Microprocessors (0630371)
Fall 2010/2011 – Lecture Notes # 17

Boolean and Comparison Instructions

Objectives of the Lecture

- **AND Instruction**
- **OR Instruction**
- **XOR Instruction**
- **NOT Instruction**
- **TEST Instruction**
- **CMP Instruction**
- **Applications**

Programs that deal with hardware devices must be able to manipulate individual bits in numbers. Individual bits must be tested, cleared and set. Data encryption and compression also rely on bit manipulation.

AND Instruction

- Performs a Boolean AND operation between each pair of matching bits in two operands
- Syntax:

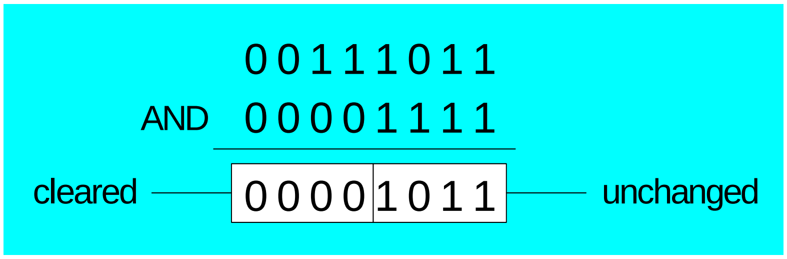
```
AND destination, source
```

- (same operand types as MOV)

```
AND reg, reg
AND reg, mem
AND reg, imm
AND mem, reg
AND mem, imm
```

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Logical truth table

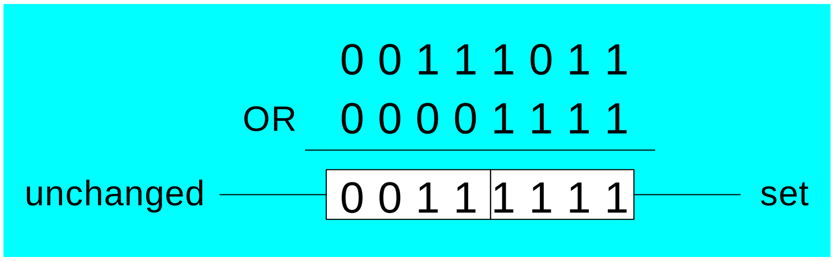


OR Instruction

- Performs a Boolean OR operation between each pair of matching bits in two operands
- Operands can be 8, 16, or 32 bits and they must be of the same size
- Syntax (the same as the AND instruction):

```
OR destination, source
```

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1



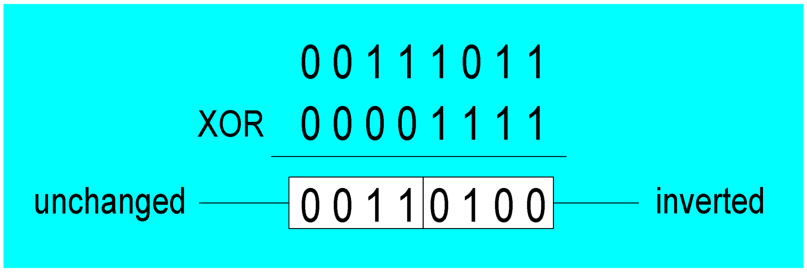
Logical truth table

XOR Instruction

- Performs a Boolean exclusive-OR operation between each pair of matching bits in two operands
- Syntax:

```
XOR destination, source
```

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0



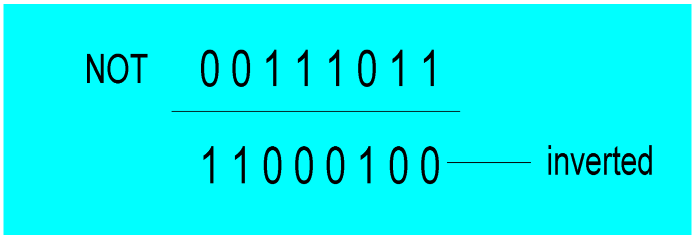
- XOR is a useful way to toggle (invert) the bits in an operand.

NOT Instruction

- Performs a Boolean NOT operation on a single destination operand
- Syntax:

```
NOT destination
```

X	¬X
F	T
T	F



TEST Instruction

- Performs a nondestructive AND operation between each pair of matching bits in two operands
- No operands are modified, but the Zero flag is affected.

CMP Instruction

- Syntax:

```
CMP destination, source
```

- The compare (CMP) instruction performs an implied subtraction of a source operand from a destination operand. Neither operand is modified.
- The Overflow, Carry, Sign, and Zero flags are updated as if the subtract instruction has been performed. The main purpose of the compare instruction is to update the flags so that a subsequent conditional jump instruction can test them.
- CMP can perform unsigned and signed comparisons
  - The *destination* and *source* operands can be unsigned or signed

- For unsigned comparison, we examine ZF and CF flags

Unsigned Comparison	ZF	CF
unsigned destination < unsigned source	0	1
unsigned destination > unsigned source	0	0
destination = source	1	0

- For signed comparison, we examine SF, OF, and ZF

Signed Comparison	Flags
signed destination < signed source	SF ≠ OF
signed destination > signed source	SF = OF, ZF = 0
destination = source	ZF = 1

```
Example1: destination == source
          mov al,5
          cmp al,5  ; Zero flag set
Example2: destination < source
          mov al,4
          cmp al,5  ; Carry flag set
Example3: destination > source
          mov al,6
          cmp al,5  ; ZF = 0, CF = 0
```

(both the Zero and Carry flags are clear)

- The comparisons shown here are performed with **signed integers**.

```
Example4: destination > source
          mov al,5
          cmp al,-2 ; Sign flag == Overflow flag
Example5: destination < source
          mov al,-1
          cmp al,5  ; Sign flag != Overflow flag
Example6:
          TITLE Demonstrating the Compare Instruction (cmp.asm)
          .686
          .MODEL flat, stdcall
          .STACK
          INCLUDE Irvine32.inc
          .data
          var1    SDWORD  -3056
          .code
          main PROC
              mov eax, 0f7893478h
              mov ebx,  1234F678h
              cmp al,  bl
              cmp ax,  bx
              cmp eax, ebx
              cmp eax, var1
              exit
          main ENDP
          END main
```

- Bitwise Logical instructions are the most primitive operations needed by every computer architecture
- bitwise logical operations are performed at **bit-by-bit basis**
- All logical instructions need two operands except **NOT** instructions which is a unary.
- The result of the operation is stored in the **Destination** except **Test** instruction, which must be a general register or a memory location.
- The Source may be an immediate value, register, or memory location.
- The Destination and Source **CANNOT** both be memory locations.
- The Destination and Source must be of the same size **(8-, 16-. 32-bit).**
- All logical instructions, except **NOT**, affect the **status flags**
- Except NOT, all logical instructions clear carry flag (**CF**) and overflow flag (**OF**).
- Remaining three flags record useful information: Zero flag (ZF), Sign flag (SF), Parity flag (PF).

Applications

1. The main usage of bitwise logical instructions is:
   - **to clear**
   - **to set**
   - **to invert**
   - **to isolate**

   some selected bits in the Destination operand.

- To do this, a Source bit pattern known as a **mask** is constructed. The mask bits are chosen so that the selected bits are modified in the desired manner when an instruction of the form:

```
LOGIC_INSTRUCTION   Destination , Mask
```

- Is executed. The Mask bits are chosen based on the following properties of **AND**, **OR**, and **XOR** :
- If X represents a bit (either 0 or 1) then:

X AND 0 = 0	X OR 0 = X	X XOR 0 = X
X AND 1 = X	**X OR 1 = 1**	**X XOR 1 = X'**

Thus,

- The **AND** instruction can be used to **CLEAR** specific Destination bits while preserving the others. A **zero mask** bit clears the corresponding Destination bit; a one mask bit preserves the corresponding destination bit.
- The **OR** instruction can be used to **SET** specific Destination bits while preserving the others. A **one mask** bit sets the corresponding Destination bit; a zero mask bit preserves the corresponding Destination bit.
- The **XOR** instruction can be used to **INVERT** specific Destination bits while preserving the others. A one mask bit inverts the corresponding Destination bit; a zero mask bit preserves the corresponding Destination bit.

Example 1

- Task: Convert the character in AL to upper case.
- Solution: Use the **AND** instruction to clear bit 5.

```
mov al,'a'      ; AL = 01100001b
and al,11011111b    ; AL = 01000001b
```

Example 2

- Task: Convert a binary decimal byte into its equivalent ASCII decimal digit.
- Solution: Use the OR instruction to set bits 4 and 5.

```
mov al,6  ; AL = 00000110b
or  al,00110000b    ; AL = 00110110b
```

The ASCII digit '6' = 00110110b

Example 3

Converting Characters to Uppercase

- AND instruction can convert characters to uppercase

```
     'a'  = 0 1 1 0 0 0 0 1   'b'  = 0 1 1 0 0 0 1 0
     'A'  = 0 1 0 0 0 0 0 1   'B'  = 0 1 0 0 0 0 1 0
```

- Solution: Use the AND instruction to clear bit 5

```
          mov  ecx, LENGTHOF mystring
          mov  esi, OFFSET mystring
L1:  and  BYTE PTR [esi], 11011111b     ; clear bit 5
          inc  esi
          loop L1
```

2. The other usage of the logical instructions is represent the set operation
   - Some application manipulate sets of items selected from a limited-sized universal set
   - To represent the **Set Complement** operation we use the **NOT** instruction
   - To represent the **Set Intersection** operation we use the **AND** instruction
   - To represent the **Set Union** operation we use the **OR** instruction

Example 1

```
SetX = 10000000 00000000 00000000 00000111
.code
mov eax, SetX
not eax ;   the complement of SetX
```

Example 2

```
SetX = 10000000 00000000 00000000 00000111
SetY = 10000011 00000110 01100001 10010001
.code
mov eax, SetX
and eax, SetY ;   the SetX and SetY intersection saved in EAX
               ; EAX = 10000000 00000000 00000000 00000001
```

Example 3

```
SetX = 10000000 00000000 00000000 00000111
SetY = 10000011 00000110 01100001 10010001
.code
mov eax, SetX
or eax, SetY ;   the SetX and SetY union saved in EAX
               ; EAX = 10000011 00000110 01100001 10010111
```