Lecture 5

Program Logic and Control

(Boolean Operation
Shift & Rotate Instructions)
Lecture Outline

• Introduction
• Logic Instructions
  • AND, OR, XOR Instructions
  • NOT Instruction
  • TEST Instruction
• Shift Instructions
  • The SHL Instruction
  • The SHR Instruction
• Rotate Instructions
  • The ROL Instruction
  • The ROR Instruction
Introduction

• Logic, shift, and rotate instructions can be used to change the bit pattern in a byte or word.

• The ability to manipulate bits is generally absent in high-level languages (except C) and is an important reason for programming in assembly language.

• It is used to check bit in a register or memory location, clear or set a register contents
Logic Instructions

• Logic instructions can be used to clear, set, and examine individual bits in a register or variable.

• Logic instructions:
  • AND.
  • OR.
  • XOR.
  • NOT.
  • TEST.
## AND, OR, XOR Instructions

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a AND b</th>
<th>a OR b</th>
<th>a XOR b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\text{AND} \quad 10101010 \\
\text{OR} \quad 11110000 \\
\text{XOR} \quad 10101010
\]

\[
\frac{10101010}{11110000} = \frac{10100000}{11110000} = \frac{01011010}{10100000}
\]
AND, OR, XOR Instructions

• Syntax

  AND  destination, source — constant, register, or memory location
  OR   destination, source
  XOR  destination, source

• The result of the operation is stored in the destination.

• Memory-to-memory operations are not allowed.

• Effect on flags:
  • SF, ZF, PF reflect the result.
  • AF is undefined.
  • CF = OF = 0.
AND, OR, XOR Instructions

• 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 a mask.

• The mask bits are chosen so that the corresponding destination bits are modified in the desired manner when the instruction is executed.

• To choose the mask bits, we make use of the following properties:
  • b AND 1 = b
  • b OR 0 = b
  • b XOR 0 = b
  • b AND 0 = 0
  • b OR 1 = 1
  • b XOR 1 = ~b

  where b represent a bit (0 or 1).
AND, OR, XOR Instructions

- From these, we may conclude that:
  
  - The AND instruction can be used to **clear** specific destination bits while preserving the others. (0 mask bit clears - 1 mask bit bit preserves).
  
  - The OR instruction can be used to **set** specific destination bits while preserving the others. (1 mask bit sets - 0 mask bit preserves).
  
  - The XOR instruction can be used to **complement** specific destination bits while preserving the others. (1 mask bit complements - 0 mask bit preserves).
AND, OR, XOR Instructions

• *Clear the sign bit of AL while leaving the other bits unchanged?*
  
  \[
  \text{AND AL, } 7\text{Fh} \quad (01111111)
  \]

• *Set the msb & lsb of AL while preserving the other bits?*
  
  \[
  \text{OR AL, } 81\text{h} \quad (10000001)
  \]

• *Change the sign bit of DX?*
  
  \[
  \text{XOR DX, } 8000\text{h} \quad (10000000)
  \]
AND, OR, XOR Instructions

• *Convert an ASCII digit to a number?*

<table>
<thead>
<tr>
<th>Character</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00110000</td>
</tr>
<tr>
<td>1</td>
<td>00110001</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>9</td>
<td>00111001</td>
</tr>
</tbody>
</table>

AND AL, 0Fh
AND, OR, XOR Instructions

- Convert the lowercase character in DL to uppercase?

<table>
<thead>
<tr>
<th>Character</th>
<th>code</th>
<th>character</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>01100001</td>
<td>A</td>
<td>01000001</td>
</tr>
<tr>
<td>b</td>
<td>01100010</td>
<td>B</td>
<td>01000010</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>z</td>
<td>01110101</td>
<td>C</td>
<td>01011010</td>
</tr>
</tbody>
</table>

AND DL, 0DFh (01001101)
AND, OR, XOR Instructions

• *Clear AX?*

  MOV AX, 0 → Three bytes machine code
  or
  SUB AX, AX → Two bytes machine code (efficient)
  or
  XOR AX, AX → Two bytes machine code (efficient)

• *Clear A?*

  MOV A, 0 (memory location)
AND, OR, XOR Instructions

- *Test CX for zero?*

**OR CX, CX**

CX is unchanged but if CX = 0 then ZF = 1.

It can be used as an alternative to

**CMP CX, 0**
**NOT Instruction**

<table>
<thead>
<tr>
<th>a</th>
<th>NOT a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\text{NOT } 10101010 = 01010101
\]
NOT Instruction

• The **NOT** instruction performs the one’s complement operation on the destination.

• Syntax:

  NOT destination

• There is no effect on the status flags.

• *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.

• Syntax:

  TEST destination, source

• Effect on flags:
  • SF, ZF, PF reflect the result.
  • AF is undefined.
  • CF = OF = 0.
TEST Instruction

• The TEST instruction can be used to examine individual bits in an operand.

• The mask should contain:
  • 1’s in the bit positions to be tested.
  • 0’s elsewhere.

• *Jump to BELOW if AL contains an even number?*

  TEST   AL,1
  JZ     BELOW
Introduction to shift and rotate instructions

• 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.

• These instructions can be used to multiply and divide by powers of 2. (shift left to multiply and shift right to divide by 2)

• For a rotate instruction, bits shifted out from one end of the operand are put back into the other end.
SHIFT Instructions

• Shift instructions:
  • SHL.
  • SHR.

• Syntax:
  Opcode destination, 1
  Opcode destination, CL ; where CL contains N

• Effect on flags:
  • SF, PF, ZF reflect the result.
  • AF is undefined.
  • CF = last bit shifted out.
  • OF = 1 if the result changes sign on last shift.
The SHL Instruction

- The SHL (shift left) instruction shifts the bits in the destination to the left.

- SHL destination, 1
  A 0 is shifted into the rightmost bit position and the msb is shifted into CF.

- SHL destination, CL ; where CL contains N
  N single left shifts are made. The value of CL remains the same.
The SHL Instruction

- 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?

- DH before

- SHL DH, CL

- CF & DH After

SAL (shift Arithmetic Left) instruction can be used instead of SHL.

Both have same machine codes.
The SHL Instruction - Multiplication by SHL

• A left shift on a binary number multiplies it by 2.

• Ex. if AL contains 5d = 00000101b
    A left shift gives 10d = 00001010b
    Another left shift gives 20d = 00010100

• When we treat left shifts as multiplication, overflow may occur.

• For 1 left shift, CF & OF accurately indicate unsigned and signed overflow, respectively.

• Overflow flags are not reliable indicators for a multiple left shift.
  Ex. SHL BL, CL ; where BL = 80h and CL = 2
  CF = OF = 0
  even though both signed & unsigned overflow occurred.
The SHR Instruction

• The SHR (shift right) instruction shifts the bits in the destination to the right

• SHR destination, 1
  A 0 is shifted into the msb position and the rightmost bit is shifted into CF.

• SHR destination, CL ; where CL contains N
  N single right shifts are made. The value of CL remains the same.
The SHR Instruction

- Suppose DH contains 8Ah and CL contains 2. What are the values of DH and of CF after the instruction SHR DH, CL is executed?

- DH before

- SHR DH, CL

- CF & DH After

SAR (shift Arithmetic Right) instruction can also used instead of SHR

Both have same machine codes
The SHR Instruction - Division by SHR

- For even numbers, a right shift divides it by 2.

- Ex. if AL contains 6d = 00000110b
  A right shift gives 3d = 00000011b

- For odd numbers, a right shift halves it and rounds down to the nearest integer.

- Ex. if AL contains 5d = 00000101b
  A right shift gives 2d = 00000010b

- *Use SHR to divide 65143 by 4, put the quotient in AX.*

  MOV  AX, 65143 ; AX has number
  MOV  CL, 2    ; CL has number of right shifts
  SHR   AX, CL  ; divide by 4
Rotate Instructions

• Rotate instructions:
  • ROL.
  • ROR.

• Syntax:

  Opcode destination, 1
  Opcode destination, CL ; where CL contains N

• Effect on flags:
  • SF, PF, ZF reflect the result.
  • AF is undefined.
  • CF = last bit shifted out.
  • OF = 1 if the result changes sign on last shift.
The ROL Instruction

• The ROL (rotate left) instruction shifts the bits in the destination to the left.

• The msb is shifted into the rightmost bit and also into the CF.

![Diagram of ROL instruction](image-url)
The ROL Instruction

- Use ROL to count the number of 1 bits in BX, without changing BX. Put the answer in AX.

XOR AX, AX ; AX counts bits
MOV CX, 16 ; loop counter

TOP:
ROL BX, 1 ; CF = bit rotated out
JNC NEXT ; 0 bit
INC AX ; 1 bit, increment total

NEXT:
LOOP TOP ; loop until done
The ROR Instruction

• The ROR (rotate right) instruction shifts the bits in the destination to the right.

• The rightmost is shifted into the msb bit and also into the CF.

![Diagram of ROR instruction]

![Diagram showing bit shifting and CF]

Shift & Rotate Instructions
APPLICATION

• shift and rotate instructions is used for binary and hex I/O.

• For binary input, we assume a program reads in a binary number from the keyboard, followed by a carriage return.

• Hex input consists of digits 0 to 9 and letters A to F followed by a carriage return.
APPLICATION (Binary I/O)

Algorithm for Binary Input

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 1sb of BX
Input a character
END_WHILE
APPLICATION

Binary Input in Assembly code

XOR BX, BX ; CLEAR BX
MOV AH, 1 ; Input character
INT 21H

WHILE_:  
CMP AL, 0DH ; CR?
JE END_WHILE ; YES, DONE
AND AL, 0FH ; CONVERT TO BINARY VALUE
SHL BX, 1 ; MAKE ROOM FOR NEW VALUE
OR BL, AL ; PUT VALUE IN BX
INT 21H ; READ A CHARACTER
JMP WHILE_ ; LOOP BACK

END_WHILE:
APPLICATION (Binary I/O)

Algorithm for Binary Output

For 16 times do
Rotate Left BX ;BX holds output value put msb in to CF

If CF= 1
THEN
  output ‘1’
ELSE
Output ‘0’
END IF
END FOR
APPLICATION

Binary Output in Assembly code

MOV CX, 16
MOV AH, 2
TOP_:  
  ROL BX, 1 ; content in BX
  JC display_one ; print bit ‘1’
  MOV DL,’0’ ; else Print bit ‘0’
JMP display
display_one :  
  MOV DL,’1’
Display:
INT 21H ; Print bit either ‘0’ or ‘1’
LOOP TOP_ ; LOOP BACK