Lecture : 6(a) (MCQ) ***

Date : ...../...../.....

* JNZ → Controls the loop
* Labels end with colon (:)
* CMP is just like SUB
* Signed Conditional Jumps → (ZF, OF, SF)
  - JG or JNLE (ZF=0, SF=OF)
  - JGE or JNL (SF=OF)
  - JL or JNGE (SF <> OF)
  - JLE or JNG (ZF=1 or SF <> OF)

* Unsigned → (ZF, CF)
  - JA or JNBE (ZF=0, CF=0)
  - JAE or JNB (CF=0)
  - JB or JNAE (CF=1)
  - JBE or JNA (ZF=1 or CF=1)

* Sign bit of a byte in a character is always zero
* Jump is primitive.

Protiva

Single-Flag Jumps

JE or JZ	• Jump if Equal • Jump if equal to Zero	ZF = 1
JNE or JNZ	• Jump if Not Equal • Jump if Not Zero	ZF = 0
JC	• Jump if Carry	CF = 1 CF = 0
JNC	• Jump if no Carry	CF=0
JO	• Jump if Overflow	CF=1 or ZF = 1
JNO	• Jump if No Overflow	OF=1
JS	Jump if Sign Negative	SF = 1
JNS	Jump if Non-Negative Sign	SF =0
JP/JPE	Jump if Parity Even	PF=1
JNP/JPO	Jump if parity Odd	PF=0

Lecture : 6(b) → Flow control.

Date : ...../...../.....

□ What is Loop ?

⇒ A loop is a sequence of instructions that is repeated.

There are 3 types of loop -

1. For loop
2. While loop
3. Repeat loop

□ What is for - loop ?

⇒ In For loop, the loop statements are repeated a known no. of times.

```
FOR loop-count times DO
        Statements
END-FOR
```

* Counter → Register CX
# Decremented

* For loop execute → At least once

If CX = 0, the LOOP instruction causes CX to be decremented to FFFF h

→ To prevent this, JCXZ (Jump if CX is zero) used. before the loop.

```
JCXZ destination_label.
```

Protiva

Date : ...../...../.....

□ What is WHILE loop?

⇒ WHILE Loop depends on a condition. The loop is executes as long as the condition is true.

* A while loop checks the terminating condition at the top of the loop.

** WHILE_ lebel is used because WHILE is a reserved word.

```
WHILE condition DO
    Statements
END-WHILE
```

□ What is Repeat loop?

⇒ In a Repeat ... UNTILL loop, the statements are executed, and then the condition is checked.

* If true → the loop terminates
  If false → Control branceches to the top of the loop

```
REPEAT
    statements
UNTILL
    Condition
```

Date : ...../...../.....

□ Difference between WHILE and REPEAT loop?

⇒ WHILE loop

1. Check condition before each iteration
2. If the condition is initially false, may not execute.
3. Code size is a little ~~shorter~~. longer.
4. Has two jumps : Conditional JMP at the top
   - a JMP at the bottom.

REREAT loop

1. Check condition after each iteration
2. Executes at least once.
3. Code size is a little ~~longer~~. shorter
4. Has only one conditional jump.

□ Logic, shift and rotate ⟶ to do binary and hexadecimal (MCQ)

Protiva

Date : ...../...../.....

Lecture 7(a)

- Logic instruction : AND, OR, XOR, NOT
  - → Converting a lowercase letters to upper case
  - → Determining an 'Even or ODD number.

** Shifting is faster than Multiplication and Division (MCQ)**

*** Memory - memory operations are not allowed.

** Source bit patterren known as mask.

Effect on flage → SF, ZF, PF reflect the result
AF is undefined
CF, OF = 0

** Instead of SUB AL, 30H we can use AND AL, 0Fh
↳ Clean the high bits of AL

- Converting Upper Case – Lower Case
  - → SET / OR operation.

A → 41 → 0100 0001
a → 61 → 0100 0001

Set → 0010 0000 [Replace with 1, and remaining will be 0

= 20 h = Mask

Protiva

Date : ...../...../.....

☐ Converting lowercase - Uppercase

→ Clear / AND operation

a → 61 → 0110 0001
A → 41 → 0100 0001

Clear → 1101 1111 [Replace with 0 and remaing will be 1]
=0DF h
↳ Alphapet এর আগে '0' বসবে (MCQ)*

☐ Clearing a Register (MCQ)*

MOV AX, 0
SUB AX, AX
XOR AX, AX
} Same

CMP CX, 0 ; to check the sign contents.

* Test instructions performed AND operation.
* Test instructions set the status flags.

OF, CF = 0
AF = Undefined
SF, ZF, PF reflect the result.

Protiva

Date : ...../...../.....

** For a shift → the bits shifted out are lost

** For intel processors → Allow the use of an 8-bit constant.

□ SHL

→ a 0 is shifted in to the right most position and msb is shifted into CF.

SHL destination, CL

Example: Suppose DH contains 8Ah and CL contains 3.

SHL DH, CL

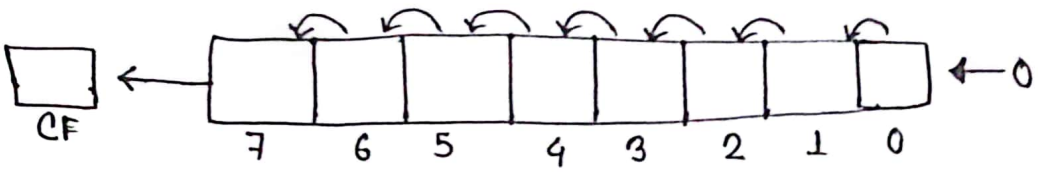
⇒ DH → 1000 1010 CF=1

0001 0100 CF=0

0010 1000 CF=0

0101 0000 CF=0

= 50H



Byte.

Protiva

Date : ...../...../.....

☐ Multiplication by left shift

AL contains 5 = 0000 0101

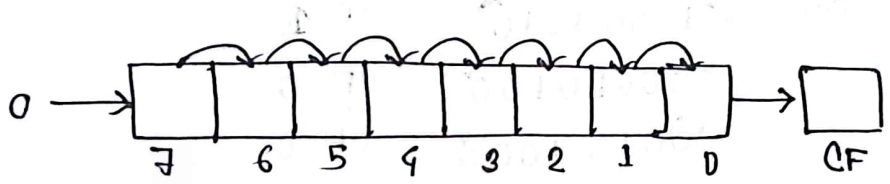
A left shift gives 0000 1010 = 10 ; Doubling its value

Another " " 0001 0100 = 20d ; Doubled again.

** Overflow flag are not reliable for this.]

☐ SHR (Right shift)

- A 0 is shifted into the msb and the rightmost bit is shifted to the CF.



Protiva

Date : ...../...../.....

Lecture -7(a) : Logic Instructions

1. AND Operation

```
 10101010
 11110000
=10100000
```

2. OR operation

```
 10101010
 11110000
=11111010
```

3. XOR operation [

```
 10101010
 11110000
=01011010
```

4. NOT Operation (1's complement)

```
 10101010
→01010101
```

- AND destination, source
- OR destination, source
- XOR destination, source

*** The bits in the mask are selected to modify the corresponding destination bits when the instruction executes.

AND → Clear
- 0 mask bit clears
- 1 mask bit preserves

OR → Set
- 1 mask bit sets
- 0 mask bit preserves.

XOR → Complement / change
- 1 mask bit complements
- 0 mask bit preserves.

Protiva

Date : ...../...../.....

Example: Clear the sign bit AL while leaving the other bit unchanged.

⇒

```
  10111011
& 01111111  → mask = 7Fh
----------
  00111011
```

mask value = 0

Thus, AND AL, 7Fh

Example: Set the most significant bit and least significant bits of AL while preserving the other bits.

⇒

```
  10111011
| 10000001  → mask = 81h
----------
  10111011
```

mask value = 1

Thus, OR AL, 81h

Example: Change the sign bit of DX

```
  10111011
^ 10000000  → 8000h
----------
  00111011
```

mask value = 1

Thus, XOR DX, 8000H

Protiva

Date : ...../...../.....

Lecture : 7(b) → Rotate Instruction

□ Suppose DH contains 8A h and CL contains 2. What are values of DH and CF after the instruction SHR DH, CL is executed.

⇒ The binary value of DH = 1000 1010 ↷ CF = Ø 1
↷ 0100 0101 ↷
↷ 0010 0D10
= 22 H

□ DH = 8A H , CL = 3 ; SHL DH, CL

DH → ↶ 1000 1010
↶ 0001 0100 ↶
CF = ✗ Ø 0 ↶ 0010 1000 ↶
0101 0000 ↶
= 50 H

□ SAR: Shift Arithmetic operates like SHR with 1 difference.
→ The msb retains its original value.

SAR destination, 1
SAR destination, CL

Protiva

Date : ...../...../.....

□ Division by SHR

→ For even no → Divided by 2

→ For Odd no. → Rounds to the nearest integer.

ex^m : BL = 00000101 = 5

SHR BL , BL = 0000 0010 = 2

* For left shift → Multiply by 2

□ Difference between Signed and Unsigned Division →

⇒ For unsigned interpretation → SHR should be used.

For signed interpretation → SAR must be used

↳ It preserves the sign.

Problem : Use right shifts to divide the unsigned number 65143 by 4, put the quotient in Ax.

⇒

```
MOV AX, 65143
MOV CL, 2  ; two Right shifts [2+2]
SHR AX, 2  ; Dividend is unsigneed.
```

Protiva

Date : ...../...../.....

Example : If AL contains -15, give the decimal value of AL after SAR AL, 1 performed.

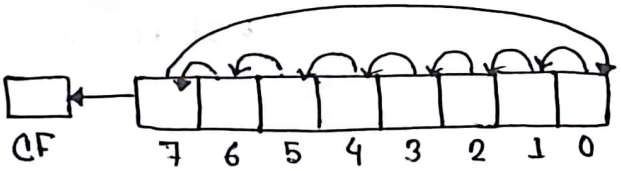
⇒ SAR AL, 1 ; CL = 1

Old AL = -15

New AL = -15/2 = -7.5 ≒ -8

□ ROL (Rotate Left)

- Shifts bits to the left
- The msb shifts into the rightmost bit.
- The CF also gets the bit shifted out of the MSB.

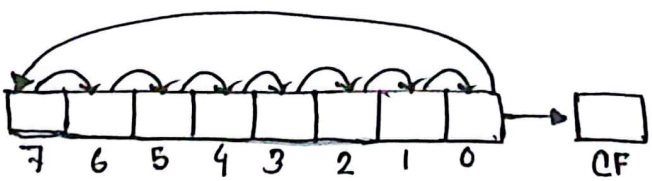


Byte

ROL destination, CL

□ ROR (Rotate Right)

- Shifts bits to the right
- The lsb/rightmost bit is shifted into the msb and also into the CF



Byte.

ROR destination, CL

Protiva

Date : ...../...../.....

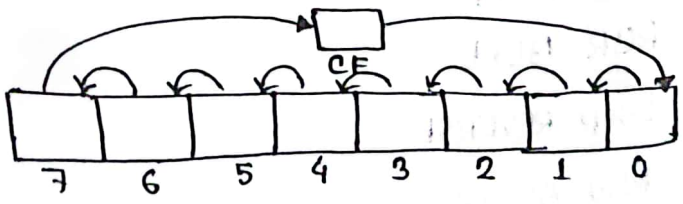
* ROL and ROR used to inspect the bits in a byte without changing the contents. (MCQ)*

Example: Use ROL to count the numbers of 1 bits in BX, without changing BX. Put the Answers in AX.

⇒
```
XOR AX, AX
MOV CX, 16
TOP:
    ROL BX, 1
    JNC NEXT
    INC AX
NEXT:
    LOOP TOP
```

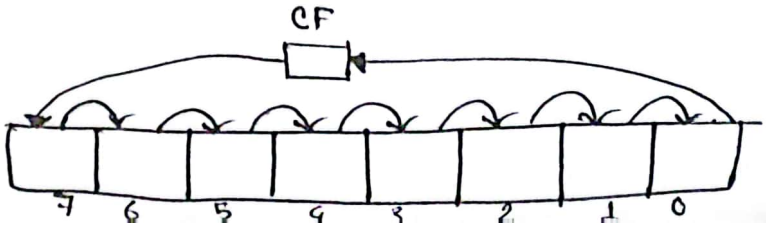
RCL (Rotate Carry Left) like ROL

- Shifts the bits to the left
- The msb is shifted into the CF and the previous
- value of CF is shifted into the rightmost bit.



RCL destination, CL

RCR



Protiva

Date : ...../...../.....

Example:

Suppose DH contains 8A h, CF = 1 and CL contains 3. What are the values of DH and CF after the instruction RCR DH, CL is executed?

⇒

DH	CF	
1000 1010 ↷	1	SF = 1
1 1000 101 ↷	0	ZF = 0
0 1100 010 ↷	1	OF = 1
1011 0001 = B1 h	0	

Prev CF value

Example:

If AL contains 11011100, we want to make it 00111011

⇒

```
    MOV CX, 8
REVERSE :
    SHL AL, 1
    RCR BL, 1
    Loop REVERSE
    MOV AL, BL
```

Date : ...../...../.....

Lecture -09

- MUL → Multiply for unsigned multiplication
  IMUL → For signed multiplication

* Byte form (→ AL) → If two bytes are multiplied → 16 bits (word)
  Word form (↓ AX) → " " words " " → 32 bits (doubleword)

** The source cannot be a constant (MCQ)*

* Doubleword product : DX : AX → least significant 16 bits
  ↳ Most significant 16 bits.

Effect on Status Flags

→ SF, AF, ZF, PF ⟶ Undefined

→ CF/OF ⟶ MUL : CF/OF = 0 if the upper half of the result is 0
CF/OF = 1 ; Otherwise.

→ IMUL : CF/OF = 0 ; if the bits of the upper half are the same as the sign bit of the lower half.

CF/OF = 1 ; Otherwise.

Date : ...../...../.....

Example.

Ax contains 1 and Bx contains FFFF h.

Instructions	Hex Product	Dx	Ax	CF/OF
MUL Bx	0000FFFF	0000	FFFF	0
IMUL Bx	FFFFFFFF	FFFF	FFFF	0

Ax = 1, Bx = -1 (2's complement) so product is -1

Example

Ax contains FFFF h and Bx contain FFFF h

Instruction	Decimal	Hex	Dx	Ax	CF/OF
MUL Bx	65535 x 65535	FFFE0001	FFFE	0001	1
IMUL Bx	1	00000001	0000	0001	0

Ax and Bx both contain -1, so the product is 1

Example

Ax contains OFFF h

MUL Ax	16969025	00FFE001	00FF	E001	1
IMUL Bx	16769025	00FF E001	00FF	E001	1

As, CF/OF = 1, this means the product is too big to fit in Ax.

Protiva

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.4 Suppose AX contains 0100h and CX contains FFFFh:

Instruction	*Decimal product*	*Hex product*	*DX*	*AX*	*CF/OF*
MUL CX	16776960	00FFFF00	00FF	FF00	1
IMUL CX	-256	FFFFFF00	FFFF	FF00	0

For MUL, the product FFFF00 is obtained by attaching two zeros to the source value FFFFh. Because the product is too big to fit in AX, CF/OF = 1.

For IMUL, AX contains 256 and CX contains -1, so the product is -256, which may be expressed as FF00h in 16 bits. DX has the sign extension of AX, so CF/OF = 0.

Signed versus Unsigned Multiplication

Example of Multiplication

Example 9.5 Suppose AL contains 80h and BL contains FFh:

Instruction	*Decimal product*	*Hex product*	*AH*	*AL*	*CF/OF*
MUL BL	128	7F80	7F	80	1
IMUL BL	128	0080	00	80	1

For byte multiplication, the 16-bit product is contained in AX.

For MUL, the product is 7F80. Because the high eight bits are not 0, CF/OF = 1.

For IMUL, we have a curious situation. 80h = –128, FFh = –1, so the product is 128 = 0080h. AH does not have the sign extension of AL, so CF/OF = 1. This reflects the fact that AL does not contain the correct answer in a signed sense, because the signed decimal interpretation of 80h is –128.

Signed versus Unsigned Multiplication

Simple Applications· of MUL and IMUL

Example 9.5 Suppose AL contains 80h and BL contains FFh:

Instruction	*Decimal product*	*Hex product*	*AH*	*AL*	*CF/OF*
MUL BL	128	7F80	7F	80	1
IMUL BL	128!	0080	00	80	1

For byte multiplication, the 16-bit product is contained in AX.

For MUL, the product is 7F80. Because the high eight bits are not 0, CF/OF = 1.

For IMUL, we have a curious situation. 80h = –128, FFh = –1, so the product is 128 = 0080h. AH does not have the sign extension of AL, so CF/OF = 1. This reflects the fact that AL does not contain the correct answer in a signed sense, because the signed decimal interpretation of 80h is –128.

Multiplication

Simple Applications (1) of MUL and IMUL

Example 9.6 Translate the high-level language assignment statement A = 5 × A – 12 × B into assembly code. Let A and B be word variables, and suppose there is no overflow. Use IMUL for multiplication.

Solution:

```
MOV   AX,5          ;AX = 5
IMUL  A             ;AX = 5 x A
MOV   A,AX          ;A = 5 x A
MOV   AX,12         ;AX = 12
IMUL  B             ;AX = 12 x B
SUB   A,AX          ;A = 5 x A - 12 x B
```

Date : ...../...../.....

Division

DIV (divide) → Unsigned division
IDIV (Integer-devide) → Signed divison

DIV divisor
IDIV divisor → 8 bit registor [not be a constant]

* 16-bit dividend → AX
* 8-bit quotient → AL
* 8-bit remainder → AH

DX : AX
↓ Remainder → Quotient.

** All status flags are undefined (MCQ)*

Divide overflow occurs —

1. When the quotient doesn't fit in the specified destination (AL or AX)
2. If the divisor is much smaller than dividend.

** Divisor and Dividend positive হলে → DIV, IDIV result will be same.

Date : ...../...../.....

Example.

Suppose Dx contains 0000h, Ax contains 0005 h and Bx contains 0002 H.

⇒ Dividend : Dx : Ax → 0000 0005 = 5
Divisor : Bx → 0002 = 2

2|5|2 → Q
4
1 → R
↑ Quotient ↑ Remainder

Instruction	Decimal quotient	Decimal remainder	AX	DX
DIV Bx	2	1	0002	0001
IDIV Bx	2	1	0002	0001

Example

Dx : Ax → 0000 0005 = 5

Bx : FFFE h → 1111 1111 1111 1110
0000 0000 0000 0001 (1's complement)
+1
0000 0000 0000 0010 (2's complement)

= -2

FFFF = -1
FFFE = -2
FFFD = -3
FFFC = -4
FFFB = -5
FFFA = -6

			AX	DX
DIV Bx	0	5	0000	0005
IDIV Bx	-2	1	FFFE	0001

-2 |5|2
+4
1

Protiva

Example 9.10 Suppose DX contains FFFFh, AX contains FFFBh, and BX contains 0002.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*DX*
IDIV BX	–2	–1	FFFE	FFFF
DIV BX	DIVIDE OVERFLOW			

For IDIV, DX:AX = FFFFFFFBh = –5, BX = 2. –5 divided by 2 gives a quotient of –2 = FFFEh and a remainder of –1 = FFFFh.

For DIV, the dividend DX:AX = FFFFFFFBh = 4294967291 and the divisor = 2. The actual quotient is 2147483646 = 7FFFFFFEh. This is too big to fit in AX, so the computer prints DIVIDE OVERFLOW and the program terminates. This shows what can happen if the divisor is a lot smaller than the dividend.

Signed versus Unsigned Division

Division Example

Example 9.11 Suppose AX contains 00FBh and BL contains FFh.

Instruction	*Decimal quotient*	*Decimal remainder*	*AX*	*AL*
`DIV BL`	0	251	FB	00
`IDIV BL`	DIVIDE OVERFLOW			

For byte division, the dividend is in AX; the quotient is in AL and the remainder in AH.

For DIV, the dividend is 00FBh = 251 and the divisor is FFh = 256. Dividing 251 by 256 yields a quotient of 0 and a remainder of 251 = FBh.

For IDIV, the dividend is 00FBh = 251 and the divisor is FFh = –1. Dividing 251 by –1 yields a quotient of –251, which is too big to fit in AL, so the message DIVIDE OVERFLOW is printed.

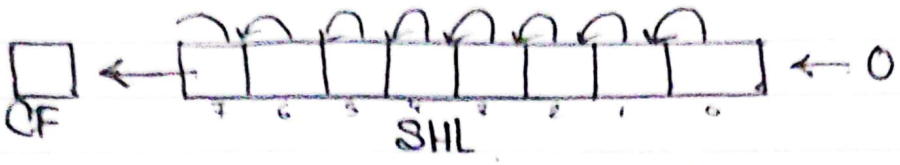
Date :

Find the max value among 3 numbers

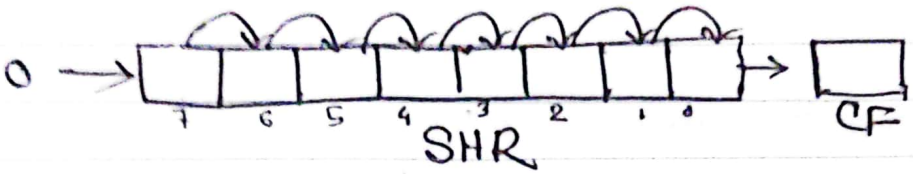
```
.data
a db ?
b db ?
c db ?
d dB " Enter 3 numbers : $ "
e dB " Is the greatest $ "

.code

Main Proc
    mov ax, @data
    mov ds, ax

    Mov ah, 9
    lea dx, d        ; string
    int 21 h

    Mov ah, 1
    int 21 h         ; 1st input
    mov a, al

    int 21 h
    mov b, al        ; 2nd input
    int 21 h
    mov c, al        ; 3rd input

    mov ah, 2        ; Display
    mov dl, 10
    int 21 h
    mov dl, 13
    int 21 h

    Cmp b, a         ; Cmp A
    Jg greaters

Greaters :
    Cmp b, c         ; Cmp c
    Jg B-greaters
    Jl C-greaters

lessers :
    Cmp a, c
    mov al, c        ; Cmp a, c
    Cmp al, a
    Jg C-greaters
    Jl A-greaters

A-greaters :
    mov ah, 2
    mov dl, a
    int 21 h
    mov ah, 9
    lea dx, e
    int 21 h
    Jmp exit.

B-greaters :
    mov ah, 2
    mov dl, b
    int 21 h
    mov ah, 9
    lea dx, e
    int 21 h
    Jmp exit
```

Protiva

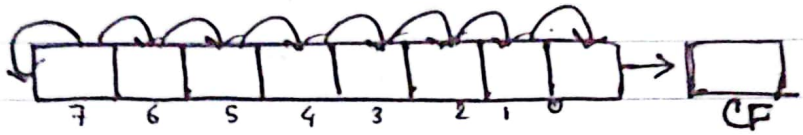
SHL 1, 2X
SHR 2, 0



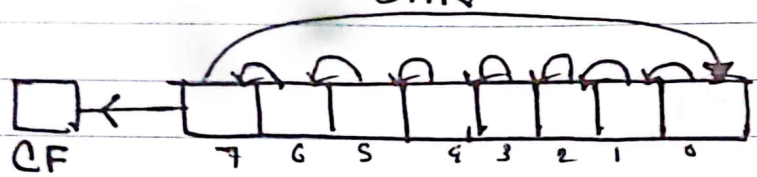
SHL



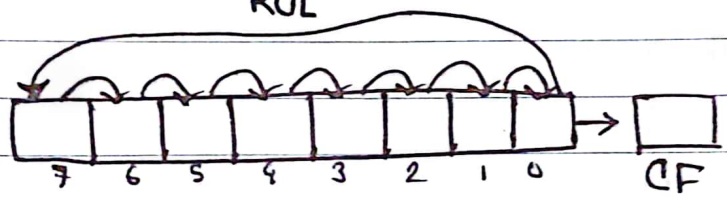
SHR



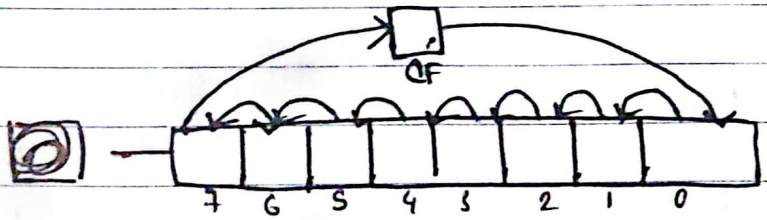
SAR



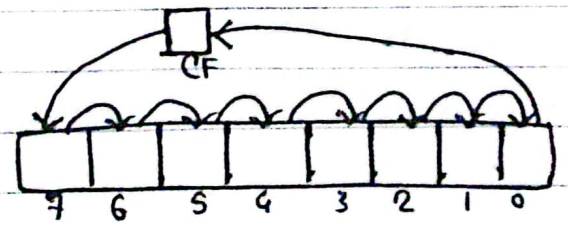
ROL



ROR



RCL



RCR

Even-ODD

```
.model small
.stack 100h
.data
A dw ?
B DB "Enter a number: $"
E DB "The number is even $"
O DB "      "   "   "  ODD $"
.code

Main PROC
        move ax, @data.
        mov ds, ax

        MOV ah, 9
        lea dx B
        int 21 h

        mov ah, 1
        int 21 h
        Sub al, 30 h

        mov ah, 00h
        mov A, ax
        ADD ax, 1
        JZ Even- lea dx, odd
        JMP print.

Even_:
        lea dx, Even

Print:
        mov ah, 9 h
        int 21 h.

        mov ah, 4ch.
        int 21 h
main endp.
```

Exercise
Counting Characters

- Model SMall
- Stack 100h
- Data.
- Code

```
Main Proc
    mov ax, @data
    mov ds, ax
    mov cx, 0

Read-char:
    Mov Ah, 1
    int 21h
    cmp AL, 0Dh      ; with carriage return.
    JE done.

    inc cx
    JMP read-char

done:
    Mov dx, cx       ; Move the count to dx
    Add dx, '0'
    mov ah, 2
    int 21h

    mov Ah, 4ch
    int 21h
    main endp.
end.
```