Bitwise Operators

Lecture 11

Administrative

- Final exam: 9:10-11:40, Dec. 8, in the lab (608B, 608C), vi or emacs only, closed book
- Topics that are out of scope of the final: Multi-threads, makefile, and bitwise operators
- Project due, 5pm, Dec. 9, send your code to yfang@scu.edu
- Office hours (final week):

Tuesday 4-5pm, Wed 3-4pm, Thursday 2-5pm

Bitwise Operators

- Positive integers are represented in the computer by standard binary numbers
  - ➢ Examples:

  short n = 13;

  →in memory - 0000 0000 0000 1101

  →$2^0 + 2^2 + 2^3 = 13$

  char c = 5;

  →in memory - 0000 0101

  →$2^0 + 2^2 = 5$

Bitwise Operators

- Bitwise operators
  - take operands of any integer type
    - char, short, int, long
  - but treat an operand as a collection of bits rather than a single number

Bitwise Negation

- Bitwise negation
  - Operand ~
    - Application of ~ to an integer produces a value in which each bit of the operand has been replaced by its negation
      - O becomes 1
      - 1 becomes 0
    - Example

      n = 0000 0000 0000 1101

      ~n = 1111 1111 1111 0010

Bitwise Shift

- Shift operators
  - shift left → <<
  - shift right → >>
- Take two integers operands
  - The value on the left is the number to be shifted
    - Viewed as a collection of bits that can move
  - The value on the right is a nonnegative number telling how far to move the bits

Bitwise Shift

- Operand
  - << shifts bits left
  - >> shifts bits right
- The bits that “fall of the end” are lost
- The “emptied” positions are filled with zeros

Bitwise Shift

- Example:

  n   0000 0000 0000 1101

  n << 1 → 0000 0000 0001 1010

  (lost 1 bit on the left)

  n << 4 → 0000 0000 1101 0000

  (lost 4 bits on the left)

  n >> 3 → 0000 0000 0000 0001

  (lost 3 bits on the right)

Bitwise Shift

- Compound assignment operators <<= and >>=
  - cause the value resulting from the shift to be stored in the variable supplied as the left operand

Bitwise AND, OR, and XOR

- The bitwise operators & (and), | (or) , and ^ (xor)
- Take two operands that are viewed as strings of bits
  - The operator determines each bit of the result by considering corresponding bits of each operand
  - For each bit i
    - $r_i = n_i \& m_i$ → 1 when both n_i and m_i are 1
    - $r_i = n_i | m_i$ → 1 when either n_i or m_i is 1
    - $r_i = n_i$ ^ m_i → 1 when n_i and m_i do not match

Bitwise AND, XOR, and OR

- Example:

n =	0000	0000	0000	1101
m =	0000	0000	0011	1100
m & n =	0000	0000	0000	1100

n =	0000	0000	0000	1101
m =	0000	0000	0011	1100
m | n =	0000	0000	0011	1101

Bitwise AND, XOR, and OR

- Example:

n =	0000	0000	0000	1101
m =	0000	0000	0011	1100
m ^ n =	0000	0000	0011	0001

Bitwise AND, XOR, and OR

- Compound assignment operators &=, |=, and ^=
  - cause the resulting value to be stored in the variable supplied as the left operand

Bitwise Operators

- Notes on shifting
  - << by 1 is the same as multiplying by 2
  - >> by 1 is the same as dividing by 2
- Notes on ~ and !
  - ~ and ! are different operators
    - ~ is a bitwise operator
      - each bit is reversed
    - ! is a logical complement or negation
      - !nonzero → false (zero)
      - !zero → true (one)

Bitwise Operators

- Notes on AND
  - x & 0 is always 0
  - x & 1 is always x
- Notes on OR
  - x | 1 is always 1
  - x | 0 is always x

Bitwise Operators

- Masks -- Used to change specific bits in an integer
  - To set specific bits
    - Use OR with a mask in which only the bits to be set have 1

```
short c =       0000 0101;
short mask =    0000 0010;
c | mask ==     0000 0111
```

  - To zero specific bits
    - Use AND with a mask in which only the bits to be zeroed have 0

```
short c =       0000 0101;
short mask =    1111 1110;
c & mask ==     0000 0100
```

Bitwise Operators

- Masks -- Used to change specific bits in an integer
  - To verify specific bit
    - Use AND with a mask in which only the bit to be verified is 1
    - Result == 0 implies that bit == 0
    - Result != 0 implies that bit == 1

short c = 0000 0101;
short mask = 0000 0100;
c & mask == 0000 0100 (not zero ==> bit is not zero)

Bitwise Operators

- Notes on XOR
  - x ^ 0 is always x
  - x ^ 1 is always ~x
  - x ^ x is always 0
  - x ^ ~x is always 1
  - if x ^ y == z, then
    - x == z ^ y and y == z ^ x