Lecture 7

Bitwise Operator

By Hafijur Rahman

Bitwise Operators

Goal: Manipulation of Individual bits.

Unsigned integers are normally used with the bitwise operators.

About Bits

- Any integral number can be thought of as a sequence of bits .
- These bits can be thought of as the representation of the number in *binary* (base-2) arithmetic
- For example, the number *87* (decimal) can be represented in binary as *1010111*

```
64   32   16   8   4   2   1
 1    0    1   0   1   1   1
```

- Individual bits are often used to represent Boolean values

The << and >> Operators

- The << (shift-left) operator is used to shift the bit pattern of a number a certain number of bits to the left
- The >> (shift-right) operator shifts the bit pattern in the opposite direction
- Shifting produces undefined results when going left “too far”, but when going right bits are simply truncated

The << and >> Operators

13 << 3 = 104

13 (*00001101*)

104 (*01101000*)

- Incidentally, $13 * 2^3 = 104$

52 >> 4 = 3

52 (*00110100*)

3 (*00000011*)

- Incidentally, $52 / 2^4 = 3$ (integer division)

The ~ Operator

- The `~ (bitwise inverse)` operator simply reverses each bit in the bit pattern (produces the "ones complement") of the number
- For example (using 8-bit numbers):

  ~211 = 44

  211 (*11010011*)

  44 (*00101100*)

- Incidentally, 211 + 44 = 255 (largest 8-bit number)

The & and | Operators

- The & ("Bitwise And") operator produces a result such that:
  - If a bit is on in *both* operands, it is on in the result
  - If a bit is off in *either* operand, it is off in the result
- The | ("Bitwise Or") operator produces a result such that:
  - If a bit is on in *either* operand, it is on in the result
  - If a bit is off in *both* operands, it is off in the result

The & Operator

The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.

Example:

104 & 13 = 8

104 (*01101000*)

13 (*00001101*)

8 (*00001000*)

Question: what is the difference between logical and(&&) and bitwise and(&)?

The | Operator

The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.

Example:

52 | 12 = 60

52 (*00110100*)

12 (*00001100*)

60 (*00111100*)

The ^ *(bitwise XOR)* Operator

The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.

Example:

15 ^ 127 = 112

15 (*00001111*)

127 (*01111111*)

112 (*01110000*)

The & and | Operators (cont.)

- "Turning a bit on" is usually achieved as follows:

```
#define FAILED     (1<<3)
status |= FAILED;
```

- "Turning a bit off" is usually achieved as follows:

```
status &= ~ FAILED;
```

- "Testing whether a bit is on" is usually achieved as follows:

```
if (status & FAILED)
if (!(status & FAILED))
```

Thanks