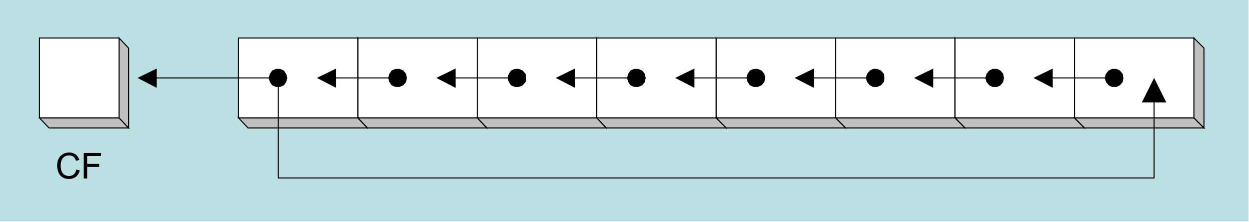
ROL Instruction

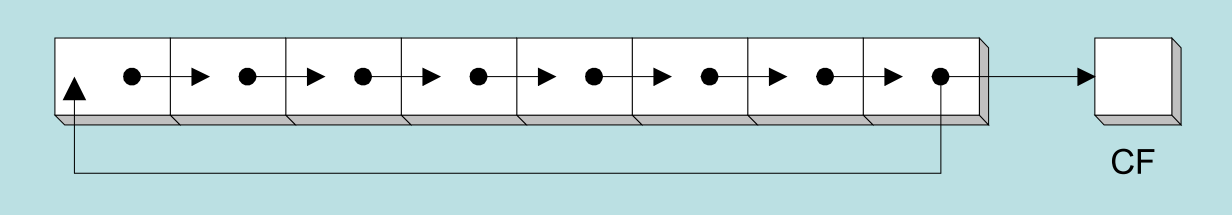
- ROL is the Rotate Left instruction
  - Rotates each bit to the left, according to the count operand
  - Highest bit is copied into the Carry Flag and into the Lowest Bit
- No bits are lost



```
mov al,11110000b
rol al,1               ; AL = 11100001b, CF = 1
mov dl,3Fh             ; DL = 00111111b
rol dl,4               ; DL = 11110011b = F3h, CF = 1
```

ROR Instruction

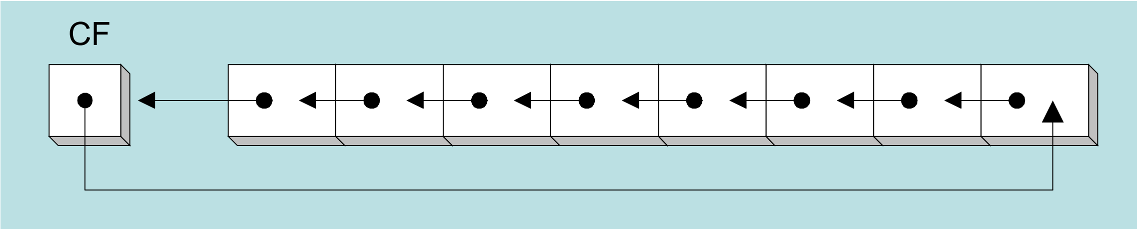
- ROR is the Rotate Right instruction
  - Rotates each bit to the right, according to the count operand
  - Lowest bit is copied into the Carry flag and into the highest bit
- No bits are lost



```
mov al,11110000b
ror al,1                ; AL = 01111000b, CF = 0
mov dl,3Fh              ; DL = 00111111b
ror dl,4                ; DL = F3h, CF = 1
```

RCL Instruction

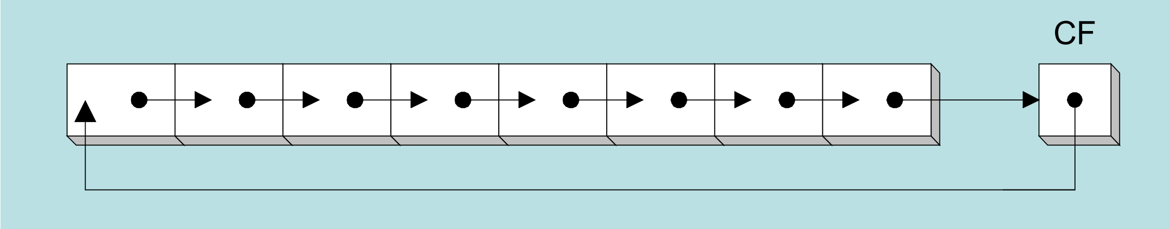
- RCL is the Rotate Carry Left instruction
  - Rotates each bit to the left, according to the count operand
  - Copies the Carry flag to the least significant bit
  - Copies the most significant bit to the Carry flag
- As if the carry flag is part of the destination operand



```
clc                 ; clear carry, CF = 0
mov bl,88h          ; BL = 10001000b
rcl bl,1            ; CF = 1, BL = 00010000b
rcl bl,2            ; CF = 0, BL = 01000010b
```

RCR Instruction

- RCR is the Rotate Carry Right instruction
  - Rotates each bit to the right, according to the count operand
  - Copies the Carry flag to the most significant bit
  - Copies the least significant bit to the Carry flag
- As if the carry flag is part of the destination operand



```
stc                  ; set carry, CF = 1
mov ah,11h           ; AH = 00010001b
rcr ah,1             ; CF = 1, AH = 10001000b
rcr ah,3             ; CF = 0, AH = 00110001b
```

Effect of Rotate Instructions on Flags

- The CF is the last bit shifted
- The OF is defined for single bit rotates only
  - It is 1 if the sign bit changes
- The ZF, SF, PF and AF are unaffected

SHLD Instruction

- SHLD is the Shift Left Double instruction
- Syntax: **SHLD *destination, source, count***
  - Shifts a *destination* operand a given *count* of bits to the left
- The rightmost bits of *destination* are filled by the leftmost bits of the *source* operand
- The *source* operand is not modified
- Operand types:

```
SHLD reg/mem16, reg16, imm8/CL
SHLD reg/mem32, reg32, imm8/CL
```

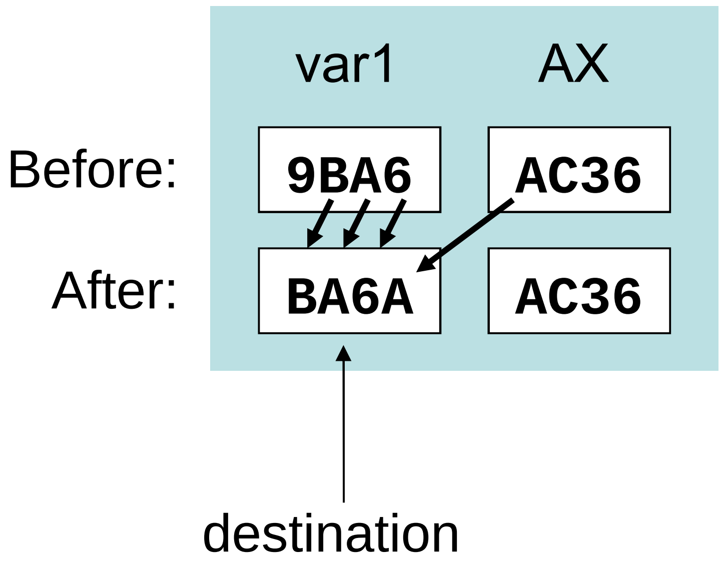
SHLD Example

Shift variable **var1** 4 bits to the left

Replace the lowest 4 bits of **var1** with the high 4 bits of AX

```
.data
var1 WORD 9BA6h
.code
mov  ax, 0AC36h
shld var1, ax, 4
```

destination source count



var1 AX

Before: 9BA6 AC36

After: BA6A AC36

destination

Only the *destination* is modified, not the *source*

SHRD Instruction

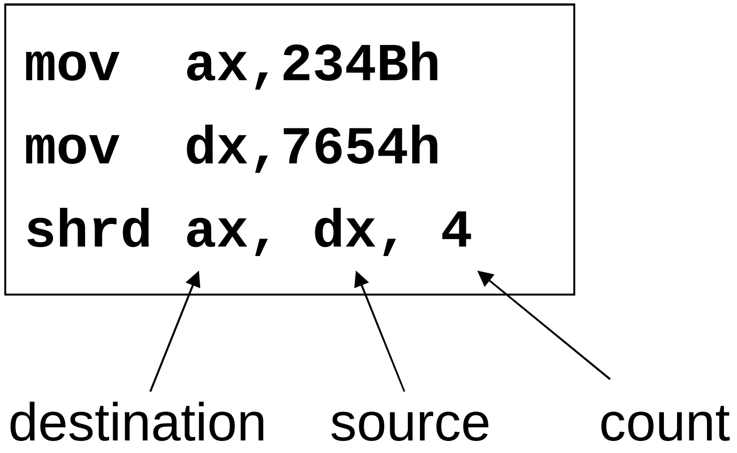
- SHRD is the Shift Right Double instruction
- Syntax: **SHRD** ***destination, source, count***
  - Shifts a *destination* operand a given *count* of bits to the right
- The leftmost bits of *destination* are filled by the rightmost bits of the *source* operand
- The *source* operand is not modified
- Operand types:

```
SHRD reg/mem16, reg16, imm8/CL
SHRD reg/mem32, reg32, imm8/CL
```

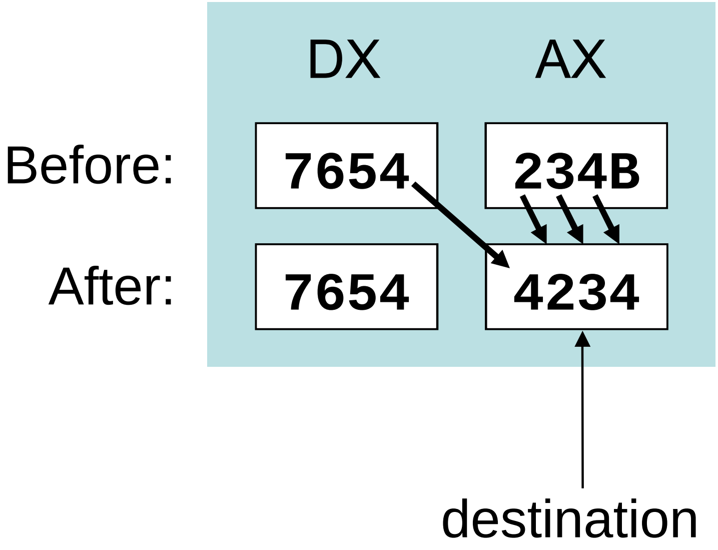
SHRD Example

Shift AX 4 bits to the right

Replace the highest 4 bits of AX with the low 4 bits of DX



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



Only the *destination* is modified, not the *source*

Your Turn . . .

Indicate the values (in hex) of each destination operand

```
mov  ax,7C36h
mov  dx,9FA6h
shld dx,ax,4      ; DX = FA67h
shrd ax,dx,8      ; AX = 677Ch
```

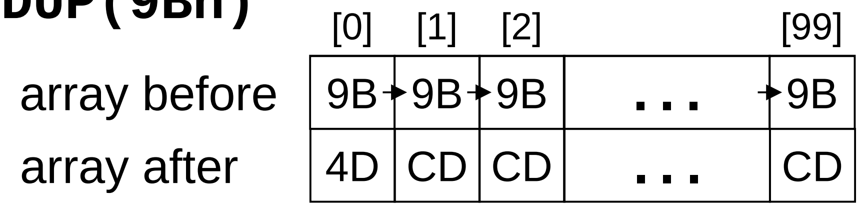
Next . . .

- ❖ Shift and Rotate Instructions
- ❖ Shift and Rotate Applications
- ❖ Multiplication and Division Instructions
- ❖ Translating Arithmetic Expressions
- ❖ Decimal String to Number Conversions

Shifting Bits within an Array

- ❖ Sometimes, we need to shift all bits within an array
  - ✧ Example: moving a bitmapped image from one screen to another
- ❖ Task: shift an array of bytes 1 bit right

```
.data
    ArraySize  EQU 100
    array BYTE ArraySize DUP(9Bh)
.code
    mov ecx, ArraySize
    mov esi, 0
    clc                 ; clear carry flag
L1:
    rcr array[esi], 1   ; propagate the carry flag
    inc esi             ; does not modify carry
    loop L1             ; does not modify carry
```



	[0]	[1]	[2]		[99]
array before	9B	9B	9B	. . .	9B
array after	4D	CD	CD	. . .	CD

Binary Multiplication

- You know that SHL performs multiplication efficiently
  - When the multiplier is a power of 2
- You can factor any binary number into powers of 2
  - Example: multiply EAX by 36
    - Factor 36 into (4 + 32) and use distributive property of multiplication
  - EAX * 36 = EAX * (4 + 32) = EAX * 4 + EAX * 32

```
mov ebx, eax          ; EBX = number
shl eax, 2            ; EAX = number * 4
shl ebx, 5            ; EBX = number * 32
add eax, ebx          ; EAX = number * 36
```

Your Turn . . .

Multiply EAX by 26, using shifting and addition instructions

Hint: 26 = 2 + 8 + 16

```
mov  ebx, eax             ; EBX = number
shl  eax, 1               ; EAX = number * 2
shl  ebx, 3               ; EBX = number * 8
add  eax, ebx             ; EAX = number * 10
shl  ebx, 1               ; EBX = number * 16
add  eax, ebx             ; EAX = number * 26
```

Multiply EAX by 31, Hint: 31 = 32 – 1

```
mov  ebx, eax             ; EBX = number
shl  eax, 5               ; EAX = number * 32
sub  eax, ebx             ; EAX = number * 31
```

Convert Number to Binary String

Task: Convert Number in EAX to an ASCII Binary String

Receives: EAX = Number
ESI = Address of binary string

Returns: String is filled with binary characters ’0’ and ’1’

```
ConvToBinStr PROC USES ecx esi
    mov  ecx,32
L1: rol  eax,1
    mov  BYTE PTR [esi],'0'
    jnc  L2
    mov  BYTE PTR [esi],'1'
L2: inc  esi
    loop L1
    mov  BYTE PTR [esi], 0
    ret
ConvToBinStr ENDP
```

Rotate left most significant bit of EAX into the Carry flag; If CF = 0, append a ’0’ character to a string; otherwise, append a ’1’; Repeat in a loop 32 times for all bits of EAX.