Embedded C Interview Questions

Part 1

1. **What is Embedded C Programming? How is Embedded C different from C language?**

   Embedded C is a programming language that is an extension of C programming. It uses the same syntax as C and it is called “embedded” because it is used widely in embedded systems. Embedded C supports I/O hardware operations and addressing, fixed-point arithmetic operations, memory/address space access, and various other features that are required to develop fool-proof embedded systems.

   Following are the differences between traditional C language and Embedded C:

C Language	Embedded C Language
It is of native development nature	It is used for cross-development purposes
C is independent of hardware and its underlying architecture	Embedded C is dependent on the hardware architecture.
C is mainly used for developing desktop applications.	Embedded C is used in embedded systems that have limited resources like ROM, RAM, etc.

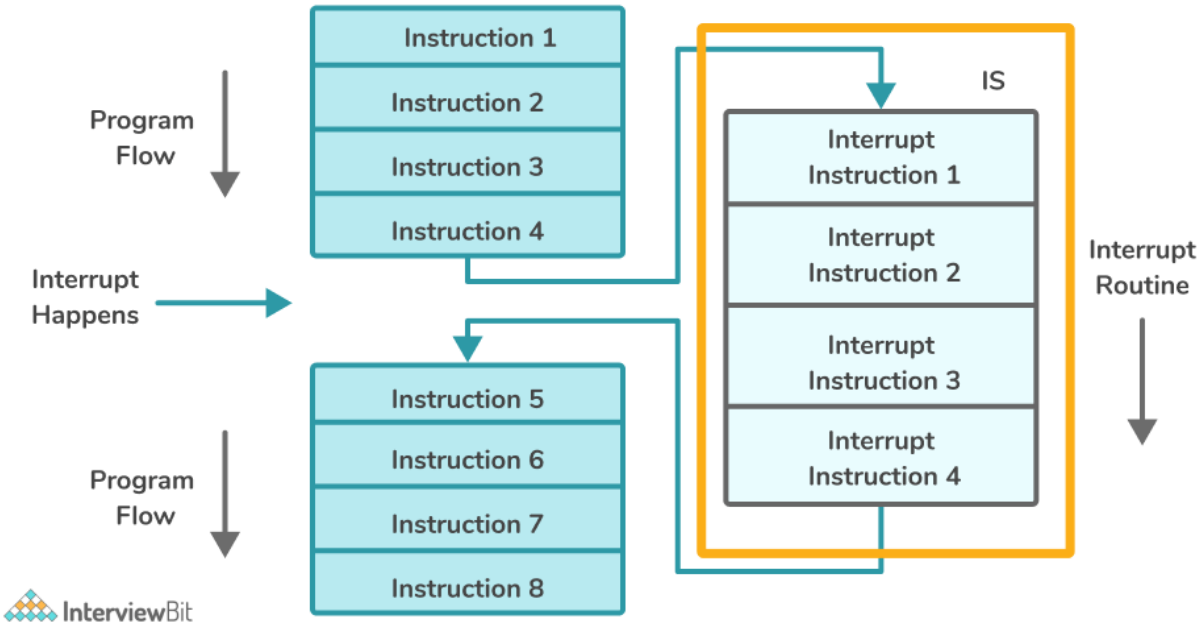
2. **What do you understand by startup code?**

   A startup code is that piece of code that is called before the execution of the main function. This is used for creating a basic platform for the application and it is written in assembly language.

3. **What is ISR?**

   ISR expands to Interrupt Service Routines. These are the procedures stored at a particular memory location and are called when certain interrupts occur. Interrupt refers to the signal sent to the processor that indicates there is a

high-priority event that requires immediate attention. The processor suspends the normal flow of the program, executes the instructions in ISR to cater for the high priority event. Post execution of the ISR, the normal flow of the program resumes. The following diagrams represent the flow of ISR.



4. What is Void Pointer in Embedded C and why is it used?

Void pointers are those pointers that point to a variable of any type. It is a generic pointer as it is not dependent on any of the inbuilt or user-defined data types while referencing. During dereferencing of the pointer, we require the correct data type to which the data needs to be dereferenced.

For Example:

```
int num1 = 20;     //variable of int datatype
void *ptr;         //Void Pointer
*ptr = &num1;    //Point the pointer to int data
print("%d",(*(int*)ptr));    //Dereferencing requires specific data type

char c = 'a';
*ptr = &c;    //Same void pointer can be used to point to data of
different type -> reusability
print("%c",(*(char*)ptr));
```

Void pointers are used mainly because of their nature of re-usability. It is reusable because any type of data can be stored.

5. **Why do we use the volatile keyword?**

   The volatile keyword is mainly used for preventing a compiler from optimizing a variable that might change its behavior unexpectedly post the optimization. Consider a scenario where we have a variable where there is a possibility of its value getting updated by some event or a signal, then we need to tell the compiler not to optimize it and load that variable every time it is called. To inform the compiler, we use the keyword volatile at the time of variable declaration.

```
// Declaring volatile variable - SYNTAX
// volatile datatype variable_name;
volatile int x;
```

   Here, x is an integer variable that is defined as a volatile variable.

6. **What are the differences between the const and volatile qualifiers in embedded C?**

const	volatile
The keyword "const" is enforced by the compiler and tells it that no changes can be made to the value of that object/variable during program execution.	The keyword "volatile" tells the compiler to not perform any optimization on the variables and not to assume anything about the variables against which it is declared.
Example: `const int x=20;` , here if the program attempts to modify the value of x, then there would be a compiler error as there is const keyword assigned which makes the variable x non-modifiable.	Example: `volatile int x;` , here the compiler is told to not assume anything regarding the variable x and avoid performing optimizations on it. Every time the compiler encounters the variable, fetch it from the memory it is assigned to.

7. **What Is Concatenation Operator in Embedded C?**

   The Concatenation operator is indicated by the usage of ##. It is used in macros to perform concatenation of the arguments in the macro. We need to keep note that only the arguments are concatenated, not the values of those arguments.

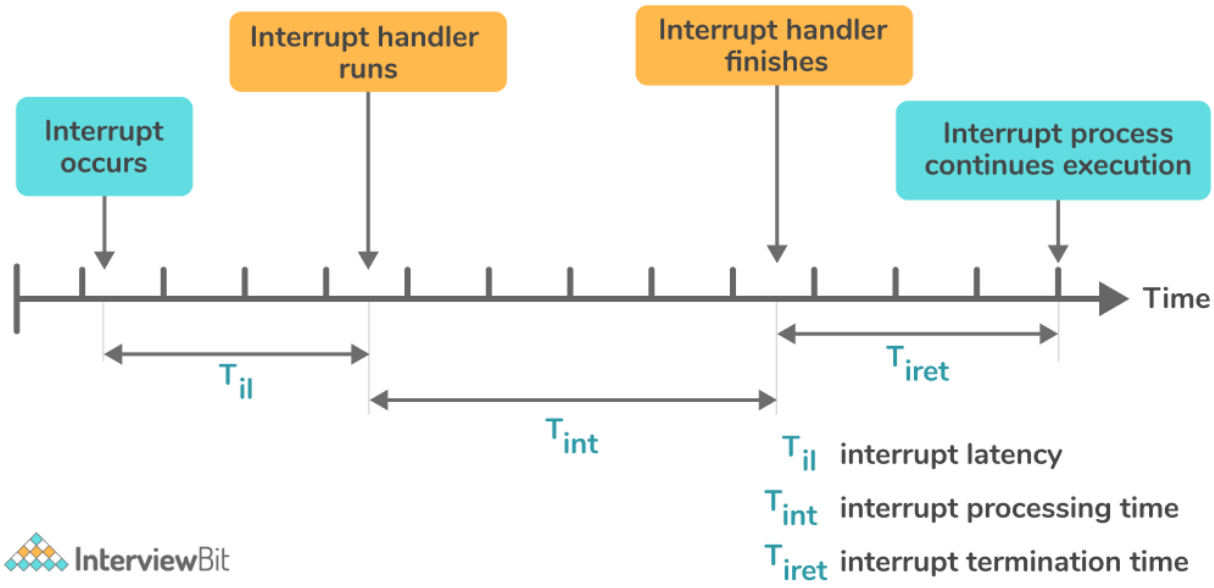
   For example, if we have the following piece of code:

```
#define CUSTOM_MACRO(x, y) x##y
main(){
  int xValue = 20;
  printf("%d", CUSTOM_MACRO(x, Value));     //Prints 20
}
```

We can think of it like this if arguments x and y are passed, then the macro just returns xy -> The concatenation of x and y.

8. What do you understand by Interrupt Latency?

Interrupt latency refers to the time taken by ISR to respond to the interrupt. The lesser the latency faster is the response to the interrupt event.



9. How will you use a variable defined in source file1 inside source file2?

We can achieve this by making use of the “extern” keyboard. It allows the variable to be accessible from one file to another. This can be handled more cleanly by creating a header file that just consists of extern variable declarations. This header file is then included in the source files which uses the extern variables. Consider an example where we have a header file named variables.h and a source file named sc_file.c.

```
/* variables.h*/
extern int global_variable_x;
/* sc_file.c*/
#include "variables.h"  /* Header variables included */
#include <stdio.h>
```

```
void demoFunction(void){
   printf("Value of Global Variable X: %d\n", global_variable_x++);
}
```

10. What do you understand by segmentation fault?

A segmentation fault occurs most commonly and often leads to crashes in the programs. It occurs when a program instruction tries to access a memory address that is prohibited from getting accessed.

11. Is it possible to declare a static variable in a header file?

Variables defined with static are initialized once and persists until the end of the program and are local only to the block it is defined. A static variables declaration requires definition. It can be defined in a header file. But if we do so, a private copy of the variable of the header file will be present in each source file the header is included. This is not preferred and hence it is not recommended to use static variables in a header file.

12. Is it possible for a variable to be both volatile and const?

The const keyword is used when we want to ensure that the variable value should not be changed. However, the value can still be changed due to external interrupts or events. So, we can use const with volatile keywords and it won't cause any problem.

13. What do you understand by the pre-decrement and post-decrement operators?

The Pre-decrement operator (--operand) is used for decrementing the value of the variable by 1 before assigning the variable value.

```
#include < stdio.h >
int main(){
    int x = 100, y;
    y = --x;   //pre-decrememt operators  -- first decrements the value
and then it is assigned
    printf("y = %d\n", y);  // Prints 99
    printf("x = %d\n", x);  // Prints 99
    return 0;}
```

The Post-decrement operator (operand--) is used for decrementing the value of a variable by 1 after assigning the variable value.

```
#include < stdio.h >
int main(){
    int x = 100, y;
    y = x--;   //post-decrememt operators  -- first assigns the value and
then it is decremented
    printf("y = %d\n", y);  // Prints 100
    printf("x = %d\n", x);  // Prints 99
    return 0;
}
```

14. What is a reentrant function?

A function is called reentrant if the function can be interrupted in the middle of the execution and be safely called again (re-entered) to complete the execution. The interruption can be in the form of external events or signals or internal signals like call or jump. The reentrant function resumes at the point where the execution was left off and proceeds to completion.

15. What are the differences between Inline and Macro Function?

Category	Macro Function	Inline Function
Compile-time expansion	Macro functions are expanded by the preprocessor at the compile time.	Inline functions are expanded by the compiler.
Argument Evaluation	Expressions passed to the Macro functions might get evaluated more than once.	Expressions passed to Inline functions get evaluated once.
Parameter Checking	Macro functions do not follow strict parameter data type checking.	Inline functions follow strict data type checking of the parameters.
Ease of debugging	Macro functions are hard to debug because it is replaced by the pre-processor as a textual representation which is not visible in the source code.	Easier to debug inline functions which is why it is recommended to be used over macro functions.
Example	#define SQUARENUM(A) A * A -> The macro functions are expanded at compile time. Hence, if we pass, the output will be evaluated to 3+2*3+2 which gets evaluated to 11. This might not be as per our expectations.	inline squareNum(int A){return A * A;} -> If we have printf(squareNum(3+2)); , the arguments to the function are evaluated first to 5 and passed to the function, which returns a square of 5 = 25.