MODULE - 1 (Introduction)

Syllabus : System Software Vs. Application Software, Different System Software– Assembler, Linker, Loader, Macro Processor, Text Editor, Debugger, Device Driver, Compiler, Interpreter, Operating System(Basic Concepts only). SIC & SIC/XE Architecture, Addressing modes, SIC & SIC/XE Instruction set, Assembler Directives. SIC/XE Programs.

1. System Software vs. Application Software:

System Software:

System software is a type of software that manages computer hardware resources and provides a platform for running application software. It acts as an intermediary between the hardware and application programs, ensuring smooth functioning of the computer system. Here are some key points about system software:

- **Operating System (OS):** The most essential system software, the operating system manages computer hardware and provides services to applications. It facilitates communication between hardware components and software programs, manages system resources such as memory and CPU, and provides a user interface for interaction.

- **Device Drivers:** Device drivers are software components that allow the operating system to communicate with hardware devices. They enable the operating system to control and interact with peripherals such as printers, keyboards, and graphics cards.

- **Utilities:** System utilities are software tools designed to perform specific tasks related to system management, maintenance, and optimization. Examples include disk cleanup tools, antivirus programs, system monitoring utilities, and backup software.

- **Compiler and Interpreter:** Compilers and interpreters are essential tools for software development. A compiler translates high-level programming languages into machine code that can be executed by the computer, while an interpreter executes high-level code directly without prior compilation.

Application Software:

Application software is designed to perform specific tasks or solve particular problems for users. Unlike system software, which is essential for the functioning of the computer system, application software serves the needs and requirements of end users. Here are some examples of application software:

- **Word Processors:** Word processors are used for creating, editing, and formatting text documents. They provide features such as spell-checking, formatting options, and document templates.

- **Spreadsheets:** Spreadsheet software is used for organizing, analyzing, and manipulating numerical data. Users can create tables, perform calculations, and generate charts and graphs using spreadsheet applications.

- **Web Browsers:** Web browsers are software tools used for accessing and navigating the World Wide Web. They allow users to view websites, download files, and interact with web-based applications.

- **Graphics Editing Software:** Graphics editing software is used for creating and editing digital images and graphics. Users can manipulate images, apply filters and effects, and create artwork using graphics editing applications.

2. Different System Software:

Assembler:

An assembler is a type of system software that translates assembly language programs into machine code that can be executed by the computer's CPU. Assembly language is a low-level programming language that uses mnemonic codes to represent machine instructions and memory addresses. The assembler converts assembly language instructions into binary code (machine code) that the computer can understand and execute.

Linker:

A linker is a program that combines multiple object files generated by the assembler into a single executable file. When a program is written in multiple source files, each file is compiled separately into an object file containing machine code and data. The linker resolves references between different object files, combines them into a single executable file, and assigns memory addresses to program elements.

Loader:

A loader is a system software component that loads executable files into memory for execution. When a program is executed, the loader reads the executable file from disk, allocates memory space for the program's code and data, and transfers control to the program's entry point. The loader also resolves external references and performs any necessary relocations to ensure that the program can run correctly in memory.

Macro Processor:

A macro processor is a program that expands macro instructions into assembly language instructions. Macros are predefined text strings that are replaced with corresponding code fragments during the assembly process. Macro processors allow programmers to define reusable code snippets and simplify the writing of assembly language programs by eliminating repetitive tasks.

Text Editor:

A text editor is a software tool used for creating and editing text files. Text editors provide features such as syntax highlighting, search and replace, line numbering, and indentation. They are commonly used by programmers to write source code and by writers to compose documents.

Debugger:

A debugger is a software tool used for finding and fixing errors (bugs) in software programs. Debuggers allow programmers to pause the execution of a program, inspect its state, and trace the flow of execution. They provide features such as breakpoints, watchpoints, and variable inspection, which help programmers identify and diagnose problems in their code.

Device Driver:

A device driver is a software component that allows the operating system to communicate with hardware devices. Device drivers act as intermediaries between the operating system and hardware components, providing a standardized interface for accessing device functionality. They enable the operating system to control and interact with peripherals such as printers, keyboards, and network adapters.

Compiler:

A compiler is a software tool that translates high-level programming languages into machine code that can be executed by the computer's CPU. The process of translation is called compilation, and the resulting machine code is often stored in an executable file that can be run by the user. Compilers perform various tasks such as lexical analysis, syntax analysis, optimization, and code generation to convert source code into executable code.

Interpreter:

An interpreter is a software tool that executes high-level programming language code directly without prior compilation. Interpreters read source code line by line and execute it immediately, translating each statement into machine code at runtime. Unlike compilers, which generate standalone executable files, interpreters require the presence of the interpreter program to run the code.

Operating System (Basic Concepts):

The operating system (OS) is a fundamental software component that manages computer hardware resources and provides services to applications. It acts as an intermediary between the hardware and software layers of the computer system, abstracting the hardware details and providing a uniform interface for application programs. Here are some basic concepts related to operating systems:

- **Process Management:** The operating system is responsible for managing processes, which are instances of executing programs. It allocates CPU time, memory, and other resources to processes, and provides mechanisms for process synchronization and communication.

- **Memory Management:** The operating system manages computer memory, allocating memory space to processes, and ensuring efficient utilization of available memory resources. It provides mechanisms for memory protection, virtual memory, and memory swapping to support multitasking and multi-user environments.

- **File System Management:** The operating system provides a file system that organizes and manages the storage of data on disk drives. It provides file access methods, directory structures, and file permissions to facilitate the creation, manipulation, and retrieval of files by applications.

- **Device Management:** The operating system manages hardware devices such as disk drives, printers, and network interfaces. It provides device drivers and input/output (I/O) subsystems to enable communication between applications and hardware devices, and handles device initialization, configuration, and error handling.

3. SIC & SIC/XE Architecture:

SIC (Simplified Instructional Computer):

The Simplified Instructional Computer (SIC) is a hypothetical computer architecture designed for educational purposes. It serves as a simplified model of a real-world computer system, with a small instruction set and limited addressing modes. The SIC architecture includes the following key components:

- **CPU (Central Processing Unit):** The CPU is the central processing unit of the SIC, responsible for executing instructions and performing arithmetic and logical operations. It consists of an arithmetic and logic unit (ALU), control unit, and registers.

- **Memory:** The SIC architecture includes a main memory unit for storing program instructions and data. The size of the memory is limited to 32,768 bytes (or 32

K), with each byte addressable by a unique memory location.

- **Instruction Set:** The SIC instruction set consists of basic instructions for arithmetic, data transfer, and control operations. Instructions are represented using mnemonic codes such as ADD, SUB, and JUMP, and are encoded as binary values for execution by the CPU.

- **Addressing Modes:** The SIC architecture supports three addressing modes: direct addressing, indirect addressing, and immediate addressing. These addressing modes determine how operands are specified in instruction operands are specified in instructions and how memory addresses are calculated during execution.

- **Input/Output:** The SIC architecture includes input/output (I/O) instructions for performing input and output operations with external devices such as keyboards, printers, and disk drives. These instructions allow data to be transferred between the CPU and external devices via predefined I/O channels.

SIC/XE (Extended SIC):

The Extended SIC (SIC/XE) architecture extends the original SIC architecture with additional features and capabilities for practical use. It addresses some of the limitations of the original SIC architecture, such as limited memory size and addressing modes, and introduces enhancements to support larger programs and more complex computations. Here are some key features of the SIC/XE architecture:

- **Expanded Memory:** The SIC/XE architecture increases the size of the main memory to 1 megabyte (or 1M), allowing larger programs to be executed and more data to be stored in memory.

- **Extended Instruction Set:** The SIC/XE instruction set includes additional instructions for arithmetic, logical, and control operations, providing more flexibility and functionality to programmers.

- **Enhanced Addressing Modes:** The SIC/XE architecture introduces new addressing modes, such as base-relative addressing and indexed addressing, which allow for more efficient memory access and manipulation of data structures.

- **Extended Registers:** The SIC/XE architecture provides additional general-purpose registers, allowing more data to be stored and manipulated directly within the CPU. This reduces the need for memory accesses and improves performance.

- **Enhanced Input/Output:** The SIC/XE architecture enhances the input/output (I/O) capabilities with support for direct memory access (DMA) and interrupt-driven I/O, improving the efficiency and responsiveness of I/O operations.

- **Compatibility:** Despite the enhancements, the SIC/XE architecture maintains backward compatibility with the original SIC architecture, allowing programs written for SIC to run on SIC/XE systems without modification.

4. Addressing Modes:

Addressing modes are mechanisms used by the CPU to specify the operand(s) of an instruction, i.e., the data on which the operation is to be performed or the address of the data. Different addressing modes provide flexibility and efficiency in accessing operands and are used based on the requirements of the instruction and the programmer's preferences. Here are some common addressing modes:

- **Direct Addressing:** In direct addressing mode, the operand is specified by a memory address directly encoded in the instruction. The CPU accesses the data stored at the specified memory location and performs the operation.

- **Indirect Addressing:** In indirect addressing mode, the operand is not directly specified in the instruction but is instead stored in a memory location whose address is encoded in the instruction. The CPU first fetches the operand address from memory and then accesses the data at that address.

- **Immediate Addressing:** In immediate addressing mode, the operand is a constant value embedded within the instruction itself. The CPU does not need to fetch the operand from memory; it simply uses the value provided in the instruction.

- **Indexed Addressing:** In indexed addressing mode, the operand is specified as the sum of a base address and an index register value. This mode is commonly used for accessing elements of arrays or data structures, where the base address points to the start of the data structure, and the index register specifies the offset or index of the desired element.

- **Base-Relative Addressing:** Base-relative addressing mode is similar to indexed addressing, but instead of using an index register, it uses a base register to specify the base address. The operand is calculated as the sum of the base address and a displacement value provided in the instruction.

- **Relative Addressing:** Relative addressing mode is used for branching and control transfer instructions. The operand is specified as a relative offset or displacement from the current instruction's address. The CPU calculates the target address by adding the displacement to the address of the current instruction.

These addressing modes provide flexibility in accessing operands and allow programmers to write more efficient and concise code by choosing the most suitable mode for each instruction.

5. SIC & SIC/XE Instruction Set:

SIC Instruction Set:

The SIC instruction set consists of basic instructions for performing arithmetic, data transfer, and control operations. Instructions are represented using mnemonic codes, which are human-readable abbreviations of the corresponding machine instructions. Here are some common instructions in the SIC instruction set:

- **ADD:** Add two operands and store the result in a register.
- **SUB:** Subtract one operand from another and store the result in a register.
- **MUL:** Multiply two operands and store the result in a register.
- **DIV:** Divide one operand by another and store the quotient in a register.
- **LOAD:** Load a value from memory into a register.
- **STORE:** Store a value from a register into memory.
- **JUMP:** Unconditionally jump to a specified memory address.
- **JLT (Jump if Less Than):** Jump to a specified address if a condition is met (e.g., if the result of the previous operation is negative).
- **JSUB (Jump to Subroutine):** Jump to a subroutine at a specified address and save the return address.
- **RSUB (Return from Subroutine):** Return from a subroutine to the saved return address.

These instructions provide the basic building blocks for writing programs in assembly language for the SIC architecture.

SIC/XE Instruction Set:

The SIC/XE instruction set extends the original SIC instruction set with additional instructions and features to support larger programs and more complex computations. In addition to the basic arithmetic, data transfer, and control instructions, the SIC/XE instruction set includes enhanced instructions for:

- **Extended Arithmetic Operations:** Additional arithmetic operations such as multiply and divide with extended precision.
- **Extended Control Flow:** Enhanced control flow instructions for conditional branching and looping.
- **String Manipulation:** Instructions for string manipulation and comparison.
- **Bitwise Operations:** Instructions for bitwise logical operations such as AND, OR, and XOR.
- **Extended Addressing Modes:** New addressing modes such as base-relative addressing and indexed addressing for more efficient memory access.

The SIC/XE instruction set provides programmers with more flexibility and functionality to write programs for the extended SIC architecture.

6. Assembler Directives:

Assembler directives are special instructions used by the assembler to control the assembly process and provide additional information to the assembler about the structure and organization of the program.

Directives are not translated into machine code but instead instruct the assembler on how to process the source code. Here are some common assembler directives:

- **START:** Specifies the starting address of the program.
- **END:** Marks the end of the program.
- **BYTE:** Defines character or binary data.
- **WORD:** Defines integer data.
- **RESB/RESW:** Reserves space in memory for data.
- **EQU:** Defines symbolic constants and equates them to specific values.
- **ORG:** Specifies the origin or starting address for subsequent instructions

or data.
- **BASE:** Sets the base address for base-relative addressing.

These directives provide programmers with the means to organize and structure their assembly language programs and provide additional information to the assembler to generate correct machine code.

7. SIC/XE Programs:

SIC/XE programs are assembly language programs written for the SIC/XE architecture. These programs consist of a combination of instructions, data definitions, and assembler directives that define the program's behavior and structure. Here are the key components of SIC/XE programs:

- **Instructions:** SIC/XE programs include instructions for performing arithmetic, data transfer, control, and input/output operations. These instructions are written in assembly language and are translated into machine code by the assembler.

- **Data Definitions:** SIC/XE programs define data variables and constants using data definition directives such as BYTE and WORD. These directives specify the size and type of data to be allocated in memory.

- **Assembler Directives:** Assembler directives provide additional instructions to the assembler on how to process the source code and generate machine code. Directives such as START, END, and EQU are used to define the structure and organization of the program.

- **Program Structure:** SIC/XE programs typically follow a modular structure, with sections for data definitions, program code, and subroutine definitions. The program code is organized into logical units such as functions or procedures, each performing a specific task.

- **Comments:** Comments are used to document the program code and provide explanations and annotations for the benefit of programmers and other readers. Comments are ignored by the assembler and have no effect on the generated machine code.

SIC/XE programs are written using the assembly language syntax and are translated into machine code using an assembler. The resulting machine code can then be executed on a computer system that supports the SIC/XE architecture.