1. Introduction to Assembly Language:

Assembly language is a low-level programming language that communicates directly with the processor. It involves writing instructions that the processor can execute. Each instruction corresponds to a machine language instruction, making assembly language specific to the hardware architecture. It provides full control over the system but requires managing details such as memory and registers.

2. Understanding the First Assembly Language Program:

The program adds three numbers using processor registers. It starts with an English explanation of each step and then translates it into assembly code. Here is the breakdown:

- *Move 5 to AX (Load number 5 into AX register)*
- *Move 10 to BX (Load 10 into BX register)*
- *Add BX to AX (Add BX to AX register)*
- *Move 15 to BX (Load 15 into BX register)*
- *Add BX to AX (Add BX to AX register)*

3. Assembly Language Syntax:

In Intel assembly language, instructions are abbreviated and follow a source-destination order. For example, the instruction format is: operation destination, source. This is more efficient and follows assembly syntax, with "mov" used instead of "move".

4. Explanation of Each Instruction:

- mov ax, 5: Loads the value 5 into AX register.
- mov bx, 10: Loads value 10 into BX register.
- add ax, bx: Adds BX to AX (AX = AX + BX).
- mov bx, 15: Loads value 15 into BX.
- add ax, bx: Adds BX to AX again, so AX becomes 30.
- mov ax, 0x4c00: Prepares to terminate the program.
- int 0x21: DOS interrupt for program termination.

5. Comments in Assembly Language:

In assembly language, comments are crucial to making code readable. They are written after a semicolon (;) and are ignored by the assembler. Comments help others (and yourself) understand the purpose of the code.

6. Understanding the Tools:

Assemblers and debuggers are tools that make assembly programming possible. NASM is used to convert assembly code into machine language, while the AFD debugger allows us to observe the program's execution step by step.

7. Memory and Opcodes:

Each assembly instruction translates into a machine opcode (binary code). Little-endian format is used by Intel, where least significant bytes are stored first in memory. For instance, the number 5 is stored as 0500 in memory.

8. Debugging the Program:

The AFD debugger steps through the program, showing how registers like AX and BX change as instructions are executed. By observing the Instruction Pointer (IP), you can follow the flow of the program. For example, after adding 5 + 10 + 15, AX holds the final value 30, which is the result of the program.

Conclusion:

This simple program illustrates the basic structure of assembly language. By understanding registers, memory, and instructions like mov and add, you can control the processor directly and perform arithmetic or more complex tasks.