Bit Manipulation

Comp 1402/1002

Octal and Hex Constants

Octal numbers are proceeded by 0 (zero)

Hexadecimal constants are proceeded by 0x

```
int octalNum = 077;
int decNum = 77;
int hexNum = 0x77;
```

Bitwise Operators

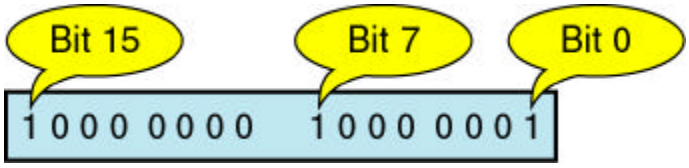
Name	Operator	Description
Bitwise And	&	Logical AND all bits
Bitwise Inclusive Or	|	Logical OR all bits
Bitwise Exclusive Or	^	Logical XOR all bits
Bitwise One's Complement	~	Negation of all bits
Shift Right	>>	Shift bits left filling with zeroes
Shift Left	<<	Shift bits right filling with zeroes

Bitwise Operators

Bits are usually numbers from left to right

In space efficient systems bits represent data

Chess – Bit board representation…



Bitwise AND

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	0
1	0	0
1	1	1

Example Bitwise AND

Octal numbers 257 and 463:

```
 010101111  (257)
&100110011  (463)
 000100011  ( 43)
```

Masking and Finding a bit

All zero except the bit to test

```
 1010101x          110011x0
&00000001 Mask    &00000010 Mask
 0000000x          000000x0
```

Note: either the result = Mask or Zero

Masking to clear a bit

All ones except the bit to clear

```
 11101111          11001110
&11111101 Mask    &10111111 Mask
 11101101          10001110
```

Bit is set to zero

Bitwise OR

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	1
1	0	1
1	1	1

Example Bitwise OR

Octal numbers 257 and 463:

```
 010101111  (257)
|100110011  (463)
 110111111  (677)
```

Masking to set a bit

All zeroes except the bit to set

```
 111011x1          1x001110
|00000010 Mask   |01000000 Mask
 11101111          11001110
```

Bit is set to one

Bitwise XOR

First Operand Bit	Second Operand Bit	Result
0	0	0
0	1	1
1	0	1
1	1	0

Example Bitwise XOR

Octal numbers 257 and 463:

```
 010101111  (257)
^100110011  (463)
 110011100  (634)
```

Masking to flip a bit

All zeroes except the bit to flip

```
 11101101          11001110
^00000010 Mask    ^01000000 Mask
 11101111          10001110
```

Bit is flipped

Bitwise Complement

Operand Bit	Result
0	1
1	0

Example Bitwise Complement

Octal numbers 257 and 463:

```
 ~100110011  (463)
  011001100  (314)
```

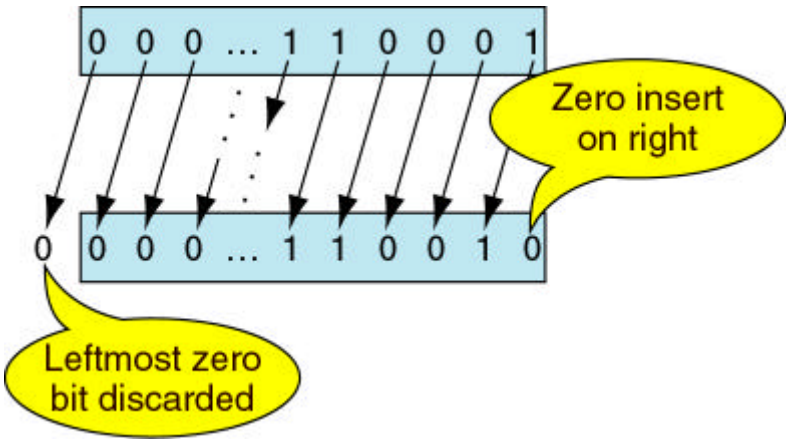
Shifting Bits

Two bit shifting operators:

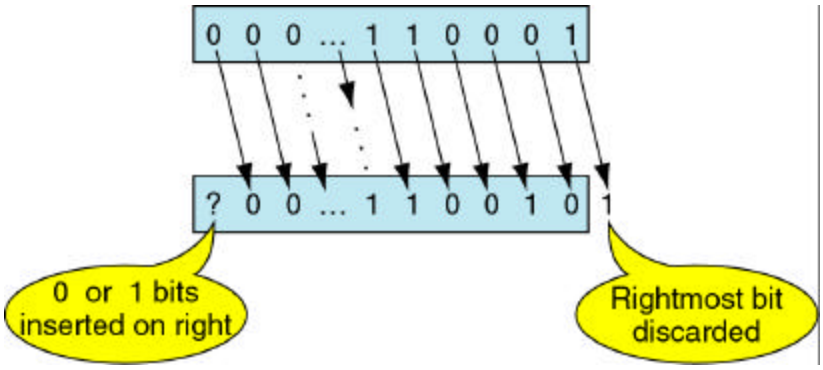
```
int x = 49;
x << 1; /* shift bits left 1 place */

x = 49;
x >> 1; /* shift bits right 1 place */
```

Shifting Bits



Shifting Bits



Shifting Bits

Shifting left adds zeros to the left

Shifting right is implementation specific:

- zeros added for **`unsigned`**
- Sign bit added otherwise

Shifting Bits

Can shift an arbitrary number of bits!

```
unsigned int x;
x = (1 << 1);  /* x = 2 */
x = (1 << 2);  /* x = 4 */
x = (1 << 15); /* x = 32768 */
```

Shifting **`unsigned ints`** by 1 is multiplying by 2

Shifting Bits

Can shift an arbitrary number of bits!

```
unsigned int x;
x = (32768 >> 1);  /* x = 16384 */
x = (32768 >> 2);  /* x =  8192 */
x = (32768 >> 14); /* x =     2 */
```

Shifting **`unsigned ints`** by 1 is dividing by 2

Creating Masks

```
m1 = 1 << 4;  /* creates 00010000 */

m2 = 1 << 7;  /* creates 10000000 */

m3 = m1 & m2; /* creates 10010000 */

m4 = ~m1;     /* creates 11101111 */

m5 = m1 – 1;  /* creates 00001111 */
```

Changing the Case

'A' and **'a'** differ in the fifth bit!

'A' is **01000001**
'a' is **01100001**

XOR with **1 << 5**
AND with **~(1 << 5)**