In this chapter, we will learn how to move bits in integer numbers, using *shift* and *rotate* operations. Such operations are particularly useful in controlling various types of hardware devices.

Shift and Rotate Instructions:

Shift operations are used to move bits right and left inside an operand. There are two types of shifting:

1- Logical Shift: It fills the shifted bits with **zeros**.
Eg: After logical shifting 11001111 by one bit right, it becomes: 01100111

2- Arithmetic Shift: It fills the shifted bits with the a copy of the original number **sign bit**. Eg: After arithmetic shifting 11001111 by one bit right, it becomes: 11100111.

The Shift and Rotate instructions listed below affect Carry and Overflow flag:

SHL: Shift Left

Performs a logical left shift on the operand, the shifted bit is filled with zero and the highest bit is moved into the Carry flag.

Syntax:
SHL destination, count

Destination may be either register or memory, and count may be 8-bit immediate.

One example of using SHL is performing high-speed multiplication by power of 2:

```
mov dl, 5        ; dl: 00000101  =  5
shl dl, 1        ; dl: 00001010  = 10
```

SHR: Shift Right

Performs a logical right shift on the operand, the shifted bit is filled with zero and the lowest bit is moved into the Carry flag.

Syntax:
SHR destination, count

Destination may be either register or memory, and count may be 8-bit immediate.

One example of using SHR is performing high-speed division by 2^n, where n is the number of shifted bits:

```
mov dl, 32       ; dl: 00100000  =  32
shr dl, 1        ; dl:  00010000  = 16
```

SAL: Shift Arithmetic Left

Similar to SHL, it performs an arithmetic left shift on the operand, the shifted bit is filled with sign bit and the highest bit is moved into the Carry flag.

Syntax:
SAL destination, count

Destination may be either register or memory, and count may be 8-bit immediate.

SAR: Shift Arithmetic Right

Performs an arithmetic right shift on the operand, the shifted bit is filled with a copy of the sign bit and the lowest bit is moved into the Carry flag.

Syntax:
SAR destination, count

Destination may be either register or memory, and count may be 8-bit immediate.

One example of using SAR is performing high-speed division by 2^n, where n is the number of shifted bits:

```
mov dl, 0F0h        ; dl:  11110000   =   -16
sar   dl, 1         ; dl:  11111000   =   -8
```

ROL: Rotate Left Instruction:

It shifts all bits left, and the highest bit is copied into both the carry flag and into the lowest bit.

ROR: Rotate Right Instruction:

It shifts all bits right, and the lowest bit is copied into both the carry flag and into the highest bit.

RCL: Rotate Carry Left Instruction:

It shifts all bits left, and the **carry flag** is copied into the **lowest bit** and the highest bit is copied into the carry flag.

RCR: Rotate Carry Right Instruction:

It shifts all bits right, and the **carry flag** is copied into the **highest bit** and the lowest bit is copied into the carry flag.

IMUL Instruction:

IMUL instruction is similar to the MUL instruction; however, it is used for signed integer multiplication.

The CF and OF are set by IMUL if the high-order product is not a sign extension of the low-order product. Eg: (48 * 4)

```
mov al, 48
mov bl, 4
imul bl          ; AX = 00C0h,  OF = 1
```

AH is not a sign extension of AL.

(-4 * 4) :

```
mov al, -4
mov bl, 4
imul bl          ; AX = FFF0h,  OF = 0
```

AH is a sign extension of AL.

IDIV Instruction:

IDIV instruction is similar to the DIV instruction; however, doing an 8-bit or 16-bit division, we must sign-extend the dividend correspondingly in AH or AX before using IDIV.

The sign-extension is done using the instructions:

- CBW: Converts BYTE to WORD: extends the sign bit of AL into AH.
- CWD: Converts WORD to DOUBLEWORD: extends the sign bit in AX into DX.
- CDQ: Converts DoubleWord into quadword: extends the sign bit in EAX into EDX.

Eg.:

```
.data
 Sval SBYTE -48
.code
  mov al, Sval
  cbw
  mov bl, 5
  idiv bl                AL= -9,  AH = -3
```

Extended Addition and Subtraction:

ADC instruction:

Add with a carry instruction, adds both the source operand and the contents of the carry flag to a destination operand.

Operand combinations:

ADC reg, reg
ADC mem, reg
ADC reg, mem
ADC mem, imm
ADC reg, imm

SBB instruction:

Subtract with a borrow instruction, subtracts both the source operand and the value of the carry flag from a destination operand.

Eg:

```
mov edx, 1      ;upper half
mov eax, 0      ; lower half
sub  eax, 1
sbb edx, 0      ; edx:eax = 00000000FFFFFFFFh
```