Bitwise Operator and Typecasting

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2017/index.html

Indian Statistical Institute Kolkata

August 17, 2017

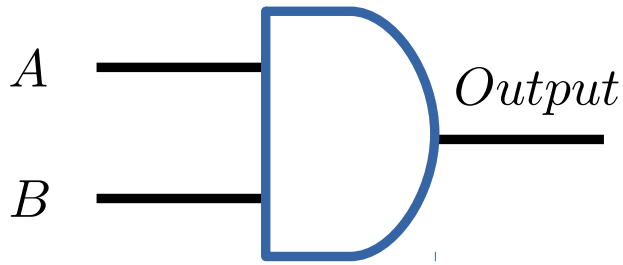
Outline

Outline

Different Types of Bitwise Operators

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
$\ll$	Left Shift
$\gg$	Right Shift
$\sim$	One's Complement

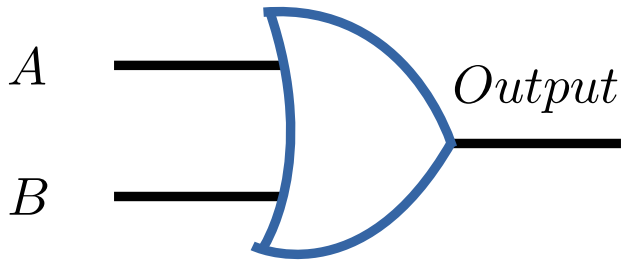
Boolean Logic: AND Gate



Truth table

A	B	Output (A & B)
0	0	0
1	0	0
0	1	0
1	1	1

Boolean Logic: OR Gate



Truth table

A	B	Output (A | B)
0	0	0
1	0	1
0	1	1
1	1	1

Performing on a sequence of bits

AND

```
1 0 1 1 1 0 1 0
1 1 1 1 0 0 0 0
---------------
1 0 1 1 0 0 0 0
```

0 or 1 with a 0 $\Rightarrow$ 0
1 with a 1 $\Rightarrow$ 1 no change
0 with a 1 $\Rightarrow$ 0 no change

OR

```
1 0 1 1 1 0 1 0
1 1 1 1 0 0 0 0
---------------
1 1 1 1 1 0 1 0
```

0 or 1 with a 1 $\Rightarrow$ 1
1 with a 0 $\Rightarrow$ 1 no change
0 with a 0 $\Rightarrow$ 0 no change

Applications

Bitwise AND operator & is often used to mask off some set of bits

$$\begin{array}{r} 1\ 0\ 1\ \mathbf{1}\ 1\ 0\ 1\ 0 \\ 1\ 1\ 1\ \mathbf{0}\ 1\ 1\ 1\ 1 \\ \hline 1\ 0\ 1\ \mathbf{0}\ 1\ 0\ 1\ 0 \end{array}$$

Bitwise OR operator | is often used to mask on some set of bits

$$\begin{array}{r} 1\ 0\ 1\ \mathbf{0}\ 1\ 0\ 1\ 0 \\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0 \\ \hline 1\ 0\ 1\ \mathbf{1}\ 1\ 0\ 1\ 0 \end{array}$$

XOR and 1's Complement

Truth table

A	B	Output (A ^ B)
0	0	0
1	0	1
0	1	1
1	1	0

Truth table

A	Output ($\sim$ A)
0	1
1	0

Left Shift and Right Shift

Left Shift $\ll$

$1\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \ll 1$

1 | 1 1 0 1 0 0 1

1 1 0 1 0 0 1 | **0**

$1\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \ll 3$

1 1 1 | 0 1 0 0 1

1 1 0 1 0 | **0 0 0**

Right Shift $\gg$

$1\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \gg 1$

1 1 1 0 1 0 0 | **1**

0 | 1 1 1 0 1 0 0

$1\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \gg 3$

1 1 1 0 1 | **0 0 1**

0 0 0 | 1 1 1 0 1

Use of Operatoes

Making the third bit position of 1 1 1 0 1 0 0 1 turned off

$1\ 1\ 1\ 0\ \mathbf{1}\ 0\ 0\ 1$

$1\ 1\ 1\ 0\ \mathbf{1}\ 0\ 0\ 1\ \&\ 1\ 1\ 1\ 1\ \mathbf{0}\ 1\ 1\ 1$

$1\ 1\ 1\ 0\ \mathbf{1}\ 0\ 0\ 1\ \&\ (\sim (0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0))$

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \ll 3 \Leftrightarrow 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0$

$1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \&\ ((\sim (0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)) \ll 3)$

$\Leftrightarrow 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ \&\ ((\sim 1) \ll 3)$

Outline

Typecasting

A type cast is basically a conversion from one type to another.

Two types of type conversion:

1. Implicit Type Conversion Also known as "automatic type conversion".

2. Explicit Type Conversion

Implicit Vs Explicit

Done by the compiler

In an expression more than one data type is present to avoid loss of data.

Data types of variables upgraded to data type of the variable with largest data type.

bool $\rightarrow$ **char** $\rightarrow$ **short int** $\rightarrow$ **int** $\rightarrow$ **unsigned int** $\rightarrow$ **long** $\rightarrow$ **unsigned** $\rightarrow$ **long long** $\rightarrow$ **float** $\rightarrow$ **double** $\rightarrow$ **long double**

Possibility of loosing information.

Signed is implicitly converted to unsigned

It is user defined.

The user can type cast the result to make it of a particular data type.

The syntax in C:

(data-type) expression

var = (float) 5 / 4;

Built-in Typecast Function

Inbuilt typecast functions in C

Typecast function	Description
atof()	Converts string to float
atoi()	Converts string to int
atol()	Converts string to long
itoa()	Converts int to string
ltoa()	Converts long to string

Example:

```
char a[10] = "5.46";
float var = atof(a);
printf("var is = %f\n");
```

```
OUTPUT:
var is = 5.460000
```

Problem 1

getbits(x, p, n) function returns the (right adjusted) n-bit field of x that begins at position p. We assume that bit position 0 is at the right end and that n and p are sensible positive values.
For example, *getbits*$(x, 4, 3)$ returns the three bits in positions 4, 3 and 2, right-adjusted.

Problem 1: Solution

getbits(x, p, n) function returns the (right adjusted) n-bit field of x that begins at position p. We assume that bit position 0 is at the right end and that n and p are sensible positive values.
For example, *getbits*$(x, 4, 3)$ returns the three bits in positions 4, 3 and 2, right-adjusted.

```
Type 1: unsigned getbits(unsigned x, int p, int n){
    int i; unsigned sum=0;
    for(i=0;i<n;i++){
        sum = sum + pow(2,i);
    }
    return ((x >> p+1-n) & sum);
}
Type 2: unsigned getbits(unsigned x, int p, int n){
    return (x >> (p+1-n)) & ~(~0 << n);
}
```

Explanation of Type 1 solution

$P = 4, n = 3$

$for(i = 0; i < n; i++)$
sum = sum + pow(2,i)

Then sum = 0 0 0 0 0 1 1 1

Let x be	0	1	0	1	1	0	1	1
We output bits	0	1	0	1	1	0	1	1

If $x \gg p + 1 - n$

0 0 0 1 0 1 1 0

Now $(x \gg p + 1 - n)$ & sum

0 0 0 0 0 1 1 0

Problem 2

Write a function $setbits(x, p, n, y)$ that returns x with the n bits that begin at position p set to the rightmost n bits of y , leaving the other bits unchanged.

Problem 3

Write a function $invert(x, p, n)$ that returns x with the n bits that begin at position p inverted (i.e., 1 changed into 0 and vice versa), leaving the others unchanged.

Problem 4

Write a function $rightrot(x, n)$ that returns the value of the integer x rotated to the right by n positions.

Problem 5

1. Implement bitwise AND operator in terms of other operators.
2. Implement bitwise OR operator in terms of other operators.
3. Implement bitwise XOR operator in terms of other operators
4. Write a C program using bitwise operators to check whether an integer is a power of two or not.