

Lecture Title

Course Code: 0052

Course Title: Computer Organization and
Architecture



Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	7(b)	Week No:	10	Semester:	Spring 21-22
Lecturer:					

SHR stands for "Shift Right" and it is used to shift the bits in a register or memory location to the right. When a number is shifted to the right using SHR, the least significant bit (LSB) is replaced by a zero, and the most significant bit (MSB) is shifted out and lost. The other bits are shifted to the right by the specified number of positions. For example, the instruction "SHR AL, 1" will shift the bits in the AL register to the right by 1 position. SAR stands for "Shift Arithmetic Right" and it is used to shift the bits in a register or memory location to the right while preserving the sign bit. When a number is shifted to the right using SAR, the least significant bit (LSB) is replaced by a zero, and the most significant bit (MSB) is shifted out and lost. However, unlike SHR, SAR preserves the sign bit by shifting it into the vacated MSB position. This ensures that the sign of the number is preserved after the shift. For example, the instruction "SAR BL, 1" will shift the bits in the BL register to the right by 1 position while preserving the sign bit.

SHL stands for "Shift Left" and it is used to shift the bits in a register or memory location to the left. When a number is shifted to the left using SHL, the most significant bit (MSB) is replaced by a zero, and the least significant bit (LSB) is shifted out and lost. The other bits are shifted to the left by the specified number of positions. For example, the instruction "SHL AX, 1" will shift the bits in the AX register to the left by 1 position. SAL stands for "Shift Arithmetic Left" and it is also used to shift the bits in a register or memory location to the left. The operation performed by SAL is identical to SHL, and both instructions can be used interchangeably. The only difference is that the Intel documentation recommends using SAL when performing arithmetic shifts, to emphasize that the sign bit is shifted along with the other bits. In summary, the main difference between SHL and SAL is that SHL performs a logical shift, while SAL performs an arithmetic shift. However, in practice, the two instructions are identical and can be used interchangeably for left shifts.

ROL: The ROL instruction performs a left rotation of the bits in a register or memory location. Each bit is shifted one position to the left, with the carry flag being shifted into the least significant bit (LSB) and the most significant bit (MSB) being shifted into the carry flag. This effectively shifts all bits one position to the left, including the carry flag. For example, the instruction "ROL AL, 1" will rotate the bits in the AL register one position to the left.

• ROR: The ROR instruction performs a right rotation of the bits in a register or memory location. Each bit is shifted one position to the right, with the carry flag being shifted into the most significant bit (MSB) and the least significant bit (LSB) being shifted into the carry flag. This effectively shifts all bits one position to the right, including the carry flag. For example, the instruction "ROR BH, 1" will rotate the bits in the BH register one position to the right.

Both ROL and ROR are circular shifts, which means that the shifted bits are rotated around to the opposite end of the register or memory location. The carry flag plays a key role in this operation, as it is shifted into the least significant bit (LSB) during a left rotation and into the most significant bit (MSB) during a right rotation.

Overview: ROTATE



- Rotates work like the shifts, except that when a bit is shifted out one end of an operand it is **put back in the other end**.
- These instructions can be used to **examine** and/or **change bits or groups of bits**.
- *** Logic, shift, and rotate instructions are used to do binary and hexadecimal I/O.
- The ability to read and write numbers will let us solve a great variety of problems.

Example SHR

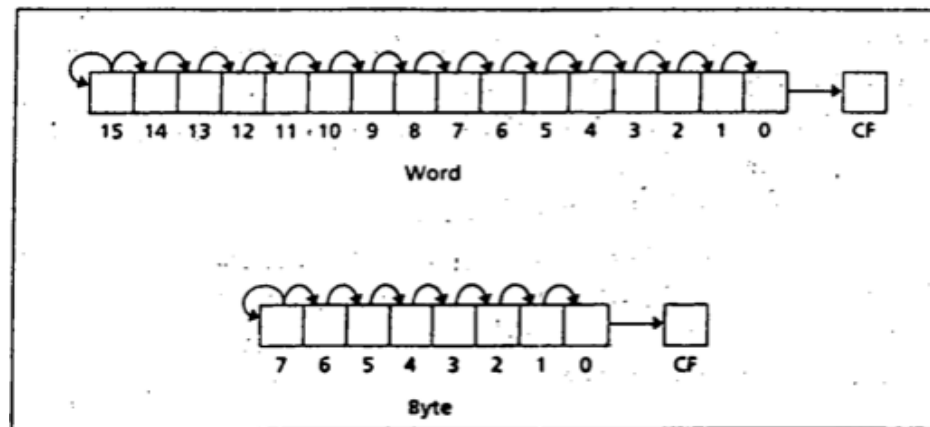


- **Problem:** Suppose DH contains 8Ah and CL contains 2. What are the values of DH and CF after the Instruction SHR DH,CL is executed?
- **Solution:** The value of DH in binary is 10001010.
- After two right shifts, CF=1
- The new value of DH is obtained by **erasing the rightmost two bits** and adding two 0 bits to the left end, thus DH =00100010b = 22h.

SAR Instruction



- The SAR Instruction (shift arithmetic right) operates like SHR , with one difference: the **msb retains its original value**. The syntax is:
- SAR destination,1
- SAR destination, CL



Division by Right Shift



- A Left shift doubles the destination's value,
- Similarly, it's reasonable to guess that a right shift might divide it by 2. This is correct for **even** numbers.
- For odd numbers, a **right shift halves it and rounds down to the nearest integer.**
- For example, if BL contains $00000101 = 5$, then after a right shift. BL will contain $00000010 = 2$

Signed and Unsigned Division



- In case of division by right shifts, we need to make a distinction between signed and unsigned numbers.
- If an **unsigned** interpretation is being given, **SHR** should be used.
- For a **signed** interpretation, **SAR** must be used, because it preserves the sign.
- **Problem:** Use right shifts to divide the unsigned number 65143 by 4. Put the quotient in AX
- To divide by 4, two right shifts are needed. Since the dividend is unsigned, we use SHR. The code is
 - MOV AX, 65143
 - MOV CL, 2
 - SHR AX,2

SAR



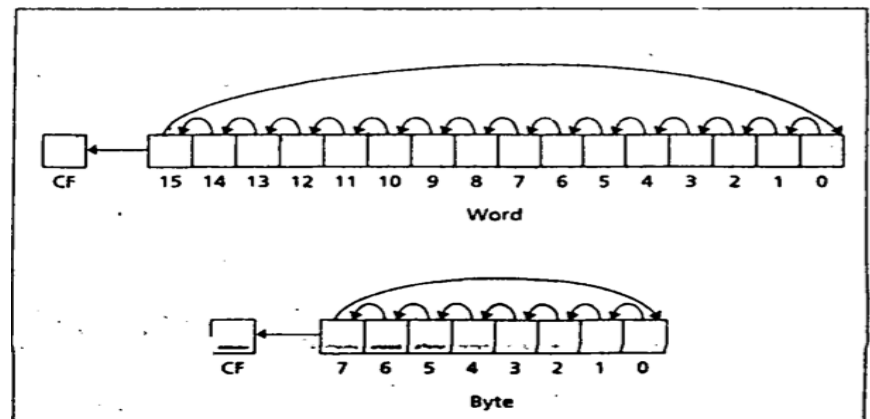
- Example: If AL contains -15, give the decimal value of AL after SAR AL,1 is performed.
- Solution: Execution of SAR AL,1 divides the number by 2 and rounds down.
- Dividing -15 by 2 yields -7.5, and after rounding down we get -8.
- In terms of the binary contents, we have $-15 = 11110001b$. After shifting, we have $11111000b = -8$.

*** We will see some MUL and DIV for multiplication operations that are not limited to power of 2 only. However, MUL and DIV is much slower than SHIFT operation

Rotate Instructions



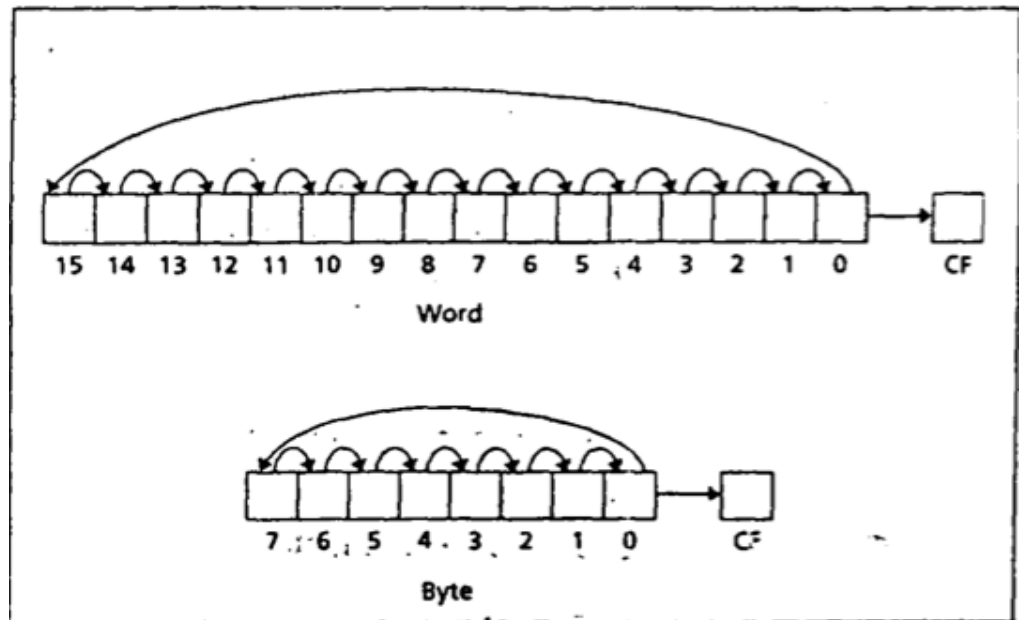
- The instruction ROL (rotate left) shifts bits to the left. The msb shifted into the rightmost bit.
- The CF also gets the bit shifted out of the msb.
- You can think of the destination bits forming a circle, with the least significant bit following the msb in the circle.
- ROL destination, 1
- and
- ROL destination, CL



Rotate Right (ROR)



- The instruction ROR (rotate right) works just like ROL except that the bits are rotated to the right.
- The rightmost bit is shifted into the msb, and also into the CF
- ROR destination, 1
- and
- ROR destination, CL



ROL, ROR and CF



- In ROL and ROR, CF reflects the bit that is rotated out.
- ROL and ROR can be used to inspect the bits in a byte or word, without changing the contents.
- Example: Use ROL to count the number of 1 bits in BX, without changing BX. Put the answer in AX.
- Solution:

```
XOR AX,AX  
MOV CX,16
```

- **TOP:**

```
ROL BX,1  
JNC NEXT
```

```
INC AX
```

- **NEXT:**

```
LOOP TOP
```

RCL (Rotate Carry Left)

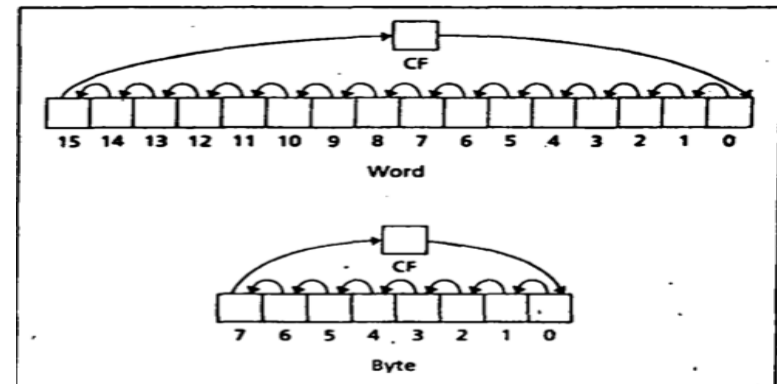


- The Instruction RCL (Rotate through Carry LEFT) shifts the bits of the destination to the left.
- The msb is shifted Into the CF and the previous value of CF is shifted Into the rightmost bit.
- In other words, RCL works like Just like ROL, except that CF is part of the circle of bits being rotated. The syntax is:

➤ RCL destination,1

➤ and

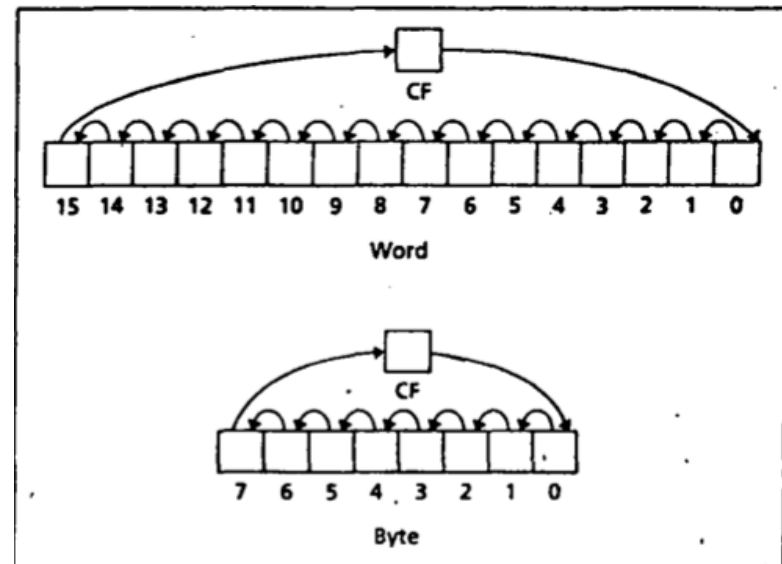
➤ RCL destination, CL



RCR (Rotate Carry Right)



- The Instruction RCR (Rotate through Carry RIGHT) works just like RCL except the bits are rotated to the right. The syntax is:
- RCR destination, 1
- and
- RCR destination, CL



Example



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

Solution:

	CF	DH
initial values	1	10001010
after 1 right rotation	0	11000101
after 2 right rotations	1	01100010
after 3 right rotations	0	10110001 = 81h

Effect of the rotate instructions on the flags

- SF, PF, ZF reflect the result
- AF is undefined
- CF = last bit shifted out
- OF = 1 if result changes sign on the last rotation

Reversing Bit Pattern Application



- **Expectation:** If AL contains **11011100**, we want to make it **00111011**
- Use **SHL** to shift the bits out the left end of AL Into CF.
- Then use **RCR** to move them Into the left end of another register (i.e. **BL**)
- Run the above operation 8 times for 8 bits

```

    MOV     CX,8       ;number of operations to do
,REVERSE:
    SHL     AL,1       ;get a bit into CF
    RCR     BL,1       ;rotate it into BL
    LOOP    REVERSE    ;loop until done
    MOV     AL,BL      ;AL gets reversed pattern
```

Binary and Hex Input & Output



- **Binary Input:** Lets assume a program reads in a binary number from the keyboard, followed by a carriage return. [i.e. string of 0's and 1's]
- Conversion in bit value needs to be done as soon as the input character is entered.
- After that collect the bits in register.
- To read a binary number from keyboard and store it in BX:

```
Clear BX      /* BX will hold binary value */
Input a character /* '0' or '1' */
WHILE character <> CR DO
    Convert character to binary value
    Left shift BX
    Insert value into lsb of BX
    Input a character
END_WHILE
```



Example: Process Input 110

Clear BX

BX = 0000 0000 0000 0000

Input character '1', convert to 1

Left shift BX

BX = 0000 0000 0000 0000

Insert value into lsb

BX = 0000 0000 0000 0001

Input character '1', convert to 1

Left shift BX

BX = 0000 0000 0000 0010

Insert value into lsb

BX = 0000 0000 0000 0011

Input character '0', convert to 0

Left shift BX

BX = 0000 0000 0000 0110

Insert value into lsb

BX = 0000 0000 0000 0110

BX contains 110b.

Assembly Conversion for input processing (110)



```
XOR  BX,BX      ;clear BX
MOV  AH,1        ;input char function
INT  21H         ;read a character
```

WHILE_:

```
CMP  AL,0DH      ;CR?
JE   END_WHILE   ;yes, done
AND  AL,0FH      ;no, convert to binary value
SHL  BX,1        ;make room for new value
OR   BL,AL       ;put value into BX
INT  21H         ;read a character
JMP  WHILE_      ;loop back
```

END_WHILE:

Binary Output



- Outputting the contents of BX in binary also involves the shift operation.
- **Algorithm for Binary output:**

```
FOR 16 times DO
    Rotate left BX /* BX holds output value,
                   put msb into CF */
    IF CF = 1
    THEN
        output '1'
    ELSE
        output '0'
    END_IF
END_FOR
```

Write an assembly code to process the Binary output for this problem.

Hex Input



- Hex input consists of digits ("0" to "9") and letters ("A" to "F") followed by a carriage return.
- For simplicity, we will assume that
- Only uppercase letters are used, and
- The user inputs no more than four hex characters.
- The process of converting characters to binary values is more involved than it was for binary input, and BX must be **shifted four times** to make room for a hex value.

Algorithm for hex input



```
Clear BX      /* BX will hold input value */
input hex character
WHILE character <> CR DO
    convert character to binary value
    left shift BX 4 times
    insert value into lower 4 bits of BX
    input a character
END_WHILE
```

Example: input 6AB



Clear BX

BX = 0000 0000 0000 0000

Input '6', convert to 0110

Left shift BX 4 times

BX = 0000 0000 0000 0000

Insert value into lower 4 bits of BX

BX = 0000 0000 0000 0110

Input 'A', convert to Ah = 1010

Left shift BX 4 times

BX = 0000 0000 0110 0000

Insert value into lower 4 bits of BX

BX = 0000 0000 0110 1010

Input 'B', convert to 1011

Left shift BX 4 times

BX = 0000 0110 1010 0000

Insert value into lower 4 bits of BX

BX = 0000 0110 1010 1011

BX contains 06ABh.

Assembly Code for Processing 6AB



```
        XOR    BX,BX        ;clear BX
        MOV    CL,4         ;counter for 4 shifts
        MOV    AH,1         ;input character function
        INT    21H          ;input a character

WHILE_:
        CMP    AL,0DH        ;CR?
        JE     END_WHILE     ;yes, exit
        ;convert character to binary value
        CMP    AL,39H        ;a digit?
        JG     LETTER.       ;no, a letter
        ;input is a digit
        AND    AL,0FH        ;convert digit to binary value
        JMP    SHIFT         ;go to insert in BX

LETTER:
        SUB    AL,37H        ;convert letter to binary value

SHIFT:
        SHL    BX,CL         ;make room for new value
        ;insert value into BX
        CR     BL,AL         ;put value into low 4 bits
                           ;of BX
        INT    21H          ;input a character
        JMP    WHILE_        ;loop until CR

END_WHILE:
```

Algorithm for Hex Output



```
FOR 4 times DO
  Move BH to DL      /* BX holds output value */
  shift DL 4 times to the right
  IF DL < 10
    THEN
      convert to character in '0'..'9'
    ELSE
      convert to character in 'A'..'F'
  END_IF
  output character
  Rotate BX left 4 times
END_FOR
```

Conversion of 4CA9h to Binary



```
BX = '4CA9h' = 0100 1100 1010 1001
Move BH to DL
DL = 0100 1100
Shift DL 4 times to the right
DL = 0000 0100
Convert to character and output
DL = '0011 0100' = 34h = '4'
Rotate BX left 4 times
BX = 1100 1010 1001 0100
Move BH to DL
DL = 1100 1010
Shift DL 4 times to the right
DL = 0000 1100
Convert to character and output
DL = 0100 0011 = 43h = 'C'
Rotate BX left 4 times
BX = 1010 1001 0100 1100
Move BH to DL
DL = 1010 1001
Shift DL 4 times to the right
DL = 0000 1010
Convert to character and output
DL = 0100 0010 = 42h = 'B'
Rotate BX left 4 times
BX = 1001 0100 1100 1010
Move BH to DL
DL = 1001 0100
Shift DL 4 times to the right
DL = 0000 1001
Convert to character and output
DL = 0011 1001 = 39h = '9'
Rotate BX 4 times to the left
BX = 0100 1100 1010 1001 = original contents
```

Write an assembly code to process the Binary output for this problem.



References

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).
- <http://faculty.cs.niu.edu/~byrnes/csci360/notes/360shift.htm>



Books

- Assembly Language Programming and Organization of the IBM PC, Ytha Yu and Charles Marut, McGraw Hill, 1992. (ISBN: 0-07-072692-2).
- Essentials of Computer Organization and Architecture, (Third Edition), Linda Null and Julia Lobur
- W. Stallings, “Computer Organization and Architecture: Designing for performance”, 67h Edition, Prentice Hall of India, 2003, ISBN 81 – 203 – 2962 – 7
- Computer Organization and Architecture by John P. Haynes.