A microprocessor is a central processing unit (CPU) designed to execute computer instructions and perform calculations in electronic devices. They are responsible for executing instructions, performing arithmetic and logical operations, managing memory, and controlling input/output devices. Microprocessors take the response from sensors in digital form and process it to produce the output in a real-time environment. They support programming languages depending on architecture and the software development tools

Microprocessor languages

1. Assembly Language:

- low-level programming language
- corresponds to the machine code instructions executed by the microprocessor)
- provides direct access to the microprocessor's registers, memory, and instructions
- specific to each microprocessor architecture

2. C :

- high-level programming language
- used for microprocessor programming
- allows developers to write code that can interact with the microprocessor's hardware
- C compilers are available for many microprocessor architectures.

Machine language (machine code)

It is a low-level programming language that directly corresponds to the instructions executed by a computer's hardware.

Machine language instructions are represented as binary patterns of 0s and 1s, which the CPU can decode and execute directly.

Each instruction performs a specific operation, such as arithmetic calculations, data manipulation, or control flow. Machine language instructions are typically

encoded as sequences of bits, where each bit represents a specific operation or operand.

Writing programs directly in machine language can be complex and error-prone, as it requires a deep understanding of the microprocessor's instruction set and binary representation. Therefore, higher-level programming languages and assemblers are often used to write programs in a more human-readable and manageable format, which are then translated into machine language instructions by an assembler or compiler.

Assembly language

- It is a low-level programming language that serves as an intermediary between machine language and higher-level programming languages.
- It represents a human-readable form of machine language instructions
- Assembly language programs consist of a series of instructions, each specifying an operation to be performed by the computer's processor.
- Assembly language is more readable and easier to work with than machine language, but it still requires a good understanding of the underlying hardware architecture.

History of programming language

1. Machine Language (1940s-1950s).

2. Assembly Language (1950s-1960s).

3. Fortran (1957): "Formula Translation".

4. LISP (1958): "LISt Processing".

5. COBOL (1959): "COmmon Business-Oriented Language".

6. ALGOL (1958-1960): "ALGOrithmic Language"

It was a collaborative effort by the international computer science community to create a universal programming language.

7. C (1972):

- C was developed by Dennis Ritchie at Bell Labs as a systems programming language. It became popular due to its efficiency, portability, and low-level capabilities, and it influenced the development of many subsequent languages.

8. C++ (1983):

- C++ was developed by Bjarne Stroustrup as an extension of the C programming language. It introduced object-oriented programming (OOP) features, such as classes and inheritance, while maintaining C's efficiency and low-level capabilities.

ANSI C , also known as C89 or C90, refers to the standard version of the C programming language. The ANSI C standard introduced several important features and clarifications to the C language to ensure portability and consistency across different platforms and compilers

Why C language is used in embedded systems?

- C is a low-level language that allows direct manipulation of hardware resources. It provides fine-grained control over memory allocation and access, making it highly efficient in terms of execution speed and memory usage.
- C is a highly portable language. It allows developers to write code that can be easily adapted to different hardware platforms and operating systems
- Embedded systems often require direct access to hardware components, such as microcontrollers, sensors, and peripherals. C allows programmers to interact with these hardware devices through direct memory manipulation

Toolchains are sets of software tools that are used together to develop and build software applications for a specific platform or target system.

1. Native Toolchains: are used when developing software on the same platform where it will be executed. Examples include the GNU toolchain for Linux (GCC, GNU Binutils) and Microsoft Visual Studio for Windows.

2. Cross-Compilation Toolchains: are used when developing software on one platform but targeting a different platform or architecture. They include compilers, linkers, and other tools specifically configured to generate executable code for the target platform. Examples include the GNU toolchain for embedded systems (GCC, GNU Binutils) and Android NDK for building Android applications.

The building or compilation process refers to the process of converting source code written in a high-level programming language into executable machine code that can be run on a computer.

- Preprocessing: In this initial step, the compiler or build system processes any preprocessor directives in the source code, such as `#include` statements or macro expansions.

- Compilation: The compiler takes the preprocessed source code and translates it into low-level, platform-specific code. This code is usually in the form of assembly language or an intermediate representation specific to the compiler.

- Linking: If the program is composed of multiple source files or relies on external libraries, the linking phase comes into play. The linker combines the compiled object code from different source files and resolves references between them.
- Output: After the compilation and linking process is complete, the final output is generated. This output can be an executable file, a dynamically linked library, or an object file, depending on the intended use and the options specified during the build process.