EE 390 : Digital System Engineering
Handout 6 by Dr Sheikh Sharif Iqbal

Reference to text book: The 8088 and 8086 Microprocessors…. by Triebel and Singh

Ref: Online course on EE 390 (KFUPM)

4.1. Converting Assembly language instruction to machine code:

Figure 4-1 to 4-6: See page 104 to 113 of text book (4-th Edition)

- Opcode field (6-bit): specifies the operation such as ADD, MOV etc.
- D bit: specifies if the register operand specified by REG in byte 2 is the source of destination operand.
- W bit: Specifies if the operation will be performed on 8-bit or 16-bit data.
- MOD (mode), REG (register) and R/M (register/memory) fields are used to specify which register is used for the 1st operand and where the 2nd operand is stored.
- SEE example 4.1 to 4.4

4.3. DEBUG: program is part of computers disk operating system (DOS), which allows us to enter assembly language programs, assemble it and execute it. We can also debug any errors in the program using this program.

Important DEBUG commands: Register command ➔ 'R' ; Assemble command ➔ 'A'; Unassembled command ➔ 'U' ; For Memory contents => Dump command ➔ 'D', Fill command ➔ 'F' ; Enter command ➔ 'E' etc.. (see pg 118 of book or lab manual)

The difference between DEBUG and TASM Program: In **Debug** the user has to be familiar with the program to use it, where as in **TASM** we can generate executable programs to be executed by an armature.

Chapter 5: Instruction set of 8088/8086 microprocessors

5.1. Data Transfer Instructions: Flags are not affected

- MOV (*copy*) instruction; MOV D,S ➔ (S) => (D) ; See book figure 5.1
- XCHG (exchange) Instruction: XCHG D,S ➔ (S) ⇔ (D) ; See figure 5.4
- XLAT (translate) instruction: will be covered after 'DB' instruction
- LEA (load effective address): calculates and loads the effective or offset address part of the physical address specified as the source-operand. Exam 1: LEA AX,[ES:49_H]
  After execution ➔ AX=49_H

Example2: LEA SI,[DI], if DI=1234_H, after executing this instruction SI➔1234_H (not the memory content of P.A.=DS:SI, as will be the case if LEA is replaced with MOV)

- LDS (load destination-operand-register and data-segment): loads the 1st word pointed by the physical address specified by the source-operand in to the destination-operand-register and the following word into the **DATA** segment register. See book fig. 5-8(a)

- LES (load operand-register and data-segment): loads the 1st word pointed by the physical address specified by the source-operand in to the destination-operand-register and the following word into the **EXTRA** segment register.

5.1. Arithmetic Instructions: Addition Instruction ➔ Flags are affected

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	**ADD D,S**	(S)+(D) ➔ (D) carry ➔ (CF)	ALL
ADC	Add with carry	**ADC D,S**	(S)+(D)+(CF) ➔ (D) carry ➔ (CF)	ALL
INC	Increment by one	**INC D**	(D)+1 ➔ (D)	ALL but CY
AAA	ASCII adjust for addition	**AAA**	If the sum is >9, AH is incremented by 1	AF,CF
DAA	Decimal adjust for addition	**DAA**	Adjust AL for decimal Packed BCD	ALL

Note: The destination on all addition instructions can not be immediate number and no memory to memory can be added.

ADD Ins.: Add. Example; ADD SI,[DI] ➔ content of SI register is added with the memory content of P.A.=DS:DI AND the result is placed as a content of SI register. If any carry has occurred, it will be placed into the CF. **Fig 5-14**

ADC Ins.: Add with carry. Example; ADC SI,[DI] ➔ content of SI register is added with the contents of CF and the result is then added with the memory contents of P.A.=DS:DI. The obtained result is placed as a content of SI register. **If any carry** has occurred in the addition process, it will be placed in the **CF**.

HW: Write a program to add the **contents** of the double word: AX,BX with the contents of another double word CX,DX and store the resulted data into the double word: AX,DX

As an example say AX,**BX** = 0123**4567**$_H$ and CX,**DX** = FEDC**BA98**$_H$

SPECIAL NOTE: Byte Ptr [2345_H] ➔ points to a byte data stored in P.A.= DS:2345_H

and Word Ptr [$87A4_H$] ➔ points to a word data stored in P.A.= DS:$87A4_H$

AAA Ins.: ASCII Adjust Addition *instruction*, is used to add to ASCII coded numbers. So in two ASCII numbers are to be added, after ADD instruction (which adds in terms of binary) use AAA instruction to convert the results into correct format.

Example: Add two ASCII values stored in AL and BL registers and store the results to AL register.

Solution: Say, AL = ***2*** $_{(\text{in ASCII})}$= 32_H *and* BL= **6** $_{(\text{in ASCII})}$ = 36_H

```
MOV AL,32H  ; Enter the 1st ASCII value in terms of hex
MOV BL,36H  ; Enter the 2nd ASCII value in terms of hex
ADD AL,BL   ; this will result in AL = 68H
AAA         ; this will adjust the result to AL = 8H
```

Remember, if the SUM is greater than "9_D", AL register contains the Least-significant-digit of result and AH register in incremented by '1_D' ***(Page 189)***

DAA Ins.: Decimal Adjust Addition *instruction*, is used to add to packed BCD (*two BCD numbers packed in one byte*) numbers.

Example: Add two BCD values stored in AL and BL registers and store the results to AL register.

Solution: Say, AL = **47** $_{(\text{in BCD})}$ *and* BL= **24** $_{(\text{in BCD})}$

```
MOV AL,47H  ; Enter the 1st BCD value in terms of hex
MOV BL,24H  ; Enter the 2nd BCD value in terms of hex
ADD AL,BL   ; this will result in AL = 6BH
DAA         ; this will adjust the result to AL = 71H
```

Remember, to include the values of CF into the result.

Example 2:

```
MOV AL,47H  ; Enter the 1st BCD value in terms of hex
MOV BL,59H  ; Enter the 2nd BCD value in terms of hex
ADD AL,BL   ; this will result in AL = A0H
DAA         ; adjust result is AL=06H  AND CF=CY  ➔106
```

Assignment: Find out the process undertaken by the software to adjust the results when 'AAA' and 'DAA' instructions are executed. (*two different explanations*)

Subtraction Instructions: Flags are also affected.

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D) - (S) ➔ (D) Borrow ➔ (CF)	All
SBB	Subtract with borrow	SBB D,S	(D) - (S) - (CF) ➔ (D)	All
DEC	Decrement by one	DEC D	(D) - 1 ➔ (D)	All but CF
NEG	Negate	NEG D	0 – (D) ➔ (D)	All
DAS	Decimal adjust for subtraction	DAS	Convert the result in AL to packed decimal format	All
AAS	ASCII adjust for subtraction	AAS	(AL) difference (AH) dec by 1 if borrow	CY, AC

Note: The subtract (SUB) instruction is used to subtract the **source** from the **destination**

SUB Ins.: Subtract. Example; SUB SI,[DI] ➔ the memory-content of P.A.=DS:DI will be subtracted from the content of SI register AND the result is placed as new content of SI register. If any carry/borrow occurs, it will be placed into the CF

SBB Ins.: Subtract with Borrow. Example; SBB SI,[DI] ➔ the memory-content of P.A.=DS:DI *AND* the content of CF will be SUBTRACTED from the content of SI register the result is stored as a new content of SI register. **If any Borrow** occurs in the subtraction process, it will be placed in the **CF**.

Solve the example problem of 5-12 in page 194 of the book.

NEG Ins.: Negate. Example; NEG AX ➔ the negative value of AX will be stored in AX. This instruction is used to perform 2's complement. (*2's comp. effects C.Flag*)

AAS Ins.: ASCII Adjust Subtraction. Used to adjust the result for ASCII subtraction.

DAS Ins.: Decimal adjust Subtraction. Used to adjust the result for P. BCD subtraction.

Exercise: Perform a subtraction of two 32 bit (double words) numbers stored in memory. Double word stored in [DS:35H] – double word stored in [DS:35H].

Multiplication Instructions: Flags are also affected.

Instruction	Meaning	Format	Operation	Flags
MUL	Multiply (unsigned)	MUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
DIV	Division (unsigned)	DIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	(AL) · (S8) → (AX) (AX) · (S16) → (DX),(AX)	OF, CF SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) Q((AX)/(S8)) → (AL) R((AX)/(S8)) → (AH) (2) Q((DX,AX)/(S16)) → (AX) R((DX,AX)/(S16)) → (DX) If Q is positive and exceeds $7FFF_{16}$ or if Q is negative and becomes less than 8001_{16}, then type 0 interupt occurs	OF, SF, ZF, AF, PF, CF undefined
AAM	Adjust AL for multiplication	AAM	Q((AL)/10) → (AH) R((AL)/10) → (AL)	SF, ZF, PF OF, AF,CF undefined
AAD	Adjust AX for division	AAD	(AH) · 10 + (AL) → (AL) 00 → (AH)	SF, ZF, PF OF, AF, CF undefined
CBW	Convert byte to word	CBW	(MSB of AL) → (All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX) → (All bits of DX)	None

- Multiply and Divide instructions works by default with AX (byte operand) and DX,AX (for word)
- Also we operate on Signed data using IMUL and IDIV Ins. and Unsigned data (MUL and DIV)

Operands

Source
Reg8
Reg16
Mem8
Mem16

Max.

MOV AX,$56F4_H$
MOV BX,$F3A4_H$
MUL BX

After execution:
AX = 5050_H
DX = $52C1_H$

MOV AX,$01FF_H$
MOV BL,02_H
DIV BL,

After execution:
AL =FF_H
AH = 01_H

MUL Ins.: Unsigned data Multiplication. *Example 1;* MUL BL ➔ byte-content of BL will be multiplied with the byte-content of AL register *AND* the resulted word will be stored in AX register. If carry occurs, CF flag becomes CY (of set)
Example 2; MUL CX ➔ the word-content of CX will be multiplied with the word-content of AX register *AND* the resulted double-word will be stored in DX,AX register. *If carry occurs, CF flag becomes CY (of set)*

DIV Ins.: Unsigned data Division. *Example 1;* DIV BL ➔ word-content of AX will be divided by the byte-content of BL register *AND* the resulted quotient will be

stored in AL and resulted remainder will stored in AH register.

Example 2; DIV CX ➔ the double-word-content of DX,AX will be divided by the word-content of CX register *AND* the resulted quotient will be stored in AX and the resulted remainder will stored in DX register. If carry ➔ CF

Note: A divide-by-zero errors occur if the resulted quotient is out of range.

IMUL Ins.: Signed data Multiplication. *Example;* if BL= -2_H= FE_H and AL= -3_H= FD_H, **IMUL BL** ➔ AX=0006_H. (*whereas,* **MUL BL** ➔ AX=$FB06_H$)

IDIV Ins.: Signed data Division. *Example;* if BX= -2_H = $FFFE_H$ and DX,AX= -8_H = $FFFFFFF8_H$, **IDIV BX** ➔ $(AX)_{Quotient} = 0004_H$. $(AH)_{Remainder} = 0000_H$ (*whereas,* **DIV BX** gives an error of 'division by zero' as the resulted quotient is greater than $7FFF_H$)

Note: A divide-by-zero errors occur if the resulted quotient is out of range.

CBW Ins.: Convert Byte to Word. By default, the byte stored in AL register is converted. *Example;* if AL= -2_H = FE_H = **1**1111110_B. Executing **CBW** instruction converts this byte to word by filling AH register with the M.S.Bit of AL register. Thus, AH = **11111111**$_B$= FF_H. Thus the resulted word is AX=$FFFE_H$

CWD Ins.: Convert Word to Double-word. By default, the word stored in AX register is converted. *Example;* if AX = $7FF2_H$ = **0**111111111110010_B. Executing **CWD** instruction converts this word in to double word by filling DX register with the M.S.Bit of AX register. Thus, DX = **00000000**$_B$= 00_H and resulted double-word is, DX,AX = $00007FF2_H$. *Note: Solve example 5.18 in page 203 of book*

Example 1: Assume AX = 0081H, BX = 0026H for question 1 and 2

1. MUL BL → AL . BL = 81H * 26H = 1300H → AX = 1326H

2. IMUL BL → AL . BL = 2'S AL * BL = 2'S (81H) * 26H
   = 7FH * 26H = 12DAH→ 2's comp → ED26H → AX.

Assume AX = 0085H, BX = 0035H for question 3 and 4

3. DIV BL → $\frac{AX}{BL} = \frac{0085H}{35H}$ = 02 (85-02*35=1B) → 1B02 ➔ AX

4. IDIV BL → $\frac{AX}{BL} = \frac{0085H}{35H}$ = 1B02 ➔ AX

Logic Instructions: Flags are affected.

Mnemonic	Meaning	Format	Operation	Flags Affected
AND	Logical AND	AND D, S	(S) · (D) → (D)	OF, SF, ZF, PF, CF AF undefined
OR	Logical Inclusive OR	OR D, S	(S) + (D) → (D)	OF, SF, ZF, PF, CF AF undefined
XOR	Logical Exclusive OR	XOR D, S	(S) ⊕ (D)→(D)	OF, SF, ZF, PF, CF AF undefined
NOT	LOGICAL NOT	NOT D	- (D) → (D)	None

Note: This 88/86 instructions perform **bit by bit** logic operation on the specified source and destination operands

AND Ins.: Bit by Bit logical AND operation. *Example;* if AL= FE_H = 11111110_B and BL= 2_H = 00000010_B. Executing, **AND AL,BL** instruction gives, AL = 00000010_B = 2_H

OR Ins.: Bit by Bit logical OR operation. *Example;* if AL= -2_H = 11111110_B and BL= 2_H = 00000010_B. Executing, **OR AL,BL** instruction gives, AL = $11111\mathbf{1}0_B$ = FE_H

XOR Ins.: Bit by Bit logical XOR operation. Only difference between OR and XOR instruction is that '1 OR 1 = 1' and '1 XOR 1 = 0'. *Example;* if AL = FE_H = 11111110_B and BL = 2_H = 00000010_B. Executing, **XOR AL,BL** instruction gives, AL = $11111\mathbf{0}0_B$ = FC_H

NOT Ins.: Bit by Bit logical NOT (inversion) operation. Used to perform 1's complement. *Example;* if AL = EE_H = 11101110_B, executing, **NOT AL** instruction gives, AL = 00010001_B = 11_H

Note: Solve the example 5.20 of page 210 ➔➔

Assambly instructions	(AL)
MOV AL,01010101B	01010101
AND AL,00011111B	00010101
OR AL,11000000B	11010101
XOR AL,00001111B	11011010
NOT AL	00100101

Example: Clear bits 0 and 1, set bits 6 and 7 and invert bit 5 of register CL

Sol:
AND CL, 1111 1100$_B$; clear bits 0 and 1
OR CL, 1100 0000$_B$; set bits 6 and 7
XOR CL, 0010 0000$_B$; invert bit 5.

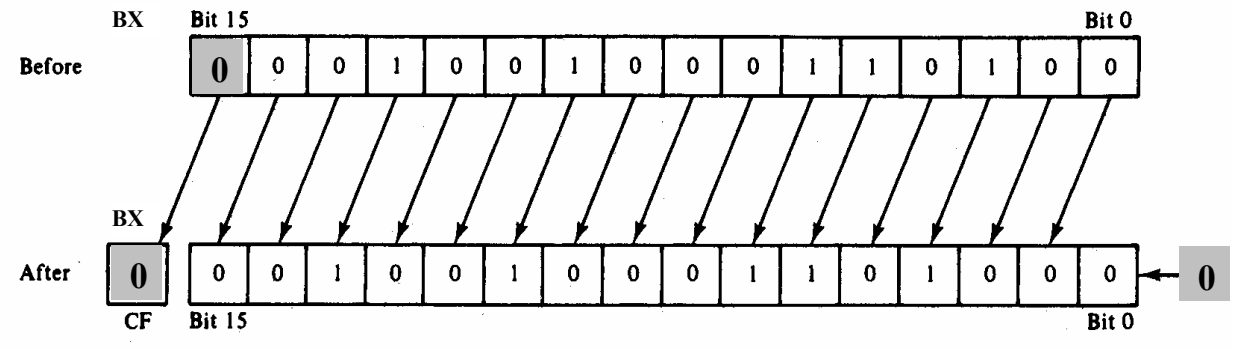
Shift Instructions: Flags are affected. Original data is lost if shift operation of performed

Mnemonic	Meaning	Format	Operation	Flags Affected
SAL/SHL	Shift arithmetic Left/shift Logical left	SAL/SHL D, CL	Shift the (D) left by the number of bit positions equal to count (CL)and fill the vacated bits positions on the right with zeros	CF,PF,SF,ZF AF undefined OF undefined if count (CL) ≠1
SHR	Shift logical right	SHR D, CL SAR D, Count	Shift the (D) right by the number of bit positions equal to count (CL) and fill the vacated bits positions on the left with zeros	CF,PF,SF,ZF AF undefined OF undefined if count (CL) ≠1
SAR	Shift arithmetic right		Shift the (D) right by the number of bit positions equal to count (CL) and fill the vacated bits positions on the left with the original most significant bit	CF,PF,SF,ZF AF undefined OF undefined if count (CL) ≠1

- Shift instructions can be used to 'test isolated bits of a register' or 'for multiplication or division operations'.

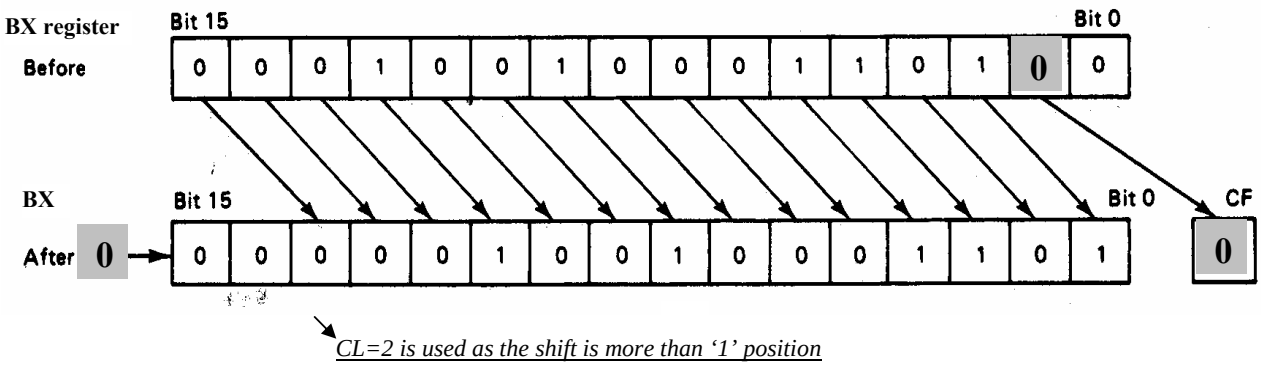
- If only one binary digit is shifted, then '1' can be used directly as a operand
- But for multiple shifts, we need to assign the count number in **CL register**

Example1: For '**SAL/SHL BX,1**' Ins. ➔ Shifted empty bits are filled with '0' and MSB goes to CF from the 1st shift. Further shift causes this data to be lost.

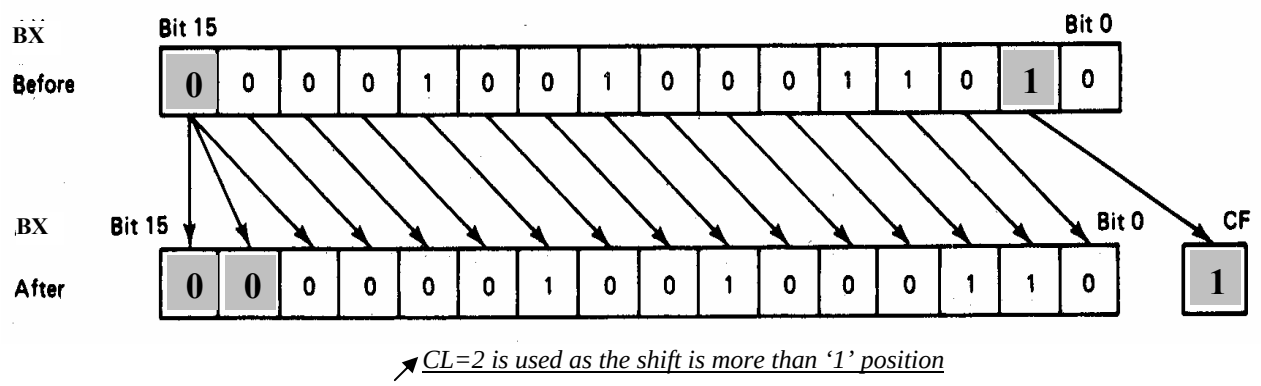


EE 390 : Digital System Engineering
Handout 9 by Dr Sheikh Sharif Iqbal

Example2: For '**SHR BX,CL**' ➔ Shifted empty bits are filled with '0' and LSB goes to CF and then data is lost.



Example3: For '**SAR BX,CL**' ➔ Shifted empty bits are filled with 'MSB value' and LSB goes to CF and then the data is lost.



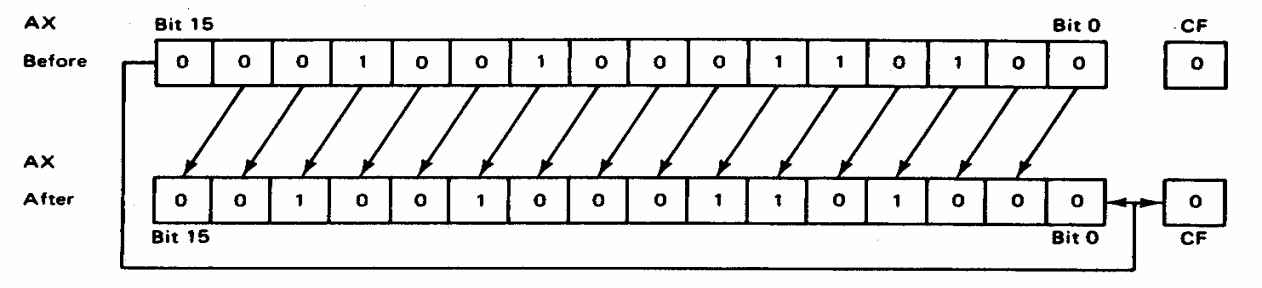
Example 4: Multiply AX by 10 using shift instructions:

Sol:
```
SHL AX, 1
MOV BX, AX
MOV CL,2          (means each shift left multiplies by 2)
SHL AX,CL
ADD AX, BX
```

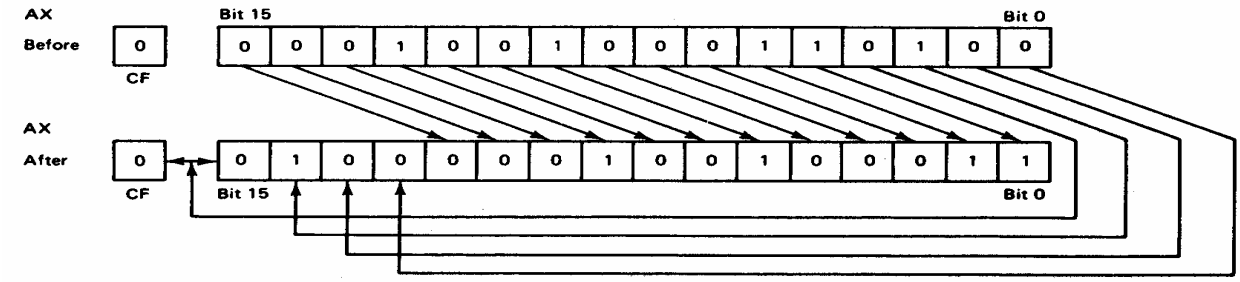
Exercise: Write a program to divide a number in AL register using shift Ins (*see lab manual for assistant*)

Rotate Instructions: Flags are affected. Original data are not lost due to this operation

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 left most bit goes back into the rightmost bit position.	CF 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 , OF undefined if count ≠ 1
RCL	Rotate left with carry	RCL D, Count	Same as ROL except carry is attached to (D) for rotation.	CF, OF undefined if count ≠ 1
RCR	Rotate right with carry	RCR D, Count	Same as ROR except carry is attached to (D) for rotation.	CF, OF undefined if count ≠ 1

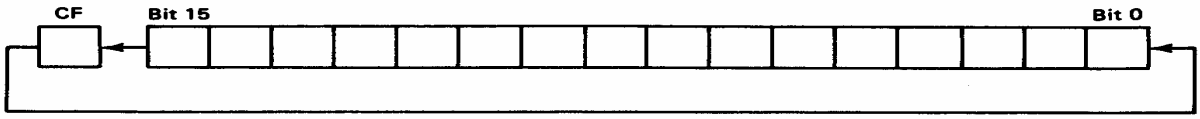


Example1: For '**ROL AX,1**' ➔ Rotated bits fill the empty slot. No data is lost.



Example2: For '**ROR AX,CX**' *(CL=4)*➔ Rotated bits fill the empty slot. No data is lost.

Rotate Instruction with carry: Data is made of '15-bit register values and CF flag content'



Example3: For '**RCR AX,1**'➔ Rotated bits include the register bits **and** the carry flag bit

Solve all the review problems of chapter 5 (pg 222-228) and pass it next class.