Lecture Title

Course Code: 0052

Course Title: : Computer Organization and Architecture

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	9	**Week No:**	9	**Semester:**	
Lecturer:					

Overview : LOGIC

- Instructions to **change the bit pattern** in a byte or word
- The ability to **manipulate bits manually** which is unlikely in high level languages (Except C)
- **Logic Instructions:** AND, OR, XOR and NOT
- Logic Instructions can be used to **clear, set, and examine** bits, a register or variable. i.e. these will be used for
    - Converting a lowercase letter to upper case
    - Determining If a register contains an even or odd number.

Overview: SHIFT

- Bits can be shifted **left or right** in a register or memory location.
- When a bit is shifted out, it goes into CF.
- Because a **left shift doubles** a number and a **right shift halves** it, these instructions give us a way to multiply and divide powers of 2.
- Shifting is much **faster** than Multiplication and Division.

LOGIC Instructions

- The ability to **manipulate individual bits** is one the main advantages of assembly language.
- Individual bits can be changed in computer by using logic operations.
- The binary values of **0 = False and 1= True**
- When a logic operation is applied to 8- or 16-bit operands, the result is obtained by applying the logic operation at **each bit position.**

Truth Table for AND, OR, XOR and NOT

a	b	a AND b	a OR b	a XOR b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	NOT a
0	1
1	0

Solve the Following

1. **AND Operation:** 10101010 **AND** 11110000
2. **OR Operation:** 10101010 **OR** 11110000
3. **XOR Operation:** 10101010 **XOR** 11110000
4. **NOT Operation:** **NOT** 10101010

Solution

1
$$\begin{array}{r} 10101010 \\ \textbf{AND}\ 11110000 \\ \hline =10100000 \end{array}$$

2
$$\begin{array}{r} 10101010 \\ \textbf{OR}\ 11110000 \\ \hline =11111010 \end{array}$$

3
$$\begin{array}{r} 10101010 \\ \textbf{XOR}\ 11110000 \\ \hline =01011010 \end{array}$$

4
$$\begin{array}{r} \textbf{NOT}\ \ 10101010 \\ \hline =01010101 \end{array}$$

AND, OR, and XOR instructions

- The **AND, OR, and XOR** instructions perform the named logic operations. The formats are:

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

- The result of the operation is stored in the destination, which must be a register or memory location.
- The source may be a constant, register, or memory location.
- However, memory-to-memory operations are not allowed.

Effect on Flags

- SF, ZF, PF reflect the result
- AF is undefined
- CF, OF= 0

- One use of AND, OR, and XOR is to **selectively modify the bits** in the destination.
- To do this, we construct a **source bit pattern** known as **mask**.
- The **mask bits** are chosen so that the **corresponding destination bits** are **modified in the desired manner** when the instruction is executed.

MASK

- To choose the mask bits, we make use of the following properties of AND, OR, and XOR:
- b AND 1 = b
- b OR 0 = b
- b XOR 0 = b
- b AND 0 =0
- b OR 1 = 1
- b XOR 1 = ~b (complement of b)

- The **AND** instruction can be used to **CLEAR** specific destination bits while preserving the others.
  - A **0 mask** bit **clears** the corresponding destination bit.
  - a **1 mask** bit **preserves** the corresponding destination bit.
- The **OR** instruction can be used to **SET** specific destination bits while preserving the others.
  - A **1 mask** bit **sets** the corresponding destination bit.
  - A **0 mask** bit **preserves** the corresponding destination bit. ·
- The **XOR** instruction can be used to **complement** specific destination bits while preserving the others.
  - A **1 mask** bit **complements** the corresponding destination bit;
  - A **0 mask** bit preserves the corresponding destination bit.

Clear bit

- **Example**

- Clear the sign bit of AL while leaving the other bits unchanged

- **Solution:**

  - Use the AND instruction with 01111111b=7Fh as the mask.

  - Thus. AND AL,7Fh

Set or Complement Bit

- **Example**: Set the most significant and least significant bits of AL while preserving the other bits.
  - **Solution:** Use the OR instruction with 10000001b =81h as the mask.
  - Thus, OR AL,81h
- **Example**: Change the sign bit of DX.
  - **Solution:** Use the XOR instruction with a mask of 8000h.
  - Thus, XOR DX,8000h

*** To avoid typing errors, it's best to express the mask in **hex rather than binary**, especially if the mask would be 16 bits long.

Converting an ASCII Digit to a Number

- when program reads a character or digit from the keyboard, AL gets the ASCII code of the character.
- For example, if the "5" key is pressed, AL gets 35h instead of *5.* To get 5 in AL, we did
  - **SUB AL,30h**
- We can also do this by using an AND instructions to **clear the high four bits of AL.**
  - **AND AL,0Fh**
- As the ASCII codes of "0" to "9" are 30h to 39h, this method will convert any ASCII digit to a decimal value.
- Using AND emphasizes on modifying bit pattern of AL and makes program more readable.

Problem: convert a stored decimal digit to Its ASCII code?

Converting a Lowercase Letter to Upper Case

- The ASCII codes range for
  - "a" to “z" is 61h to 7Ah
  - "A" to "Z" is 41h to 5Ah.
- So, to convert a lowercase to UPPERCASE we use the following operation:
  - **Sub DL,20h**
- However, if we compare binary codes of corresponding lower and uppercase letters,

Character	Code	Character	Code
a	0110[illegible]001	A	010000[illegible]
b	01100010	B	01000010
	.	.	.
	.	.	
	.	.	
	01111010	Z	01011010

Conversion using AND

- To convert lower to upper case we need to clear only bit 5. This can be done by using an AND instruction with the mask **11011111** or **0DFh**. So if the lowercase character to be converted is In DL, we execute
- AND DL, 0DFh

Clearing a Register

- MOV AX,0
- SUB AX,AX
- XOR AX,AX [1 XOR 1 = 0 and 0 XOR 0=0]

Testing a Register for Zero

- To test the contents of a register for zero, or to check the sign of the contents, we may use:
- CMP CX,0

Not Instruction

- The NOT instruction performs the **one's complement** operation on the destination. The format is:
- **NOT destination** (**No effect on status flags)
- Example: Complement the bits in AX:
- **NOT AX**

TEST Instruction

- The **TEST** Instruction performs an AND operation of the destination with the source but **does not change** the destination contents.

- The purpose of the test instruction is to **set the status flags**. The format is:

  - TEST destination, Source

- Effects of flags on test operation:

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

Bit Examination on TEST

- TEST instruction can be used to examine individual bits in operand.
- The mask should contain **1's** in the bit positions to be tested and **0's** elsewhere
  - As **1 AND b = b**, **0 AND b = 0**
- The operation **TEST destination, mask**
- Will have 1's in the tested bit positions if and only if the destination has 1's in these positions; and 0's elsewhere.
- if the destination has 0's in all the tested positions, the result will be 0 and thus ZF=1

Find Even Number

- **Example:** Jump to label BELOW If AL contains an even number.
- **Solution:** Even numbers have a 0 in bit 0. Thus, the mask is 00000001b=1
    - TEST AL, 1
    - JZ BELOW

- The shift and rotate instructions **shift the bits** in the **destination operand** by one or more positions either to the left or right.
- For a shift instruction, the bits shifted out are lost
- For a rotate instruction, bits shifted out from one end of the operand are put back into the other end.
- The instruction have two possible formats. For a single shift or rotate, the form is
  - **Opcode destination,1**
- For a shift or rotate of **N** positions, the form is
  - **Opcode destination, CL**
- Where CL contains N In both cases, destination is an 8- or 16-bit register or memory location.

Shift Instructions…

- Shift or Rotate instructions can be used to **multiply and divide by powers of 2**, and we will use them in programs for binary and hex I/O

***Note that for Intel's more advanced processors, a shift or rotate instruction also allows the use of an 8-bit **constant**.

Left Shift (SHL) Instructions

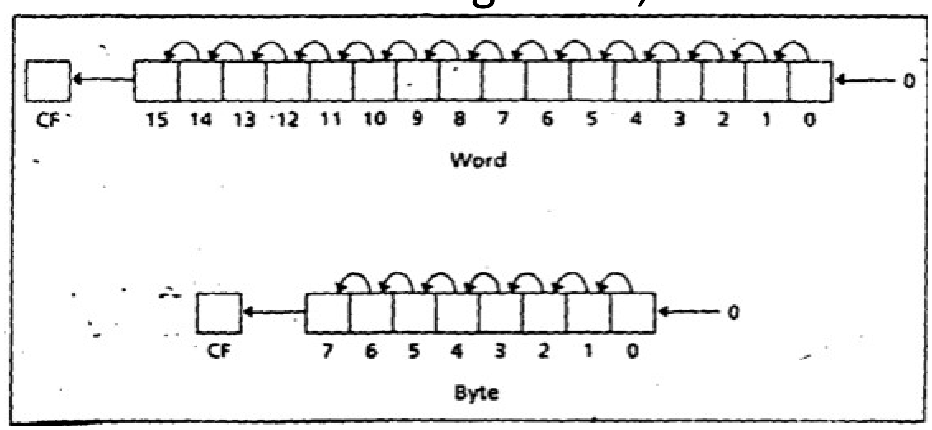
- The SHL (shift left) instruction shifts the bits in the destination to the left. The format for a single shift is
    - SHL destination, 1
- A 0 is shifted into the **rightmost bit position** and the **msb is shifted into CF**. If the shift count ***N*** is different from 1, the instruction takes the form
    - **SHL destination, CL** (Here **CL** contains ***N*** and the above instruction made N single shifts)
        - The value of CL remains the same after the shift operation

Effect on flags

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

SHL Instruction

- Example: Suppose DH contains **8Ah** and **CL** contains **3**. What are the values of DH and of CF after the instruction ***SHL DH,CL*** is executed?
- The binary value of DH is **10001010**. After **3 left shifts,** CF will contain 0. The new contents of DH may be obtained by
- Erasing the leftmost three bits
- Adding three zero bits to the right end, thus 01010000b = 50h.



Multiplication by Left Shift

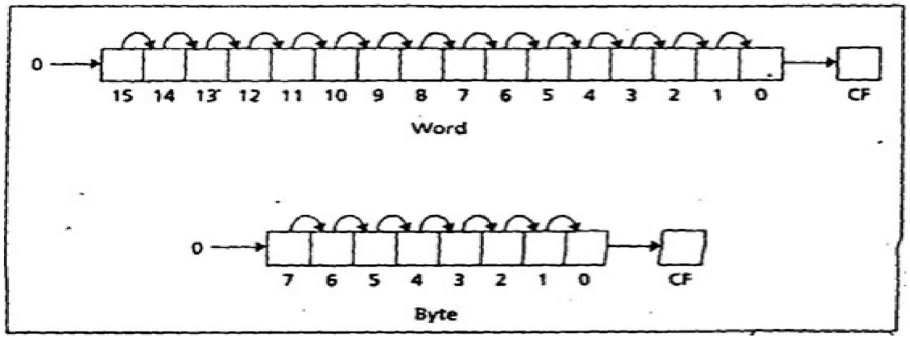
- Let us consider a decimal number 235.
  - If each digit is shifted left and 0 is attached on the right end, we get 2350 which is same as multiplying by 10.
  - Similarly, a left shift on a binary number multiplies it by 2.
- For example, suppose that AL contains 5=00000101b
  - A left shift gives 00001010b = 10 thus doubling its value.
  - Another left shift yields 00010100= 20d, so it is doubled again.

Shift Arithmetic Left (SAL)

- SHL Instruction can be used to multiply an operand by multiples of 2.
- However, to emphasize the arithmetic nature of the operation the opcode SAL (shift arithmetic left)often used in instances for numeric multiplication.
- Both instructions generate the same ma chine code.
- When *we* treat left shifts as multiplication, overflow may occur.
- For a single left shift, CF and OF accurately indicate unsigned and signed over- flow, respectively.
- However, the overflow flags are not reliable indicators for a multiple left shift as multiple shift is really a series of single shifts, and OF and CF only reflect the **result of the last shift.**

Right Shift (SHL) Instructions

- The instruction SHR (shift right) performs right shifts on the destination operand
- SHR destination, 1
- A 0 is shifted Into the msb position, and the rightmost bit is shifted
- SHR destination,CL
- ** here CL contains N In this case N single right shifts are made. The effect on the flags is the same as for SHL



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.