "Time division multiplexing" is used that allow the reading and writing of the data to be done from the same bus line. This requires lots of time to be given to the bus so that it can complete the read and write operation of the data in the memory. This is very expensive process due to the data transfer technique that is used in between the processor and the memory. This also gives the concept of cache and provides algorithms to solve the problems occurring in read and writes operations.

34. Give some examples of serial and parallel buses. ★★★

Ans. Serial buses include I2C (used for communication between chip on single board or module), SPI (used for communicating with fast peripherals and memories), CAN (used in automobiles for connecting difference control systems having individual processors), USB (for connecting peripherals with PCs like mouse, keyboard, memory devices etc) etc.
Examples of parallel buses are ISA, EISA, PCI, PC104, VME etc which are used to in PCs and network devices.

Communication Protocols

CAN

1. What is a CAN network, why is it so popular? Where it is used dominantly? ★★★

Ans. CAN is a serial asynchronous communication protocol mainly used when 2 or more microprocessor units need to communicate with each other. CAN is a multi-master bus access protocol dominantly used in automotive industries.
A CAN network basically consists of number of CAN nodes (also called Electronic Control Units or ECUs typically about 70 in a vehicle). Each of these nodes has one host processor including CAN controller inside the chip and a CAN transceiver. CAN protocols works on the principle of message broadcasting. There is no node id. CAN works on message ID only.
CAN become popular because the complete protocol (physical + data link layers) is integrated in a module (called CAN controller) hence a developer need not to worry about protocol functions. Hence CAN is very easy to use. Even if one needs to implement advanced protocol features then he may buy higher level protocols like CANOpen or Device net.
Also the implementation of CAN is low cost.

2. Which layers of OSI model involved in CAN Protocol? ★★★

Ans. Physical layer, Data link layer (Medium Access Control and Logical Link Control) and the Application layer. All other layers are bypassed by basic CAN protocol. They are implemented in other higher level protocols like CAN Open, Device net, J1939 etc. Medium Access Layer is responsible for bus arbitration, message framing (encapsulation and decapsulation) and error detection while Logical Link Layer looks after message filtering, error recovery management and overload protection features.

3. What are the areas other than automotive industries, where CAN is used? ★★★

Ans. Industrial Automation, SAE J1939 is used in Agriculture vehicles, Semiconductor Industry Wafer handlers, Robotic and Motion control applications (CANOpen, Device Net), Medical Equipments like X-Ray, CAT Scanners etc. (CANOpen due to its reliability), Coffee vending machine

4. Why CAN is said to be so reliable? ★★★

Ans. There are many reasons.
When a node sends (rather broadcasts) a message over the bus, the same message is received by each node present in the network so say if 999 nodes in a 1000 nodes network say that the message is correct but 1 says message is not correct i.e. reports an error so that particular node is need to be removed. This is called a majority vote scheme.

There is one more reason for calling it reliable. No message on the CAN network is lost because when any node loses its arbitration it can send its message again after the higher priority message of the other node has been sent. Also CAN has got very effective error detection techniques.

5. Which industrial standard does CAN follow? ★★★
Ans. ISO11898

6. How CAN is better than Ethernet. ★★★
Ans. The messages in Ethernet are very long and Ethernet can't be used in real time applications hence because of length of message they can't be interrupted.

7. How does CAN replace the dual port RAM technology? ★★★

8. What is mask filter in CAN? ★★★
Ans. Most of the CAN controllers provide you a mechanism to filter out the reception of messages at hardware level. You set the mask filter and if the ID of an incoming message passes that filter only then it is received and the controller is interrupted. In mask filter you can specify which bits of an incoming acceptable message must be DOMINANT, which must be RECESSIVE and which one are don't care.

9. Draw a block diagram of a typical CAN network and briefly explain.
Ans.

10. How many maximum nodes are possible in a CAN network? ★★★
Ans. Although no. of nodes are not specified by a CAN network. 64 nodes network is common

11. Why Remote Frame is not usually recommended to be used? ★★★
Ans. Remote Frames are identical to Data Frames with Data field missing. This frame is not generally used because today various manufacturers provide CAN chips and most of them interpret remote frames differently.

12. What are the main frames of a CAN Protocol? ★★
Ans. Data Frame, Remote Frame, Error Frame and Overload Frame.

13. If 2 message IDs 0x74 and 0x7D happen to win the bus, which one will win? ★★★
Ans. 0x74. As this no. is less than the other one and in CAN network lower is the value of message ID higher is its priority.

14. What is dominant and recessive state? ★★★

Ans Dominant – 0, recessive – 1

15. What's maximum speed and length of a CAN network? ★★

Ans. 1Mbps for 40-45m and 500kbps for 100m. Speed of CAN communication can be configured using CAN controller registers.

16. What is meant by bus arbitration? ★★★

Ans. Suppose initially the bus is idle and two or more nodes try to access the bus at the same time. So first all of them transmit their SOF bit (0). Then they start transmitting their message IDs bit by bit. After every bit transmitted they sense the bus whether the same bit is there or not. Suppose there are three nodes A, B and C sending message IDs. Now if A sends a recessive bit and senses bus back (called bit monitoring) and finds it to be dominant, node A immediately stops transmitting its ID and goes into receiver/slave mode. This happened because at that time B was sent the 0bit and C sent the 1 bit. CAN bus is a wired AND logic hence if any of the node sends 0, CAN will be in dominant stage (0).
Now only nodes B and C are in the race. Same process happens with these two as well and one of them wins the bus. Say if B wins the bus then bus is said to be arbitrated to B i.e. B will be master and rest other nodes will be slaves.

17. What is the disadvantage of a CAN network? ★★★

Ans. Due to the signal propagation time in bit monitoring process, the maximum length of a CAN network is limited to few meters. At 1Mbps of speed length of 40-45m is possible. If you want to go for longer network, you need to compromise with the speed.

18. Briefly describe the functioning of a CAN network. ★★★

Ans. After winning the bus, when one of the node broadcasts a message it is seen by each of the node present in the network. Each node contains a message filter inside the CAN controller which decides whether the message is intended for it or not. If message is intended for particular receiver (filtering is done at LLC level), it is transferred to application other wise discarded.

19. What are the other higher layer protocols of CAN? ★★★

Ans. J1939, Device Net (by Allen Bradley now Rockwell Automation), CAN Open, OSEK Net etc. if more data need to be transmitted or some additional features need to be added, these protocols are useful.

20. What is standard and extended CAN? ★★★

Ans. Standard CAN or CAN 2.0A involves an 11 bit message ID while extended CAN or CAN 2.0B involves a 29 bit message ID in the data frame.

21. Which bit of CAN frame decides whether the ID is standard or extended? ★★★

Ans. It's IDE (ID-Extension) bit inside the control field that decides the CAN frame format. 0 for 11bit and 1 for 29bit format.

22. Can both the formats (standard and extended) exist on the same CAN network? ★★★★

Ans. Although not used so much but YES they can exist on the same bus. There is a standard called MilCAN or Military CAN which uses CANOpen (uses 11bit ID) and J1939 (29 bit ID) both on the same bus.

23. What if both 11bit and extended 29bit protocol chips happen to broadcast message at the same time. ★★★★

Ans. Standard CAN i.e. 11bit version will win the bus because it always pursue higher priority over its 29bit counterpart. This is because of the dominant IDE bit of 11bit identifier. On the other hand RTR bit of 29bit format is shifted at the end of arbitration field and the older place is replaced by SRR (Substitute Remote Request) which is always recessive and prevents 29bit format to be of higher priority.

24. Which encoding technique is used for CAN Protocol? ★★★

Ans. It's NRZ (Non Return to Zero) because NRZ principles provides higher data rates as compared to Manchester encoding but on the other hand NRZ gives less no. of signal edges for synchronization purpose.

25. What do you mean by bit stuffing, why it is used? ★★★

Ans. To avoid the problem with NRZ encoding, bit stuffing is used. Bit stuffing is a method in which an extra complement bit is stuffed with the stream after every 5 successive bits of same polarity. Bit stuffing is done by CAN controller. On the receiver side stuffed bits are removed by CAN controller hence they are not seen by the application.

26. Which fields of a CAN data frame are affected by bit stuffing? ★★★

Ans. Bit stuffing affects starting from SOF (Start of Frame), Arbitration field including (11bit message ID and 1 bit Remote Transmission Request), Data Field and CRC sequence. All other fields like CRC delimiter acknowledgement, acknowledgement delimiter, end of frame etc. are unaffected. The unaffected fields are called fixed fields

27. What is disadvantage of bit stuffing? ★★★★

Ans. The problem with bit stuffing is that it makes the length of CAN frames unpredictable. Because of the bit stuffing, message length varies between 49 bits to 114 bits considering an 11 bit ID and 64 to 154 bits for 29bit ID

28. What do you mean by bit sample point? Where exactly does it exist? ★★★

Ans. Bit sample point is the time when CAN node senses the bus considering the bus propagation time. A CAN bit time is divided in four segments called Synchronization segment, Propagation segment, Phase segment 1 and Phase segment 2. The bit sample point lies between phase segment 1 and 2. This is again duty of CAN controller to adjust the sample point. There are two types of synchronizations – hard synchronization and resynchronization.

29. What are various error detection methods involved in a CAN network? ★★★

Ans. Bit monitoring (detected by transmitter) – When a node sends a dominant bit (0) over the bus it must sense the dominant bit only i.e. it should never happen that node sends dominant and senses recessive bit back. If it so then reported as error.
Bit stuffing (detected by receiver) – More than 5 bits of same polarity outside the bit-stuffed segment is also reported as error.
CRC (detected by receiver) – Cyclic redundancy Check
Form (detected by receiver)- If delimiter is not in correct position i.e. error in fixed fields.
Acknowledgement error (detected by transmitter) – When transmitter doesn't receive an acknowledgement of the message it transmitted.

30. What is the probability that a faulty message in CAN is not detected? ★★★

Ans. 1 in 1000 years

31. What happens when an error is reported by a particular node on the bus? ★★★

Ans. When any of the CAN node detects an error on the bus, it sends an error frame with 6 consecutive dominant bits. When all other nodes see 6 consecutive bits on the bus (which is normally not acceptable because of bit stuffing), they come to know that there is some problem with the message over the bus so they will also send their error flags. By the time error delimiter edge starts, all nodes come to know that the message over the bus need to be destroyed so none of the node accepts the message. After the message is deleted and the inter-frame space after the error frame, the node which transmitted that faulty message reattempts to transmit the message.

32. How much approximately the total error recovery time in worst case? ★★★

Ans. Error flag + Error delimiter + Intermission field = 12 + 8 + 3 = 23bits hence 23 us for a speed of 1Mbps.

33. What if one of the node continuously produces faulty messages or continuously reports an error on reception? What do you mean by fault confinement? ★★★

Ans. Actually this is the case when fault confinement comes into the picture. In this particular case that node has capability to remove itself from the network (called Bus-Off state). Faulty node is pinpointed by the majority vote of other nodes.

34. What do you mean by transmit error and receive error? ★★★

Ans. When a node broadcasts a faulty message an error is reported against that message by majority of the nodes then this error is said to be transmit error (Transmit Error Counter incremented). On the other hand when one of the node sends a healthy message. Majority of nodes treat the message healthy (i.e. don't report any error) but one of the node reports an error then it is called a receive error (Receive Error Counter Incremented).

35. What are different states of a node? ★★★

Ans. There are 3 states of a node error active, error passive and bus-off. A CAN node uses error counters (TEC and REC) to control the transition within these states. If the counter is less than 128, the node is said to be in error active state. If counter equals or exceeds 128 but less than 255 than it is in error passive state. If counter equals or exceeds 255 the node goes in bus-off state hence this node cannot take part in communication. An error active node always transmits an active error frame (dominant bits) when detecting an error while an error-passive node transmits passive error flags (recessive bits) when detects an error.

36. What is function of CAN transceiver? What are different bus levels? ★★★

Ans. It is a chip that converts TTL voltage into CAN differential voltages and vice versa. CAN differential voltages include CAN_High (3.5V), CAN_Low(1.5V). Default 2.5V. Hence a recessive bit means the differential voltage between CAN_high and CAN_Low is 0V i.e. both are at 2.5V and dominant bit intends the voltage difference of 2V between the two.

37. Why CAN transceiver is a separate chip? Why it is not integrated as part of microprocessor unit? ★★★

Ans. Because it has to do the power conversion operations and hence its power requirement is more than the usual microprocessor unit.

38. What is meant by inhibit time? ★★★

Ans. Sometimes there may be cases when one node continuously sends high priority messages and no other node gets chance to win the bus. So this leads bus jamming by that particular node. In this case inhibit time is introduces i.e. after the message is transmitted for next few amount of time same message can't be sent again so meanwhile other nodes have chance to win the bus. But surprisingly inhibit time is not developed in CAN protocol. It must be developed at higher level as CANOpen or Device net.

39. What development tools you need to work on CAN? ★★★

Ans. We basically need cross compilers, emulators, CAN analyzers etc.

40. Why higher layer protocols are required over a CAN implementation? ★★★

Ans. If we need to add extra features to the existing CAN protocol like increasing data length to more than 8 bytes, to establish a master slave configuration, Network management features like node monitoring, node synchronization etc. then we go for higher layer protocols. Most popular higher layer protocols are CANOpen (11bit ID), Device net (11bit ID) and SAE J1939 (29 bit ID).

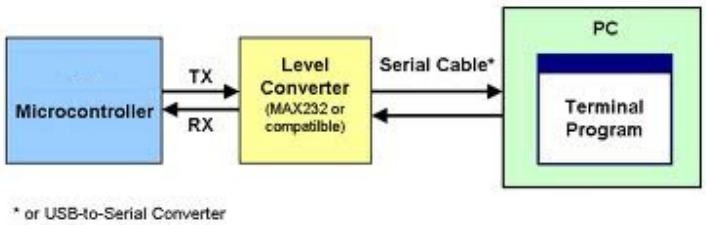
41. How to you select a protocol for your application? ★★★

Ans. Selecting a protocol for particular application is very important process. You first need to identify the speed requirement, memory size, no. of communication units and the maximum communication length, cost feasibility of the system and ease of implementation.

42. What's maximum length that can be achieved for RS485 and CAN protocols? ★★★

Ans. RS485 was implemented to provide long distance data communication ranging up to 1200m. CAN protocol at its maximum speed (i.e.) can work up to 40m while at 125kbps it can go up to 500m.

43. Block diagram of RS232 connection between controller and PC and controller to controller. ★★★



44. Name some serial communication protocols. Which of them are synchronous and which are asynchronous. ★★★

Ans. CAN, SPI, I2C, RS232, RS485, USB etc.
CAN, RS232/485 and USB are asynchronous as they don't include clock for communication while SPI and I2C are synchronous protocols.

45. What is RS232 Protocol and application?★★★

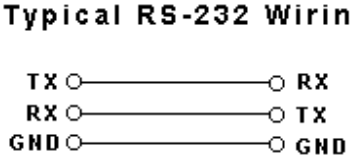
Ans. RS-232
In telecommunications, **RS-232** is the traditional name for a series of standards for serial binary single-ended data and control signals connecting between a *DTE* (Data Terminal Equipment) and a *DCE* (Data Circuit-terminating Equipment). It is commonly used incomputer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pinout of connectors.

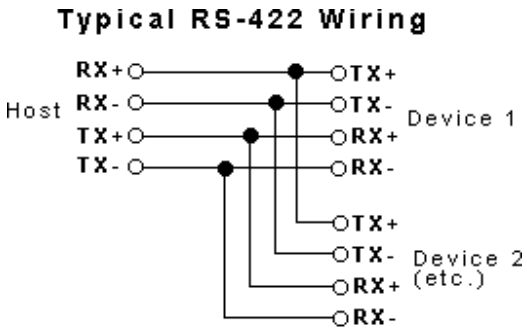
Definitely the most popular interface, also being one of the first. However, things may soon change for obvious reasons. Any PC that is purchased will have one (and sometimes more) RS-232 port. Sometimes, they are simply referred to as SERIAL PORTS, however this may cause confusion since there are other Serial interfaces available. RS-232 is widely used because it is so readily available. You don't usually need to purchase an RS-232 port since it is available on any PC. However, it does have some disadvantages. Here are a few:
1. Limited Distance - Cable lengths are limited to 50 ft or less. Many will claim to go further, but this is not recommended, and is not part of the RS-232 specification.
2. Susceptible to Noise - RS-232 is single-ended, which means that the transmit and receive lines are referenced to a common ground
3. Not Multi-drop - You can only connect one RS-232 device per port. There are some devices designed to echo a command to a second unit of the same family of products, but this is very rare. This means that if you have 3 meters to connect to a PC, you will need 3 ports, or at least, an RS-232 multiplexor.

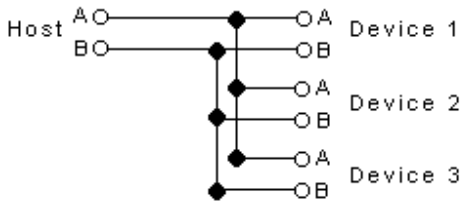
46. What's difference between RS232, RS422 and RS485? ★★★

Specifications	RS-232	RS-422	RS-485
Mode of Operation	Single-Ended	Differential	Differential
Total Number of Drivers and Receivers on One Line (One driver active at a time for RS-485 networks)	1 Driver 1 Receiver	1 Driver 10 Receiver i.e. one talker many listeners	32 Drivers 32 Receivers.. i.e. many talker many listeners
Maximum Cable Length	50 ft (2500 pF)	4000 ft	4000 ft
Maximum Data Rate (40 ft - 4000 ft for RS-422/RS-485)	160 kbits/s (can be up to 1MBit/s	10 MBit/s	10 MBit/s

Here are some typical wiring diagrams for each interface type:







47. What can be done to avoid EMI interference? ★★★

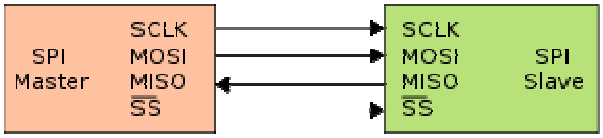
Ans. Differential signaling is the option like in RS485/ RS422 and CAN etc. Use of differential signaling not only improves signal strength but also improves noise immunity of the signal hence protects it from the surrounding EMI interferences.

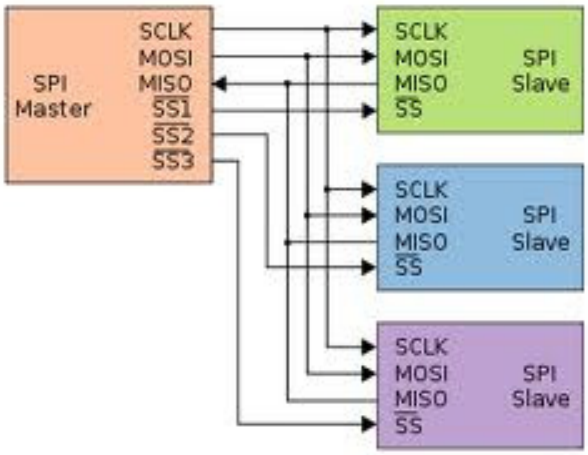
48. Have you worked on SPI protocol? Application of SPI protocol? ★★★

Ans. Generally used for external flash interfacing and ISP (in system programming of microcontrollers)

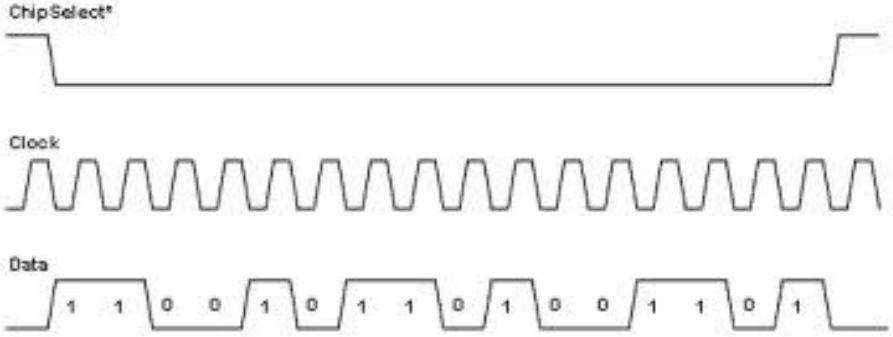
The **Serial Peripheral Interface Bus** or **SPI** (pronounced as either *ess-pee-eye* or *spy*) bus is a synchronous serial data link standard, named by Motorola, that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a *four-wire* serial bus, contrasting with three-,two-, and one-wire serial buses. SPI is often referred to as SSI (Synchronous Serial Interface).

Speed of SPI is from 1 to 100MHz





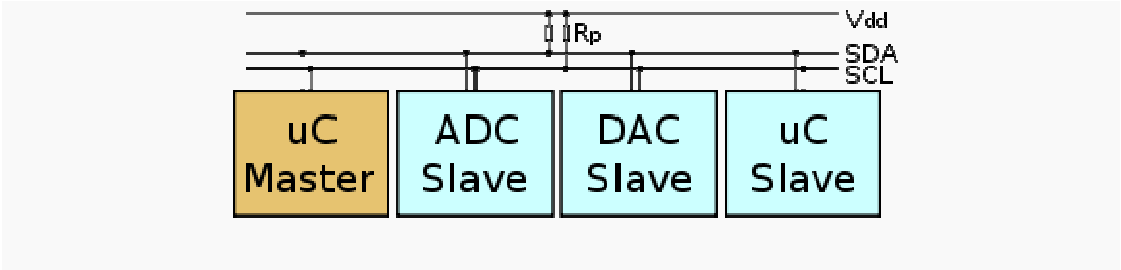
Here follows the timing diagram for SPI protocol…



49. What is an I2C protocol? State its application and features. ★★★★

Ans. It's a synchronous serial communication protocol with basically 2 wires – SCL(Serial Clock) and SDA (Serial Data)

Design



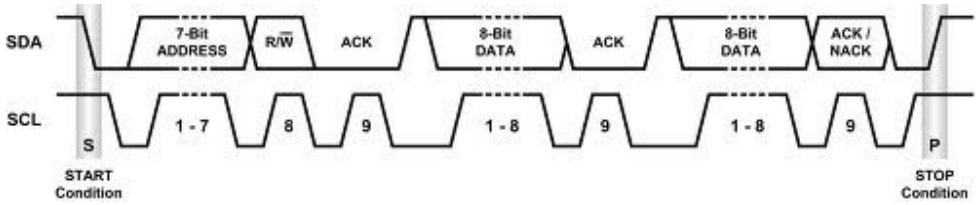
A sample schematic with one master (a microcontroller), three slave nodes (an ADC, a DAC, and a microcontroller), and pull-up resistors R_p

I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other voltages are permitted.

The I²C reference design has a 7-bit or a 10-bit (depending on the device used) address space. Common I²C bus speeds are the 100 kbit/s *standard mode* and the 10 kbit/s *low-speed mode*, but arbitrarily low clock frequencies are also allowed. Recent revisions of I²C can host more nodes and run at faster speeds (400 kbit/s *Fast mode*, 1 Mbit/s *Fast mode plus* or Fm+, and 3.4 Mbit/s *High Speed mode*). These speeds are more widely used on embedded systems than on PCs. There are also other features, such as 16-bit addressing.

Note that the bit rates quoted are for the transactions between master and slave without clock stretching or other hardware overhead. Protocol overheads include a slave address and perhaps a register address within the slave device as well as per-byte ACK/NACK bits. So the actual transfer rate of user data is lower than those peak bit rates alone would imply. For example, if each interaction with a slave inefficiently allows only 1 byte of data to be transferred, the data rate will be less than half the peak bit rate.

The maximum number of nodes is limited by the address space, and also by the total bus capacitance of 400 pF, which restricts practical communication distances to a few meters. Here is the timing diagram for I2C protocol…



Operating System and RTOS concepts

1. What do you mean by an Operating System? ★★★★

Ans. Operating System is a system-software. It is a program that acts as an intermediary between a user of a computer and the computer hardware. Goals of an Operating system involve Executing user programs efficiently and make the computer system convenient to use. Examples of Operating systems are MS DOS, MS Windows, UNIX, Linux, MacOS, Android etc.

Operating system can be defined as

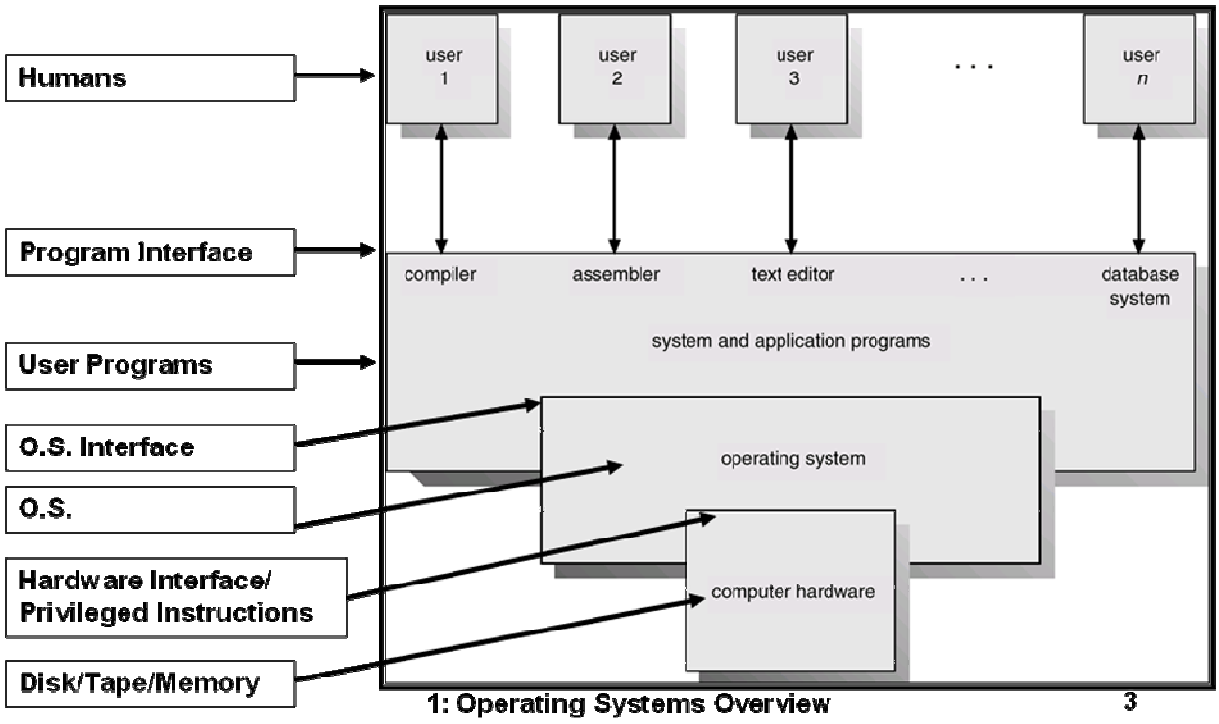
A mechanism for scheduling jobs or processes. Scheduling can be as simple as running the next process, or it can use relatively complex rules to pick a running process.

OR

A method for simultaneous CPU execution and IO handling. Processing is going on even as IO is occurring in preparation for future CPU work.

2. What is structure of an Operating System? Where does it lie in a computer system? ★★★★

Ans.



3. What are different tasks that an OS handles? ★★★

Ans. An OS usually handles following tasks

a. Resource allocation: OS manages all resources like memory; peripherals etc. and decide between conflicting requests for efficient and fair resource use.
b. OS is a control program that schedules the execution of programs or program segments to prevent errors

4. What do you mean by Kernel? ★★★

Ans. Kernel is the main and essential component of an Operating System. A kernel provides the lowest level of abstraction for the resources (especially processors and I/O devices) that application software must control to perform its function. Kernel is a central component of any OS that acts as an interface between user applications and the hardware.
Kernel is the heart of OS. Main tasks of a kernel include

- Process management – Consists of creation, activation, running, blocking, resumption, deactivation and deletion of processes.
- Device management – Managing physical and virtual devices like pipes and sockets. Three standard approaches for three types are device drivers are Programmed I/O polling, Interrupt driven through ISR, DMA. A device manager includes functions like device detection & addition, device allocation and registration, device sharing, device buffer management, device access management.
- Memory management – Consists of fixed or dynamic Memory allocation to process/task and deallocation.
- Interrupt handling- Searching for Interrupt addresses in vector tables and context switching.
- I/O communication-
- File system organization- Include Open, Write, Read, Seek and Close functions
- Inter Process Communication

5. What do you mean by a Wakeup call? ★★★

Ans. When you turn on the PC, the first program that runs is a set of instructions kept in computers ROM. It checks to make sure everything is functioning properly. It checks the CPU memory and Basic Input Output Systems (BIOS) for errors. Once successful, the software will begin to activate computer's disk drives. It then finds the first piece of operating system called 'bootstrap loader'.
This complete process is called a wakeup call.

6. What is Bootstrap program? ★★★

Ans. Bootstrap is a program that is loaded at the power-up or reboot. This is typically stored in ROM or EPROM, generally known as firmware. This program initializes all the aspects of the system. This also loads OS kernel and starts execution.
A booting process involves…

- First the boot strap loader loads the operating system into memory and allows it to begin operation.

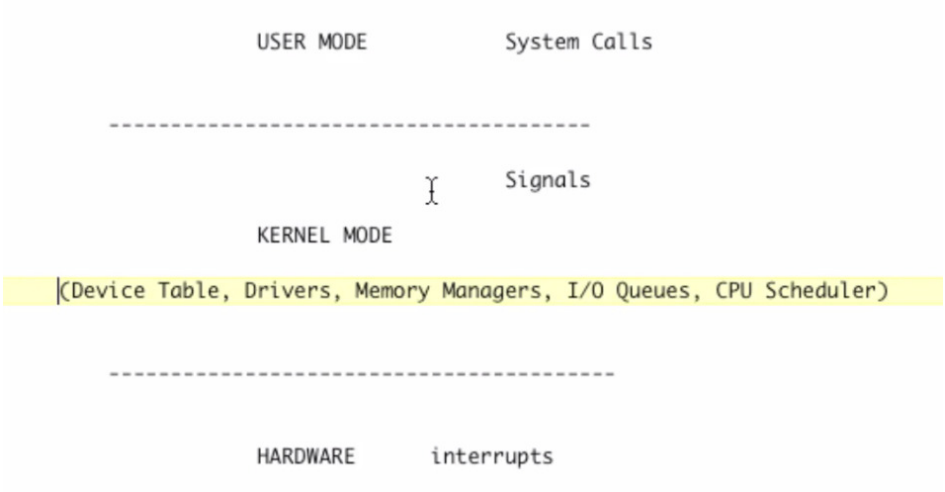
- The bootstrap loader also sets up the small driver programs that interface with and control various hardware.
- It also sets up the divisions of memory, user information and applications
- It establishes the data structures needed to communicate within and between the subsystems and applications of the computer.
- Then it turns control of the computer over to operating system.

7. Why is it needed to load OS from flash to memory? Why can't it be directly executed from flash? ★★★★

Ans. It is because the flash memory is usually the serial NOR memory for storing large programs. Hence due to serial interface the program can't be addressed instruction by instruction hence first the essential piece of code i.e. the OS is loaded into the main memory by boot loader then it can be addressed directly and executed.

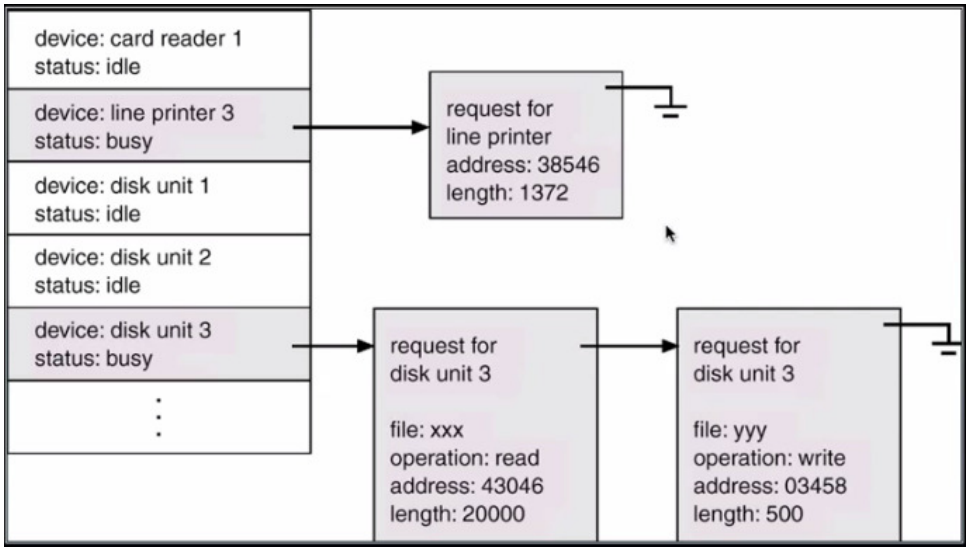
8. Differentiate between system calls, signals and interrupts. ★★★★

Ans. The complete computer system can be thought of three essential layers called User mode, Kernel mode and the Hardware. User mode is the uppermost layer where user application programs run. Kernel mode is the middle layer and is an interface between user mode and the hardware. System calls are the request from User mode to the kernel for example "open a CDROM drive". After receiving the System call Kernel invokes particular driver for the hardware. Driver is responsible for controlling the particular hardware. After completion of the said task, hardware generates and interrupt for the Kernel mode saying the task is finished. Signals are then sent to the user mode from the kernel mode about completion of task.



9. What is device status table? ★★★★

Ans. It is part of the kernel mode and contains entry for each I/O device including its type, address and state.



10. What is DMA (Direct Memory Access)? What's its advantage? ★★★

Ans. Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU initiates the transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller when the operation is done. This feature is useful any time the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

Direct memory access is mainly used to overcome the disadvantages of interrupt and program controlled I/O.

DMA modules usually take the control over from the processor and perform the memory operations and this is mainly because to counteract the mismatch in the processing speeds of I/O units and the processor. This is comparatively faster. It is an important part of any embedded systems, and the reason for their use is that they can be used for burst data transfers instead of single byte approaches. It has to wait for the systems resources such as the system bus in case it is already in control of it.

11. What's advantage of using a Cache memory? ★★★

Ans. A **CPU cache** is a cache used by the central processing unit of a computer to reduce the average time to access memory. The cache is a smaller, faster memory which stores copies of the data from the most frequently

used main memory locations. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory.

When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory.

Most modern desktop and server CPUs have at least three independent caches: an **instruction cache** to speed up executable instruction fetch, a **data cache** to speed up data fetch and store, and a translation look aside buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data. The data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc). Usual cache sizes vary from kBs to few Megabytes (around 16MB). Cache is usually handled by computer hardware itself.

12. What is virtual memory? State its advantage. ★★★

Ans. For example, if you load the operating system, an e-mail program, a Web browser and word processor into RAM simultaneously, 32 megabytes is not enough to hold it all. If there were no such thing as virtual memory, then once you filled up the available RAM your computer would have to say, "Sorry, you can not load any more applications. Please close another application to load a new one." With virtual memory, what the computer can do is look at RAM for areas that have not been used recently and copy them onto the hard disk. This frees up space in RAM to load the new application.

Because this **copying** happens automatically, you don't even know it is happening, and it makes your computer feel like is has unlimited RAM space even though it only has 32 megabytes installed. Because hard disk space is so much cheaper than RAM chips, it also has a nice economic benefit.

The read/write speed of a hard drive is much slower than RAM, and the technology of a hard drive is not geared toward accessing small pieces of data at a time. If your system has to rely too heavily on virtual memory, you will notice a significant performance drop. The key is to have enough RAM to handle everything you tend to work on simultaneously -- then, the only time you "feel" the slowness of virtual memory is is when there's a slight pause when you're changing tasks. When that's the case, virtual memory is perfect.

Swap file is an area of hard disk that is reserved for virtual memory. Swap files can be temporary or permanent.

13. What is thrashing? ★★★

Ans. It is a phenomenon in virtual memory schemes when the processor spends most of its time in swapping pages, rather than executing instructions. This is due to an inordinate number of page faults.

14. Explain the concept of reentrancy. ★★★

Ans. It is a useful, memory-saving technique for multiprogrammed timesharing systems. A Reentrant Procedure is one in which multiple users can share a single copy of a program during the same period. Reentrancy has 2 key aspects: The program code cannot modify itself, and the local data for each user process must be stored separately. Thus, the permanent part is the code, and the temporary part is the pointer back to the calling program and local variables used by that program. Each execution instance is called activation. It executes the code in the permanent part, but has its own copy of local variables/parameters. The temporary part associated with each activation is the activation record. Generally, the activation record is kept on the stack.
Note: A reentrant procedure can be interrupted and called by an interrupting program, and still execute correctly on returning to the procedure.

15. Explain Belady's Anomaly? ★★★

Ans. Also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns

16. Difference between General Purpose Operating System and Real Time Operating System? ★★★

Ans. General purpose OSes don't have timing limitations for performing the tasks while real time OSes have. In real time environment if the task is not completed within specified time for it, it is considered to be failure. Although real time operating systems are further classified as soft real time and hard real time.

17. What do you mean by an Embedded OS? What are its various features? ★★★

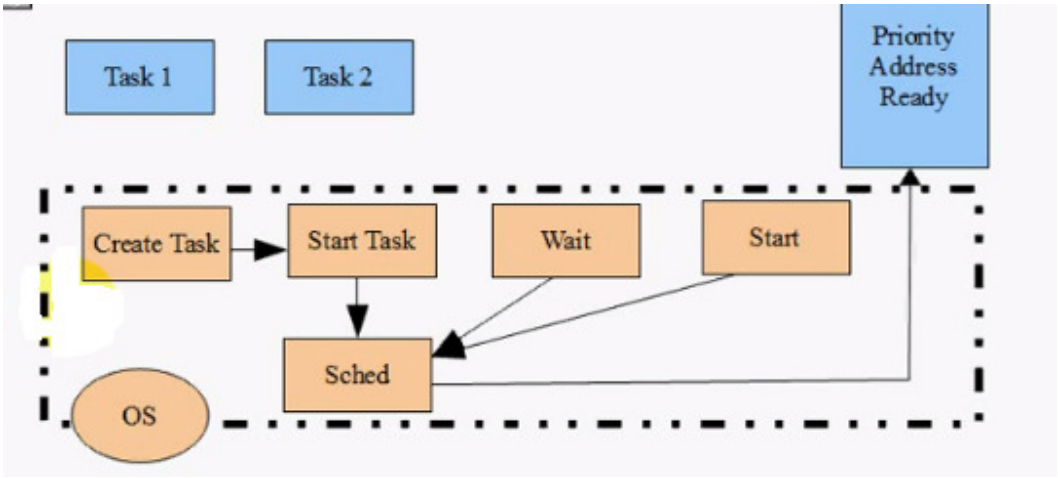
Ans. An Embedded OS is dedicated to particular embedded system, its foot print is small and supports real time functionality. Features of an Embedded OS include Modular, Scalable, Configurable, Small footprint, CPU support, Device drivers etc.

18. Difference between soft real time and hard real time system. ★★★

Ans. In soft real time environment, timing violations are acceptable but hard real time systems must complete task within specified limit otherwise treated as failure of system or may cause damage to system or even the user. Examples of

hard real time system using tasks like applying brake in vehicle launching a missile etc. which cannot afford delay in execution while soft real time tasks include displaying some information etc.

19. Say 2 tasks T1 and T2 are running at a time how they will be managed in RTOS given that T1 has higher priority over T2?
Ans.



Information about Task1 and Task2 is stored in task table while creating them. The information includes address, priority, ID and the ready flag for each task. 'Create' function then invokes 'Start Task'. 'Start Task' function then invokes 'Scheduler' which checks in the table for the highest priority task and starts it (in this case it is Task1).
Now suppose at some point of time Task1 needs some resource which is not available then it invokes a service called 'Wait' which sets its ready flag to 0 and invokes the scheduler again. Scheduler again checks for the higher priority task in the table which is Task1 then it checks whether it is ready or not. As Task1 is not ready it will check the next priority task which is Task2 which is ready hence it is started.
Now suppose the resource needed by Task1 is available in this case it invokes another start routine which sets its ready flag in the table and then invokes the scheduler. Scheduler again checks the table in same way i.e. first the highest priority which is Task1 and whether it is ready or not so now YES it is ready hence starts the Task1 again. In this case Task2 is said to be preempted by Task1.

20. What features an RTOS must possess?
Ans.

- Should be able to group tasks having similar time constraints
- Should be able to assign priority
- User defined priority of interrupts
- Priority or deadline oriented scheduling
- It must be multithreaded and preemptible.
- Thread priority must exist as no deadline driven OS exist.
- It must support predictable thread synchronization mechanisms.

- A system of priority inheritance must exist.

21. What are the built-in hardware features that a processor must have to support an RTOS or RTOS implementation?

Ans. There are 3 basic features

- Powerful Interrupt Structure
  - Interrupts must have priority levels
  - Mechanism to enable/disable interrupts
  - Small interrupt latencies (vectored interrupts have low latencies)
  - Provision for dynamic change of priority levels
- Programmable Timers
- Good no. of register banks to avoid overhead of context saving.

22. What do you mean by EDF (Early Deadline Fast)?

Ans. This is similar to priority based scheduling but here priorities of tasks are defined by their deadline times. Task having minimum deadline time will have the higher priority.

23. Name various RTOSs available in the market.

Ans. Microsoft – Embedded XP/NT, Windows CE, Pocket PC 2002, **Windriver Systems** – VxWorks, pSOS. Other popular RTOSs are uCOS-2, RT-Linux, QNX, Symbian, Android, Palm OS etc.

24. What are various scheduling mechanisms?

Ans. There are various scheduling mechanisms like preemptive, non-preemptive,

In preemptive scheduling, a process that is being allotted processor is forcely removed and the processor is given to some other process.
In non-preemptive scheduling, there are no forcible removals.
FCFS (first come first serve), SJF are non-preemptive.
Round Robin is preemptive. If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list.
Round Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in time-sharing environments in which the system needs to guarantee reasonable response times for interactive users.

25. What are various RTOS Task Scheduling Models?

Ans.

Control Flow strategy-

- Complete control on I/Os
- Co-operative scheduler adopts this strategy
- Worst case latencies are well defined

Data Flow Strategy-

- Interrupt occurrences are predictable
- Task control not deterministic e.g. network packet arrival

- Preemptive scheduler adopts this strategy

Control Data Flow strategy-

- Task scheduler functions are designed with predefined time-out delays
- Worst case latency is deterministic, because the maximum delay is predefined
- Cyclic co-op scheduling, Pre-emptive scheduling, Fixed time scheduling, Dynamic real time scheduling use this strategy

26. What do you mean by preemption?

Ans. Suppose a low priority task is running and suddenly a high priority task starts, the scheduler pauses the low priority task and starts the task with higher priority hence the low priority task is said to be preempted by high priority task.

27. What is difference between preemptive scheduling and time slicing?

Ans. Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

28. Why time slicing or round robin scheduling can't be used in real time applications?

Ans. In time slicing or round robin mechanism each task is treated equally i.e. there is no priority of any task i.e. all the tasks are of same priority and hence there is no dead line for particular task. But in real time system tasks have timing constraints or the deadlines that's why time slicing scheduling can't be used.

29. What are short, long and medium-term scheduling?

Ans.

Long term scheduler determines which programs are admitted to the system for processing. It controls the degree of multiprogramming. Once admitted, a job becomes a process.

Medium term scheduling is part of the swapping function. This relates to processes that are in a blocked or suspended state. They are swapped out of real-memory until they are ready to execute. The swapping-in decision is based on memory-management criteria.

Short term scheduler, also known as a dispatcher executes most frequently, and makes the finest-grained decision of which process should execute next. This scheduler is invoked whenever an event occurs. It may lead to interruption of one process by preemption.

30. What is difference between turn-around time and response time?

Ans. Turn-around time is the interval between the submission of a job and its completion. Response time is the time interval between submission of the request and the first response to the request.

31. What are typical elements of a process image? ★★★★

Ans.

32. What do you mean by IPC? ★★★

Ans. Inter-process communication (**IPC**) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.

There are several reasons for providing an environment that allows process cooperation:

- Information sharing
- Computational Speedup
- Modularity
- Convenience
- Privilege separation

IPC may also be referred to as *inter-thread communication* and *inter-application communication*.

The combination of IPC with the address space concept is the foundation for address space independence/isolation

33. What is difference between process and thread? ★★★★

Ans. Thread is said to be a light weight process and doesn't have a Thread control block like process has process control block. Process control block or PCB is area where Process information like process ID, process variables, parent process, child process information etc are stored.

34. Define Process, Task and Thread. ★★★★

Ans.

Process: It is sequentially running program with its state.

Task: An application consisting of task controlled by scheduling mechanism of an OS

Thread: Light weighted sub-process in application program, controlled by process control entity

35. What is semaphore? ★★★

Ans. Semaphores let processes query or alter status information. They are often used to monitor and control the availability of system resources such as shared memory segments.
Semaphores are a programming construct designed by E. W. Dijkstra in the late 1960s. Dijkstra's model was the operation of railroads: consider a stretch of railroad in which there is a single track over which only one train at a time is allowed. Guarding this track is a semaphore. A train must wait before entering the single track until the semaphore is in a state that permits travel. When the train enters the track, the semaphore changes state to prevent other trains from entering the track. A train that is leaving this section of track must again change the state of the semaphore to allow another train to enter. In the computer version, a semaphore appears to be a simple integer. A process (or a thread) waits for permission to proceed by waiting for the integer to become 0. The signal if it proceeds signals that this by performing incrementing the integer by 1. When it is finished, the process changes the semaphore's value by subtracting one from it.

36. What is binary semaphore? What is its use? ★★★

Ans. A binary semaphore takes only two values either 1 or 0. It is used to implement mutual exclusion and synchronize concurrent processes.
A binary semaphore is a synchronization object that can have only two states:
Not taken.
Taken.
Two operations are defined:
Take Taking a binary semaphore brings it in the "taken" state, trying to take a semaphore that isalready taken enters the invoking thread into a waiting queue.
Release Releasing a binary semaphore brings it in the "not taken" state if there are not queued threads. If there are queued threads then a thread is removed from the queue and resumed, the binary semaphore remains in the "taken" state. Releasing a semaphore that is already in its "not taken" state has no effect.

37. Difference between signal and semaphore? ★★★★

Ans. Signals are relatively low level inter process communication methods which is used in Unix. The signal arranges for user defined function to be called, when a particular event occurs.

38. Difference between Binary semaphore and a Mutex? ★★★★

Ans. Mutex can be released only by thread that had acquired it, while you can signal semaphore from any other thread (or process), so semaphores are more suitable for some synchronization problems like producer-consumer.
On Windows, binary semaphores are more like event objects than mutexes.
Mutex can be released only by the thread that had acquired it where as in semaphore any thread can signal the semaphore to release the critical section.
There are 3 major differences between mutex and binary semaphore.
a. In case of mutex semaphore the task that had taken the semaphore can only give it, however in the case of binary semaphore any task can give the semaphore.

b. Calling SemFlush() function in mutex is illegal.
c. Mutex semaphore can't be given from an ISR.
d. Semaphores is a synchronization tool to overcome the critical section problem.
- A semaphore S is basically an integer variable that apart from initialisation is accesses only through atomic operations such as wait() and signal().
- Semaphore object basically acts as a counter to monitor the number of threads accessing a resource.
- Mutex is also a tool that is used to provide deadlock free mutual exclusion. It protects access to every critical data item.if the data is locked and is in use, it either waits for the thread to finish or awakened to release the lock from its inactive state.

39. What is deadlock? Why does it happen? ★★★

Ans. A **deadlock** is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. In an operating system, a deadlock is a situation which occurs when a process enters a waiting state because a resource requested by it is being held by another waiting process, which in turn is waiting for another resource. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.
A simple computer-based example is as follows. Suppose a computer has three CD drives and three processes. Each of the three processes holds one of the drives. If each process now requests another drive, the three processes will be in a deadlock. Each process will be waiting for the "CD drive released" event, which can be only caused by one of the other waiting processes. Thus, it results in a circular chain.
A task will not get scheduled if any resource it may lock actually has been locked by another task, and therefore the priority ceiling protocol prevents deadlocks.

40. What are Coffman's conditions that lead to a deadlock? ★★★★

- **Mutual Exclusion**: Only one process may use a critical resource at a time.
- **Hold & Wait**: A process may be allocated some resources while waiting for others.
- **No Pre-emption**: No resource can be forcibly removed from a process holding it.
- **Circular Wait**: A closed chain of processes exist such that each process holds at least one resource needed by another process in the chain.

41. What is meant by atomic instruction? ★★★

Ans. In computing, this means that the operation, well, *happens*. There isn't any intermediate state that's visible before it completes. So if your CPU gets interrupted to service hardware (IRQ), or if another CPU is reading the same

memory, it doesn't affect the result, and these other operations will observe it as either completed or not started.

Atomicity is a key concept when you have any form of parallel processing (including different applications cooperating or sharing data) that includes shared resources.
The problem is well illustrated with an example. Let's say you have two programs that want to create a file but only if the file doesn't already exist. Any of the two programs can create the file at any point in time.
If you do (I'll use C since it's what's in your example):

```
...
f = fopen ("SYNCFILE","r");
if (f == NULL) {
  f = fopen ("SYNCFILE","w");
}
...
```

You can't be sure that the other program hasn't created the file between your open for read and your open for write.
There's no way you can do this on your own, you need help from the operating system, that usually provides synchronization primitives for this purpose, or another mechanism that is guaranteed to be atomic (for example a relational database where the lock operation is atomic, or a lower level mechanism like processors "test and set" instructions).

42. What is shared data problem? What can be done to avoid it? ★★★

Ans. Conflict arising on a common variable due to usage by multiple tasks / processes on it.e.g. Interrupt changing the subsequent bits while processing a 32 bit data on an 8-bit CPU. Following are the ways to avoid shared data problem.

a. Use of volatile variables.
b. Use of reentrant functions.
c. Putting shared variables in circular queues
d. Disabling interrupt on execution of critical section.
e. Using semaphores.

43. Differentiate between multiprogramming, multiprocessing, and multithreading. ★★★★

Ans. "Multiprogramming" means the ability to run several programs at once. We take this for granted today, but early systems could run one program at a time. Almost all computer systems today use multiprogramming, so's there's really no need for the term anymore.

"Multiprocessing" is the ability to have several cpu's in one system to distribute the load. There are still plenty of uni-processor systems out there, so this term still sees frequent use.

"Multithreading" is the ability to have multiple threads in a process. For example, one thread may be reading the next block of data while another thread processes the current block of data. When I heard "multitasking" used in the non-unix world, it was referring to something close to today's multithreading.

44. What do you mean by multi user operating system? Give some examples. ★★★

Ans. Multi user operating system is one that can be used by multiple users at a time. The OS is installed at a centralized server and logged in through remote systems. Mainframe, Unix and Linux are some examples of such OSes.

45. Difference between semaphore and mailbox? ★★★

Ans. Both are RTOS elements and defer in way that semaphore is used to resolve shared data or shared resource problem. Mailboxes are used to send/receive messages from one process/thread to another.

46. Difference between Process and Task? ★★★★

Ans. Both are program elements and have same features. Process term is generally used in Unix/Linux OSes and Task is used in VxWorks RTOS**.**

47. What is Priority Inversion? ★★★★

Ans. In computer science, **priority inversion** is a problematic scenario in scheduling when a higher priority task is indirectly preempted by a lower priority task effectively "inverting" the relative priorities of the two tasks.

This violates the priority model that high priority tasks can only be prevented from running by higher priority tasks and briefly by low priority tasks which will quickly complete their use of a resource shared by the high and low priority tasks.

Consider there is a task L, with low priority. This task requires resource R. Now, there is another task H, with high priority. This task also requires resource R. Consider H starts *after* L has acquired resource R. Now H has to wait until L relinquishes resource R.

Everything works as expected up to this point, but problems arise when a new task M (which does not use R) starts with medium priority during this time. Since R is still in use (by L), H can not run. Since M is the highest priority unblocked task, it will be scheduled before L. Since L has been preempted by M, L cannot relinquish R. So M will run till it is finished, then L will run - at least up to a point where it can relinquish R - and then H will run. Thus, in the scenario above, a task with medium priority ran before a task with high priority, effectively giving us a priority inversion.

48. What is Priority Inheritance? ★★★★★

Ans. This is one of the methods to avoid priority inversion. Under the policy of priority inheritance, whenever a high priority task has to wait for some resource shared with an executing low priority task, the low priority task is temporarily

assigned the priority of the highest waiting priority task for the duration of its own use of the shared resource, thus keeping medium priority tasks from pre-empting the (originally) low priority task, and thereby affecting the waiting high priority task as well. Once the resource is released, the low priority task continues at its original priority level.

49. What do you mean by priority ceiling? ★★★★

Ans. In real-time computing, the **priority ceiling protocol** is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections. In this protocol each resource is assigned a priority ceiling, which is a priority equal to the highest priority of any task which may lock the resource.

For example, with priority ceilings, the shared mutex process (that runs the operating system code) has a characteristic (high) priority of its own, which is assigned to the task locking the mutex. This works well, provided the other high priority task(s) that tries to access the mutex does not have a priority higher than the ceiling priority.

In the **Immediate Ceiling Priority Protocol** (ICPP) when a task locks the resource its priority is temporarily raised to the priority ceiling of the resource, thus no task that may lock the resource is able to get scheduled. This allows a low priority task to defer execution of higher-priority tasks.

The **Original Ceiling Priority Protocol** (OCPP) has the same worst-case performance, but is subtly different in the implementation which can provide finer grained priority inheritance control mechanism than ICPP. Under this approach, a task can lock a resource only if its dynamic priority is higher than the current system priority. (The dynamic priority of a task is the maximum of its own static priority and any it inherits due to it blocking higher-priority processes. The system current priority is the ceiling of the resource having the highest ceiling among those locked by *other* tasks currently.) Otherwise the task is blocked, and its priority is inherited by the task currently holding the resource that gives the current system priority. A task will not get scheduled if any resource it may lock actually has been locked by another task, and therefore the priority ceiling protocol prevents deadlocks.

50. Why does pre-emptive multi-threading used to solve the central controller problem? ★★★

Ans. Multi-threading provide lot of functionality to the system to allow more than one task can be run at a time. It allows a process to execute faster with less difficulty. But, if there any problem comes in any program or the process than the entire system comes to a halt and slows down the whole system. To control the behavior of this the preemptive multi-threading is used. The control in this case is being shifted from one process to another at any time according to the requirement provided. It allows the program to give the control to another

program that is having the higher priority. It includes of many problems like giving of a control by a process half way through in execution and the preemption of the process takes place then the data will be entered as corrupted in the memory location, multi-threading keeps the synchronization that is to be performed between different components of the system and the program and try to avoid the problem mentioned above.

51. What are the basic rules followed by Mutexes? ★★★

Ans. Mutexes follow these rules…

• Mutex is also called as Mutual Exclusion is a mechanism that is used to show the preemptive environment and allow providing security methods like preventing an unauthorized access to the resources that are getting used in the system. There are several rules that has to be followed to ensure the security policies:
• Mutex are directly managed by the system kernel that provides a secure environment to allow only the applications that passes the security rules and regulations. The mutex consists of objects that are allowed to be called by the kernel.
• Mutex can have only one process at a time in its area that is owned by the process using it. This allows less conflict between the different applications or processes that wait for their turn to execute it in the kernel area.
• Mutex can be allocated to another mutex that is running some task at a particular time and allow the kernel to have synchronization in between them.
• If Mutex is allocated to some other process then the area will consist of the process till the area is having the process in it.

52. What is the purpose of using critical sections? ★★★

Ans. Critical section allows the process to run in an area that is defined by that process only. It is a sequence of instructions that can be corrupted if any other process tries to interrupt it. This process allow the operating system to give the synchronization objects that are used to monitor the processes that are up and running so that no other process will get executed till the critical region consists of a process that is already running. The example includes removal of the data from a queue running in a critical section and if not protected then it can be interrupted and the data have chances of getting corrupted. The processes exit from the critical section as soon as they finish the execution so that the chances can be given to other processes that are waiting for their chance to come.

Embedded Linux

1. Is Linux a Kernel or an OS? ★★★★★

Ans. Well, there is a difference between kernel and OS. Kernel is the heart of OS which manages the core features of an OS while if some useful applications and utilities are added over the kernel, then the complete package becomes an OS. So, it can easily be said that an operating system consists of a kernel space and a user space.

So, we can say that Linux is a kernel as it does not include applications like file-system utilities, windowing systems and graphical desktops, system administrator commands, text editors, compilers etc. So, various companies add these kinds of applications over Linux kernel and provide their operating system like ubuntu, suse, centOS, redHat etc.
Linux follows the monolithic modular approach.

2. Why Command Line Interface is more popular in Linux? ★★★

Ans. The command line interface is the standard interface for every version of Linux whether it is RedHad, Ubuntu, Fedora, Caldera, SuSE etc. but the graphical user interface might vary for different versions that's the reason command line interface is more popular and used widely. Also system administration and diagnostic activities become very easy in command line interface.
Development or kernel enhancements are also done through CLI hence it's more popular.

3. Why to use Linux? ★★★

Ans. There are reasons to prefer Linux over other Embedded OSes

- Linux distribution has software worth thousands of dollars, for virtually no cost
- Linux OS is reliable, stable and very powerful
- Linux comes with complete development environment, including compilers, toolkit and scripting languages by different distributers
- Linux comes with networking facilities allowing you to share your hardware
- Linux utilizes your memory, CPU and other hardware to the fullest
- A wide variety of commercial software is also available
- Linux is very easily upgradable
- Supports multiple processors as standard
- True multitasking, so many apps, all at once
- The GUIs are more powerful than Mac!
- Triggers time to market the product
- Reduces overall system cost as time to market is decreased.

4. What are design challenges for using an Embedded OS? ★★★

Ans. Although Linux is a powerful OS but still there are following challenges when using it for an Embedded Application…

- Memory restrictions
- Booting from flash is slower
- Diskless operations
- Requirement for deterministic operations

5. What are the Criteria to evaluate and choose CPU/SoC for Linux? ★★★

Ans.

- End of Life
- Performance
- Application support
- Ecosystem
- Features
- Cost
- Resource availability

  Usual processors supported by Linux are x86, ARM, Hitachi SH, PowerPC, MIPS

  Although other processors can also be used but some more efforts will be required while Linux on them.

6. Give some examples of Embedded Systems where Linux is used or can be used. ★★★★

Ans. Examples of such sophisticated embedded systems include high end measurement systems with no.of communication interfaces like Ethernet, USB etc. Other examples include Routers, Gateways, Switches, PDAs and there can be many such applications. We can divide such embedded systems in following categories…

- Consumer Electronics
  - Smart phones based on android
  - Set-top boxes
  - Smart TV's
- Industrial Electronics
  - Automated meter reading
  - Asset and inventory tracking
  - Assembly line
  - Image processing for surveillance and security
- Automotive
  - Infotainment

7. What is minimum hardware requirement for Linux in an Embedded System? ★★★

Ans. For Linux to run in an Embedded System, it needs

- CPU with MMU

- Min 4 MB of RAM, 128 MB for most of the functionality
- Min 4MB of FLASH, 256 MB for most of the functionality
- Clock
- Timers
- Serial Port

8. What are different directories involved in Linux OS? ★★★

Ans. /bin, /boot, /dev, /etc, /home, /lib, /lost+found, /mnt, /opt, /proc, /root, /sbin, /usr, /var etc. These all directories lie under / or root directory.

/ -root

- Every single file and directory starts from the root directory
- Only root user has write privileges
- Please note that /root is root user's home directory which is not /.

/bin – User binaries

- Contains binary executables
- Common Linux commands you need to use in single-user modes are located under this directory
- Commands used by all users of the system are located here e.g. ls, ps, ping, grep, cp etc.

/sbin – System binaries

- Just like /bin, /sbin also contains binary executables but the Linux commands located under this directory are used typically by system administrator for system maintenance
- Examples of such commands are iptables, reboot, fdisk, ifconfig, swapon

/etc – Configuration files

- Contains configuration files required by all programs
- This also contains startup and shutdown shell scripts used to start/stop individual programs.
- E.g. /etc/resolv.conf, /etc/logrotate.conf

/dev – Device files

- Contains device files
- These include terminal devices, usb, or any device that is attached to the system.
- E.g. /dev/tty1, /dev/usbmon0 etc

/proc – process information

- Contains information about system processes
- This is a pseudo file system contains information about running process. For example /proc/{pid} directory contains information about the process with that particular ID.
- This is a virtual file system with text information about system resources e.g. /proc/uptime

/var – variable files

- Content of the file that are expected to grow can be found under this directory

- This includes – system log files (/var/log), packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files need across reboots (/var/tmp)

/tmp – temporary files
/usr – user programs
/home – home directories of users
/boot – boot loader files
/lib – System libraries
/opt – Optional add-ons apps
/mnt - mount directory
/media – removable devices
/srv – service data

9. What is difference between Monolithic Kernels and MicroKernels? ★★★
Ans. Kernels may be classified mainly in two categories
Monolithic
Micro Kernel

Monolithic Kernels
Earlier in this type of kernel architecture, all the basic system services like process and memory management, interrupt handling etc were packaged into a single module in kernel space. This type of architecture led to some serious drawbacks like 1) Size of kernel, which was huge. 2) Poor maintainability, which means bug fixing or addition of new features resulted in recompilation of the whole kernel which could consume hours.

In a modern day approach to monolithic architecture, the kernel consists of different modules which can be dynamically loaded and un-loaded. This modular approach allows easy extension of OS's capabilities. With this approach, maintainability of kernel became very easy as only the concerned module needs to be loaded and unloaded every time there is a change or bug fix in a particular module. So, there is no need to bring down and recompile the whole kernel for a smallest bit of change. Also, stripping of kernel for various platforms (say for embedded devices etc) became very easy as we can easily unload the module that we do not want.
Linux follows the monolithic modular approach

Microkernels

This architecture majorly caters to the problem of ever growing size of kernel code which we could not control in the monolithic approach. This architecture allows some basic services like device driver management, protocol stack, file system etc to run in user space. This reduces the kernel code size and also increases the security and stability of OS as we have the bare minimum code

running in kernel. So, if suppose a basic service like network service crashes due to buffer overflow, then only the networking service's memory would be corrupted, leaving the rest of the system still functional.
In this architecture, all the basic OS services which are made part of user space are made to run as servers which are used by other programs in the system through inter process communication (IPC). eg: we have servers for device drivers, network protocol stacks, file systems, graphics, etc. Microkernel servers are essentially daemon programs like any others, except that the kernel grants some of them privileges to interact with parts of physical memory that are otherwise off limits to most programs. This allows some servers, particularly device drivers, to interact directly with hardware. These servers are started at the system start-up.

So, what the bare minimum that microKernel architecture recommends in kernel space?

- Managing memory protection
- Process scheduling
- Inter Process communication (IPC)

Apart from the above, all other basic services can be made part of user space and can be run in the form of servers.
QNX follows the Microkernel approach.

10. What is UNIX? How it is different from Windows? ★★★

Ans. UNIX is an operating system which was first developed in 1960s and has been under constant development ever since. Unix is stable, multi user, multitasking system for servers, desktops and laptops.
Difference between windows and UNIX is that UNIX possesses the multiuser functionality i.e. at a time no. of users can login to the same OS. This functionality is not there in Windows.

11. What is kernel boot loading sequence? ★★★★

Ans.

a. SoC Reset controller loads the boot loader

b. Boot loader performs clock and memory initialization and copies the kernel and initial RAM disk (initrd) to memory from flash, decompresses the kernel (as kernel image is in compressed form in the flash), passes the kernel boot arguments (like what kind of file system you have, what kind of console like TFT, Serial port etc you have, parameters for that console etc) and jumps to kernel entry point.

c. Kernel entry point _stext (arch/arm/kernel/head.S for ARM) performs the startup functions, checks the machine ID, setup the page table, turns the MMU and jumps to start kernel.

d. Start kernel (arch/arm/init/main.c) performs the hardware initialization such as console, timer, interrupts and performs the Linux kernel subsystem initialization (mm, proc, scheduler, fs, network etc) and mounts the initrd (initial RAM disk)

e. If the initrd is successfully mounted the start_kernel forks the init process in the /bin

f. Init process forks other user processes and finally forks the shell

g. User can now interact with the kernel through shell.

12. What's an initial RAM disk? ★★★

Ans. The *initial RAM disk (initrd)* is an initial root file system that is mounted prior to when the real root file system is available. The initrd is bound to the kernel and loaded as part of the kernel boot procedure. The kernel then mounts this initrd as part of the two-stage boot process to load the modules to make the real file systems available and get at the real root file system.

The initrd contains a minimal set of directories and executables to achieve this, such as the insmod tool to install kernel modules into the kernel.

In the case of desktop or server Linux systems, the initrd is a transient file system. Its lifetime is short, only serving as a bridge to the real root file system. In embedded systems with no mutable storage, the initrd is the permanent root file system.

13. How does boot loader helps in development? ★★★★

Ans. A boot loader also gives us the opportunity to fetch the kernel from the host machine over the network using certain commands. Once we get the kernel image we can do some development or modification required, again we compile it and dump into the target host.

14. What do you mean by U-Boot and Red Boot?

15. What do you mean by Board support packages? ★★★★

Ans. BSPs are the programs dedicated for particular architecture like ARM, Power PC etc. A BSP usually contains…

- Arch: cpu architecture specific code
- Include: common include files and architecture includes
- Init: kernel initialization code
- Mm: common mmu handling code, arch/mm for CPU
- Drivers: device drivers
- Ipc: inter-process communications
- Modules: dynamic modules handling code
- Fs: file system
- Kernel: scheduler, process management, arch/kernel is CPU specific
- Networking sub system
- Lib: kernel library code, architecture library is in arch/lib
- Scripts: scripts for kernel configuration

16. What are basic Linux development tools a developer needs to have? ★★★

Ans.

- Gnu tools

    - Compiler, linker, objcopy, objdump, objsize
    - Make, uclibc (c library for compiler)
- Kernel source
    - Kernel.org, patches for the SoC and reference board
- Tftp
    - Trivial file transfer protocol (server and client)
- Minicom
    - Serial port terminal utility on Linux (similar to Hyper terminal)
- Network utilities
    - Ifconfig, Iptables, Firewall setting tools
- Busybox
    - Shell commands (also the shell itself) are packaged together keeping the embedded requirement in consideration

17. What is a development environment? What are various development environments? ★★★

Ans. It is a complete setup with all the required tools to enable the kernel and Linux application development.

- Linux host machine
    - Fedora
    - Ubuntu
- Buildroot
    - An integrated development environment for package and rootfs management
        - Build environment to compile the cross compiler
        - Build environment to create the root file system
- Commercial Development environments
    - Montavista
    - Timesys
    - LynxOS
    - WindRiver Linux
- Open source development environment
    - Builtroot
    - LTIB (Linux Target Image Builder)
    - Angstrom (Open distribution for Embedded environment like fedora, ubuntu for desktop enviroment)
    - Android

18. What do you mean by an elf image how it is different from bin image? ★★★★

Ans. Analogous to obj file, elf image is Embedded Loadable Format image which is further converted to bin format (equivalent to exe/hex)

19. What do you mean by uImage? ★★★

Ans. 'uImage' is the final image of kernel with header information needed by the boot loader.

20. What do you mean by a root file system? ★★★

Ans. The *root filesystem* is the *filesystem* that is contained on the same *partition* on which the *root directory* is located, and it is the filesystem on which all the other filesystems are *mounted* (i.e., logically attached to the system) as the system is *booted up* (i.e., started up).

A partition is a logically independent section of a hard disk drive (HDD). A filesystem is a hierarchy of directories (also referred to as a *directory tree*) that is used to organize files on a computer system. On Linux and and other Unix-like operating systems, the directories start with the root directory, which contains a series of subdirectories, each of which, in turn, contains further subdirectories, etc. A variant of this definition is the part of the entire hierarchy of directories (i.e., of the directory tree) that is located on a single partition or disk.

The exact contents of the root filesystem will vary according to the computer, but they will include the files that are necessary for booting the system and for bringing it up to such a state that the other filesystems can be mounted as well as tools for fixing a broken system and for recovering lost files from backups. The contents will include the root directory together with a minimal set of subdirectories and files including */boot*, */dev*, */etc*, */bin*, */sbin* and sometimes */tmp* (for temporary files).

Only the root filesystem is available when a system is brought up in *single user mode*. Single user mode is a way of booting a damaged system that has very limited capabilities so that repairs can be made to it. After repairs have been completed, the other filesystems that are located on different partitions or on different media can then be *mounted* on (i.e., attached to) the root filesystem in order to restore full system functionality. The directories on which they are mounted are called *mount points*.

The root filesystem should generally be small, because it contains critical files and a small, infrequently modified filesystem has a better chance of not becoming corrupted. A corrupted root filesystem will generally mean that the system becomes *unbootable* (i.e., unstartable) from the HDD, and must be booted by special means (e.g., from a *boot floppy*).

A filesystem can be mounted anywhere in the directory tree; it does not necessarily need to be mounted on the root filesystem. For example, it is possible (and very common) to have one filesystem mounted at a mount point on the root filesystem, and another filesystem mounted at a mount point contained in that filesystem.

21. What do you mean by a 'Shell'? ★★★

Ans. A shell is basically a command line interface responsible for executing user commands. Shell is the program that interprets the command you give.

There are various shells like bourn shell, kourn shell, C-shell etc.

22. What do you mean by parent and child processes? ★★★

Ans. As we know that our running program is a process. From this process we can create another process. There is a parent child relationship between the two processes. The way to achieve this is by using the library function (or a system call) fork(). This function splits the running process into two processes, the existing one is the parent and the new process is called the child process. The function actually copies the code of parent process into child process.

23. Why do we need to create child processes? ★★★

Ans. When we want our program to perform two jobs simultaneously. Since these jobs may be interrelated and we may not want create separate programs for the same. Suppose we want to copy contents of a source file to target file and display an animated GIF file indicating that the copy is in progress. The GIF file should continue to play till the copy is in progress. Once the copying is over the playing of GIF should be stopped. Since both the jobs are interrelated they cannot be performed in two different programs. Also they cannot be performed one after another. Both jobs should be performed simultaneously.
Hence we need to create parent-child relationship in processes. In this case copy is done by parent process and displaying GIF is done by child process

24. Which process in linux is called the father of all other processes? ★★★

Ans. The 'init' is called the father of all processes**.**

25. Explain the concept of Zombies and Orphans? ★★

Ans. Linux maintains a table containing information about all the processes called a Process table (in /proc/ directory). Apart from other information the process table contains an entry of 'exit code' of the process. This integer value indicates why the process was terminated. Even if the process is terminated its information is still maintained in the process table until its parent queries for the exit code. As soon as the parent queries the exit code terminated child, its information is deleted from the process table.
Now there can be two possibilities…
Child terminated earlier than parent: If it is the case then until the parent process queries the exit code of terminated child the information about the child continues to exist in the process table. Such a process in Linux terminology is called a Zombie process.
What if the parent process terminates without querying. In such case the zombie child process is treated as an 'Orphan' process. Immediately the father of all processes 'init' adopts the orphaned process. Next as a responsible parent init queries the process table as a result of which the child process entry is eliminated from the list.

Parent terminates before child: Since every parent is launched from Linux shell, the parent of parent is shell process. When our parent terminates, the shell queries the process table. Thus a proper cleanup happens for parent process.. However, the child which is still running is left orphaned. Immediately the init process would adopt it and when its execution is over init would query the process table to clean up the entry for the child process. Note that in this case the child process doesn't become a Zombie.

26. Why it is important to get exit code of terminated process? ★★

Ans. It is because; it is the exit code that would give indication about whether the job assigned to the process was completed successfully or not.

27. How to avoid child becoming Zombie or Orphan? ★★★

Ans.

28. What are various system calls related to process? ★★

Ans. fork(), exec(), wait(), exit()

29. What is a device driver? How it is different from a kernel module? ★★★★

Ans. A device driver is a piece of specific software that communicates with particular device. The device driver resides in kernel space in case of monolithic kernels. The device driver registers can only be accessed in privileged mode. Applications in OS environment use drivers to operate on hardware. So in Linux environment we can say that the drivers are kernel services for applications to access hardware functionalities.

A kernel module is a bit of compiled code that can be inserted into the kernel at run-time, such as with insmod or modprobe.

A driver is a bit of code that runs in the kernel to talk to some hardware device. It "drives" the hardware. Most every bit of hardware in your computer has an associated driver[*]. A large part of a running kernel is driver code; the rest of the code provides generic services like memory management, IPC, scheduling, etc.

A driver may be built statically into the kernel file on disk. (The one in /boot, loaded into RAM at boot time by the boot loader early in the boot process.) A driver may also be built as a kernel module so that it can be dynamically loaded later. (And then maybe unloaded.)

Standard practice is to build drivers as kernel modules where possible, rather than link them statically to the kernel, since that gives more flexibility. There are good reasons not to, however:

- Sometimes a given driver is absolutely necessary to help the system boot up. That doesn't happen as often as you might imagine, due to the initrd feature.
- Statically built drivers may be exactly what you want in a system that is statically scoped, such as an embedded system. That is to say, if you know in

advance exactly which drivers will always be needed and that this will never change, you have a good reason not to bother with dynamic kernel modules.

Not all kernel modules are drivers. For example, a relatively recent feature in the Linux kernel is that you can load a different process scheduler.

30. Why do we need virtual device drivers when we have physical device drivers? ★★★★

Ans. Device drivers are basically a set of modules/routines so as to handle a device for which a direct way of communication is not possible through the user's application program and these can be thought of as an interface thus keeping the system small providing for minimalistic of additions of code, if any. Physical device drivers can't perform all the logical operations needed in a system in cases like IPC, Signals and so on...
The main reason for having virtual device drivers is to mimic the behaviour of certain hardware devices without it actually being present and these could be attributed to the high cost of the devices or the unavailability of such devices. These basically create an illusion for the users as if they are using the actual hardware and enable them to carry out their simulation results. Examples could be the use of virtual drivers in case of Network simulators, also the support of virtual device drivers in case a user runs an additional OS in a virtual box kind of software.

31. What is major number and minor number in linux? ★★★

Ans. One of the basic features of the Linux kernel is that it abstracts the handling of devices. All hardware devices look like regular files; they can be opened, closed, read and written using the same, standard, system calls that are used to manipulate files. Every device in the system is represented by a file. For block (disk) and character devices, these device files are created by the mknod command and they describe the device using major and minor device numbers. Network devices are also represented by device special files but they are created by Linux as it finds and initializes the network controllers in the system.

The kernel needs to be told how to access the device. Not only does the kernel need to be told what kind of device is being accessed but also any special information, such as the partition number if it's a hard disk or density if it's a floppy, for example. This is accomplished by the major number and minor number of that device.

All devices controlled by the same device driver have a common major device number. The minor device numbers are used to distinguish between different devices and their controllers. Linux maps the device special file passed in system calls (say to mount a file system on a block device) to the device's device driver using the major device number and a number of system tables, for example the character device table, chrdevs. The major number is actually the offset into the kernel's device driver table, which tells the kernel what kind of device it is (whether it is a hard disk or a serial terminal). The minor number tells the kernel special characteristics of the device to be accessed. For example, the second hard

disk has a different minor number than the first. The COM1 port has a different minor number than the COM2 port, each partition on the primary IDE disk has a different minor device number, and so forth. So, for example, /dev/hda2, the second partition of the primary IDE disk has a major number of 3 and a minor number of 2.

32. What is char module, block module and network module? ★★★

Ans.

Character devices

A character (char) device is one that can be accessed as a stream of bytes (like a file); a char driver is in charge of implementing this behavior. Such a driver usually implements at least the *open*, *close*, *read*, and *write* system calls. The text console (*/dev/console*) and the serial ports (*/dev/ttySo* and friends) are examples of char devices, as they are well represented by the stream abstraction. Char devices are accessed by means of filesystem nodes, such as */dev/tty1* and */dev/lpo*. The only relevant difference between a char device and a regular file is that you can always move back and forth in the regular file, whereas most char devices are just data channels, which you can only access sequentially. There exist, nonetheless, char devices that look like data areas, and you can move back and forth in them; for instance, this usually applies to frame grabbers, where the applications can access the whole acquired image using *mmap* or *lseek*.

Block devices

Like char devices, block devices are accessed by filesystem nodes in the */dev* directory. A block device is a device (e.g., a disk) that can host a filesystem. In most Unix systems, a block device can only handle I/O operations that transfer one or more whole blocks, which are usually 512 bytes (or a larger power of two) bytes in length. Linux, instead, allows the application to read and write a block device like a char device—it permits the transfer of any number of bytes at a time. As a result, block and char devices differ only in the way data is managed internally by the kernel, and thus in the kernel/driver software interface. Like a char device, each block device is accessed through a filesystem node, and the difference between them is transparent to the user. Block drivers have a completely different interface to the kernel than char drivers. CDROM, HDD are examples of block drivers

Network interfaces

Any network transaction is made through an interface, that is, a device that is able to exchange data with other hosts. Usually, an *interface* is a hardware device, but it might also be a pure software device, like the loopback interface. A network interface is in charge of sending and receiving data packets, driven by the network subsystem of the kernel, without knowing how individual transactions map to the actual packets being transmitted. Many network connections (especially those using TCP) are stream-oriented, but network devices are, usually, designed around the transmission and receipt of packets. A network driver knows nothing about individual connections; it only handles packets.

33. What is proc file system in linux? ★★★★

Ans. The proc file system is a pseudo-file system rooted at /proc that contains user-accessible objects that pertain to the runtime state of the kernel and, by extension, the executing processes that run on top of it. "Pseudo" is used because the proc file system exists only as a reflection of the in-memory kernel data structures it displays. This is why most files and directories within /proc are 0 bytes in size.

Broadly speaking, a directory listing of /proc reveals two main file groups. Each numerically named directory within /proc corresponds to the process ID (PID) of a process currently executing on the system. The following line of **ls** output illustrates this:

```
dr-xr-xr-x  3 noorg  noorg      0 Apr 16 23:24 19636
```

Directory 19636 corresponds to PID 19636, a current bash shell session. These per-process directories contain both subdirectories and regular files that further elaborate on the runtime attributes of a given process. The proc(5) manual page discusses these process attributes at length.

The second file group within /proc is the non-numerically named directories and regular files that describe some aspect of kernel operation. As an example, the file /proc/version contains revision information relevant to the running kernel image.

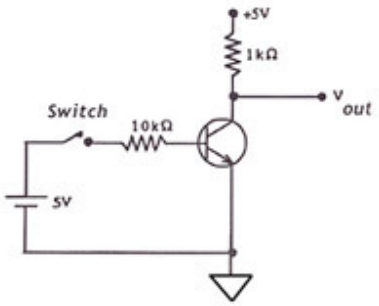
Proc files are either read-only or read-write. The /proc/version file above is an example of a read-only file. Its contents are viewable by way

of cat(1), and they remain static while the system is powered up and accessible to users. Read-write files, however, allow for both the display and modification of the runtime state of the kernel. /proc/sys/net/ipv4/ip_forwarding is one such file. Using cat(1) on this file reveals if the system is forwarding IP datagrams between network interfaces--the file contains a 1--or not--the file contains a 0. In echo(1)ing 1 or 0 to this file, that is, writing to the file, we can enable or disable the kernels ability to forward packets without having to build and boot a new kernel image. This works for many other proc files with read-write permissions.

34. What is sysfs file system? Why is it a good replacement for proc?

Basic Electronics

1. Explain transistor as a switch. Draw circuit diagram.
★★★★★

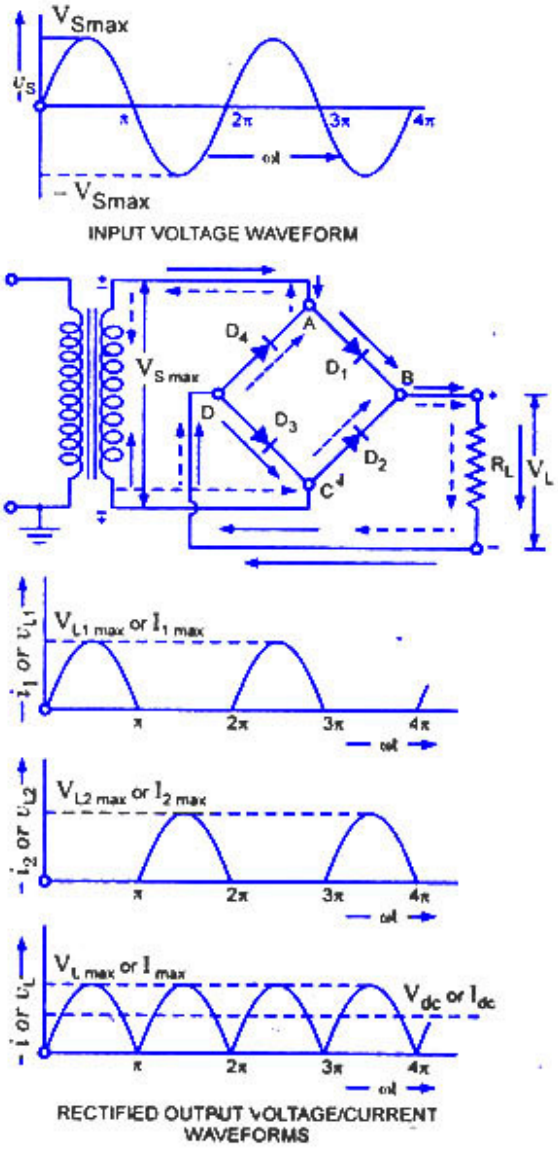


Ans.
A transistor as a switch represents a logical NOT gate which gives output LOW whenever there is an input current (base current) i.e. HIGH input state. This base current makes transistor go in saturation causing output terminal shorted to the ground i.e. at 0 potential. On the other hand when the input switch is open i.e. there is no input current (or base current), the transistor behaves like an open circuit and the +5V Vcc is available at output showing the output high state.

2. Can you make transistor as switch to work as active high application i.e. for logic high input there must be a logic high output.
★★★★

Ans. Yes, it's quite easy. Just connect another transistor at Vout terminal so that Vout behaves input to the second transistor and output is taken from the collector of it. This configuration represents 2 NOT gates connected in series.

3. Draw a full wave bridge rectifier circuit with corresponding wave forms. ★★★



Bridge Rectifier

4. What is difference between a 2 diode rectifier and a bridge rectifier? ★★★

5. What do you mean by dc value? How is it related with rms value? ★★★

6. What is charging and discharging time of a capacitor? How is it controlled? ★★

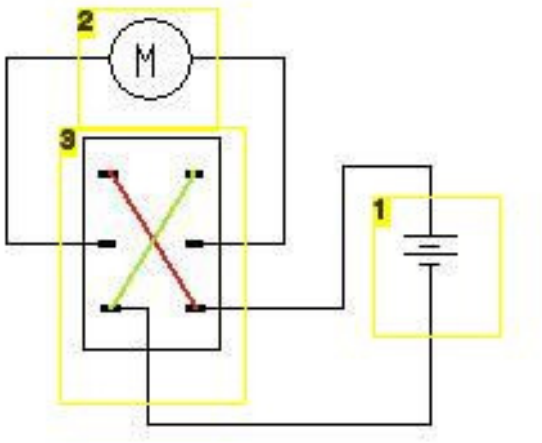
7. How to improve the current rating of power supply? ★★★★

Ans. Current rating refers to the capability of any device to handle certain amount of current. Improving current rating involves modification/replacement of all components of power supply to sustain desired current.

8. Draw circuit diagram for 2 directional DC motor controller using microcontroller. ★★★★★

Ans. The required circuit contains 3 elements.

1. DC Supply
2. DC Motor
3. DPDT Switch i.e. dual pole dual throw switch.



9. Why CMOS devices have larger propagation delays than TTL ones? ★★★

Ans. This is because of the capacitance present at GATE and substrate terminal. This capacitance leads CMOS devices to slow down the performance by increasing signal propagation delays.

10. How 2x1 MUX can be used as a 2 input AND gate. ★★★★

Ans. We know that the output of 2x1 MUX is given by equation
Y = S'A + SB
Now if we put S = A in above equation, it becomes
Y = A'A + AB.
By laws of boolean algebra, term A'A cancels hence the output is
Y = AB.
So to make an AND gate out of a 2x1 MUX, one just needs to connect the select line with the first input of MUX.

Referring above example, other gates can also be realized.

Note: Although there can be many such questions but space and time doesn't allow to include them all. Also chances are rare that questions for basic electronics are asked. Usually interviewers with old mentality and bold visibility (people who need bold characters to read properly) show interest in such questions. They want to prove their knowledge instead of testing yours hence throw questions from electronics and circuit designs!!
Anyways, I would recommend concentrating on digital electronics rather than the analog electronics.

Project/Assignment related questions

1. What do you mean by an SRS (Software Requirements and Specifications)? ★★★

Ans. SRS refers to a basic idea and technical details required to start building software. It's the process of knowing inputs and outputs of the software and its expected functionality.

2. What were specification and requirements of your projects? ★★★

Ans. The Specifications and Requirements document is usually generated by the end user or the client who is finally going the use the product in live. Document broadly states what product is all about, what functionality is expected from it. What and how many no. of inputs/outputs will be there. For hardware products, electrical specifications are also specified. Sometimes user may specify the internal details as well like what data length of the processor and particular operating system must be used.

3. What were your responsibilities towards the project? ★★★

Ans. Here you can explain about your role in the project you were involved. What were your deliverables and how many departments you communicated with like testing, production, stores etc. You should also explain about your performance at work if it was really good. Problem solving attitude and friendly behavior with team members are the key skills that every employer looks for.

4. How many members were involved? What were their responsibilities? ★★★

Ans. To answer this question, you need to explain role of your team members the way you explained yours. You should commit that each and every team member was important and without their involvement project would have not been possible.

5. Draw block diagram of your project. Explain each block. ★★★

Ans. This is very critical questions specially when you are being interview by the competitor of your company. You should be very careful while answering such type of questions. Try to be diplomatic this time. If the project related information is not so generic then without giving so much of technical details, you can just draw a broader view of the project.

6. Why did you select particular protocol? ★★★

Ans. If the protocol requirements are mentioned in Specifications and Requirement documents then tell so. Otherwise what was the speed and interfacing requirements to select the particular protocol need to be explained.

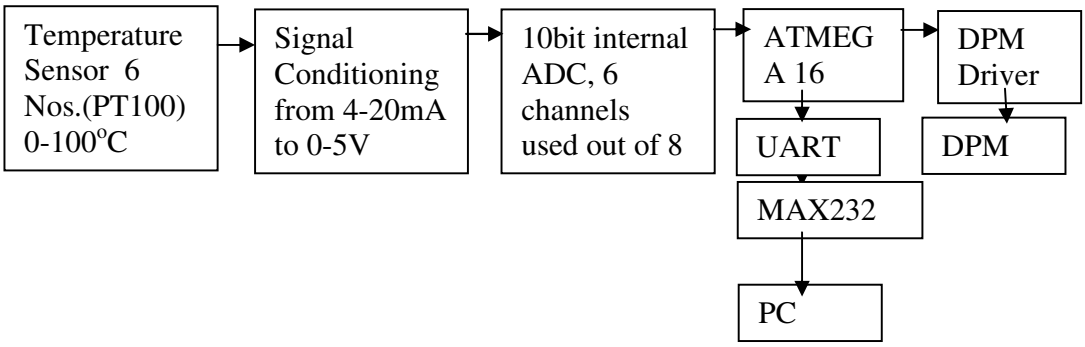
7. Why did you select particular microcontroller for this project?

★★★

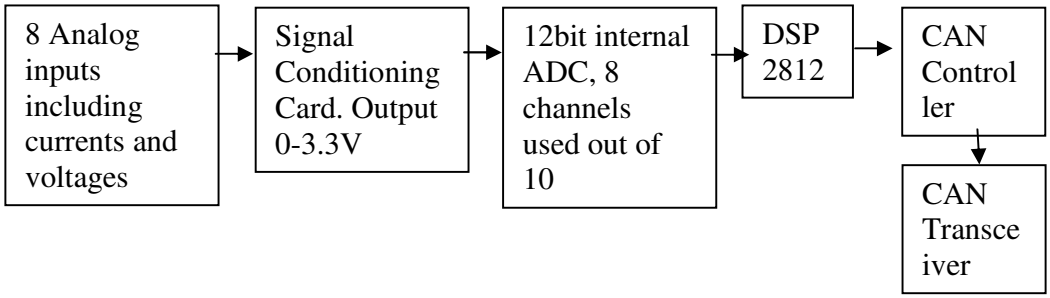
Ans. Speed, Cost, Memory, Peripherals and Ease of availability are the important aspects while selecting any microcontroller or processing mother board for your application. So what were your dominant criteria for a microcontroller selection like to meet temperature specifications you might have compromised with speed etc.

Ans.

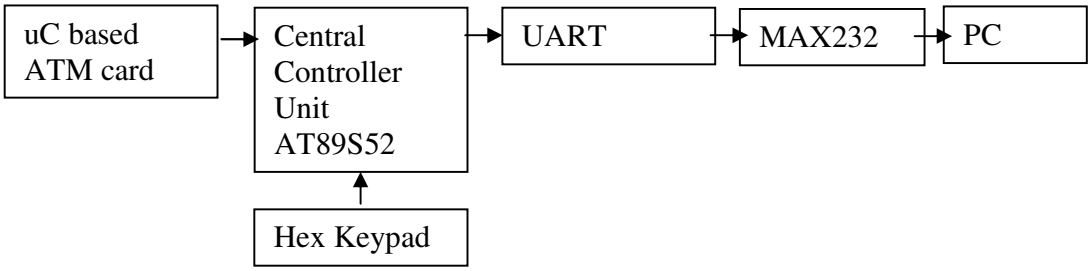
6 channel temperature logger. Based on ATMEGA16



8 channel AI card with CAN communication. Based on DSP TMS320F2812



ATM Demonstrator with PC based Application software



Embedded Testing

What is testing? Testing is a disciplined process that consists of evaluating the application (including its components) behavior, performance, and robustness -- usually against expected criteria. One of the main criteria, although usually implicit, is to be as defect-free as possible. Expected behavior, performance, and robustness should therefore be both formally described and measurable. Verification and Validation (V&V) activities focus on both the quality of the software product and of the engineering process. These V&V activities can be sub-classified as either preventative, detective, or corrective measures of quality. While testing is most often regarded as a detective measure of quality, it is closely related to corrective measures such as debugging. In practice, software developers usually find it more productive to enact testing and debugging together, usually as an interactive process. Debugging literally means removing defects ("bugs").

1. What are the main aspects of Embedded Testing? ★★★★

Ans.

Six Aspects of testing embedded systems:
1. Software unit testing
2. Software integration testing
3. Software validation testing
4. System unit testing
5. System integration testing
6. System validation testing

2. What is functional testing? ★★★

Ans. Functional testing is done with users perspective as if user is using the product and its functionality. Functional testing is used to check whether the system is performing as per user specifications or not.

3. What is difference between functional testing and unit testing? ★★★

Ans. Unit tests tell a developer that the code is doing things right; functional tests tell a developer that the code is doing the right things.

Unit Tests are written from a *programmers* perspective. They are made to ensure that a particular method (or a *unit*) of a class performs a set of specific tasks.

Functional Tests are written from the *user's* perspective. They ensure that the system is *functioning* as users are expecting it to function.

Unit testing is analogous to a building inspector visiting a house's construction site. He is focused on the various internal systems of the house, the foundation, framing, electrical, plumbing, and so on. He ensures (tests) that the parts of the house will work correctly and safely, that is, meet the building code.

Functional tests in this scenario are analogous to the homeowner visiting this same construction site. He assumes that the internal systems will behave appropriately, that the building inspector is performing his task. The homeowner is focused on what it will be like to live in this house. He is concerned with how the house looks, are the various rooms a comfortable size, does the house fit the family's needs, are the windows in a good spot to catch the morning sun.

4. Difference between functional testing and structural testing? ★★★

Ans. Functional testing (also known as black-box testing), is a software testing approach in which:

The tester will have a user perspective in mind,
not knowing and doesn't mind how the program works.
Input and output are the only things that matter.
The tester acts as if he/she is the final user of the program.

On the other hand, structural testing (also known as white-box testing), is a software testing approach in which:

The tester will have a developer perspective in mind,
knowing how the program works behind the scene,
such that the test will test all algorithm paths in the program.
Everything does matter.

The tester acts as a developer of the program who knows the internal structure of the program very well.

5. Difference between functional testing and integration testing? ★★★

Ans. Integration testing is when you test more than one component and how they function together. For instance how another system interacts with your system or the database interacts with your data abstraction layer. Usually this requires a fully installed system, although in its purest forms it does not.

Functional testing is when you test the system against the functional requirements of the product. Product/Project management usually writes these up and QA formalizes the process of what a user should see and experience, and what the end result of those processes should be. Depending on the product this can be automated or not.

6. What is regression testing? ★★★

Ans. Regression testing is any type of software testing that seeks to uncover new software bugs, or *regressions*, in existing functional and non-functional areas of a system after changes, such as enhancements, patches or configuration changes, have been made to them.

The intent of regression testing is to ensure that a change such as those mentioned above has not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.

Common methods of regression testing include rerunning previously-completed tests and checking whether program behavior has changed and whether previously-fixed faults have re-emerged. Regression testing can be used to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.

7. What is difference between verification and validation? ★★★

Ans.

Verification is Static while Validation is Dynamic. This means in Verification the s/w is inspected by looking into the code going line by line or function by function. In Validation, code is executed and s/w is run to find defects. Since in verification code is reviewed, location of the defect can be found which is not possible in validation.

Verification - is to determine the right thing, which involves the testing the implementation of right process. Ex: Are we building the product right?

Validation - is to perform the things in right direction, like checking the developed software adheres the requirements of the client. Ex: right product was built

Index