THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 8

Bit operations in Assembly Language

Agenda

- **Bit operations in Assembly Language**
  - NOT operation
  - And operation
  - Test operation
  - Or operation
  - Exclusive OR operation
  - Logical Shift operations
  - Arithmetic Shift operation
  - Rotate Shift operations
  - Bit testing and setting (Bitmasks)
  - Extracting and filling a bit field
  - SHLD Instruction
  - SHRD Instruction

NOT Instruction

NOT "~" Logic Truth Table

1	0
0	1

- NOT instruction gets the one's complement of a number by converting each bit to its complement
- Operand can be a register or memory location
- Assume that content of register AL is (1010 0101)
  - NOT AL (AL = 0101 1010)

OR – Logical inclusive OR

OR “|” Logic Truth Table

0	0	0
0	1	1
1	0	1
1	1	1

OR des, src

- Destination operand can be memory location or register
- Source operand can be memory location, register or immediate
- Destination and source operands can’t be memory locations at same time

	00110011 (EAX: 0x33)
OR	01010101 (EBX: 0x55)
Result	01110111 (EAX: 0x77)

	00110011b (EAX: 0x33)
OR	01000010b (IMM: 0x42)
Result	01110011b (EAX: 0x73)

OR EAX, EBX

OR EAX, 0x42

AND – Logical bitwise AND

AND “&” Logic Truth Table

0	0	0
0	1	0
1	0	0
1	1	1

AND des, src

- Destination operand can be memory location or register
- Source operand can be memory location, register or immediate
- Destination and source operands can’t be memory locations at same time

	00110011 (EAX: 0x33)
AND	01010101 (EBX: 0x55)
Result	00010001 (EAX: 0x11)

AND EAX, EBX

	00110011 (EAX: 0x33)
AND	01000010 (IMM: 0x42)
Result	00000010 (EAX: 0x02)

AND EAX, 0x42

Test Instruction

- The test instruction performs an AND, but does not store the result
- The test instruction only sets the FLAG bits
- Example:

```
        mov al, 0FFh   ;   assume ZF = 0
        test al, 00h   ; result =0, ZF = 1 (set ZF)
        jz function    ; jump to function if result =0 (ZF = 1)
                       (jnz function, it means jump when result not zero (ZF = zero))
          -----
         -------
function:
       ----
       ----
```

- Note that all Boolean bitwise instructions (except not operation) set the FLAG bits

XOR – Logical exclusive OR

XOR “^” Logic Truth Table

0	0	0
0	1	1
1	0	1
1	1	0

XOR des, src

- Destination operand can be memory location or register
- Source operand can be memory location, register or immediate
- Destination and source operands can’t be memory locations at same time
- One way to zero a register is to XOR it with itself

	00110011b (EAX: 0x33)
XOR	00110011b (EAX: 0x33)
Result	00000000b (EAX: 0x00)

XOR EAX, EAX

	00110011b (AL: 0x33)
XOR	01000010b (IMM: 0x42)
Result	01110001b (AL: 0x71)

XOR AL, 0x42

Uses of Bitwise operations

- Bitwise operations provide ways to manipulate individual bits in multi-byte values
- Bitwise operations are useful to modify individual bits within data

- This is done via bit masks, i.e., constant (immediate) quantities with carefully chosen bits
- Example:

How could you turn on bit#3 of a 2-byte value stored in ax (counting from the right)

Answer: Execute OR operation the stored value with 0000000000001000 (= 8 d)

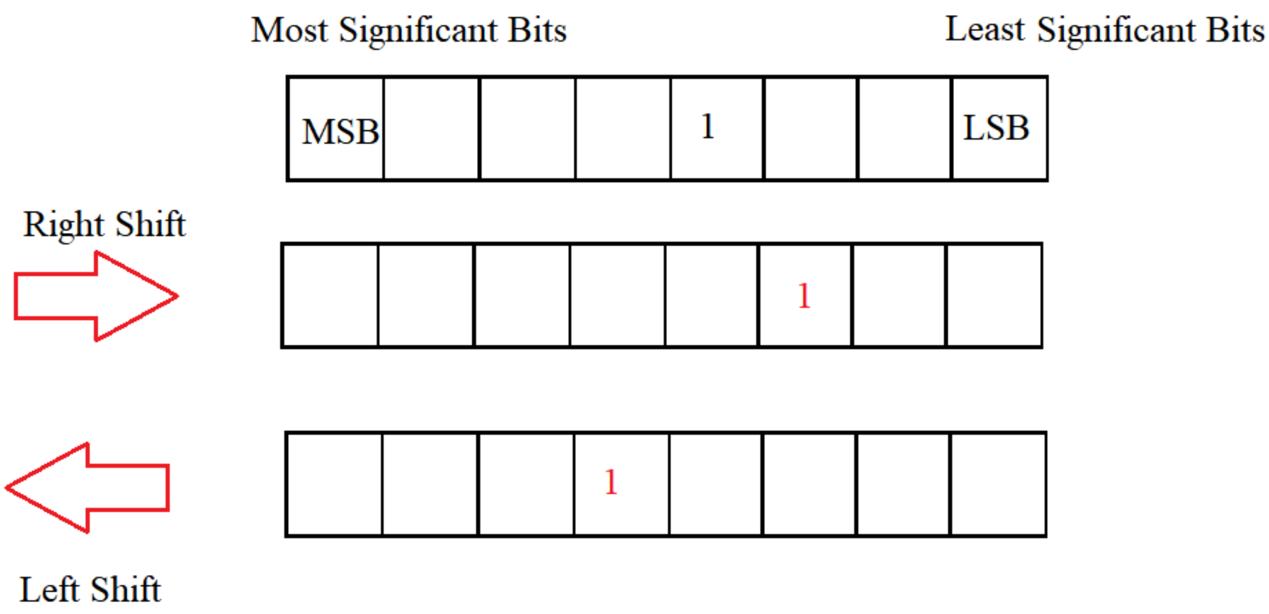
```
or    ax, 8       ; turns on bit 3 in ax
```

- Rules

- To turn on bits: use OR (with appropriate 1's in the bit mask)
- To turn off bits: use AND (with appropriate 0's in the bit mask) (111111 …1111 0111)
- To flip bits: use XOR (with appropriate 1's in the bit mask)

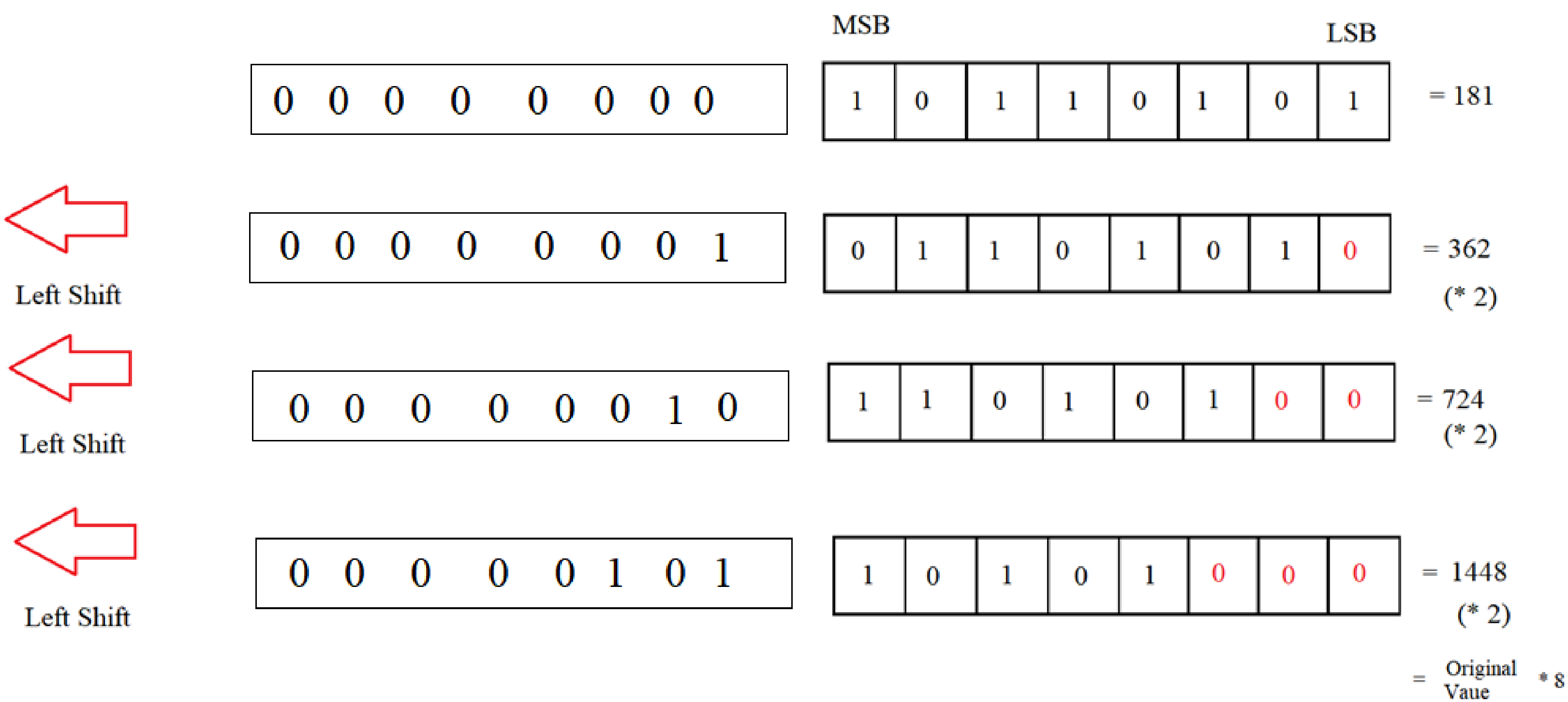
Shift Operations

- A shift moves the bits around in some data
- A shift can be toward the right (i.e., toward the least significant bits (LSB))
- A shift can be toward the left (i.e., toward the most significant bits (MSB)),
- There are two kinds of shifts:
  1- Logical Shifts
  2- Arithmetic Shifts



Left Logical Shift Operations

- In the logical shift operations, a disappeared bit at one end is replaced by zero



$$\text{New value} = \text{Old Value} * 2^{\#\text{left shift bits}}$$

Left Logical Shift Example

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
```

Right Logical Shift Operations

- In the logical shift operations, a disappeared bit at one end is replaced by zero

$$\text{New value} = \text{int}\left(\text{old value} / 2^{\#\ \text{right shift bits}}\right)$$

Shift Operations

- Two instructions: shl and shr
- One specifies by how many bits the data is shifted
  - Either by just passing a constant to the instruction

    shl eax, 2 ; shift 2 bits to the left

    shl, [memory location], 2
  - Or by using whatever is stored in the **ECL** register

    mov ecl, 3

    shl eax, ecl

    shl, [memory location], ecl
- After the instruction executes, the carry flag (CF) contains the (last) bit that was shifted out
- Examples:

mov al, 0C6h	; al = 1100 0110	
shl al, 1	; al = 1000 1100	CF = 1
shr al, 1	; al = 0100 0110	CF = 0
shl al, 3	; al = 0011 0000	CF = 0

Logical Shift Operations Issue-1

- The logical left shift instruction (shl) could cause overflow issue if the new value is larger than the maximum value that a register can have
- In that case, the new product value is wrong
- If a number is too large, then we need more bits to get the correct multiplication value
- Example:

  10000000 (128d) cannot be left-shifted to obtain 256

Logical Shift Operations Issue-2

- The logical instructions (shl and shr) work fine with unsigned numbers

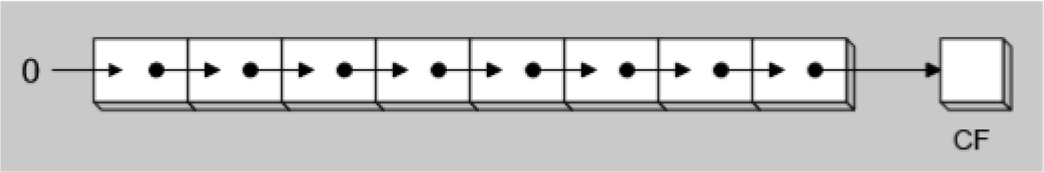
- With signed numbers, the shift operation could change the value of the sign bit
- In this case, the new value can't be obtained as multiplying or dividing the original number by powers of 2

Example: Consider the 1-byte number FE

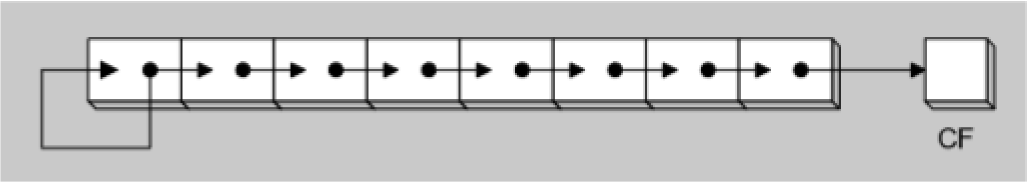
- If unsigned number:
  - FE = 254d = 11111110
  - Right shift: 01111111 = 7Fh = 127d (which is 254/2)
- If signed number:
  - FE = - 2d = 11111110
  - Right shift: 01111111 = 7Fh = +127d (which is NOT -2/2)

Arithmetic Right Shift Operations

- Arithmetic shift works only with signed numbers
- Arithmetic right shift reserves the sign bit of the shifted number by filling the new sign bit with a copy of the original number's sign bit



Logical Right Shift (shr)



Arithmetic Righ Shift (sar)

Arithmetic Left Shift Operations

- Arithmetic left shift works only with signed numbers
- Signed numbers represented by 2's complement
- Arithmetic left shift (sal) works exactly as logical left shift (shl) but with signed number
- As long as the sign bit is not changed by the shift, the result will be correct (i.e., will be multiplied by 2). Issue could happen if you perform many left shifts!!

Example: Store -8 in 1-byte

\- 8

8 = 00001000
1st complement = 1111 0111
$+ \quad 1$

1111 1000

MSB ... LSB

1	1	1	1	1	0	0	0	= (-8)

1	1	1	1	0	0	0	0	= ??? = (2* (-8)) = -16

1111 0000
0000 1111
$+ \quad 1$

00010000 = (16)

1	1	1	0	0	0	0	0	= ?? = -32

1110 0000
0001 1111
$+ 1$

00100000 = (32)

Arithmetic Shift Examples

- Arithmetic left shift works only with signed numbers

```
mov    al, 0C3h     ; al = 1100 0011 (-61d)
sal    al, 1        ; al = 1000 0110 (86h = -122d)

mov    al, 0C3h     ; al = 1100 0011 (-61d)
sar    al, 1        ; al =  1110 001 (-30d)
sar    al, 1        ; al = 1111 0001 (-15d)
sar    al, 1        ; al = 1111 1000  (-8)
```

```
1111 1000              1111 0001
0000 0111              0000 1110
       +1                    + 1
__________             --------------
0000 1000 (8)          0000 1111 (15)
```

Arithmetic Left Shift Operations

- Arithmetic left shift works only with signed numbers
- Signed numbers represented by 2's complement
- Arithmetic left shift (sal) works exactly as logical left shift (shl)
- As long as the sign bit is not changed by the shift, the result will be correct (i.e., will be multiplied by 2). Issue could happen if you perform many left shifts!!
- The following is not a correct multiplication by 16!
- Assume al= 1100 0011 (-ve)

  sal al, 4 ; al will become positive

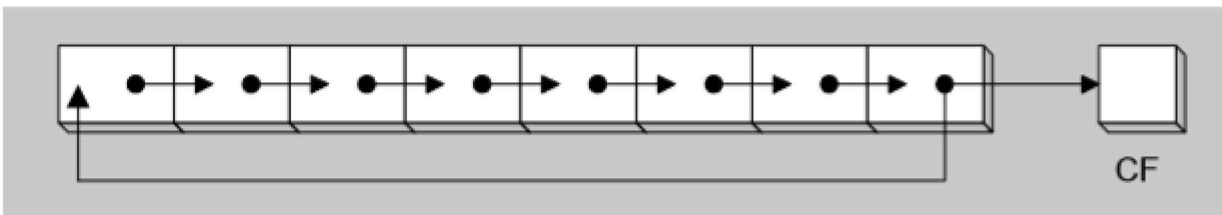
al = 1100 0011 > 1000 0110 > 0000 1100 > 0001 1000 > 0011 0000

Resolution: If the goal of shifting operation to multiply or divide a number

- It is safer to use imul and idiv with signed numbers
- It is safer to use mul and div with unsigned numbers

Rotate Right Shift

- ROR (Rotate Right) shifts each bit to the right
- The LSB is copied into both the CF (Carry Flag) and into the MSB
- No bits are lost
- N bit operation



- EXAMPE:

  mov al, 11110000 ; al =240d
  ror al, 1 ; al = 01111000 = 120 d, CF =0

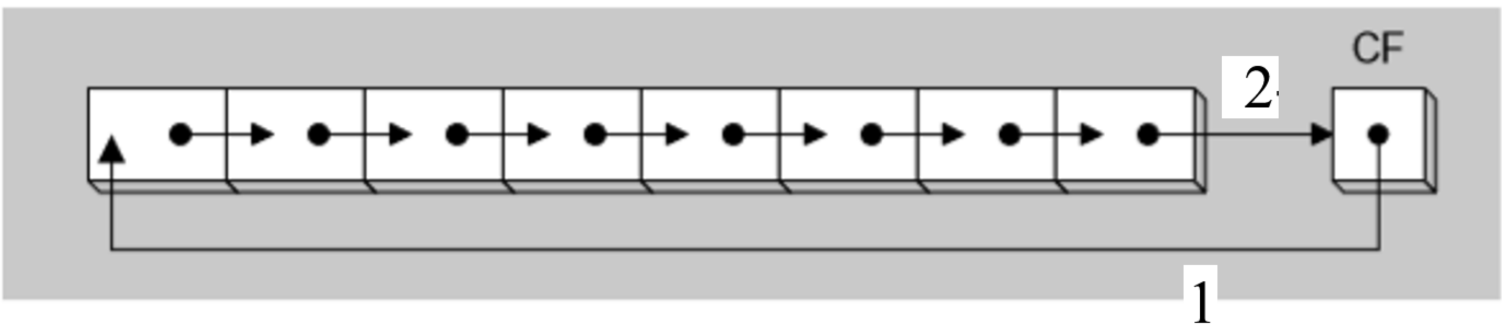
  mov al, 01110001 ; al =113d
  ror al, 1 ; al = 10111000 = 184 d, CF =1

Rotate Carry Right Shift

- RCR (Rotate Carry Right) shifts each bit to the right

  1- Copies the old Carry flag to the MSB

  2- Copies the LSB to the Carry flag
- (N+1) bit rotation

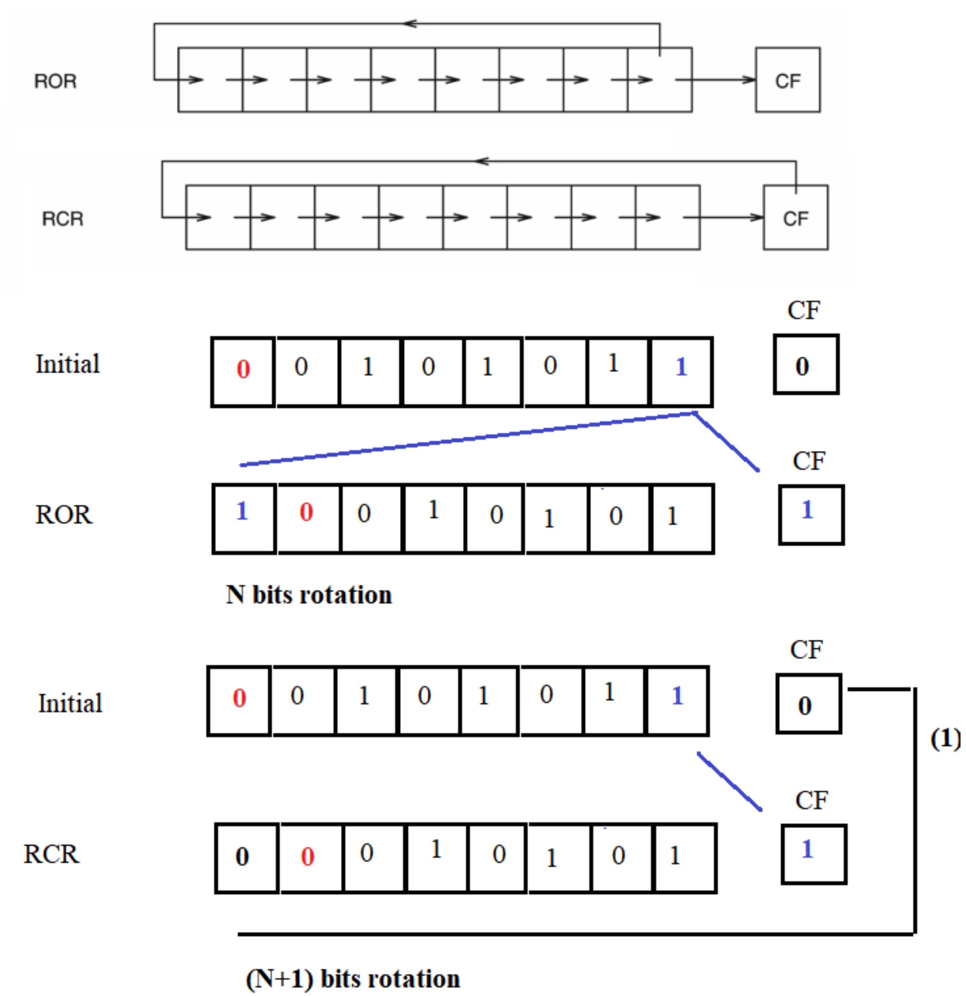


- EXAMPE:

```
; assume CF =1
mov al, 10h          ; CF = 1,   al = 00010000
rcr al, 1            ;  CF = 0,   al = 10001000
```

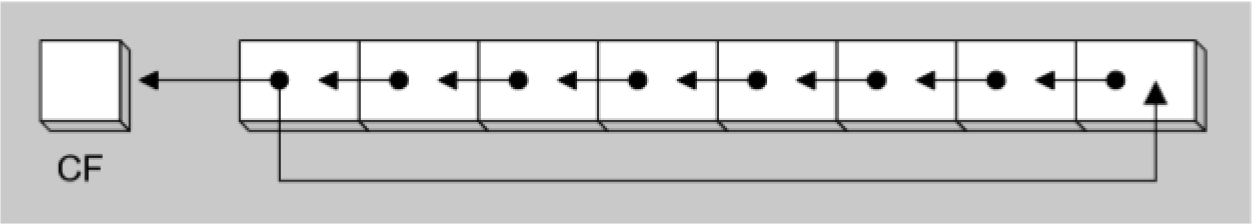
Difference Between ROR and RCR

- ROR doesn't include CF in the rotation. It is N bit rotation
- RCR includes the CF in the rotation. It is (N+1) bit rotation



Rotate Left Shift

- ROL (rotate Left) shifts each bit to the left
- The MSB is copied into both the CF (Carry Flag) and into the LSB
- No bits are lost



- EXAMPE:

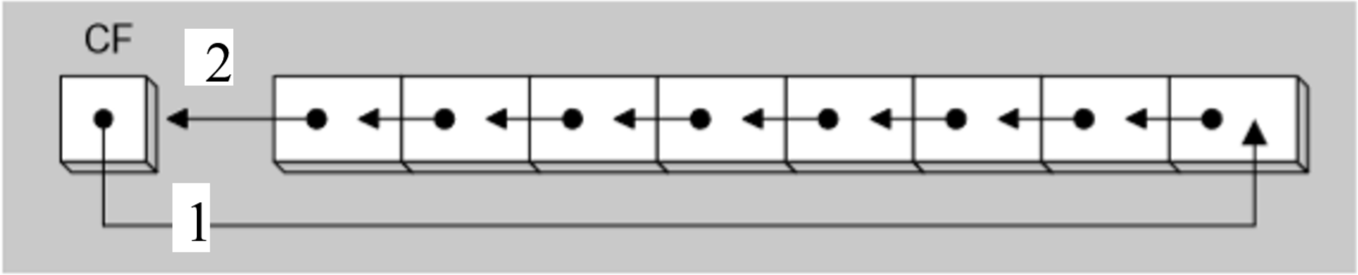
```
mov al, 11110000  ;  al =240d
rol al, 1         ;  al = 1110 0001 =  225d

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

Rotate Carry Left Shift

- RCL (Rotate Carry Left) shifts each bit to the left

  1- Copies the Carry flag (CF) to the LSB

  2- Copies the MSB to the Carry flag



- EXAMPE:

```
clc                 ; CF =0
mov al, 88h         ; CF = 0,   al = 10001000
rcl al, 1           ;  CF = 1,   al = 00010000
rcl, al, 1          ; CF = 1,    al = 00100001
```

SHLD Instruction

- Shift a destination operand a given number of bits (count) to the left
- Fill up the least significant bits of the destination operand by the most significant bits of the source operand
- The source operand is not affected

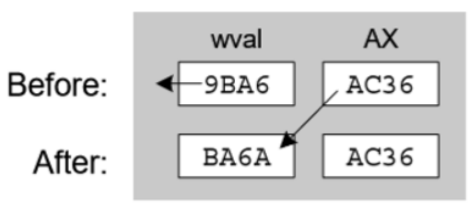
SHLD destination, source, count

```
.data
 var1  word  9BA6h
.text
mov ax, 0AC36h   ;
```

var 1= 1001 1011 1010 0110

ax = $\boxed{1010}$ 1100 0011 0110

```
shld var1, ax, 4       ; ax = 0AC36h (no change)
                       ; var1 = BA6A
```



Initial

var 1= 1001 $\boxed{\text{1011 1010 0110}}$

1- Shift var1 to the left by 4 bits

$\boxed{\text{1011 1010 0110}}$ 0000

Lowest 4 bits

2- Replace lowest 4 bits with highest 4 bit of ax

$\boxed{\text{1011 1010 0110}}$ 1010

9 B 6 A

SHRD Instruction

- Shift a destination operand a given number of bits (count) to the right
- Fill up the most significant bits of the destination operand by the least significant bits of the source operand
- The source operand is not affected

SHRD destination, source, count

```
.data
 var1  word  9BA6h
.text
mov ax, 0AC36h   ;

shld var1, ax, 4      ; ax = 0AC36h (no change)
                      ; var1 = BA6A
```

var 1= 1001 1011 1010 $\boxed{0110}$

ax = $\boxed{1010}$ 1100 0011 $\boxed{0110}$

(1)

9BA6 ⇨ 09BA

AX
AC36

2

69BA

Initial
var 1= 1001 1011 1010 $\boxed{0110}$

1- Shift var1 to the right by 4 bits

0000 1001 1011 1010

2- Replace lowest 4 bits with highest 4 bit of ax

$\boxed{0110}$ 1001 1011 1010

6 9 B A

Putting It All Together

You should know:

- **Assembly Language Instructions**
  - NOT operation
  - And operation
  - Test operation
  - Or operation
  - Exclusive OR operation
  - Logical Shift operations
  - Arithmetic Shift operations
  - Rotate Shift operations
  - Bit testing and setting (Bitmasks)
  - How could you extract and fill a bit field
  - SHLD Instruction
  - SHRD Instruction

Questions?

Coming Next Week
Branching and Looping

Week 8 Assignments

- **Learning Materials**

  1- Week 5 Presentation

  2- Reading Ch.8 and 9: Duntermann, Jeff. Assembly Language Step by Step, Programming

  3- Reading Ch3: PCASM textbook

- **Assignment**

  1- Complete "Lab 8 assignment" by coming Sunday 11:59 PM.