

Assembly Language for Intel-Based Computers, 4th Edition

Kip R. Irvine

Chapter 7: Integer Arithmetic

Chapter Overview

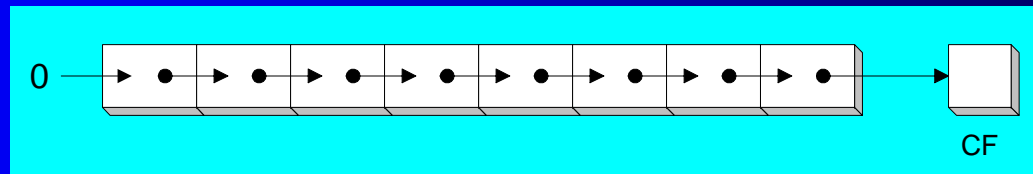
- Shift and Rotate Instructions
- Shift and Rotate Applications
- Multiplication and Division Instructions
- Extended Addition and Subtraction
- ASCII and Packed Decimal Arithmetic

Shift and Rotate Instructions

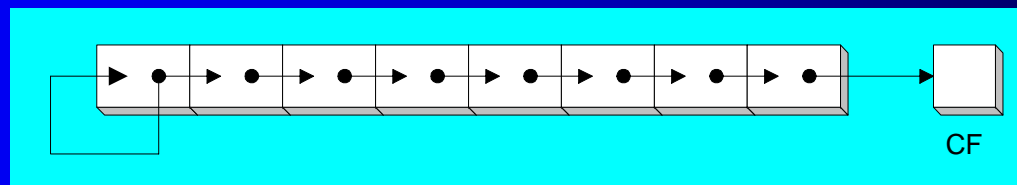
- Logical vs. Arithmetic Shifts
- SHL Instruction
- SHR Instruction
- SAL and SAR Instructions
- ROL Instruction
- ROR Instruction
- RCL and RCR Instructions
- SHLD/SHRD Instructions

Logical vs Arithmetic Shifts

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

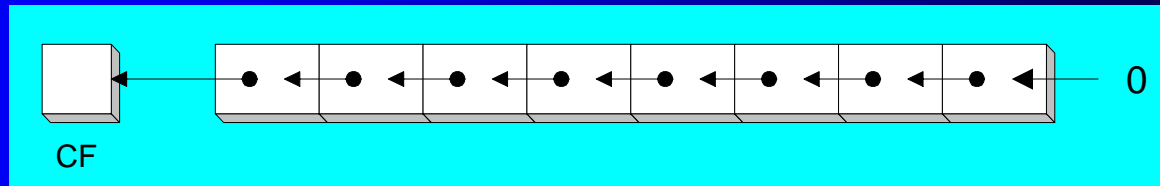


- An arithmetic shift fills the newly created bit position with a copy of the number's sign bit:



SHL Instruction

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



- Operand types:

`SHL reg,imm8`

`SHL mem,imm8`

`SHL reg,CL`

`SHL mem,CL`

Fast Multiplication

Shifting left 1 bit multiplies a number by 2

```
mov dl,5  
shl dl,1
```

Before: 0 0 0 0 0 1 0 1 = 5
After: 0 0 0 0 1 0 1 0 = 10

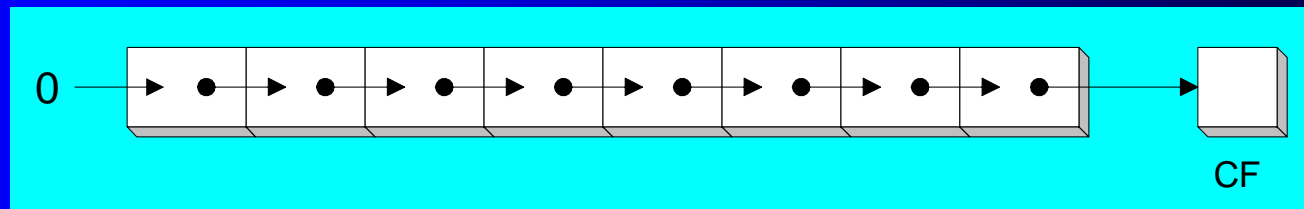
Shifting left n bits multiplies the operand by 2^n

For example, $5 * 2^2 = 20$

```
mov dl,5  
shl dl,2  
; DL = 20
```

SHR Instruction

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

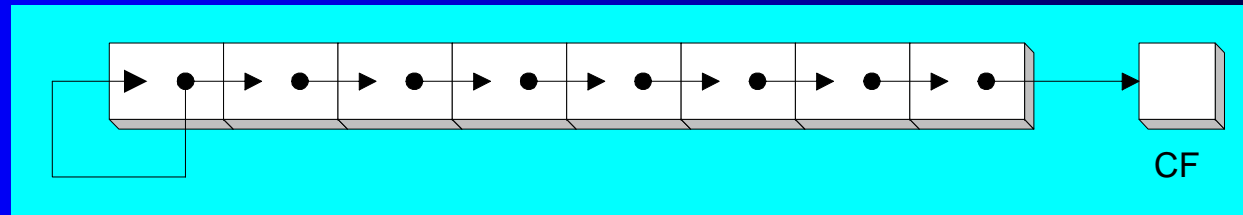


Shifting right n bits divides the operand by 2^n

```
mov dl,80
shr dl,1           ; DL = 40
shr dl,2           ; DL = 10
```

SAL and SAR Instructions

- SAL (shift arithmetic left) 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.

```
mov dl,-80
sar dl,1      ; DL = -40
sar dl,2      ; DL = -10
```


Your turn . . .

Indicate the hexadecimal value of AL after each shift:

```
mov al,6Bh
```

```
shr al,1
```

```
shl al,3
```

```
mov al,8Ch
```

```
sar al,1
```

```
sar al,3
```

a. 35h

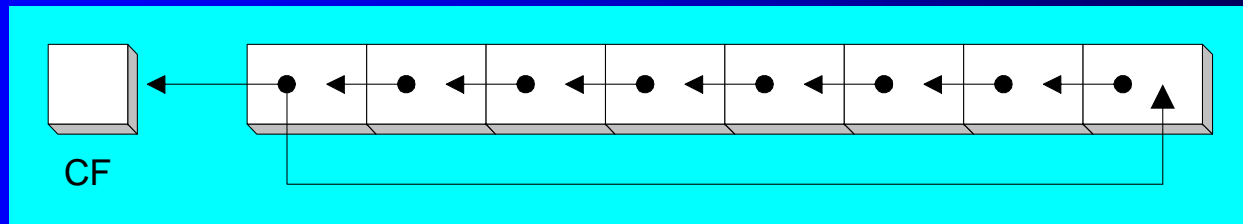
b. A8h

c. C6h

d. F8h

ROL Instruction

- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost

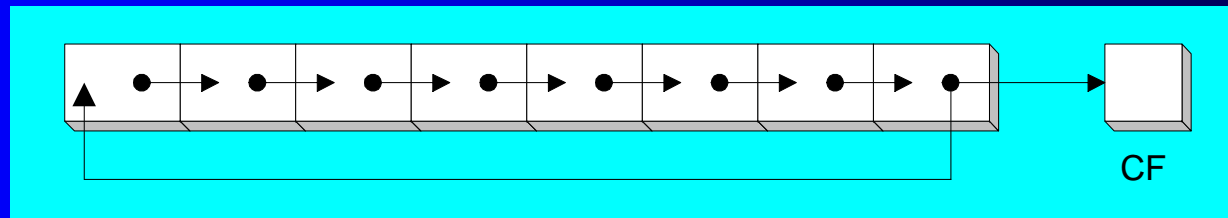


```
mov al,11110000b
rol al,1                ; AL = 11100001b

mov dl,3Fh
rol dl,4                ; DL = F3h
```

ROR Instruction

- ROR (rotate right) shifts each bit to the right
- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost



```
mov al,11110000b
ror al,1                ; AL = 01111000b

mov dl,3Fh
ror dl,4                ; DL = F3h
```

Your turn . . .

Indicate the hexadecimal value of AL after each rotation:

```
mov al,6Bh
```

```
ror al,1
```

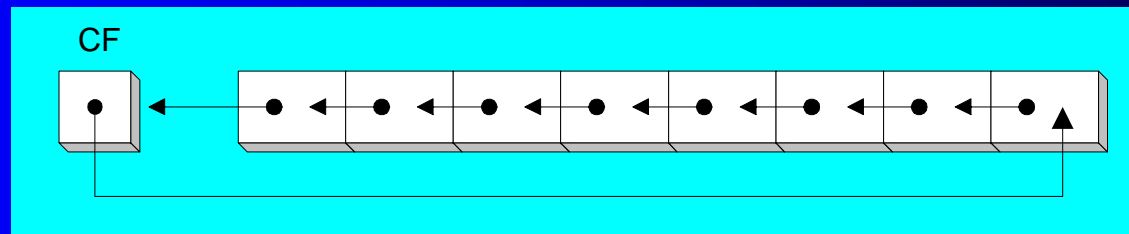
```
rol al,3
```

a. B5h

b. ADh

RCL Instruction

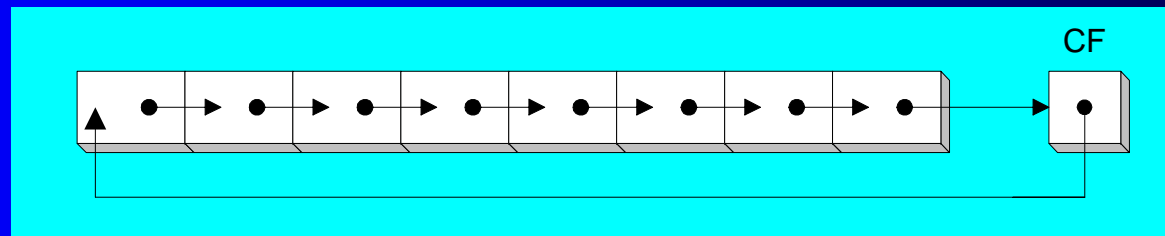
- RCL (rotate carry left) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag



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

RCR Instruction

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag



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

Your turn . . .

Indicate the hexadecimal value of AL after each rotation:

```
stc  
mov al,6Bh  
rcr al,1  
rcl al,3
```

a. B5h

b. AEh

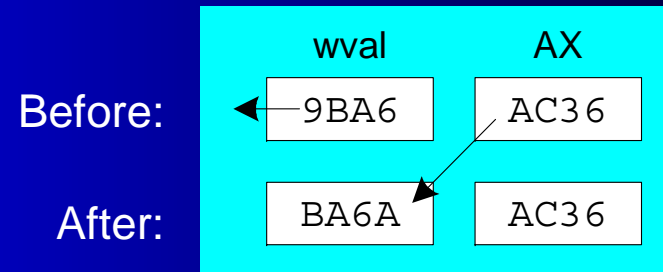
SHLD Instruction

- Shifts a destination operand a given number of bits to the left
- The bit positions opened up by the shift are filled by the most significant bits of the source operand
- The source operand is not affected
- Syntax:
SHLD destination, source, count

SHLD Example

Shift **wval** 4 bits to the left and replace its lowest 4 bits with the high 4 bits of AX:

```
.data
wval WORD 9BA6h
.code
mov  ax,0AC36h
shld wval,ax,4
```



SHRD Instruction

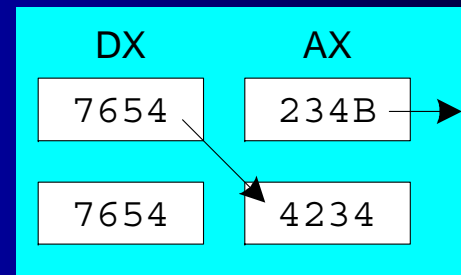
- Shifts a destination operand a given number of bits to the right
- The bit positions opened up by the shift are filled by the least significant bits of the source operand
- The source operand is not affected
- Syntax:
SHRD destination, source, count

SHRD Example

Shift **AX** 4 bits to the right and replace its highest 4 bits with the low 4 bits of **DX**:

```
mov  ax,234Bh  
mov  dx,7654h  
shrd ax,dx,4
```

Before:



After:

Your turn . . .

Indicate the hexadecimal values of each destination operand:

```
mov    ax,7C36h
mov    dx,9FA6h
shld   dx,ax,4           ; DX = FA67h
shrd   dx,ax,8           ; DX = 36FAh
```

Shift and Rotate Applications

- Shifting Multiple Doublewords
- Binary Multiplication
- Displaying Binary Bits
- Isolating a Bit String

Shifting Multiple Doublewords

- Programs sometimes need to shift all bits within an array, as one might when moving a bitmapped graphic image from one screen location to another.
- The following shifts an array of 3 doublewords 1 bit to the right (view complete [source code](#)):

```
.data
ArraySize = 3
array DWORD ArraySize DUP(99999999h)      ; 1001 1001...
.code
mov esi,0
shr array[esi + 8],1      ; high dword
rcr array[esi + 4],1      ; middle dword, include Carry
rcr array[esi],1          ; low dword, include Carry
```

Binary Multiplication

- We already know that SHL performs unsigned multiplication efficiently when the multiplier is a power of 2.
- You can factor any binary number into powers of 2.
 - For example, to multiply $EAX * 36$, factor 36 into $32 + 4$ and use the distributive property of multiplication to carry out the operation:

```
EAX * 36
= EAX * (32 + 4)
= (EAX * 32) + (EAX * 4)
```

```
mov eax,123
mov ebx,eax
shl eax,5      ; mult by 25
shl ebx,2      ; mult by 22
add eax,ebx
```

Your turn . . .

Multiply AX by 26, using shifting and addition instructions.

Hint: $26 = 16 + 8 + 2$.

```
mov ax,2                ; test value

mov dx,ax
shl dx,4                ; AX * 16
push dx                ; save for later
mov dx,ax
shl dx,3                ; AX * 8
shl ax,1                ; AX * 2
add ax,dx               ; AX * 10
pop dx                 ; recall AX * 16
add ax,dx               ; AX * 26
```


Displaying Binary Bits

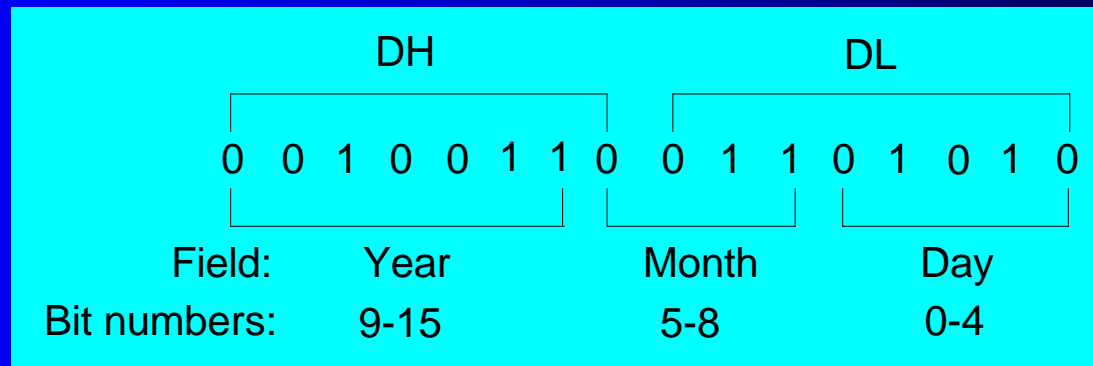
Algorithm: Shift MSB into the Carry flag; If CF = 1, append a "1" character to a string; otherwise, append a "0" character. Repeat in a loop, 32 times.

```
    mov ecx,32
    mov esi,offset buffer
L1: shl eax,1
    mov BYTE PTR [esi],'0'
    jnc L2
    mov BYTE PTR [esi],'1'

L2: inc esi
    loop L1
```

Isolating a Bit String

- The MS-DOS file date field packs the year, month, and day into 16 bits:



Isolate the Month field:

```
mov ax,dx          ; make a copy of DX
shr ax,5           ; shift right 5 bits
and al,00001111b   ; clear bits 4-7
mov month,al        ; save in month variable
```