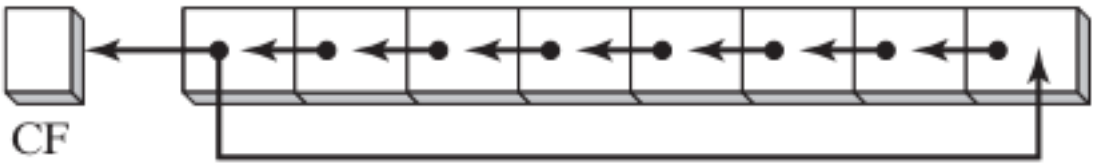
Rotate Left/Rotate Right

==========

ROL Instruction

==========

ROL (rotate left) instruction is used to perform bitwise rotation, where bits are shifted in a circular fashion. There are different ways to achieve this, including one where the bit leaving one end of the number is immediately copied into the other end.



ROL can also use the Carry flag as an intermediate point for shifted bits.

When you use the ROL instruction, each bit in the operand is shifted to the left. The highest bit (the one farthest to the left) is copied into **both the Carry flag and the lowest bit position** (farthest to the right). The instruction format is similar to that of the SHL instruction.

Preservation of Bits: Bit rotation is unique in that it doesn't lose any bits during the process. If a bit is rotated off one end of a number, it appears again at the other end. For example, let's consider the following code:

```
13 mov al, 40h ; AL = 01000000b
14 rol al, 1    ; AL = 10000000b, CF = 0
15 rol al, 1    ; AL = 00000001b, CF = 1
16 rol al, 1    ; AL = 00000010b, CF = 0
```

In this example, we start with the value 01000000b in AL, and after three rotations, the high bit is copied into both the Carry flag (CF) and bit position 0.

Multiple Rotations: When you use a rotation count greater than 1, the Carry flag contains the last bit rotated out of the most significant bit (MSB) position. For instance:

```
20 mov al, 00100000b
21 rol al, 3 ; CF = 1, AL = 00000001b
```

Here, when we rotate AL three times to the left, the Carry flag (CF) holds the last bit shifted out of the most significant bit position.

Exchanging Groups of Bits: ROL can be employed to exchange groups of bits within a byte. For example, by rotating a value like 26h four bits in either direction, you can effectively swap the upper (bits 4-7) and lower (bits 0-3) halves of a byte:

```
26 mov al, 26h
27 rol al, 4 ; AL = 62h
```

Furthermore, when you rotate a multibyte integer by four bits, you are effectively shifting each hexadecimal digit one position to the right or left. For instance:

```
31 mov ax, 6A4Bh
32 rol ax, 4 ; AX = A4B6h
33 rol ax, 4 ; AX = 4B6Ah
34 rol ax, 4 ; AX = B6A4h
35 rol ax, 4 ; AX = 6A4Bh
```

In this sequence, you can observe how each four-bit chunk of the value is rotated, eventually returning to the original value.

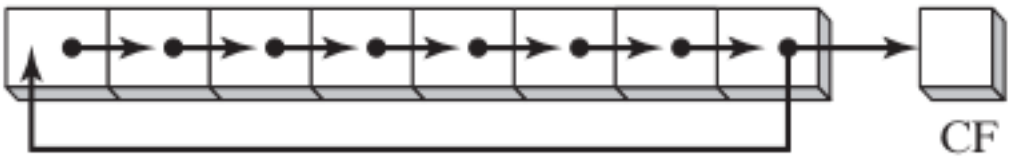
These instructions provide valuable tools for bit manipulation and data reordering in assembly language programming.

==========

ROR Instruction

==========

The **ROR (rotate right) instruction** is used to perform bitwise rotation to the right. In this operation, each bit is shifted to the right, and the lowest bit (the one farthest to the right) is copied into both the Carry flag (CF) and the highest bit position (the one farthest to the left). The instruction format for ROR is the same as that for SHL.



When you use the ROR instruction, each bit in the operand is shifted to the right. The lowest bit is copied into both the Carry flag (CF) and the highest bit position. Here's an example to illustrate this:

```
39 mov al, 01h ; AL = 00000001b
40 ror al, 1    ; AL = 10000000b, CF = 1
41 ror al, 1    ; AL = 01000000b, CF = 0
```

In this example, the initial value in AL is 00000001b, and after two rotations to the right, the lowest bit is copied into both the Carry flag (CF) and the highest bit position.

Multiple Rotations: When you perform multiple rotations with a count greater than 1, the Carry flag (CF) contains the last bit rotated out of the least significant bit (LSB) position. For example:

```
46 mov al, 00000100b
47 ror al, 3 ; AL = 10000000b, CF = 1
```

In this case, after three right rotations, the lowest bit (00000100b) is shifted to the right, and the Carry flag (CF) holds the last bit, which is 1.

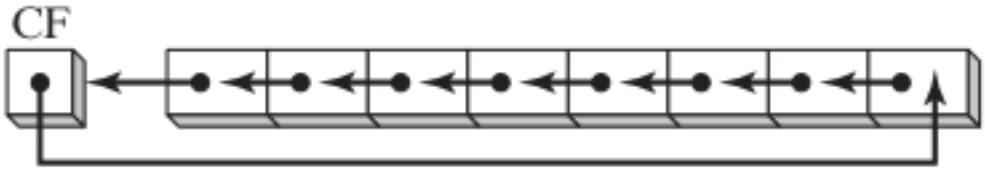
The ROR instruction is valuable for tasks that involve bitwise rotation to the right and is essential for various bit manipulation operations in assembly language programming.

==========

RCL(Rotate Carry Left) Instruction

==========

RCL (Rotate Carry Left) Instruction: The RCL instruction is used to perform bitwise rotation to the left, similar to the ROL instruction.



However, it has the additional feature of incorporating the Carry flag (CF) as an extra bit at the high end of the operand. Here's how it works:

• Each bit is shifted to the left, and the Carry flag is copied to the least significant bit (LSB).

• The most significant bit (MSB) is then copied into the Carry flag.

Example:

```
52 clc              ;CF = 0
53 mov bl, 88h      ;CF,BL = 0 10001000b
54 rcl bl, 1        ;CF,BL = 1 00010000b
55 rcl bl, 1        ;CF,BL = 0 00100001b
```

In this example, the initial value of BL is 10001000b, and after two RCL instructions, the bits are shifted to the left, with the CF acting as an extra bit and being involved in the rotation.

Recovering a Bit from the Carry Flag

RCL can be used to recover a bit that was previously shifted into the Carry flag.

In the following example, it checks the lowest bit of the "testval" variable by shifting its lowest bit into the Carry flag.

If the lowest bit is 1, a jump is taken; otherwise, RCL restores the number to its original value:

```
.data
    testval BYTE 01101010b
.code
    shr testval, 1 ; Shift LSB into Carry flag
    jc exit        ; Exit if Carry flag is set
    rcl testval, 1 ; Restore the number
```

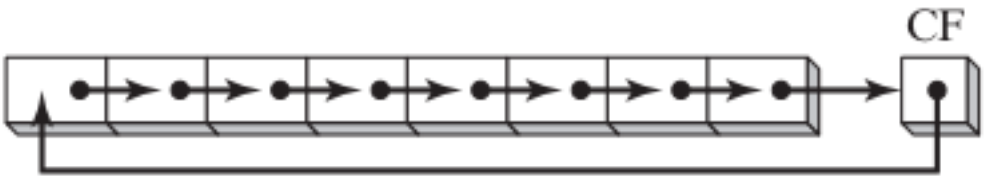
Here, the SHR instruction shifts the LSB into the Carry flag, and if it's 1, it takes a jump; otherwise, RCL restores the number.

==========

RCR (Rotate Carry Right)

==========

The RCR instruction is similar to RCL but rotates bits to the right. Here's how it operates:Each bit is shifted to the right, and the Carry flag is copied into the **most significant bit (MSB).**



The least significant bit (LSB) is copied into the Carry flag.

Example:

```
stc             ;CF = 1
mov ah, 10h     ;AH, CF = 00010000 1
rcr ah, 1       ;AH, CF = 10001000 0
```

In this example, the STC instruction sets the Carry flag to 1, and then AH is rotated to the right, incorporating the Carry flag in the

process.

These instructions are crucial for bit manipulation and advanced operations, allowing you to shift and rotate bits while incorporating the Carry flag for more intricate calculations in assembly language programming.

==========

Visualizing the integer

==========

When working with rotate and shift instructions, visualizing the integer as a 9-bit value can be helpful.

In this representation, the Carry flag is positioned to the right of the least significant bit (LSB), acting as an extra bit during the rotation.

Example: RCR (Rotate Carry Right)

In the following code example, the STC instruction sets the Carry flag to 1, and then a rotate carry right operation is performed on the AH register:

```
076 stc     ; CF = 1
077 mov ah, 10h ; AH, CF = 00010000 1
078 rcr ah, 1 ; AH, CF = 10001000 0
```

This code demonstrates how the Carry flag is used during the right rotation, influencing the resulting bits in AH.

==========

Signed Overflow

==========

The Overflow flag (OF) is set if the act of shifting or rotating a

signed integer by one bit position generates a value outside the signed integer range of the destination operand.

In other words, this means that the sign of the number is reversed. Two examples illustrate this concept:

Positive Integer Becoming Negative (ROL):

A positive integer (+127) stored in an 8-bit register becomes negative (-2) when rotated left.

```
mov  al,+127                    ; AL = 01111111b
rol  al,1                       ; OF = 1, AL = 11111110b
```

Initially:

```
AL = 01111111b
```

After rotation:

```
OF = 1, AL = 11111110b
```

Negative Integer Changing Sign (SHR):

When -128 is shifted one position to the right, the Overflow flag is set.

The result in AL (+64) has the opposite sign.

```
mov  al,-128                    ; AL = 10000000b
shr  al,1                       ; OF = 1, AL = 01000000b
```

Initially:

```
AL = 10000000b
```

After shift:

```
OF = 1, AL = 01000000b
```

The Overflow flag is set when the signed integer range is exceeded during a shift or rotation by one bit position.

The **value of the Overflow flag** is undefined when the shift or rotation count is greater than 1.

It's important to note that the value of the Overflow flag is undefined when the shift or rotation count is greater than 1.