College of Engineering and Technology
Department of Electrical and Computer Engineering,
Wolkite University

8086 Assembly Language Lab Manual (Emu8086 Based)

Course: Microprocessor and Assembly Language

Platform: emu8086 Emulator

Lab #3: Logical Instructions

Prepared By: Abebe Birhanu (MSc.)

Date: December 2, 2025

Wolkite, Ethiopia

LAB 3— Logical Instructions

Objective: To study and perform logical operations (bitwise and Boolean) using 8086 logical instructions in the emu8086 environment, and to observe the effects on registers and flags.

Background Theory:

1. Overview

Logical instructions operate bit-by-bit on data. They are used for masking, comparison, testing specific bits, and performing bitwise operations. These instructions are non-arithmetic — they manipulate bits without affecting numerical magnitude.

2. Logical Instruction Set of 8086

Category	Instructions	Description
AND Operations	AND, TEST	Bitwise AND of operands
OR Operations	OR	Bitwise OR of operands
Exclusive OR	XOR	Bitwise XOR (toggle bits)
Complement	NOT	Bitwise negation (inversion)
Shift/Rotate	(done later in LAB 4)	Shift and rotate bits

3. Effects on Flags

Instruction	CF	OF	ZF	SF	PF
AND	0	0	✓	✓	✓
OR	0	0	✓	✓	✓
XOR	0	0	✓	✓	✓
NOT	Unaffected	Unaffected	Unaffected	Unaffected	Unaffected
TEST	0	0	✓	✓	✓

4. Typical Uses

- **AND:** Masking bits (clearing certain bits)
- **OR:** Setting bits
- **XOR:** Toggling bits (used for encryption, swapping)
- **NOT:** Inverting all bits
- **TEST:** Bit testing without changing the operand

Example Programs

All examples can be directly assembled and run in **emu8086**.

Example 1: AND Instruction

```
org 100h
mov al, 0F3h   ; AL = 11110011b
mov bl, 0A7h   ; BL = 10100111b
and al, bl
hlt
```

Explanation:

Performs bitwise AND:

AL = 11110011
BL = 10100111

AL = 10100011 (A3h)

Register	Before	After
AL	F3h	A3h
BL	A7h	A7h

Flags affected: ZF=0, SF=1, PF=1

Example 2: OR Instruction

```
org 100h
mov al, 0F0h   ; 11110000b
mov bl, 00Fh   ; 00001111b
or al, bl
hlt
```

Result:

11110000
00001111

11111111 (FFh)

Register	Result
AL	FFh

Example 3: XOR Instruction (Bit Toggle)

```
org 100h
mov al, 0AAh   ; 10101010b
mov bl, 0FFh   ; 11111111b
xor al, bl
hlt
```

Result:

10101010 XOR 11111111 = 01010101 (55h)

Tip: XOR is used to invert bits or swap values without a temporary register.

Example 4: XOR for Swapping Two Registers (without using MOV)

```
org 100h
mov al, 12h
mov bl, 34h
xor al, bl
xor bl, al
xor al, bl
hlt
```

Before	After
AL=12h, BL=34h	AL=34h, BL=12h

Example 5: NOT Instruction (One's Complement)

```
org 100h
mov al, 0F0h
not al
hlt
```

Result:

0F0h → 0F0h = 11110000 → NOT → 00001111 (0Fh)

Register	Before	After
AL	F0h	0Fh

Example 6: TEST Instruction

```
org 100h
mov al, 0A5h   ; 10100101b
mov bl, 80h    ; 10000000b
test al, bl
hlt
```

Explanation:

Performs bitwise AND but **does not store result**.

ZF = 0 (since bit 7 is set in AL).

💡 Used for checking whether a specific bit is 1 or 0.

Example 7: Combining Logical Operations

```
org 100h
mov al, 0C3h  ; 11000011b
and al, 0F0h  ; Mask lower nibble
or  al, 05h   ; Set last 2 bits
hlt
```

Result:

After AND → AL = C0h

After OR → AL = C5h

Observation Table

No	Instruction	Operation	Example	Result	Flags Affected
1	AND	Bitwise AND	F3h AND A7h	A3h	ZF,SF,PF
2	OR	Bitwise OR	F0h OR 0Fh	FFh	ZF,SF,PF
3	XOR	Bitwise XOR	AAh XOR FFh	55h	ZF,SF,PF
4	NOT	Bitwise NOT	F0h → 0Fh	0Fh	None
5	TEST	Bitwise AND (no store)	A5h, 80h	Flags set only	ZF,SF,PF
6	XOR Swap	Data swap	12h↔34h	Swapped	ZF,SF,PF

Exercises

1. Write an assembly program to mask the **upper nibble** of AL and keep the lower nibble unchanged.
2. Write a program to **set the lowest 3 bits** of a number using OR.
3. Demonstrate **bit toggling** using XOR on any 8-bit register.
4. Write a program using **TEST** to check whether bit 5 of AL is set or not.
5. Combine **AND**, **OR**, and **NOT** to convert an input byte into its inverted lower nibble.
6. Implement XOR swapping for 16-bit registers (AX and BX).
7. Write a program to **clear AL** only if its most significant bit (bit 7) is 1.
8. Perform AND & OR between two memory operands using `MOV` & `AND` instructions indirectly.

Viva Questions

1. What is the difference between `AND` and `TEST` instructions?
2. How does `XOR` differ from `OR` in functionality?
3. Which logical instruction can be used to clear a specific bit?
4. Explain how `NOT` instruction works.
5. Why doesn't `NOT` affect any flags?
6. What happens to the carry and overflow flags during logical operations?
7. How can `XOR` be used for data encryption?
8. How is `AND` used for masking bits? Give an example.
9. Can you swap two registers using only `XOR`? Explain the process.
10. What is the significance of the **TEST** instruction in bit testing?

Summary

In this lab, you learned:

- How **logical instructions** manipulate data at the bit level.
- How AND, OR, XOR, NOT, and TEST affect processor flags.
- How to perform **masking, setting, and toggling** bits.
- How to use logical operations for **data manipulation, testing, and conditional branching**.

These concepts are crucial for understanding bitwise operations, conditional logic, and low-level control in microprocessor programming.