Bitwise Operators

Saturday, September 19, 2020 2:03 PM

Bitwise operators

Motivation: manipulate binary representations at the bit level

And (&): 1 if both bits are 1, 0 otherwise

ex.

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

& w/ 1 to let a bit through, & w/ 0 to zero it out

Or (|): 1 if either one or both are 1, zero otherwise

ex.

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

| w/ 1 to force a bit on, | w/ 0 to let it go through

Not (~): Unary operator, 1 if bit is 0, vice versa

ex.

a	~a
0	1
1	0

Xor (^): 1 if exactly one is 1, 0 otherwise

ex.

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

^ w/1 to flip a bit, ^ w/0 to let it through

Operation on multiple bits.

Applied to corresponding bits in each number.

ex. $\begin{array}{r} 0110 \\ \& 1100 \\ \hline 0100 \end{array}$ $\quad \begin{array}{r} 0110 \\ | 1100 \\ \hline 1110 \end{array}$ $\quad \begin{array}{r} 0110 \\ \wedge 1100 \\ \hline 1010 \end{array}$ $\quad \begin{array}{r} \sim 1100 \\ \hline 0011 \end{array}$

Bitmasks

Bit vectors and sets.

Ordered collection of bits to represent data

ex.

0	0	1	0	1	0	1	0
CS161	CS109	CS103	CS110	CS107	CS106X	CS106B	CS106A

Union -> use or

Intersection -> use and

Bitmasks.

Defn. Constructed bit pattern to manipulate or isolate

Specific bits in a bit vector

ex. Make nth bit 1 -> or it w/ bitvec of 0s with nth digit 1

Code.

```
#define CS106A  0x1    // 0000 0001
#define CS106B  0x2    // 0000 0010
#define CS106X  0x4    // 0000 0100
#define CS107   0x8    // 0000 1000
#define CS110   0x10   // 0001 0000
#define CS103   0x20   // 0010 0000
#define CS109   0x40   // 0100 0000
#define CS161   0x80   // 1000 0000

char classes = "...";
classes = classes | CS107;   // add CS107

classes = classes & ~CS103;  // remove CS103

if (classes & CS106B) {
    // taken CS106B
}
```

Demo: Powers of 2

Get the lowest byte in a 32-bit int.

```
int j = ...;
int k = j & 0xff;
```

Setting least significant byte to 1s.

int j = $\underbrace{\text{\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_}}_{0\text{'s}}\ \underbrace{\text{\_\_\_\_\_\_\_\_}}_{1}$;

```
int k = j | 0xff;
```

Flipping all but least significant byte.

```
int k = j ^ ~(0xff);
```

Detecting if a 32-bit int is a power of two.

```
int j = ________ ________ ________ ________;
// need exactly 1 1 and everything else 0
// subtract 1 to get all places below to flip to 1
// no digits in 2^k and 2^k -1 overlap
bool is_power = (j & (j - 1) == 0);
```

Bit Shift Operators

Left Shift (<<) Shifts bit pattern number of positions to the left; new bits on right are 0's, left bits shifted are lost.

ex.

```
00110111 << 2 = 11011100
01100011 << 4 = 00110000
10010101 << 4 = 01010000
```

Right Shift (>>) Shifts bit pattern number of positions to the right, new bits on left are filled w/ 0's (unsigned), or filled w/ MSB (signed) right bits shifted are lost.

Notes. Addition/subtraction have higher precedence than shifts - use parentheses

Integer literals are signed ints - specify type w/ L and U