3.4 Relative Addressing

Understanding Relative Addressing

Relative addressing is a method of addressing in assembly language that refers to a location relative to the current instruction pointer (IP) rather than specifying an exact address in memory. This contrasts with absolute addressing, where a specific memory address is used directly.

Example of a JMP Instruction

In the context of a **JMP** instruction, when we see something like `JMP 0119`, it means that the program is instructed to jump to the instruction located at address `0119`. However, the way this jump is represented in machine code involves using relative addressing.

- For instance, the opcode might be shown as **`F21600`**, where:

  - **`F2`** is the opcode that indicates the operation (in this case, an unconditional jump).

  - **`1600`** is the operand, which specifies how far to jump from the current position.

Position Relative Addressing

- In relative addressing, the operand (`1600`) represents an offset rather than an exact address.

- This offset is added to the current value of the instruction pointer (IP) to determine the target address.

For example:

- If the instruction is executed at address `0100`, and the IP is pointing to the next instruction (let's say at `0103`), adding the relative offset `0016` (which is `16` in decimal) to the current IP means:

  \[

\text{New IP} = 0103 + 0016 = 0119

\]

Advantages of Relative Addressing

1. **Flexibility**: Relative addressing allows the code to be more flexible, as the exact addresses can change when the program is loaded into memory. As long as the offsets are correct, the instructions will still point to the correct locations.

2. **Easier Code Modification**: If you insert or remove code, the relative addresses automatically adjust based on the new positions of instructions, reducing the need for manual updates to absolute addresses.

3. **Use of Labels**: When using relative addressing, programmers can simply use labels (like `start`, `end`, etc.) with jump instructions. The assembler takes care of calculating the correct offsets during assembly. This makes the code more readable and easier to maintain.

Conclusion

Relative addressing is a vital concept in assembly language programming that allows for more dynamic and flexible code execution. By specifying offsets rather than exact addresses, it simplifies the development process and makes the code more adaptable to changes in the program structure. The use of labels with instructions like `JMP` ensures that the intended logic is executed correctly, regardless of where the program is loaded in memory.

If you have any further questions or need additional details, feel free to ask!