Chapter-7

Logic, Shift, and Rotate Instructions

7.1 Logic Instructions

The ability to manipulate individual bits is one of the advantages of assembly language. We can change individual bits in the computer by using logic operations. When a logic operation is applied to 8-bit or 16-bit operands, the result is obtained by applying the logic operation at each bit position.

7.1.1 AND, OR, and XOR Instructions

The formats of AND, OR, and XOR operations are as follows:

AND destination, source

OR destination, source

XOR destination, source

The result of the operation is stored in the destination, which must be a register or memory location. The source may be a constant, register, or memory location. However, memory-to-memory operations are not allowed.

Effect on Flags:

SF, ZF, PF reflect the result

AF is undefined

CF, OF = 0

Mask: One use of AND, OR, and XOR is to selectively modify the bits in the destination. To do this, we construct a source bit pattern known as a **mask**. The mask bits are chosen so that the corresponding destination bits are modified in the desired manner when the instruction is executed.

Properties of AND, OR, and XOR

1. The AND instruction can be used to **clear** specific destination bits while preserving the others. A 0 mask bit clears the corresponding destination bit; a 1 mask bit preserves the corresponding destination bit.
2. The OR instruction can be used to **set** specific destination bits while preserving the others. A 1 mask bit sets the corresponding destination bit; a 0 mask bit preserves the corresponding destination bit.
3. The XOR instruction can be used to **complement** specific destination bits while preserving the others. A 1 mask bit complements the corresponding destination bit; a 0 mask bit preserves the corresponding destination bit.

Example 7.2: Clear the sign bit of AL while leaving the other bits unchanged.

Solution: Use AND instruction with a mask bit pattern 01111111=7FH

```
AND AL, 7FH
```

Example 7.3: Set the most significant and least significant bits of AL while preserving the other bits.

Solution: Use OR instruction with 10000001b = 81H as the mask. Thus,

```
OR  AL, 81H
```

Example 7.4: Changes the sign bit of DX.

Solution: Use XOR instruction with a mask of 8000H. Thus,

```
XOR  DX, 8000H
```

Converting an ASCII Digit to a Number

To convert an ASCII digit to a number we use an arithmetic operation:

```
SUB  AL, 30H
```

Another method is to use the AND instruction to clear the high nibble (high four bits) of AL:

```
AND  AL, 0FH
```

Converting a Lowercase Letter to Uppercase

To convert a lower case to upper case we use an arithmetic operation:

```
SUB  DL, 20H
```

Character	Code	Character	Code
a	01100001	A	01000001
b	01100010	B	01000010
:	:	:	:
:	:	:	:
z	01111010	Z	01011010

It is apparent that to convert lower to upper case we need only clear bit 5. This can be done by using an AND instruction with the mask 11011111b, or 0DFH. Thus,

```
AND  DL, 0DFH
```

Clearing a Register

MOV AX, 0

or,

SUB AX, AX

or,

XOR AX, AX

Testing a Register for Zero

OR CX, 0

or,

CMP CX, 0

7.1.2 NOT Instruction

The **NOT** instruction performs the one's complement operation on the destination. The format is

NOT destination

There is no effect on the status flags. Example: NOT AX.

7.1.3 TEST Instruction

The **TEST** instruction performs an AND operation of the destination with the source but does not change the destination contents. The purpose of the TEST instruction is to set the status flags. The format is:

TEST destination, source

Effect on Flags:

SF, ZF, PF reflect the result

AF is undefined

CF, OF = 0

Examining Bits:

The TEST can be used to examine individual bits in an operand. Thus,

TEST destination, mask

Example 7.6: Jump to BELOW if AL contains an even number.

Solution: Even numbers have a 0 in bit 0. Thus the mask is 00000001b = 1.

```
TEST  AL, 1
JZ  BELOW
```

7.2 Shift Instructions

The shift and rotate instructions shift the bits in the destination operand by one or more positions either to the left or right. For a shift instruction, the bits shifted out are lost; for a rotate instruction, bits shifted out from one end of the operand are put back into the other end.

The instructions have two possible formats. For a single shift or rotate, the form is

Opcode destination, 1

For a shift or rotate of N position, the form is

Opcode destination, CL

where CL contains N. In both cases, destination is an 8-bit or 16-bit register or memory location.

7.2.1 Left Shift Instructions

The SHL (shift left) **Instruction**:

The **SHL** (shift left) instruction shifts the bits in the destination to the left. The format for a single shift is

SHL destination, 1

A 0 is shifted into the rightmost bit position and the msb is shifted into CF (Figure 7.2).

If the shift count N is different from 1, the instruction takes the form

SHL destination, CL

Effect on Flags:

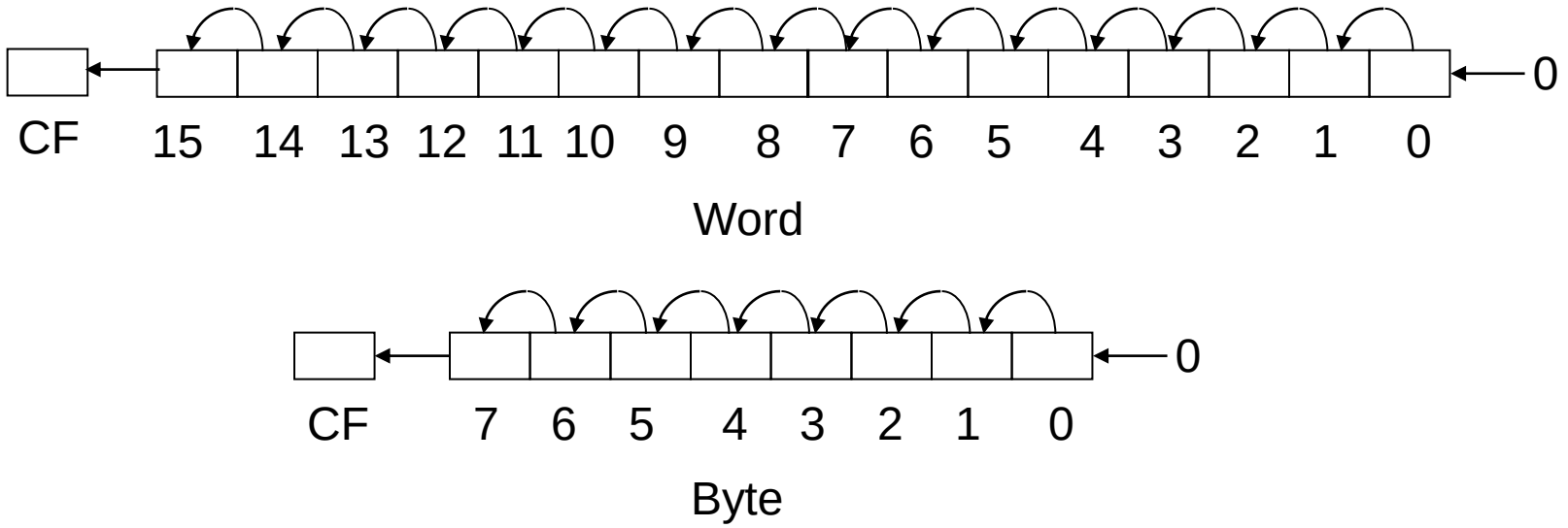
SF, PF, ZF reflect the result

AF is undefined

CF = last bit shifted out

OF = 1 if result changes sign on last shift

Figure 7.2 SHL and SAL



Example 7.7: Suppose DH contains 8AH and CL contains 3. What are the values of DH and CF after the instruction SHL DH, CL is executed?

Solution: The binary value of DH is 10001010. After 3 left shifts, CF will contain 0. The new contents of DH is 01010000b = 50H.

Multiplication by Left Shift

Consider the decimal number 235. If each digit is shifted left one position multiplying 235 by ten (2350 for 1 bit left shift). In the same way, a left shift on a binary multiplies it by 2. For example, suppose AL contains 5=00000101b. A left shift gives 0001010b=10d, another left shift yields 00010100=20d, and so on.

The SAL (Shift Arithmetic Left) Instruction

The **SAL** instruction is often used for arithmetic multiplication. However, both SHL and SAL instructions generate the same machine code.

Negative number can also be multiplied by powers of 2 by left shifts.

Suppose AX=FFFFH (-1), CL=03H, then SAL AX, CL gives AX=FFF8H (-8).

Overflow

For a single left shift, CF and OF accurately indicated unsigned and signed overflow respectively. However, the overflow flags are not reliable indicators for a multiple left shift. For example, if BL contains 80H, CL contains 2 and we execute SHL BL, CL, then CF=OF=0 even though both signed and unsigned overflow occur.

Example 7.8: Write some code to multiply the value of AX by 8.

Solution:

```
MOV  CL, 3
SAL  AX, CL
```

7.2.2 Right Shift Instructions

The SHR Instruction:

The **SHR** (shift right) instruction shifts the bits in the destination to the right. The format for a single shift is

SHR destination, 1

A 0 is shifted into the msb position, and the rightmost bit is shifted into CF (Figure 7.3).

If the shift count N is different from 1, the instruction takes the form

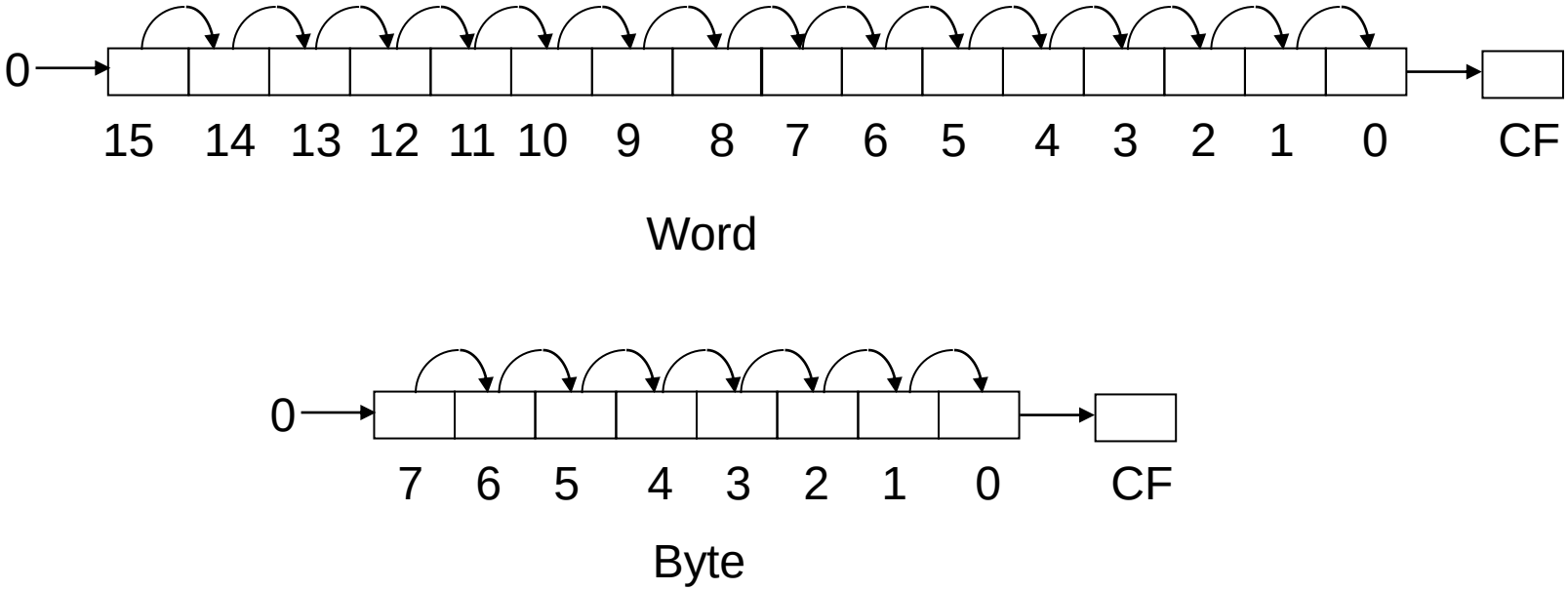
SHR destination, CL

The effect on the flags is the same as for SHL.

Example 7.9: Suppose DH contains 8AH and CL contains 2. What are the values of DH and CF after the instruction SHR DH, CL is executed?

Solution: The binary value of DH is 10001010. After 2 right shifts, CF will contain 1. The new contents of DH is 00100010b = 22H.

Figure 7.3 SHR



The SAR (Shift Arithmetic Right) Instruction

The **SAR** instruction operates like SHR, with one difference; the msb retains its original value (Figure 7.4). The syntax is

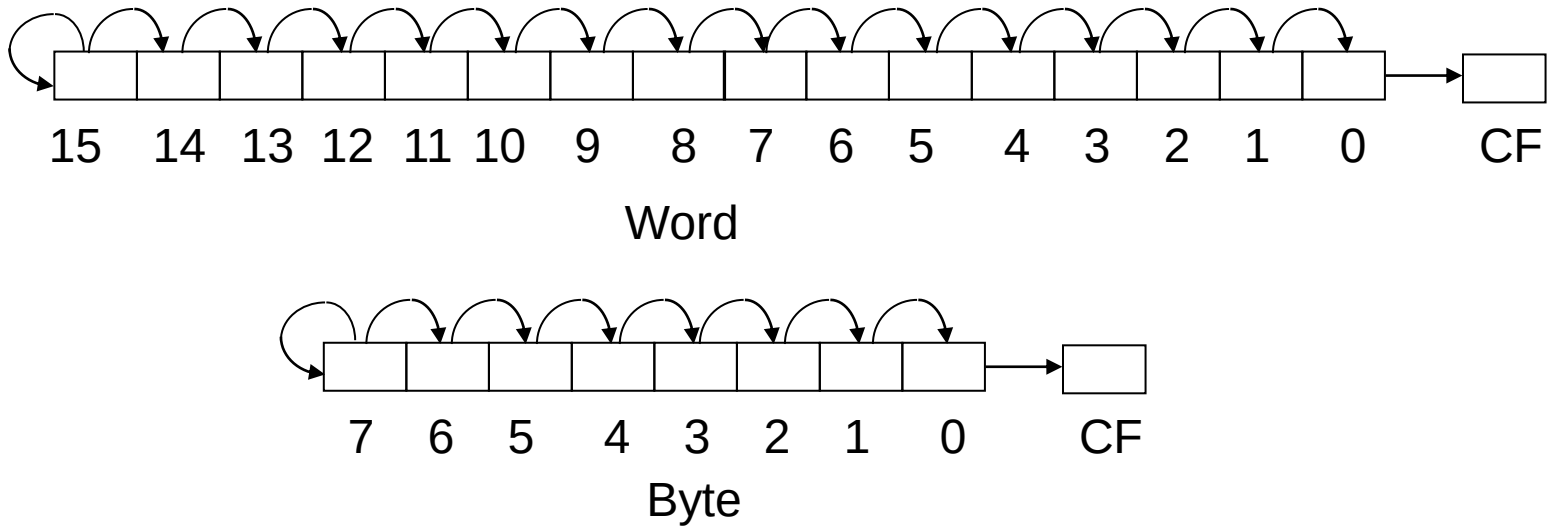
SAR destination, 1

and

SAR destination, CL

The effect on flags is the same as for SHR.

Figure 7.4 SAR



Division by Right Shift

A right shift divide a destination by 2. This is correct for even numbers. For odd numbers, a right shift halves it and rounds down to the nearest integer. For example, if BL contains 00000101b = 5, the after shift BL will contain 00000010 = 2.

Signed and Unsigned Division

If an unsigned interpretation is being given, SHR should be used. For a signed interpretation, SAR must be used, because it preserves the sign.

Example 7.10: Use right shifts to divide the unsigned number 65143 by 4. Put the quotient in AX.

Solution:

```
MOV  AX, 65143
MOV  CL, 2
SHR  AX, CL
```

Example 7.11: If AL contains -15, give the decimal value of AL after SAR AL, 1 is performed.

Solution: After execution -15 is divided by 2 and rounds down to nearest integer. Thus, -15=11110001b, after shifting 11111000b=-8.

7.3 Rotate Instructions

Rotate Left

The instruction **ROL** (rotate left) shifts bits to the left. The msb is shifted into rightmost bit. The CF also gets the bit shifted out of the msb. The destination bits forming a circle with the least significant bit following the msb in the circle (Figure 7.5). The syntax is

ROL destination, 1

and

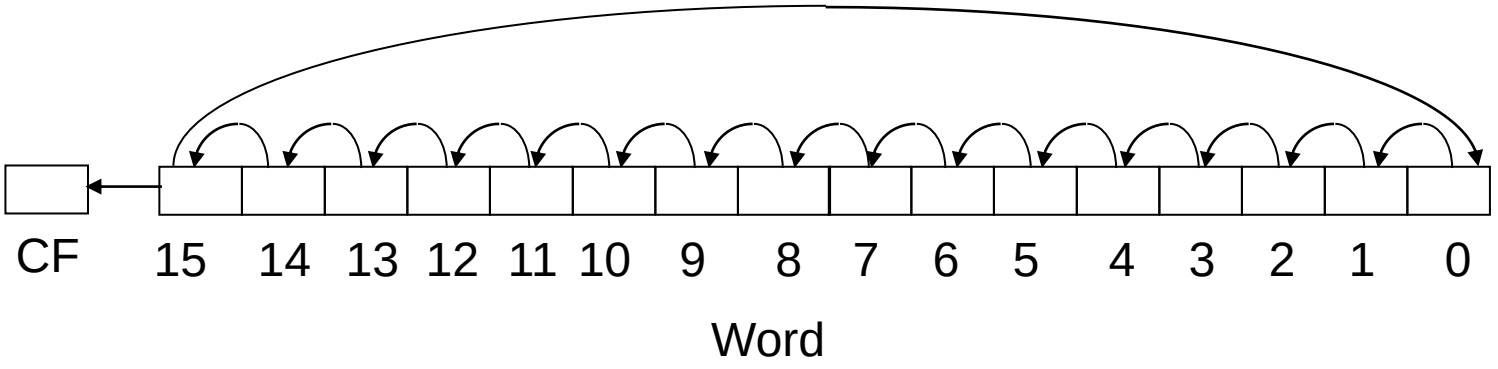
ROL destination, CL

Rotate Right

The instruction **ROR** (rotate right) works just like ROL, except that the bits are rotated to the right. The rightmost bit is shifted into the msb, and also into the CF (Figure 7.6).

In ROL and ROR, CF reflects the bit that is rotated out.

Figure 7.5 ROL



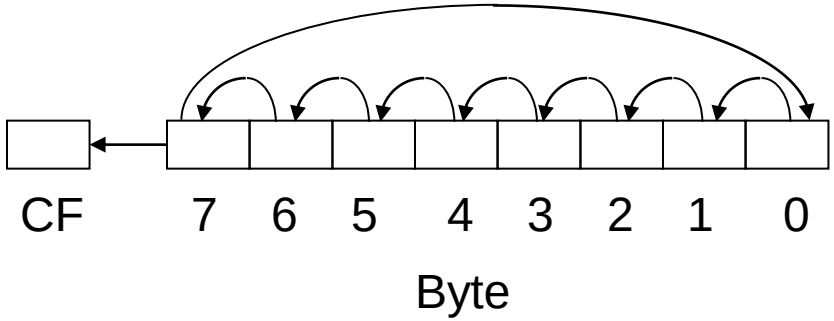
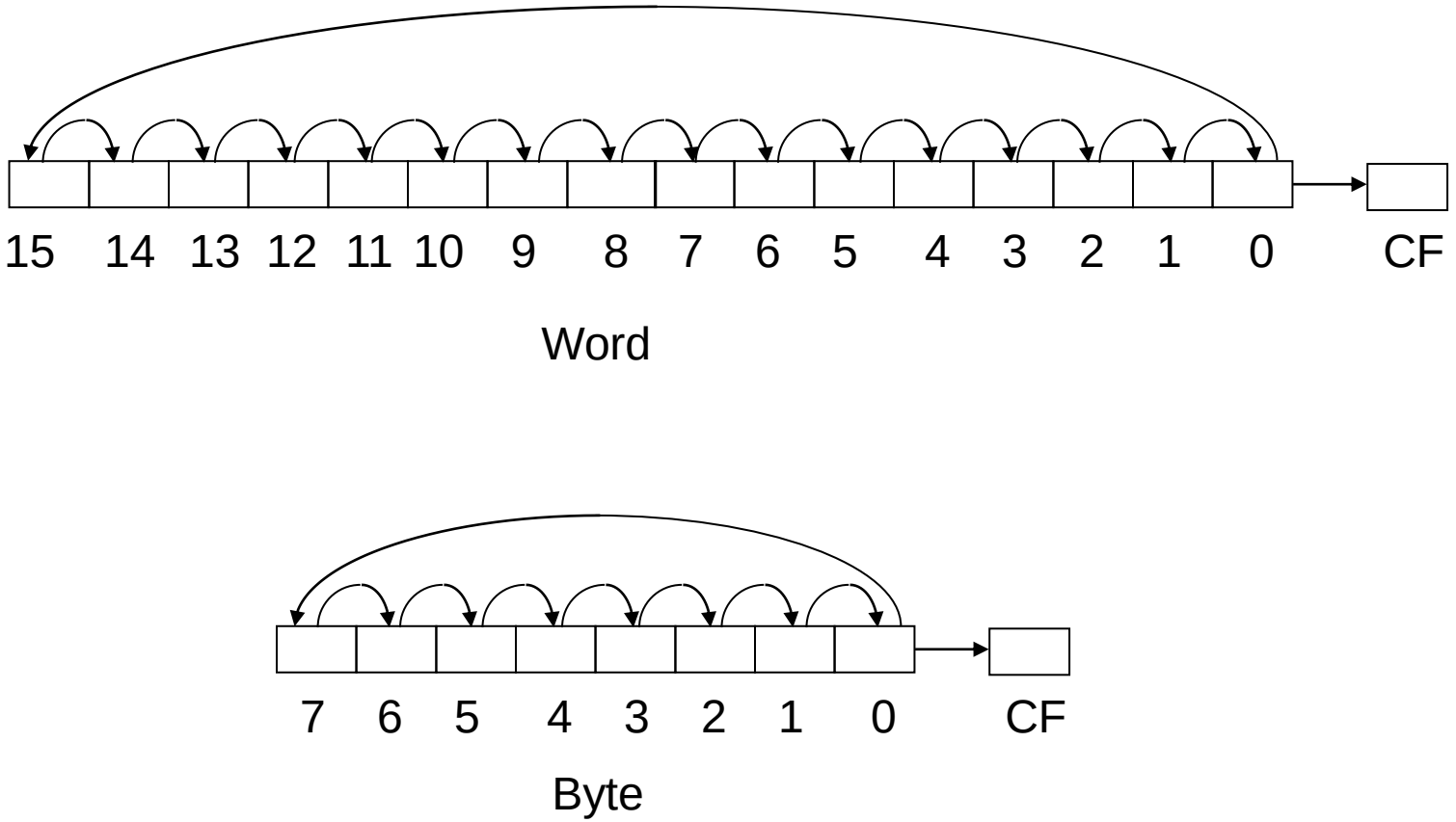


Figure 7.6 ROR



Example 7.12: Use ROL to count the number of 1 bits in BX, without changing BX. Put the answer in AX.

Solution:

```
        XOR  AX, AX
        MOV  CX, 16
TOP:
        ROL  BX, 1
        JNC NEXT
        INC AX
NEXT:
        LOOP TOP
```

Rotate Carry Left

The instruction **RCL** (Rotate through Carry Left) shifts the bits of the destination to the left. The msb is shifted into the CF, and the previous value of CF is shifted into the rightmost bit. In other words, RCL works just like ROL, except that CF is part of the circle of bits being rotated (Figure 7.7). The syntax is

RCL destination, 1

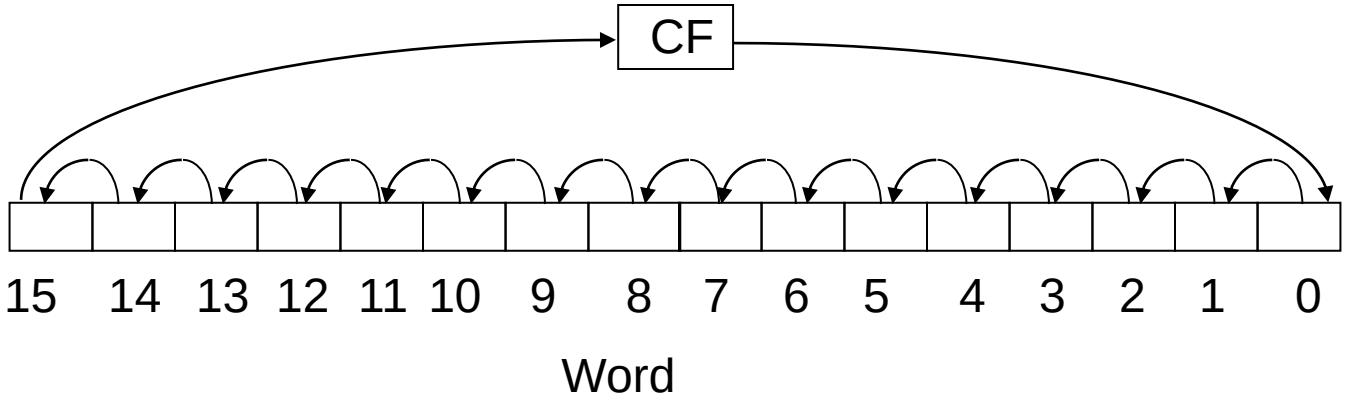
and

RCL destination, CL

Rotate Carry Right

The instruction **RCR** (Rotate through Carry Right) works just like RCL, except that the bits are rotated to the right (Figure 7.8). The syntax is

Figure 7.7 RCL



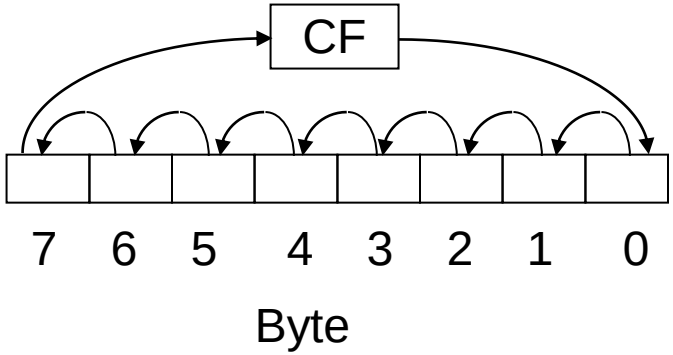
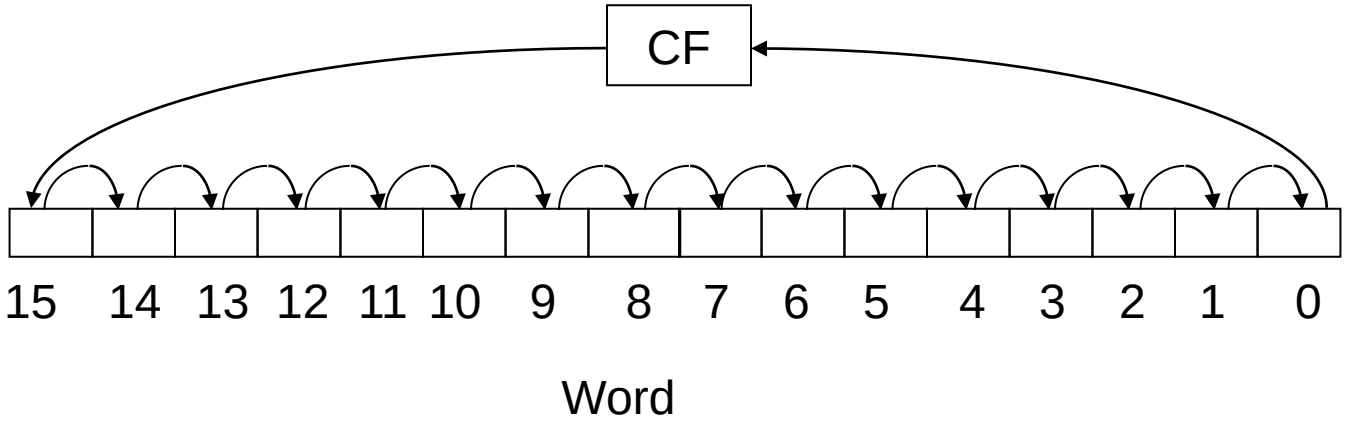
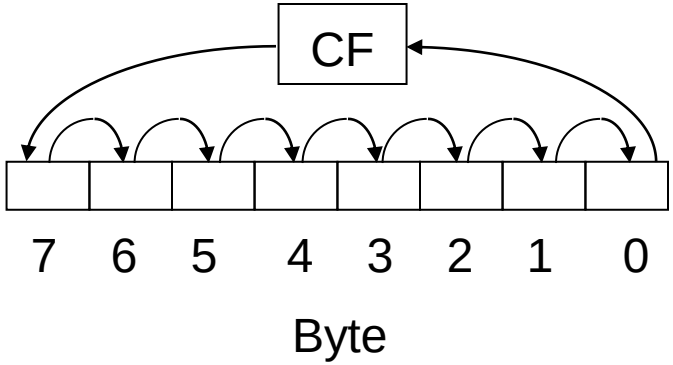


Figure 7.8 RCR





An Application: Reversing a Bit Pattern

An easy way to do this is to use SHL and then RCR or SHR and then RCL.

Solution:

```
        MOV  CX, 8
REVERSE:
        SHL  AL, 1
        RCR  BL, 1
        LOOP  REVERSE
        MOV  AL, BL
```