**Lab 07: Shift and Rotate Instructions**

**OBJECTIVE**

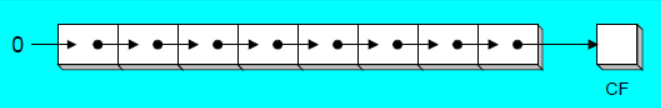
To learn the basic ‘shift and rotate’ instructions and their use.

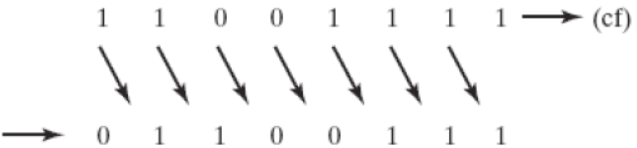
**Shift Instructions**

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

**Logical Shift:** Fills the newly created bit position with 0 (zero):

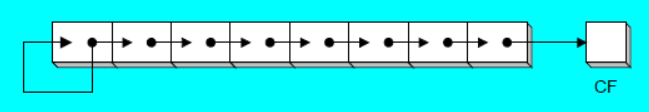
**SHR**

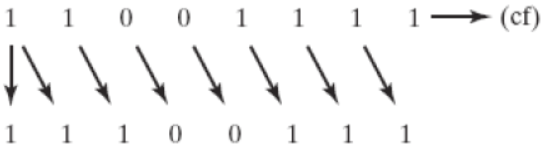
****

****

**Arithmetic Shift:** Right shift fills the newly created bit positions with a copy of the number’s sign bit (MSB):

**SAR**





‘Shifting Left’ by 1-bit **multiplies** a number by 2, and shifting left **n**-bits multiplies the operand by **2n**.

‘Shifting Right’ by 1-bit **divides** a number by 2, and shifting right by **n**-bits divides the operand by **2n**.

|  |  |  |
| --- | --- | --- |
| **Command** | **Meaning** | **Format** |
| **SAL** | Shift Arithmetic Left | SAL D, count |
| **SAR** | Shift Arithmetic Right | SAR D, count |
| **SHL** | Shift Logical Left | SHL D, count |
| **SHR** | Shift Logical Right | SHR D, count |

**Note:** Count can be an ‘immediate value’ or may be contained in ‘**CL**’ register.

If the source operand is specified as CL instead of a number, then the count in this register represents the bit positions the contents of the source 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.

|  |  |  |
| --- | --- | --- |
| **Command** | **Addressing Format** | **Description** |

|  |  |  |
| --- | --- | --- |
| **SHL** | REG, immediate REG, CL memory, immediate memory, CL | **Shift Left instruction performs a ‘Logical Left Shift’ on the destination operand, filling the lowest bit with 0(zero).**  Shifting Left 1-bit multiplies a number by 2.  **Example:**  mov al,2  **shl** al,2 **;** al = 2 \* 2**2** = 2 \* 4 = 8 ; shift left 2-bits.  **Note: SAL** (Shift Arith. Left) is identical to **SHL**. |
| **SHR** | REG, immediate REG, CL memory, immediate memory, CL | **It performs a ‘Logical Right Shift’ on the destination operand.** **The highest bit positon is filled with a 0 (zero).** See ‘SHR’ by 1-bit divides a number by 2. Fig  Shifting Right 1-bit divides a number by 2.  **Example:**  mov al,8  **shr** al,2 **;** al = 8 / 2**2**= 8 / 4 = 2 ; shift right 2-bits. |
| **SAR** | REG, immediate REG, CL memory, immediate memory, CL | **It performs a ‘Right Arithmetic Shift’ on the destination operand.** See ‘SAR’ Figure.  **Note:** An ‘Arithmetic Shift’ **preserves the no’s sign**.  **Example:**  mov al, -8  **sar** al,1 ; al = -4  mov cl,2  **sar** al,2 ; al = -1 |

**Multiplication and Division using Shift Instructions:**

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’ instruction than to use the MUL/DIV instructions. For example, ‘MUL BL’ when BL=2, takes many more clock cycles than ‘SHL BL,1’.

**Code-01: Write a program to multiply AX=2 by 7 using only Shift and ADD instruction. You should not use the MUL instruction.**

**Hint:** Recall that shifting left n-bit 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 power of 2.

For example, to 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.

**Code:**

.model small

.stack 100h

.data

.code

Main proc

Mov AX,2 ; operand to be multiplied

Mov BX,AX

Mov CX, AX

SHL BX,2 ; AX\*4 = 2 \* 4 = 8

SHL CX,1 ; AX\*2 = 2 \* 2 = 4

Add BX,CX ; 4\*AX + 2\*AX = 6\*AX = 8 + 4 = 12

Add AX,BX ; 6\*AX + 1\*AX = **7**\*AX = 12 + 2 = 14

Main endp

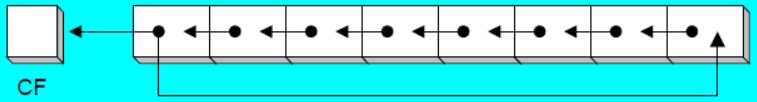
End main

**Code: To multiply AX by 36. Use the Hint 36 = 32 + 4. Also 26 = 16+8+2.**

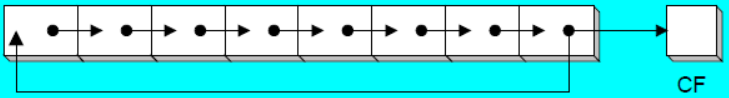
**Rotate Instructions**

The 8086 can perform two types of rotate instructions; the *rotate without carry* and the *rotate through carry*. There are four rotate operations (ROL, ROR, RCL and RCR). **Note:** In a ‘Rotate operation’, NO bits are lost.

**Rotate without Carry:** Shifts each bit to the left or the right. The highest/lowest bit is copied into both the carry flag and the highest/lowest bit.

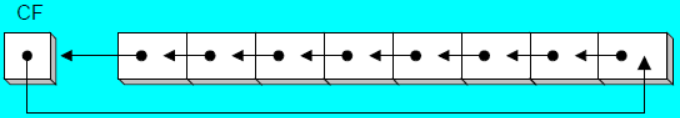
****

**ROL**

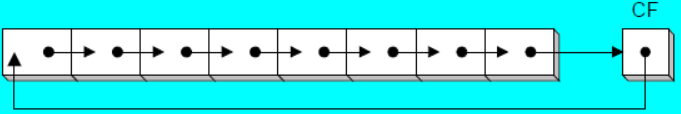
****

**ROR**

**Rotate with Carry:** Shifts each bit to the left or right. The carry flag bit (CF) is inserted into left/right bit position based on the respective operation.



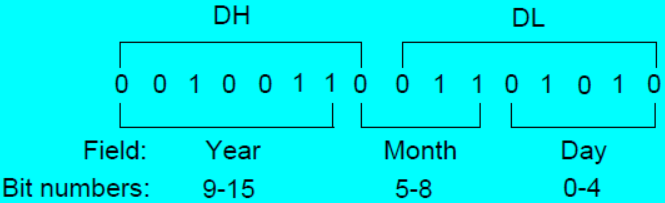
**RCL**



**RCR**

|  |  |  |
| --- | --- | --- |
| **ROL** | REG, immediate REG, CL memory, immediate memory, CL | ROL shifts each bit to the left. The highest bit (MSB) is copied into both the Carry Flag (CF) and into the lowest bit (LSB). No bits are lost. See ‘ROL’ Fig.  **Example:**  mov al, 11110000b  **rol** al,1 ; al = 1110000**1**, CF = 1. |
| **ROR** | REG, immediate REG, CL memory, immediate memory, CL | ROR shifts each bit to the right. The lowest bit (LSB) is copied into both the Carry Flag (CF) and into the highest bit (MSB). No bits are lost. See ‘ROR’ Fig. **Example:**  mov al, 11110000b  **ror** al,1 ; al = **0**1111000, CF = 0. |
| **RCL** | REG, immediate REG, CL memory, immediate memory, CL | RCL (Rotate Carry Left) shifts each bit to the left. Copies the Carry Flag (CF) to the Least Significant Bit (LSB). Copies the Most Significant Bit (MSB) to the Carry Flag (CF). See ‘RCL’ Figure.  **Example:**  **clc** ; clear carry flag, CF = 0  mov al, 88h ; CF,AL = 0 **1**0001000b  **rcl** al,1 ; CF,AL = **1** 0001000**0**b |
| **RCR** | REG, immediate REG, CL memory, immediate memory, CL | RCR (Rotate Carry Right) shifts each bit to the right. Copies the Carry Flag (CF) to the Most Significant Bit (MSB). Copies the Least Significant Bit (LSB) to the Carry Flag (CF). See ‘RCR’ Figure.  **Example:**  **stc** ; set carry flag, CF = 1  mov al, 10h ; CF,AL = **1** 0001000**0**b  **rcr** al,1 ; CF,AL = **0** **1**0001000b |

**Code-02: The MS-DOS file Date field packs the ‘year, month, day’ into 16-bits. Isolate the ‘month’ field from the bit stream given below. [2Ah]**

****

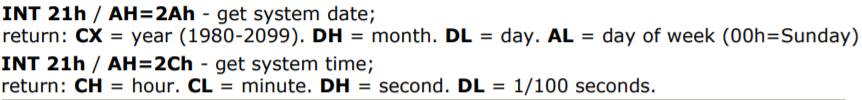
**Code:** mov ax,dx ; make a copy of **DX = 0010\_0110\_0110\_1010b**

**Shr** ax,5 ; shift right 5 bits

And **al**,**0000**1111b ; clear bits 4 to 7

mov month,al ; save in month. **Note:** same to hours**:**min**:**sec. **[2Ch]**

**< The End >**

****