# 8086 Microprocessor Laboratory Experiments

## Experiment 6: Logical, Shift and Rotate Instructions

Murtadha Hssayeni, Ph.D.

m.hssayeni@uobabylon.edu.iq

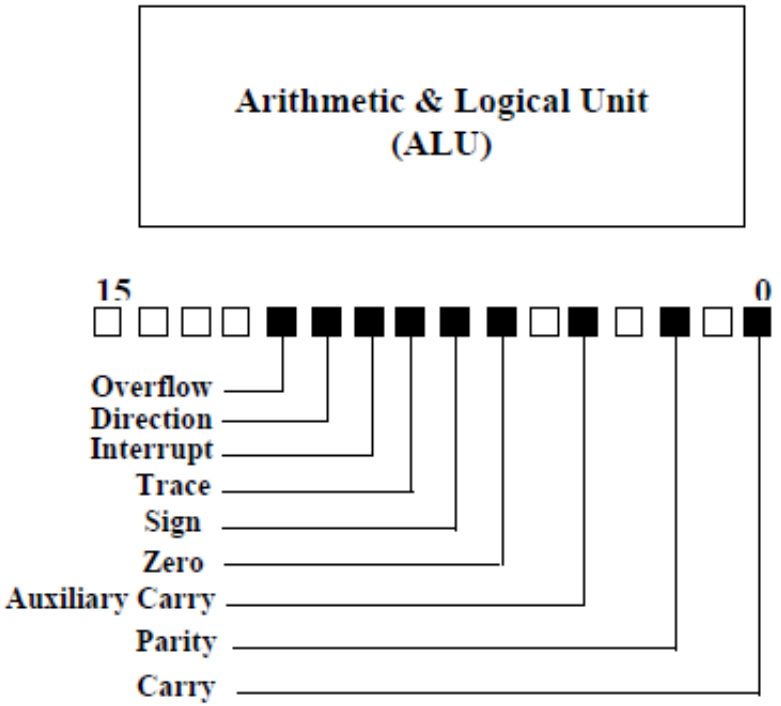

# Outlines

- Logical Instructions
  - AND, OR, XOR, CMP and TEST
  - NOT, NEG
- SHIFTS
- ROTATE
- Procedure and Discussion

# Logical Instructions

- Most Logic Instructions affect the processor status register (or **Flags**).

- **There are 2 groups of instructions:**
  - AND, OR, XOR, CMP and TEST.
    - Two-address instructions
  - NOT, NEG.



# First group: AND, OR, XOR, CMP and TEST

- These types of operands are supported:
  - REG, memory
  - memory, REG
  - REG, REG
  - memory, immediate
  - REG, immediate
  - REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
  - memory: [BX], [BX+SI+7], variable, etc.
  - Immediate: 5, -24, 3Fh, 10001101b, etc.
- After operation between operands, result is always stored in **first** operand.
- **CMP** and **TEST** instructions affect flags only and do not store a result.
- These instructions affect these flags only: CF, ZF, SF, OF, PF, AF.

# Logical Instructions: AND

- Logical AND between all bits of two operands.
- AND destination, source
  - The destination operand can be a register or in memory.
  - The source operand can be a register, in memory, or immediate.
- First operand = First operand AND second operand.
- **Example**:

*MOV AL, 'a' ; AL = 01100001b*

*AND AL, 11011111b ; AL = 01000001b ('A')*

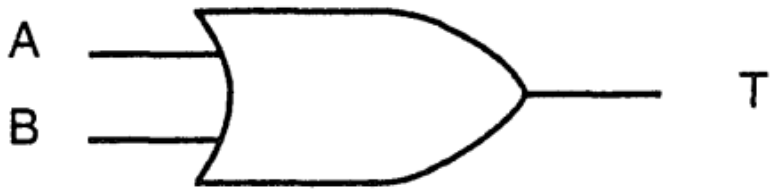

| X | Y | X AND Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Logical Instructions: OR

- Logical OR between all bits of two operands.
- OR destination, source
  - The destination operand can be a register or in memory.
  - The source operand can be a register, in memory, or immediate.
- First operand = First operand OR Second operand.
- **Example**:

*MOV AL, 'A' ; AL = 01000001b*

*OR AL, 00100000b ; AL = 01100001b ('a')*




| X | Y | X OR Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# Logical Instructions: XOR

- Logical Exclusive OR (XOR) between all bits of two operands.
- XOR destination, source
- First operand = First operand XOR Second operand.
- XOR sets the result bits to 1 if the two operands are not equal; otherwise, they are reset to 0.
- **Example**:

*MOV AL, 00000111b*

*XOR AL, 00000010b ; AL = 00000101b*


| X | Y | X XOR Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Logical Instructions: XOR

- CF = 0 and OF = 0 are set internally and the rest are changed according to the result of the operation.
- XOR can be used to see if two registers have the same value. XOR BX, CX will make ZF = 1 if both registers have the same value.
- The XOR can also be used to clear the contents of a register by XORing it with itself.

- **Example**:
- Show how "XOR AH, AH" clears AH, assuming that AH = 45H

**Solution:**

| | |
|---|---|
| 45H | 01000101 |
| 45H | 01000101 |
| 00 | 00000000 |

- Flag setting will be: SF = 0, ZF = 1, PF = 1, CF = OF = 0.

# Logical Instructions: CMP

- Subtract second operand from first for **flags** only.
- CMP destination, source
- Compare Algorithm: operand1 - operand2
- The CMP compares two operands and changes the flags according to the result of the comparison.
- **The operands remain unchanged.**
- Although all the CF, AF, SF, PF, ZF, and OF flags reflect the result, only the CF and ZF are used.

**Flag Settings for Compare Instruction**

| Compare operands | CF | ZF |
|---|---|---|
| destination > source | 0 | 0 |
| destination = source | 0 | 1 |
| destination < source | 1 | 0 |

- **Example**:

*MOV AL, 5*
*MOV BL, 5*
*CMP AL, BL ; AL = 5, ZF = 1 (so equal!)*

# Logical Instructions: TEST

- The same as AND but for **flags** only.
- These flags are affected: ZF, SF, PF.
- Result is not stored anywhere.
- **Example**:

*MOV AL, 00000101b*

*TEST AL, 1 ; ZF = 0*

*TEST AL, 10b ; ZF = 1 means the second bit is 0.*

| *Assembly Language* | *Operation* |
|---|---|
| TEST DL,DH | DL is ANDed with DH |
| TEST CX,BX | CX is ANDed with BX |
| TEST AH,4 | AH is ANDed with 4 |

# Second group: NOT, NEG

- These types of operands are supported:
  - REG
  - memory
  - REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
  - memory: [BX], [BX+SI+7], variable, etc.
- Logical inversion or the one's complement **(NOT)** instruction does not affect any flags.
- Arithmetic sign inversion or the two's complement **(NEG)** instruction affects these flags only: CF, ZF, SF, OF, PF, AF.

| *Assembly Language* | *Operation* |
|---|---|
| NOT CH | CH is one's complemented |
| NEG CH | CH is two's complemented |
| NEG AX | AX is two's complemented |

- **Example**:

*MOV AL, 00011011b*

*NOT AL ; AL = 11100100b*

- **Example**:

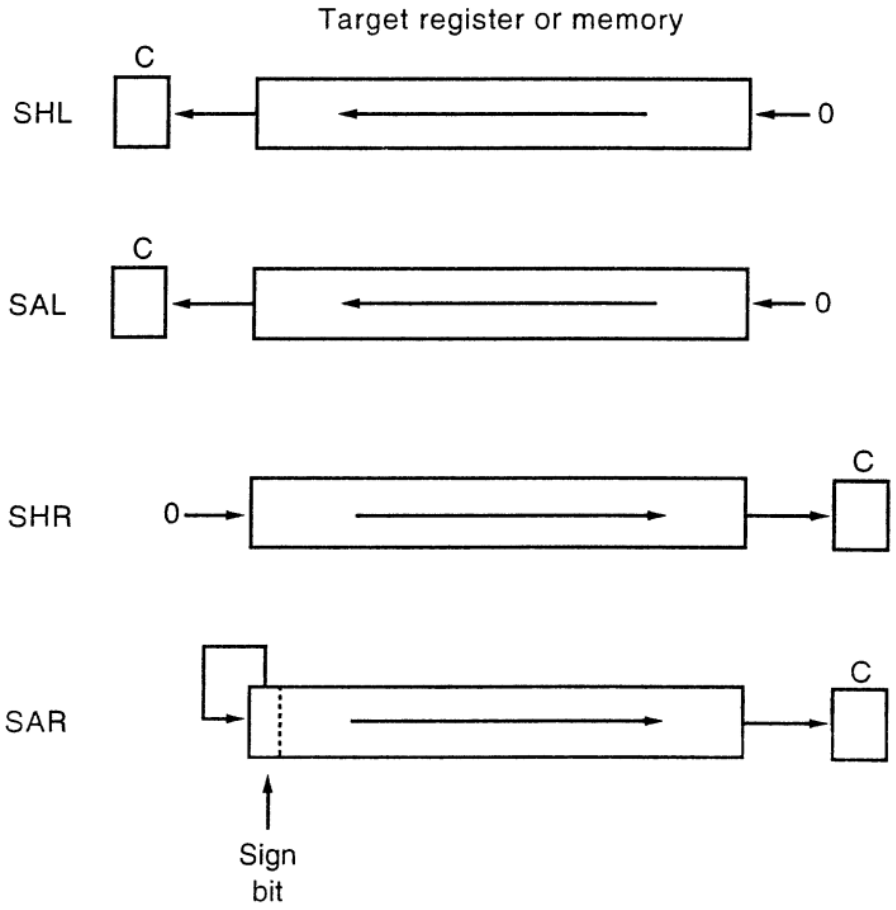
*MOV AL, 5 ; AL = 05h*

*NEG AL ; AL = 0FBh (-5)*

*NEG AL ; AL = 05h (5)*

# SHIFTS

- Shift instructions position or move numbers to the left or right within a register or memory location.
- The microprocessor's instruction set contains four different shift instructions: **two are logical shifts** and **two are arithmetic shifts**.
  - The logical shifts move a 0 into the rightmost bit position for a **logical left shift**.
  - The logical shifts move a 0 into the leftmost bit position for a **logical right shift**.
  - The **arithmetic and logical left shifts are** identical. While the right shifts are different because the arithmetic right shift copies the sign-bit through the number.



# SHIFTS

- Logical shift operations function with unsigned numbers.
- Arithmetic shifts function with signed numbers.
- The number of times (or bits) that the operand is shifted can be specified directly if it is once only, or through **the CL register if it is more than once.**

```
MOV   BX,0FFFFH          ;BX=FFFFH
SHR   BX,1               ;shift right BX once only
```

- **BX = 7FFFH and CF = 1**

**Example**

Show the result of SHR in the following:

```
MOV   AL,9AH
MOV   CL,3        ;set number of times to shift
SHR   AL,CL
```

**Solution:**

```
9AH = 10011010
      01001101    CF=0   (shifted once)
      00100110    CF=1   (shifted twice)
      00010011    CF=0   (shifted three times)
```

After three times of shifting right, AL = 13H and CF = 0.

# SHIFTS

- Immediate addressing mode is not allowed for shift instructions.
  - For example, “SHR CL, 25” is not allowed.
- **Example**:

**Example**

```
MOV    AL,-10      ;AL=-10=F6H=1111 0110
SAR    AL,1        ;AL is shifted right arithmetic once
                   ;AL=1111 1011=FDH=-5
```

Show the effects of SHL in the following:

```
MOV    DH,6
MOV    CL,4
SHL    DH,CL
```

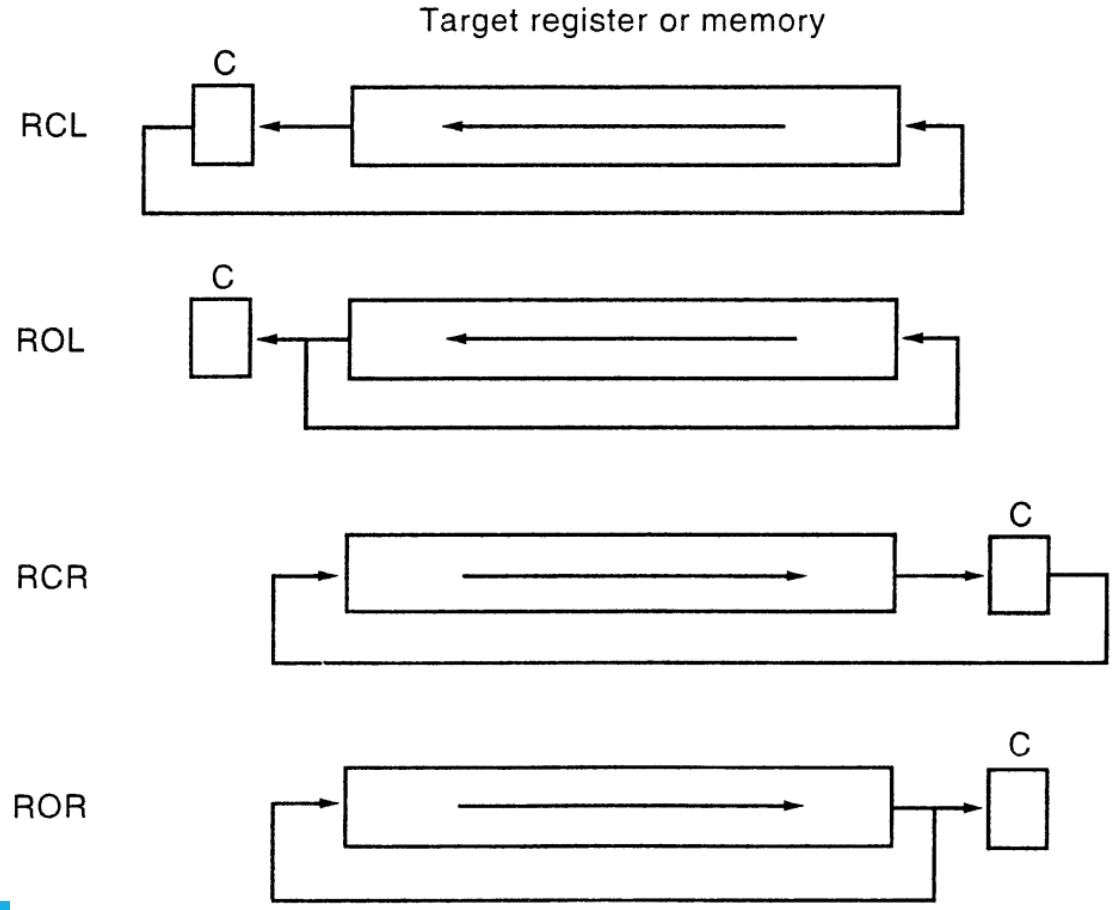
**Solution:**

```
                00000110
CF=0            00001100        (shifted left once)
CF=0            00011000
CF=0            00110000
CF=0            01100000        (shifted four times)
```

After the four shifts left, the DH register has 60H and CF = 0.

# ROTATE

- Rotate instructions position binary data by rotating the information in a register or memory location **either from one end to another or through the carry flag**.
- With either type of rotate instruction, it can be either a left or a right rotate.
- Addressing modes used with rotate are the same as used with the shifts.
- A rotate count can be immediate or located in register CL.



# Procedure

1. Write 8086 program to calculate the result of the **logical** expression:
   - X=(A ⊕ B)+AB
   - A = 1010110101011000B, B = 0000000011111111B
2. Execute above program and write down the results.
3. Store the result into memory location 47511H.
   - Assume the offset is 0011H.
4. Repeat the previous steps to solve and save the results of the following **logical** expression:
   - X=(A B)+ (3 A)
   - A = 2FH, B = 20H

# Discussion

1. How can use XOR for encryption. Give a simple example.
2. Write 8086 program to evaluate the logical expression:
   - X = A B + A C + (B ⊕ C).
   - A = 5FH, B = 10H, C = 10010010B.
3. Execute above program and write down the results.
4. Is it possible to multiply A=15H by 2 using a shift operation? If yes, perform the operation in an assembly language and store the results in a memory location of your choice.
5. In cryptography, RC5 is a symmetric-key block cipher notable for its simplicity. What operation that you studied in this lecture is applicable in RC5.