3.3 Unconditional Jump

Introduction

In assembly language programming, data can be placed anywhere in the code. Unlike some high-level programming languages, there are no strict restrictions on where to define data segments. However, it's crucial to ensure that the program starts executing at a valid instruction.

Problem with Data Placement

- In the example given, if you place data at the start of the code (where the program execution is supposed to begin), the debugger will attempt to execute that data as if it were instructions.

- According to the COM file definition, the first executable instruction should be located at offset `0100`. If data is placed there instead, the debugger will misinterpret this data as opcodes (machine instructions) and will fail to execute correctly.

Unconditional Jump (JMP)

To manage the flow of execution and ensure that the program starts at the correct point, we introduce an instruction called **`JMP`**.

- **`JMP`** stands for **unconditional jump**. This instruction causes the program to jump to a specified location in the code, regardless of the current state of any flags (like Zero Flag, Carry Flag, etc.).

- This means that when the `JMP` instruction is executed, the flow of control will always go to the target instruction specified after the jump, enabling proper execution of the program.

Example Program

The provided example illustrates how to use the `JMP` instruction effectively:

```assembly
```

```
[org 0x0100]
jmp start ; unconditionally jump over data
num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50
total: dw 0

start:
mov bx, 0 ; initialize array index to zero
mov ax, 0 ; initialize sum to zero
l1:
add ax, [num1 + bx] ; add number to ax
add bx, 2 ; advance bx to next index
cmp bx, 20 ; are we beyond the last index
jne l1 ; if not, add the next number
mov [total], ax ; write back sum in memory
mov ax, 0x4c00 ; terminate program
int 0x21
```

Key Points in the Example:

1. **JMP Instruction**:

   - The program starts with the `jmp start` instruction. This line tells the CPU to jump immediately to the label `start`, effectively skipping over the data declarations that follow.

   - This ensures that execution resumes from a valid instruction instead of trying to execute data.

2. **Data Declarations**:

   - The data segment, defined with `num1: dw 10, 20, 30, 40, 50, 10, 20, 30, 40, 50`, is now safely placed below the code.

- The `total: dw 0` declaration initializes a location to store the result of the sum.

3. **Program Logic**:

- The program initializes the array index (`bx`) and the sum (`ax`) to zero.

- It enters a loop (`l1`) where it adds values from the `num1` array to `ax`, advances the index (`bx`), and checks if it has summed ten numbers.

- Finally, it stores the total in memory and terminates the program.

Conclusion

The **unconditional jump** instruction (`JMP`) is a powerful tool in assembly language that allows programmers to control the flow of execution without relying on flags. By strategically placing jumps, developers can navigate around data declarations and ensure that their programs run smoothly. This capability is vital for constructing well-structured assembly programs that handle data efficiently while maintaining logical flow.