**Computer Organization & Assembly Language**

**Lab 06**

**Topics:**

1. **AND instruction**
2. **OR instruction**
3. **NOT instruction**
4. **TEST instruction**

**AND Instruction:**

**Definition:** The AND instruction is a bitwise logical operation that performs a bitwise AND operation between the bits of two operands, resulting in a new value where each bit is set to 1 only if both corresponding bits in the operands are 1.

**Explanation:** The AND instruction is used to manipulate individual bits within data. It takes two operands, the source, and the destination, and performs a bitwise AND operation. The result is stored in the destination operand.

**Examples:**

**Example 1:** Let's say we have two 8-bit binary numbers, A = 11001100 and B = 10101010. When we perform the AND operation between them, we get the result C = 10001000.

**Example 2:** Consider another example with A = 11110000 and B = 11001100. When we apply the AND operation, the result is D = 11000000.

**Example 3:** Clearing bits in a register or memory location. For example, to clear the high-order bits of a register, you could AND it with a mask of all 0s.

; Clear the high-order bits of the AL register.

**and al, 0fh**

**OR Instruction:**

**Definition:** The OR instruction is a bitwise logical operation that performs a bitwise OR operation between the bits of two operands, resulting in a new value where each bit is set to 1 if at least one of the corresponding bits in the operands is 1.

**Explanation:** The OR instruction is used to combine individual bits within data. It takes two operands, the source, and the destination, and performs a bitwise OR operation. The result is stored in the destination operand.

**Examples:**

**Example 1:** Let's say we have two 8-bit binary numbers, A = 11001100 and B = 10101010. When we perform the OR operation between them, we get the result C = 11101110.

**Example 2:** Consider another example with A = 11110000 and B = 11001100. When we apply the OR operation, the result is D = 11111100.

**Example 3:** Setting bits in a register or memory location. For example, to set the high-order bits of a register, you could OR it with a mask of all 1s.

; Set the high-order bits of the AL register.

**or al, 0xf0**

**NOT Instruction:**

**Definition:** The NOT instruction is a bitwise logical operation that performs a bitwise negation operation on each bit of the operand, flipping 0s to 1s and 1s to 0s.

**Explanation:** The NOT instruction is used to invert or complement individual bits within data. It takes a single operand and negates all its bits, creating a new value.

**Examples:**

**Example 1:** Let's say we have an 8-bit binary number A = 11001100. When we perform the NOT operation on it, we get the result B = 00110011.

**Example 2:** Consider another example with an 8-bit binary number C = 11110000. When we apply the NOT operation, the result is D = 00001111.

**Example 3:**

; Invert the value of the AL register.

not al

; Create a mask of all 0s.

mov ax, 0xffff

not ax

**TEST Instruction:**

**Definition:** The TEST instruction is a bitwise logical operation that performs a bitwise AND operation between the bits of two operands, similar to the AND instruction. However, it does not store the result but only sets the condition flags in the CPU based on the result.

**Explanation:** The TEST instruction is used to check the bits of two operands without changing them. It takes two operands, the source, and the destination, and performs a bitwise AND operation. The result is not stored, but the condition flags (such as zero flag) are updated based on the result.

**Examples:**

**Example 1:** Let's say we have two 8-bit binary numbers, A = 11001100 and B = 10101010. When we perform the TEST operation between them, the condition flags are updated based on the result without storing it.

**Example 2:** Consider another example with A = 11110000 and B = 11001100. The TEST operation updates the condition flags based on the bitwise AND result.