# **Experiment #4**

# **Shift and Rotate Instructions**

### 4.0 Objectives:

The objective of this experiment is to write programs demonstrating the applications of Shift and Rotate instructions.

In this experiment, you will do the following:

- Learn to use Shift and Rotate instructions
- Write programs demonstrating the applications of Shift/Rotate instructions
- Execute programs using Turbo Debug and TASM

### **4.1 Introduction:**

#### **Shift Instructions**

The 8086 can perform two types of Shift operations; the *logical* shift and the *arithmetic* shift. There are four shift operations (SHL, SAL, SHR, and SAR).

| Mnemonic | Meaning                | Format       | Allowed operands |       |  |  |
|----------|------------------------|--------------|------------------|-------|--|--|
| SAL      | Shift Arithmetic Left  | SAL D, count |                  | ~     |  |  |
| 0111     |                        |              | Destination(D)   | Count |  |  |
| SHL      | Shift Logical Left     | SHL D, count | Register         | 1     |  |  |
| SAL      | Shift Arithmetic Right | SAR D, count | Register         | CL    |  |  |
|          |                        | ,            | Memory           | 1     |  |  |
| SHL      | Shift Logical Right    | SHR D, count | Memory           | CL    |  |  |

If the source operand is specified as CL instead of 1, then the count in this register represents the number of bit positions the contents of the operand are to be shifted. This permits the count to be defined under software control and allows a range of shifts from 1 to 255 bits.

A logical shift fills the newly created bit position with zero:



An arithmetic shift fills the newly created bit position with a copy of the number's sign bit.



The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.



Shifting left 1 bit multiplies a number by 2 and shifting left *n* bits multiplies the operand by  $2^n$ . For example:

| MOV BL, 5 | Before: | Before: 00000101 |        |  |  |
|-----------|---------|------------------|--------|--|--|
| SHL BL, 1 | After:  | 00001010         | ] = 10 |  |  |

The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.



Shifting right 1 bit divides a number by 2 and shifting right *n* bits divides the operand by  $2^{n}$ .

#### For example:



SAL is **identical** to SHL. SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand. An arithmetic shift preserves the number's sign.



For example:

$$\begin{array}{|c|c|c|c|c|}\hline MOV & BL, -40 \\ SAR & BL, 1 \\ \hline BL = -20 \\ \hline \end{array}$$

#### **Rotate Instructions**

The 8086 can perform two types of rotate operations; the *rotate* without carry and the *rotate* through carry. There are four rotate operations (ROL, ROR, RCL, and RCR).

| Mnemonic | Meaning                    | Format       | Allowed oper   | ands  |
|----------|----------------------------|--------------|----------------|-------|
| ROL      | Rotate Left                | ROL D, count |                |       |
| DOD      | D. J. D. L.                | DODD         | Destination(D) | Count |
| ROR      | Rotate Right               | ROR D, count | Register       | 1     |
| RCL      | Rotate Left through carry  | RCL D, count | Register       | CL    |
| DCD      | D + D + 1 = 1              |              | Memory         | 1     |
| KCR      | Kotate Kight through carry | KCK D, count | Memory         | CL    |

ROL shifts each bit of a register to the left. The highest bit is copied into both the Carry flag and into the lowest bit of the register. No bits are lost in the process.



For example:

| MOV AL,11100010B<br>ROL AL,1 | ; AL = 11000101B |
|------------------------------|------------------|
| MOV BL,0A5H                  |                  |
| MOV CL, 4                    |                  |
| ROL BL, CL                   | ; $BL = 5AH$     |

ROR shifts each bit of a register to the right. The lowest bit is copied into both the Carry flag and into the highest bit of the register. No bits are lost in the process.



For example:



RCL (rotate carry left) shifts each bit to the left. It copies the Carry Flag to the least significant bit and copies the most significant bit to the Carry flag.



#### For example:

| CLC        | ; clear carry flag, $CF = 0$ |
|------------|------------------------------|
| MOV BL,A4H | ; CF = 0, BL = 10100100B     |
| RCL BL,1   | ; CF = 1, BL = 01001000B     |
| RCL BL,1   | ; CF = 0, BL = 10010001B     |

RCR (rotate carry right) shifts each bit to the right. It copies the Carry Flag to the most significant bit and copies the least significant bit to the Carry flag.



#### For example:

| STC        | ; set carry flag, CF = 1 |
|------------|--------------------------|
| MOV AH,14H | ; CF = 1, AH = 00010100B |
| RCR AH,1   | ; CF = 0, AH = 10001010B |

### 4.2 Pre-lab:

Run the following instructions in Turbo Debugger and fill the corresponding column for each Shift or Rotate instruction.

NOTE: Include the status of flags before and after the execution of shift and rotate instructions in Table 1.

- 1. MOV AL, 6BH SHR AL,1 SHL AL,3
- 2. MOV AX, 0AAAAH MOV CL,8 SHL AX,CL
- 3. MOV AL, 8CH MOV CL,3 SAR AL,CL
- 4. MOV DI, 1000H MOV [DI], 0AAH MOV CL,3 SHL BYTE PTR [DI],CL
- 5. MOV AL, 6BH ROR AL,1 ROL AL,3
- 6. STC MOV AL, 6BH RCR AL,3
- 7. CLC MOV DI,2000H MOV [DI],0AAH MOV CL,1 RCL BYTE PTR [DI],CL

### TABLE 1

|                         | Source                  |              | Destination             |                         | Status Flags           |        |        |        |        |        |
|-------------------------|-------------------------|--------------|-------------------------|-------------------------|------------------------|--------|--------|--------|--------|--------|
| Statement               | Register<br>/Memor<br>y | Content<br>s | Registe<br>r/Mem<br>ory | before<br>executio<br>n | after<br>executio<br>n | A<br>F | P<br>F | S<br>F | Z<br>F | C<br>F |
|                         |                         |              |                         |                         |                        |        |        |        |        |        |
| SHR AL,1                |                         |              |                         |                         |                        |        |        |        |        |        |
| SHL AL,3                |                         |              |                         |                         |                        |        |        |        |        |        |
|                         |                         |              |                         |                         |                        |        |        |        |        |        |
| SHL AX,CL               |                         |              |                         |                         |                        |        |        |        |        |        |
| SAR AL,CL               |                         |              |                         |                         |                        |        |        |        |        |        |
|                         |                         |              |                         |                         |                        |        |        |        | <br>   |        |
| SHL BYTE PTR<br>[DI],CL |                         |              |                         |                         |                        |        |        |        |        |        |
|                         |                         |              |                         |                         |                        |        |        |        |        |        |
| ROR AL,1                |                         |              |                         |                         |                        |        |        |        |        |        |
| ROL AL,3                |                         |              |                         |                         |                        |        |        |        |        |        |
|                         |                         |              |                         |                         |                        |        |        |        |        |        |
| RCR AL-3                |                         |              |                         |                         |                        |        |        |        |        |        |
|                         |                         |              |                         |                         |                        |        |        |        |        |        |
| RCL BYTE PTR            |                         |              |                         |                         |                        |        |        |        |        |        |
| [DI],CL                 |                         |              |                         |                         |                        |        |        |        |        |        |

# 4.3 Lab Work:

### Multiplication and Division using Shift instructions

We have seen earlier that the SHL instruction can be used to multiply an operand by  $2^n$  and the SHR instruction can be used to divide an operand by  $2^n$ .

The MUL and DIV instructions take much longer to execute than the Shift instructions. Therefore, when multiplying/dividing an operand by a small number it is better to use Shift instructions than to use the MUL/DIV instructions. For example MUL BL where BL = 2 takes many more clock cycles than SHL AL, 1.

In **Exercise 1, and 2**, you will write programs to multiply, and divide respectively, using shift instructions.

Write each of the programs using the TASM assembler format. Programs 1, 2, and 3 must be executed using the Turbo Debugger (TD) program. Program 4 must be directly executable from the DOS prompt.

1. Write a program to multiply AX by 27 using only Shift and Add instructions. You should not use the MUL instruction.

Recall that shifting left *n* bits multiplies the operand by  $2^n$ . If the multiplier is not an absolute power of 2, then express the multiplier as a sum of terms which are absolute powers of 2. For example, multiply AX by 7.  $(7 = 4 + 2 + 1 = 2^2 + 2^1 + 1)$ Answer = AX shifted left by 2 + AX shifted left by 1 + AX. **Note:** Only the original value of AX is used in each operation above. 2. Write a program to divide AX by 11 using Shift and Subtract instructions. You should not use the DIV instruction. Assume AX is a multiple of 11.

Recall that shifting right *n* bits divides the operand by  $2^n$ . If the divisor is not an absolute power of 2, then express the divisor as a sum of terms which are absolute powers of 2. For example, divide AX by 5.  $(5 = 4 + 1 = 2^2 + 1)$ Answer = AX shifted right by 2 - AX. **Note:** Only the original value of AX is used in each operation above.

3. Write a program to check if a byte is a Palindrome. [Hint: Use Rotate instructions]. If the byte is a Palindrome, then move AAh into BL. Otherwise move 00h in BL.

A Palindrome looks the same when seen from the left or the right.

For example, 11011011 is a Palindrome but 11010011 is not a Palindrome

4. Write a program to display the bits of a register or memory location. Use the INT 21H interrupts to display data on the display monitor.

[Hint: Use logical shift instruction to move data bit into the carry flag]

For example, if AL = 55H, then your program must display:

AL = 0 1 0 1 0 0 1 0 1