# Logic Instructions

- The logic instructions include
  - AND
  - OR
  - XOR (Exclusive-OR)
  - NOT

| Mnemonic | Meaning | Format | Operation | Flags affected |
|---|---|---|---|---|
| AND | Logical AND | AND D, S | (S) ·(D)→(D) | OF, SF, ZF, PF, CF<br>AF undefined |
| OR | Logical Inclusive-OR | OR D, S | (S) ·(D)→(D) | OF, SF, ZF, PF, CF<br>AF undefined |
| XOR | Logical exclusive-OR | XOR D, S | (S)⊕(D)→(D) | OF, SF, ZF, PF, CF<br>AF undefined |
| NOT | Logical NOT | NOT D | (NOT D)→(D) | None |

# Logic Instructions

Truth tables for AND, OR, XOR, NOT

| | | |
|---|---|---|
| 0 AND 0 = 0 | 0 OR 0 = 0 | 0 XOR 0 = 0 |
| 0 AND 1 = 0 | 0 OR 1 = 1 | 0 XOR 1 = 1 |
| 1 AND 0 = 0 | 1 OR 0 = 1 | 1 XOR 0 = 1 |
| 1 AND 1 = 1 | 1 OR 1 = 1 | 1 XOR 1 = 0 |

NOT 0 = 1 ; NOT 1 = 0

- OR
  - If you OR in a 1 bit, you force the result to be a 1 bit
  - If you OR in a 0 bit, you do not change the original bit
  - Thus OR can be used to turn bits on (SET a bit)
- AND
  - If you AND in a 1 bit, you do not change the original bit
  - If you AND in a 0 bit, you force the result to be a 0 bit
  - Thus AND can be used to turn bits off (CLEAR bits)
- XOR
- If you XOR a bit with itself, you force the result to be a 0 bit
- If you XOR a 0 bit, you do not change the original bit
- If you XOR a 1 bit, you flip the state of the original bit
- Thus XOR can be used to COMPLEMENT individual bits or to CLEAR a group of bits

- NOT
  - NOT flips the state of the original bits
  - Thus NOT can be used to COMPLEMENT all the bits in an operand
- To modify bits in the destination field, you usually construct a source bit pattern called a MASK field
- The MASK field can be expressed in Binary, Hex, or Decimal. For example, the following three mask fields are all equivalent:

  01000000b in Binary is the same as

  40h in Hex is the same as

  64 in Decimal

NOT: 1's complement
Logic inversion

AND and OR instructions are often used to mask out data

- a mask value is used to force certain bits to zero or one within some other value
- a mask typically affects certain bits and leaves other bits unaffected
  - AND forces selected bits to zero AND CL, 0Fh
  - OR forces selected bits to one OR CL, 0Fh

NEG: 2's complement
Logic inversion + 1

# Logic examples

| AL | 1100 1010 |
|---|---|
| NOT AL | |
| AL | 0011 0101 |

| AL | 0011 0101 |
|---|---|
| BL | 0110 1101 |
| AND AL, BL | |
| AL | 0010 0101 |

| AL | 0011 0101 |
|---|---|
| BL | 0000 1111 |
| AND AL, BL | |
| AL | 0000 0101 |

| AL | 0011 0101 |
|---|---|
| BL | 0110 1101 |
| OR AL, BL | |
| AL | 0111 1101 |

| AL | 0011 0101 |
|---|---|
| BL | 0000 1111 |
| OR AL, BL | |
| AL | 0011 1111 |

| AL | 0011 0101 |
|---|---|
| BL | 0110 1101 |
| XOR AL, BL | |
| AL | 0101 1000 |

1. Before the instruction executes: AX: 8935

```
AND  AX,0000h   A MASK of 0000000000000000b is ANDed
with            the AX register
```

After the instruction executes: AX: 0000

2. Before the instruction executes: AL: 01

```
OR  AL,40h      A MASK of 01000000b is ORed with
                the AL register
```

After the instruction executes: AL: 41

3. Before the instruction executes: AX: 7849

```
XOR AX,AX       The AX register is XORed with itself
```

After the instruction executes: AX: 0000

4. Convert the ASCII digit in AL to a pure number (remember the ASCII codes for 0-9 are 30h-39h)

```
SUB     AL,30h  or
AND     AL,00001111b or
AND  AL,Ofh
```

5. Convert the digit in AL to its ASCII code

```
OR  AL,00110000b or
OR  AL,30h or
ADD  AL,30h
```

6. Convert the upper case letter at CHAR to its corresponding lower case letter

```
OR  CHAR,00100000b  or
OR  CHAR,20h  or
ADD  CHAR,20h
```

## Shift Instructions

- Shift instructions: SHL, SHR, SAL, SAR

| Mnemonic | Meaning | Format | Operation | Flags affected |
|---|---|---|---|---|
| SAL/SHL | Shift arithmetic left / Shift logical left | SAL D, Count<br>SHL D, Count | Shift the (D) left by the number of bit positions equal to Count and fill the vacated bits positions on the right with zeros | CF, PF, SF, Z<br>AF undefined<br>OF undefined if count ≠1 |
| SHR | Shift logical right | SHR D, Count | Shift the (D) right by the number of bit positions equal to Count and fill the vacated bits positions on the left with zeros | CF, PF, SF, Z<br>AF undefined<br>OF undefined if count ≠1 |
| SAR | Shift arithmetic right | SAR D, Count | Shift the (D) right by the number of bit positions equal to Count and fill the vacated bits positions on the left with the original most significant bits | CF, PF, SF, Z<br>AF undefined<br>OF undefined if count ≠1 |

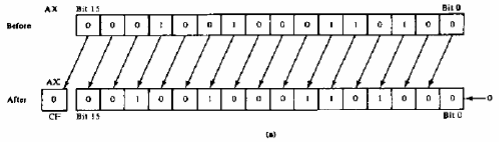
## Shift Instructions

- Shift instructions: SHL, SHR, SAL, SAR

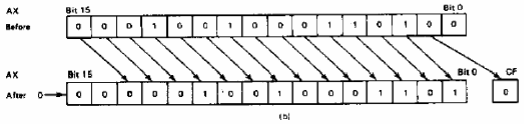
| Destination | Count |
|---|---|
| Register | 1 |
| Register | CL |
| Memory | 1 |
| Memory | CL |

Allowed operands for shift instructions

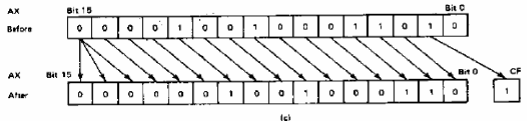
## Shift instructions: SHL, SHR, SAL, SAR



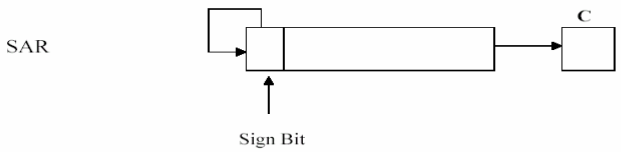

SHL AX, 1




SHR AX, CL




SAR AX, CL

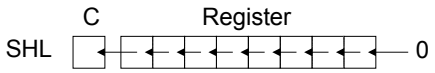


# Shift left

- Syntax: SHL reg, count (immediate count)
  SHL reg, CL (count stored in CL)
  - moves each bit of the operand one bit position to the left the number of times specified by the count operand
  - zeros fill vacated positions at the L.O. bit
  - H.O. bit shifts into the carry flag
- a quick way to multiply by two
- useful in *packing data,* e.g., consider two nibbles in AL and AH that we want to combine

```
shl ah, 4
or al,ah
```

- *the 8086 and 8088 allow an immediate shift of 1 only*

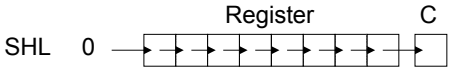



# Shift right

- Syntax: SHR reg, count (immediate count)
  SHR reg, CL (count stored in CL)
  - moves each bit of the operand one bit position to the right the number of times specified by the count operand
  - zeros fill vacated positions at the H.O. bit
  - L.O. bit shifts into the carry flag
- a quick way to divide by two
- useful in *unpacking data,* e.g., suppose you want to extract the two nibbles in the AL register, leaving the H.O. nibble in AH and the L.O. nibble in AL

```
mov ah, al
shr ah, 4
and al, 0Fh
```

- the 8086 and 8088 allow an immediate shift of 1 only



# Shift arithmetic right

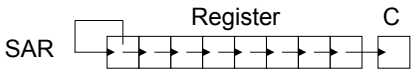
- Syntax: SAR reg, count (immediate count)
  SAR reg, CL (count stored in CL)
  - moves each bit of the operand one bit position to the left the number of times specified by the count operand
  - H.O. bit replicates
  - L.O. bit shifts into the carry flag
- main purpose is to perform a *signed division* by some power of two

```
mov ax, -15
sar ax, 1          ; result is -8
```

- In 80286 and later you can use SAR to sign extend one register into another

```
mov ah, al
sar ah, 7
```

if AL contains 1111 0001 then AX will be 1111 1111 1111 0001 after execution




1. Shift the contents of register AX to the left by 3 bits

Before: AX: 0001 CL:??

```
MOV  CL,3     ;Puts a 3 into register CL
SHL  AX,CL    ;Shifts register AX 3 bits to the left
```

After: AX: 0008 CL:03

2. Multiply the contents of BX by 2

Before: BX: 0001

```
SAL BX,1     ;Shifts register BX 1 bit to the left
```

After: BX: 0002

SAL is equivalent to multiplying by 2.

SAR is equivalent to dividing by 2. The sign bit is retained.

Assume that CL contains $02_{16}$ and AX contains $091_{16}$. Determine the new contents of AX and the carry flag after the instruction SAR AX, CL is executed.

Solution:

$$(AX)=0000001001000110_2=0246_{16}$$

and the carry flag is $(CF)=1_2$

EXAMPLE

Verify the previous example using DEBUG program.

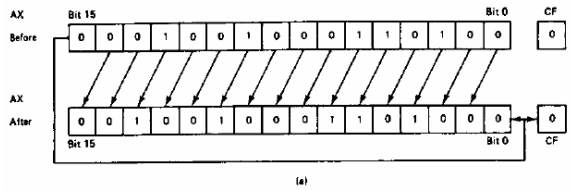
Solution:


# 5.5 Rotate Instructions

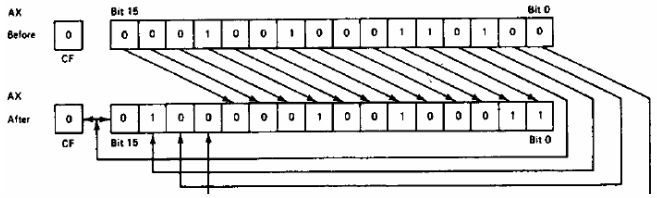
- Rotate instructions: ROL, ROR, RCL, RCR

| Mnemonic | Meaning | Format | Operation | Flags affected |
|---|---|---|---|---|
| ROL | Rotate left | ROL D, Count | Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the leftmost bit goes back into the rightmost bit position. | CF<br>OF undefined if count ≠1 |
| ROR | Rotate right | ROR D, Count | Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes back into the leftmost bit position. | CF<br>OF undefined if count ≠1 |
| RCL | Rotate left through carry | RCL D, Count | Same as ROL except carry is attached to (D) for rotation. | CF<br>OF undefined if count ≠1 |
| RCR | Rotate right through carry | RCR D, Count | Same as ROR except carry is attached to (D) for rotation. | CF<br>OF undefined if count ≠1 |

- Rotate instructions: ROL, ROR, RCL, RCR

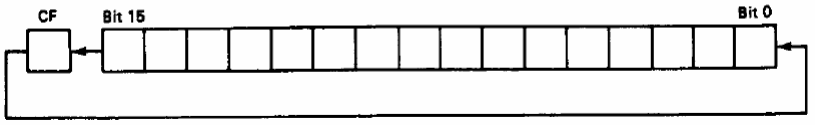



ROL AX, 1



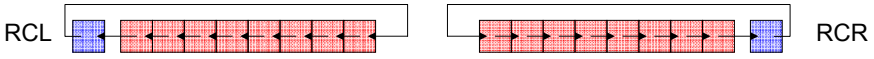

ROL AX, CL
(CL)=4

For RCL, RCR, the bits are rotate through the carry flag



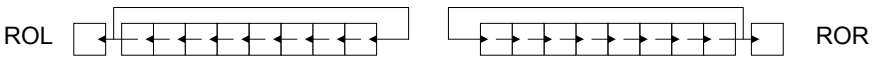

# Rotate through carry L/R

- Syntax: RCL reg, count (immediate count)
  RCL reg, CL (count stored in CL)
  - rotates bits to the left, through the carry flag
  - bit in the carry flag is written back into bit zero on the right
- Syntax: RCR reg, count (immediate count)
  RCR reg (count stored in CL)
  - rotates bits to the right, through the carry flag
  - bit in the carry flag is written back into the H.O. bit on the left



# Rotate left/right

- Syntax: ROL reg, count (immediate count)
  ROL reg, CL (count stored in CL)
  - rotates bits to the left
  - H.O. bit goes to carry flag and L.O. bit
- Syntax: ROR reg, count (immediate count)
  ROR reg (count stored in CL)
  - rotates bits to the right
  - L.O. bit goes to carry flag and H.O. bit




- `mov ah,40h` ;ah = 0100 0000b
- `rol ah,1` ;ah = 1000 0000b, CF = 0
- `rol ah,1` ;ah = 0000 0001b, CF = 1
- `rol ah,1` ;ah = _____b,
  CF = ___
- `mov ax,1234h` ;ax = 0001 0010 0011 0100b
  = 1234h
- `ror ax,4` ;ax = 4123h
- `ror ax,4` ;ax = 3412h
- `ror ax,4` ;ax = 2341h

EXAMPLE

What is the result in BX and CF after execution of the following instructions?

RCR BX, CL

Assume that, prior to execution of the instruction, (CL)=$04_{16}$, (BX)=$1234_{16}$, and (CF)=0

Solution:

The original contents of BX are

(BX) = $0001001000110100_2 = 1234_{16}$

Execution of the RCR command causes a 4-bit rotate right through carry to take place on the data in BX, the results are

(BX) = $1000000100100011_2 = 8123_{16}$

(CF) = $0_2$

```
-
-A
0B35:0100 RCR BX,CL
0B35:0102
-R BX
BX 0000
:1234
-R CX
CX 0000
:4
-R F
NV UP EI PL NZ NA PO NC  -
-T

AX=0000  BX=8123  CX=0004  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B35  ES=0B35  SS=0B35  CS=0B35  IP=0102   OV UP EI PL NZ NA PO NC
0B35:0102 D3F8          SAR     AX,CL
-
```

;write a program that counts the number of 1's in a byte and writes it into BL

```
DATA1  DB 97                ; 61h
       SUB   BL,BL          ;clear BL to keep the number of 1s
       MOV    DL,8          ;rotate total of 8 times
       MOV    AL,DATA1
AGAIN: ROL    AL,1          ;rotate it once
       JNC    NEXT          ;check for 1
       INC    BL            ;  if CF=1 then add one to count
NEXT:  DEC    DL            ;go through this 8 times
       JNZ    AGAIN         ;if not finished go back
       NOP
```