Chapter 6 (Part a)

Conditional Processing

Eng. Eman R. Habib

April, 2014
6.2 Boolean and Comparison Instructions

**AND Instruction**
- Boolean (bitwise) AND between each pair of matching bits in two operands and places the result in the destination operand:
\[
\text{AND \ destination, source}
\]
- The following operand combinations are permitted:
  \[
  \text{AND \ reg, reg}
  \text{AND \ reg, mem}
  \text{AND \ reg, imm}
  \text{AND \ mem, reg}
  \text{AND \ mem, imm}
  \]
- The operands can be 8, 16, or 32 bits, and they must be the same size.
- Clear 1 or more bits in an operand without affecting other bits.
- Always clears the Overflow and Carry flags. It modifies the Sign, Zero, and Parity flags in a way that is consistent with the value assigned to the destination operand.

**Converting Characters to Upper Case:**
\[
\begin{align*}
0 & \ 1 & 1 & 0 & 0 & 0 & 0 & 1 = 61h \ (\text{"a"}) \\
0 & \ 1 & 0 & 0 & 0 & 0 & 0 & 1 = 41h \ (\text{"A"})
\end{align*}
\]
The rest of the alphabetic characters have the same relationship.
If we AND any character with 11011111 binary, all bits are unchanged except for bit 5, which is cleared.

**Converting ASCII to Binary Digits:**
\[
\begin{align*}
0 & \ 0 & 1 & 1 & 0 & 0 & 0 & 1 = \ '1' \\
0 & \ 0 & 0 & 0 & 0 & 0 & 1 & 1 = \ 1
\end{align*}
\]
Use the AND instruction to clear bits 4 to 7 and a1,0Fh ; Convert ASCII '0' to '9' to binary

**OR Instruction**
- Boolean OR between each pair of matching bits in two operands and places the result in the destination operand:
\[
\text{OR \ destination, source}
\]
- The OR instruction uses the same operand combinations as the AND instruction:
- The operands can be 8, 16, or 32 bits, and they must be the same size.
- Set 1 or more bits in an operand without affecting any other bits.
- Always clears the Overflow and Carry flags. It modifies the Sign, Zero, and Parity flags in a way that is consistent with the value assigned to the destination operand.

**Converting Characters to Lowercase:**
\[
\begin{align*}
0 & \ 1 & 0 & 0 & 0 & 0 & 0 & 1 = 41h \ (\text{"A"}) \\
0 & \ 1 & 1 & 0 & 0 & 0 & 0 & 1 = 61h \ (\text{"a"})
\end{align*}
\]
The rest of the alphabetic characters have the same relationship.
If we OR any character with 00100000 binary, all bits are unchanged except for bit 5, which is set.
Converting Binary Digits to ASCII:

0 0 0 0 0 0 0 1 = 1
0 0 1 1 0 0 0 1 = ’1’

Use the OR instruction to set bits 4 and 5
or al,30h ; Convert binary digit 0 to 9 to ASCII

❖ XOR Instruction
- Boolean exclusive-OR between each pair of matching bits in two operands and stores the result in the destination operand:
  \[
  \text{XOR} \text{ destination,source}
  \]
- The XOR instruction uses the same operand combinations and sizes as the AND and OR instructions.
- A bit exclusive-ORed with 0 retains its value, and a bit exclusive-ORed with 1 is toggled (complemented).
- XOR reverses itself when applied twice to the same operand.
- Always clears the Overflow and Carry flags. It modifies the Sign, Zero, and Parity flags in a way that is consistent with the value assigned to the destination operand.

❖ NOT Instruction
- Toggles (inverts) all bits in an operand. The result is called the one’s complement.
- The following operand types are permitted:
  NOT \text{ reg}
  NOT \text{ mem}
- No flags are affected by the NOT instruction.

❖ TEST Instruction
- The TEST instruction performs an implied AND operation between each pair of matching bits in two operands and sets the Sign, Zero, and Parity flags based on the value assigned to the destination operand.
- The only difference between TEST and AND is that TEST does not modify the destination operand.
- The TEST instruction permits the same operand combinations as the AND instruction.
- TEST is particularly valuable for finding out whether individual bits in an operand are set.

❖ CMP Instruction
- The CMP (compare) instruction performs an implied subtraction of a source operand from a destination operand. Neither operand is modified:
  \[
  \text{CMP} \text{ destination,source}
  \]
- CMP uses the same operand combinations as the AND instruction.
- The CMP instruction changes the Overflow, Sign, Zero, Carry, Auxiliary Carry, and Parity flags according to the value the destination operand would have had if actual subtraction had taken place.
- When two unsigned operands are compared, the Zero and Carry flags indicate the following relations between operands:
- When two signed operands are compared, the Sign, Zero, and Overflow flags indicate the following relations between operands:

<table>
<thead>
<tr>
<th>CMP Results</th>
<th>ZF</th>
<th>CF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination &lt; source</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Destination &gt; source</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Destination = source</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

![Table of CMP Results and Flags](image)

**Section 6.2 Review**

4. Write a single instruction using 16-bit operands that clears the high 8 bits of AX and does not change the low 8 bits.
   
   ```
   and ax,00FFh
   ```

5. Write a single instruction using 16-bit operands that sets the high 8 bits of AX and does not change the low 8 bits.
   
   ```
   or ax,0FF00h
   ```

6. Write a single instruction (other than NOT) that reverses all the bits in EAX.
   
   ```
   xor eax,0FFFFFFFFh
   ```

7. Write instructions that set the Zero flag if the 32-bit value in EAX is even and clear the Zero flag if EAX is odd.
   
   ```
   test eax,1 ; (low bit set if eax is odd)
   ```

8. Write a single instruction that converts an uppercase character in AL to lowercase but does not modify AL if it already contains a lowercase letter.
   
   ```
   or al,00100000b
   ```

9. Write a single instruction that converts an ASCII digit in AL to its corresponding binary value. If AL already contains a binary value (00h to 09h), leave it unchanged.
   
   ```
   and al,00001111b
   ```
6.3 Conditional Jumps

- **Jcond Instruction**

A conditional jump instruction branches to a destination label when a status flag condition is true. Otherwise, if the flag condition is false, the instruction immediately following the conditional jump is executed. The syntax is as follows:

```
Jcond destination
```

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags / Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JZ</td>
<td>Jump if zero</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>JNZ</td>
<td>Jump if not zero</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>JC</td>
<td>Jump if carry</td>
<td>CF = 1</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if not carry</td>
<td>CF = 0</td>
</tr>
<tr>
<td>JO</td>
<td>Jump if overflow</td>
<td>OF = 1</td>
</tr>
<tr>
<td>JNO</td>
<td>Jump if not overflow</td>
<td>OF = 0</td>
</tr>
<tr>
<td>JS</td>
<td>Jump if signed</td>
<td>SF = 1</td>
</tr>
<tr>
<td>JNS</td>
<td>Jump if not signed</td>
<td>SF = 0</td>
</tr>
<tr>
<td>JP</td>
<td>Jump if parity (even)</td>
<td>PF = 1</td>
</tr>
<tr>
<td>JNP</td>
<td>Jump if not parity (odd)</td>
<td>PF = 0</td>
</tr>
</tbody>
</table>

- **Equality Comparisons**

In some cases, two operands are compared; in other cases, a jump is taken based on the value of CX or ECX. In the table, the notations leftOp and rightOp refer to the left (destination) and right (source) operands in a CMP instruction:

```
CMP leftOp, rightOp
```

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE</td>
<td>Jump if equal (leftOp = rightOp)</td>
</tr>
<tr>
<td>JNE</td>
<td>Jump if not equal (leftOp ≠ rightOp)</td>
</tr>
<tr>
<td>JCXZ</td>
<td>Jump if CX = 0</td>
</tr>
<tr>
<td>JECXZ</td>
<td>Jump if ECX = 0</td>
</tr>
</tbody>
</table>
Untsised Comparisons

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>Jump if above (if leftOp &gt; rightOp)</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not below or equal (same as JA)</td>
</tr>
<tr>
<td>JAE</td>
<td>Jump if above or equal (if leftOp ≥ rightOp)</td>
</tr>
<tr>
<td>JNB</td>
<td>Jump if not below (same as JAE)</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if below (if leftOp &lt; rightOp)</td>
</tr>
<tr>
<td>JNAE</td>
<td>Jump if not above or equal (same as JB)</td>
</tr>
<tr>
<td>JBE</td>
<td>Jump if below or equal (if leftOp ≤ rightOp)</td>
</tr>
<tr>
<td>JNA</td>
<td>Jump if not above (same as JBE)</td>
</tr>
</tbody>
</table>

Signed Comparisons

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JG</td>
<td>Jump if greater (if leftOp &gt; rightOp)</td>
</tr>
<tr>
<td>JNLE</td>
<td>Jump if not less than or equal (same as JG)</td>
</tr>
<tr>
<td>JGE</td>
<td>Jump if greater than or equal (if leftOp ≥ rightOp)</td>
</tr>
<tr>
<td>JNL</td>
<td>Jump if not less (same as JGE)</td>
</tr>
<tr>
<td>JL</td>
<td>Jump if less (if leftOp &lt; rightOp)</td>
</tr>
<tr>
<td>JNGE</td>
<td>Jump if not greater than or equal (same as JL)</td>
</tr>
<tr>
<td>JLE</td>
<td>Jump if less than or equal (if leftOp ≤ rightOp)</td>
</tr>
<tr>
<td>JNG</td>
<td>Jump if not greater (same as JLE)</td>
</tr>
</tbody>
</table>

Section 6.3 Review

1. Which jump instructions follow unsigned integer comparisons?
   JA, JNBE, JAE, JNB, JB, JNAE, JBE, JNA

2. Which jump instructions follow signed integer comparisons?
   JG, JNLE, JGE, JNL, JL, JNGE, JLE, JNG

3. Which conditional jump instruction branches based on the contents of ECX?
   JECXZ

4. (Yes/No): Are the JA and JNBE instructions equivalent? Explain your answer.
5. Suppose the CMP instruction compares the integers 7FFFh and 8000h. Show how the
JB and JL instructions would generate different results if used after comparing these
values.

```
mov ax,7FFFh ; 7FFFh = 32767d 8000h = 32768d
cmp ax,8000h
JB L1 ; unsigned cmp, and the jump will be taken.

mov ax,7FFFh ; 7FFFh = +32767d 8000h = -32768d
cmp ax,8000h
JL L1 ; signed cmp, and the jump will not be taken.
```

6. Which conditional jump instruction is equivalent to the JNA instruction?
   JBE

7. Which conditional jump instruction is equivalent to the JNGE instruction?
   JL

8. (Yes/No): Will the following code jump to the label named Target?
   mov ax,8109h
cmp ax,26h
jg Target

9. (Yes/No): Will the following code jump to the label named Target?
   mov ax,-30
cmp ax,-50
jg Target

10. (Yes/No): Will the following code jump to the label named Target?
    mov ax,-42
cmp ax,26
ja Target

11. Write instructions that jump to label L1 when the unsigned integer in DX is less than or
    equal to the integer in CX.
    
    ```
cmp dx, cx
jbe L1
```
12. Write instructions that jump to label L2 when the signed integer in AX is greater than
the integer in CX.
    \texttt{cmp ax, cx}
    \texttt{jg L2}

13. Write instructions that first clear bits 0 and 1 in AL. Then, if the destination operand is
equal to zero, the code should jump to label L3. Otherwise, it should jump to label L4
    \texttt{and al, 1111100b}
    \texttt{jz L3}
    \texttt{jmp L4}

😊 Best Wishes 😊