Logical Operations
3/9/01 Lecture #13 16.070

• We have been performing arithmetic operations
  ➢ Use arithmetic operators; e.g., +, -
  ➢ Are performed on values represented as binary patterns; e.g., integer, float

• Logical operations are another class of operations
  ➢ Use logical operators; e.g., AND, OR
  ➢ Are performed on binary patterns

• Logical operations are used in computer science
  ➢ To express conditionals; e.g., in if construct
  ➢ To perform bit manipulation; e.g., masking
  ➢ To construct the basic components in a computer; i.e., logic gates

• Refer to C book, pp. 365-370
**Boolean Algebra**

- **Boolean Algebra** or **Boolean Logic** is the Algebra of Logic

- Handy for when you need to perform logical operations on logical variables

  - A **Logical Variable** has a value of 1 or 0, True or False
  
  - Performing Boolean Algebra on logical variables results in a 1 or 0, True or False

  - **C implementation of Logical Operators**
    
    - Zero is interpreted as False and non-zero is interpreted as True
    
    - Operations return zero if False and one if True
Overview of Logical Operators

- Logical operators, their functions, and their representations in C

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th># of Inputs</th>
<th>Function</th>
<th>C Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>1</td>
<td>Negate/complement</td>
<td>!</td>
</tr>
<tr>
<td>AND</td>
<td>2</td>
<td>Result is T iff both inputs are T</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>OR</td>
<td>2</td>
<td>Result is T if either input is T</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>2</td>
<td>Result is 1 if inputs are different</td>
<td></td>
</tr>
<tr>
<td>NAND</td>
<td>2</td>
<td>Result is F iff both inputs are T</td>
<td></td>
</tr>
<tr>
<td>NOR</td>
<td>2</td>
<td>Result is F if either input is T</td>
<td></td>
</tr>
</tbody>
</table>
AND ("ALL") - Binary Function (denoted by && in C)

- Result is True (1) if and only if (IFF) both inputs are True; else Result is False (0)
  
  \[
  \begin{array}{c|c|c}
  x & y & x \text{ AND } y \\
  \hline
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 0 & 0 \\
  1 & 1 & 1 \\
  \end{array}
  \]

- Truth Table representation

- Gate Representation

\[ Z = x_1 \land x_2 \land \ldots \land x_n \]
Truth Table for && Operator

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x &amp;&amp; y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>non-zero</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>non-zero</td>
<td>0</td>
</tr>
<tr>
<td>non-zero</td>
<td>non-zero</td>
<td>1</td>
</tr>
</tbody>
</table>
AND Examples

Logical AND can be used in if statement to determine hardware state of health

/* Determine if reaction wheel is spinning */

if ((rw == 1) && (torque_cmd > 0))
{
    printf ("Reaction wheel spinning\n")
} /* end if */

Given:   a = 1,  b = 1,  c = 0;  then solve the following

    a AND b =
    a AND c =
    b AND c =
Bitwise AND Logical Operation (denoted by & in C)

- Perform bit-by-bit comparison between two operands. For each bit position, resulting bit is 1 iff both corresponding bits in operand are 1.
- Examples of performing bitwise AND on bytes

```
  11111111  10101010
  AND 10001000   AND 10000010
      10001000      10000010
```

10001000  10000010

10001000  10000010
AND Exercises (&&)

• Evaluate the following expressions. True or False?
  
  \( (3 < 5) \)
  
  \( ((10/3) > 3) \And (3 > (10/4)) \)
  
  \( ((100 \cdot 3.5) / 2.94) < 120) \And \text{FALSE} \)

Bitwise AND Exercises (&)

• Perform the following bitwise AND logical operations
  
  \( (1110)_2 \And (0000)_2 = \)
  
  \( (10)_{10} \And (05)_{10} = \) \hspace{1cm} \text{(hint: convert to binary)}
  
  \( (F)_{16} \And (E)_{16} = \)
OR ("ANY") - Binary Function (denoted by || in C)

- Result is True (1) if either input is True; else Result is False (0)
  - 0 OR 0 = 0
  - 0 OR 1 = 1
  - 1 OR 0 = 1
  - 1 OR 1 = 1

- Truth Table representation

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Gate Representation

\[ Z = x_1 + x_2 + \ldots + x_n \]
# Truth Table for || Operator

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>non-zero</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>non-zero</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>non-zero</td>
<td>non-zero</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
OR Examples

• Logical OR can by used in if statement to check user input

/* If user enters 'Y' or 'y', say Hello! */
char response;
scanf ("%c", &response);
if ((response == 'Y') || (response == 'y'))
{
    printf ("Hello!\n")
}/* end if */

• Given: a = 1, b = 1, c = 0; then solve the following

    a OR b =
    a OR c =
    b OR c =
Bitwise OR Logical Operation ((denoted by | in C))

- Perform bit-by-bit comparison between two operands. For each bit position, resulting bit is 1 if *either* corresponding bit in operands is 1

```
11111111 OR 10001000 OR 10000010
```

```
11111111
10001000
11111111
```
OR Exercises (||)

• Evaluate the following expressions. True or False?
  
  $((10/3) > 3) \text{ || } (3 > (10/4))$
  
  $((100 \times 3.5) / 2.94) < 120) \text{ || } \text{TRUE}$
  
  $((3 < 5) \&\& (5 < 7))) \text{ || } ((12/4) > 3)$

Bitwise OR Exercises (|)

• Perform the following bitwise OR logical operations
  
  $(1110)_2 \text{ OR } (0000)_2 =$
  
  $(10)_{10} \text{ OR } (05)_{10} =$  \hspace{1cm} (hint: convert to binary)
  
  $(F)_{16} \text{ OR } (E)_{16} =$
NOT - Unary Function (denoted by ! in C)

- Performs the Complement: Result is True (1) if input is False; else Result is False (0)
  \[
  \text{NOT } 1 = 0 \\
  \text{NOT } 0 = 1
  \]

- Truth Table representation

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

- Truth Table for ! Operator

<table>
<thead>
<tr>
<th>x</th>
<th>!x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>\text{non-zero}</td>
<td>0</td>
</tr>
</tbody>
</table>

- Gate Representation (Inverter)
NOT Examples

• 🚨 Careful when using Logical NOT as conditional for loop

    /* Count down by twos */
    int i, countdown = 99;
    for (i = countdown, !i, i = i - 2)
    {
        printf ("Countdown = %d\n", i)
    } /* end if */

• Given:   a = 1,  b = 2,  c = 0; then solve the following
    NOT a =
    NOT b =
    NOT c =
Bitwise NOT Logical Operation, "One's Complement"

(denoted by ~ in C)

• For each bit position, change each 1 to a 0 and each 0 to a 1

\[ \neg(10101010) = (01010101) \]
\[ \neg(11111111) = (00000000) \]
XOR - Exclusive OR Binary Function (not represented in C)

• Result is True (1) if the two inputs are different; else Result is False (0)
  0 XOR 0 = 0
  0 XOR 1 = 1
  1 XOR 0 = 1
  1 XOR 1 = 0

• Truth Table representation

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Gate Representation
**XOR Examples**

- Given: \( a = T, \ b = T, \ c = F; \) then solve the following
  - \( a \text{ XOR } b = \)
  - \( a \text{ XOR } c = \)
  - \( b \text{ XOR } c = \)

**Bitwise Logical Operation ((denoted by \(^\wedge\) in C))**

- Perform bit-by-bit comparison between two operands. For each bit position, resulting bit is 1 if corresponding bits in operands are different
  - \((10101010) \text{ XOR } (10000010) = (00101000)\)
  - \((11111111) \text{ XOR } (10001000) = (01110111)\)
DeMorgan's Law

• Negate the inputs and output of an AND gate:

\[ x = 0 \quad y = 1 \quad z = 1 \]

• Create the truth table that corresponds with this circuit

\[
\begin{array}{c|c|c|c|c|c}
  x & y & x' & y' & \text{AND} & z \\
  \hline
  0 & 0 & 1 & 1 & 1 & 0 \\
  0 & 1 & 1 & 0 & 0 & 1 \\
  1 & 0 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

• This can be described algebraically: \((x' \text{ AND } y')' = x \text{ OR } y\)

• DeMorgan's Law: \((x \text{ AND } y)' = x' \text{ OR } y', \quad (x \text{ OR } y)' = x' \text{ AND } y'\)
Summary

- Logical Operators evaluate the truth or falseness of expressions and returns a TRUE (=1) or FALSE (=0)

<table>
<thead>
<tr>
<th>Operator</th>
<th>C Logical</th>
<th>C Bitwise</th>
<th>00</th>
<th>01 or 10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
<td>&amp;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>n/a</td>
<td>^</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
<td>~</td>
<td></td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>