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
- <u>Logical operations</u> 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

Logical Operator	# of Inputs	Function	C Representation
NOT	1	Negate/complement	**!**
AND	2	Result is T iff both inputs are T	**&&**
OR	2	Result is T if either input is T	||
XOR	2	Resut is 1 if inputs are different	
NAND	2	Result is F iff both inputs are T	
NOR	2	Result is F if either input is T	

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)

0 AND 0 = 0

0 AND 1 = 0

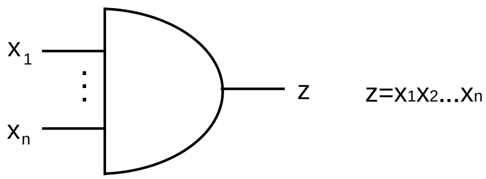
1 AND 0 = 0

1 AND 1 = 1

- <u>Truth Table representation</u>

x	y	AND
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

- <u>Gate Representation</u>



Truth Table for && Operator

x	y	x && y
0	0	0
non-zero	0	0
0	non-zero	0
non-zero	non-zero	1

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
```

AND Exercises (&&)

- Evaluate the following expressions. True or False?

  (3 < 5)

  ((10/3) > 3) AND (3 > (10/4))

  ((100 * 3.5) / 2.94) < 120) AND FALSE

Bitwise AND Exercises (&)

- Perform the following bitwise AND logical operations

  $(1110)_2$ AND $(0000)_2$ =

  $(10)_{10}$ AND $(05)_{10}$ = (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

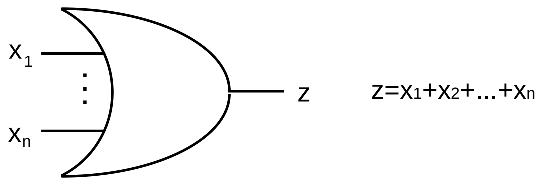
1 OR 0 = 1

1 OR 1 = 1

- <u>Truth Table representation</u>

x	y	OR
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

- <u>Gate Representation</u>



Truth Table for || Operator

x	y	x || y
0	0	0
non-zero	0	1
0	non-zero	1
non-zero	non-zero	1

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              10101010
OR    10001000        OR    10000010
      --------              --------
      11111111              10101010
```

OR Exercises (||)

- Evaluate the following expressions. True or False?

((10/3) > 3) || (3 > (10/4))

((100 * 3.5) / 2.94) < 120) || TRUE

((3 < 5) && (5 < 7))) || ((12/4) > 3)

Bitwise OR Exercises (|)

- Perform the following bitwise OR logical operations

$(1110)_2$ OR $(0000)_2$ =

$(10)_{10}$ OR $(05)_{10}$ = (hint: convert to binary)

$(F)_{16}$ 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)

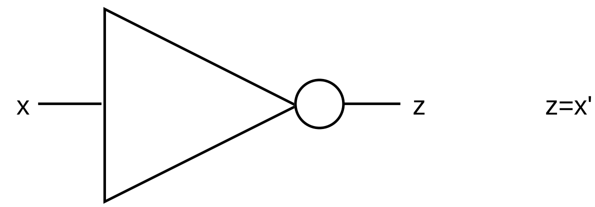
NOT 1 = 0

NOT 0 = 1

- Truth Table representation

x	NOT
0	**1**
1	**0**

- Gate Representation (Inverter)



- Truth Table for ! Operator

x	!x
0	**1**
non-zero	**0**

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

~(10101010) = (01010101)

~(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

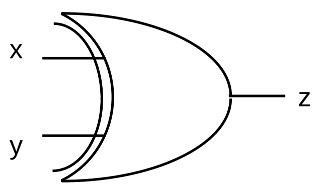
1 XOR 0 = 1

1 XOR 1 = 0

- Truth Table representation

x	y	XOR
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

- Gate Representation



$z = x \oplus y$

XOR Examples

- Given: a = T, b = T, c = F; then solve the following

  a XOR b =

  a XOR c =

  b XOR c =

Bitwise Logical Operation ((denoted by ^ in C))

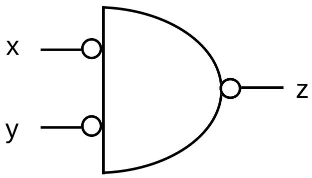
- Perform bit-by-bit comparison between two operands. For each bit position, resulting bit is 1 if corresponding bits in operands are different

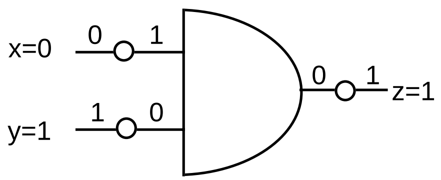
  (10101010) XOR (10000010) = (00101000)

  (11111111) XOR (10001000) = (01110111)

DeMorgan's Law

- Negate the inputs and output of an AND gate:





- Create the truth table that corresponds with this circuit

x	y	x'	y'	AND	z
0	0	1	1	**1**	**0**
0	1	1	0	**0**	**1**
1	0	0	1	**0**	**1**
1	1	0	0	**0**	**1**

- This can be described algebraically: (x' AND y')' = x OR y
- DeMo<u>rgan's Law</u>: (x AND y)' = x' OR y', (x OR y)' = x' AND y'

Summary

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

Operator	C Logical	C Bitwise	00	01 or 10	11
AND	&&	&			
OR	||	|			
XOR	n/a	^			
NOT	!	~		--	