Here’s a detailed explanation of **3.5 Types of Jump** in assembly language, broken down into simpler terms:

3.5 Types of Jump

In assembly programming, there are three main types of jumps: **near**, **short**, and **far**. These types differ in their instruction sizes and the ranges of memory addresses they can jump to. Let’s break down each type:

1. Near Jump

- A **near jump** uses a relative address that is stored in **16 bits** (2 bytes).

- This type of jump allows the program to jump anywhere within the same segment of memory.

- If the offset exceeds the bounds of the segment, it will **wrap around** to the lower part of the segment.

- For instance, if you jump beyond the maximum addressable range in a segment, the address will circle back to the start of that segment.

- A negative offset in this context is interpreted as a large number due to this wraparound behavior, effectively allowing backward jumps.

Instruction Format:

- Example: `EB Disp`

  - `EB` is the opcode for the near jump.

  - `Disp` indicates the displacement (offset) from the current instruction pointer (IP).

2. Short Jump

- A **short jump** uses a **single byte** (8 bits) to store the offset.

- This jump can be represented by instructions like `75F2`, where `75` is the opcode, and `F2` is the operand.

- The operand is treated as a signed byte:

- If the byte is **positive**, it adds to the current IP.

- If the byte is **negative**, it subtracts from the current IP by using two's complement representation.

- The range for a short jump is limited:

- It can jump **+127 bytes forward** or **-128 bytes backward** within the code.

Instruction Format:

- Example: `75 Disp`

- `75` is the opcode for a conditional short jump.

- `Disp` is the signed offset.

3. Far Jump

- A **far jump** is not position-relative but is instead **absolute**.

- It requires both a **segment** and an **offset** to determine the jump target.

- Far jumps are used when you need to jump from one code segment to another, which cannot be achieved using near or short jumps.

- The instruction for a far jump loads the **Code Segment (CS)** register with the segment part and the **Instruction Pointer (IP)** with the offset part.

- Execution resumes from the specified physical memory location, which may be in a different segment than where the jump originated.

Instruction Format:

- Example: `JMP [segment:offset]`

- Both segment and offset must be specified in the instruction.

Summary

- **Near Jump**: Uses a 16-bit relative address, can jump anywhere within the current segment, and can wrap around if the address exceeds the segment's limits.

- **Short Jump**: Uses an 8-bit signed offset, allowing jumps of +127 or -128 bytes, and is primarily used for conditional jumps.

- **Far Jump**: Uses both a segment and an offset to perform absolute jumps, enabling jumps across different segments in memory.

Conclusion

Understanding the differences between near, short, and far jumps is essential for effective assembly programming. Each type serves specific purposes and has unique limitations and capabilities regarding how program flow can be controlled. This knowledge allows programmers to write more efficient and flexible assembly code. If you have any further questions or need additional clarification, feel free to ask!