CME 112 Programming Languages II
Lecture 4- Bitwise Operators
Assist. Prof. Dr. Caner ÖZCAN

I'm human, so I'm free. If I'm free, I must be responsible.
SARTRE
Binary number system uses 0 or 1 for each digit.

For computer systems everything is coded in binary.

\[ (d_4d_3d_2d_1d_0)_2 = (d_0 \cdot 2^0) + (d_1 \cdot 2^1) + (d_2 \cdot 2^2) + (d_3 \cdot 2^3) + (d_4 \cdot 2^4) \]

\[ (10011)_2 = (1 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0) = 19 \]
HEXADECIMAL NUMBER SYSTEM

- Hexadecimal number system has 16 different symbols.

\[
(3FC)_{16} = (3 \cdot 16^2) + (F \cdot 16^1) + (C \cdot 16^0) = 768 + 240 + 12 = 1020
\]

\[
(1FA9)_{16} = (1 \cdot 16^3) + (F \cdot 16^2) + (A \cdot 16^1) + (9 \cdot 16^0) = 4096 + 3840 + 160 + 9 = 8105
\]

\[
(75)_{16} = (7 \cdot 16^1) + (5 \cdot 16^0) = 112 + 5 = 117
\]
SIGNED NUMBERS in BINARY

- Variables in C can be signed or unsigned.
- Think of a 8 bits (1 byte) number.

- If the number is negative then highest level bit (7th bit in this sample) is considered as **sign bit**.
- If the sign bit is 1 then number is negative, otherwise number is positive.
SIGN **ED NUMBERS in BINARY**

- Decimal equivalent of a signed binary number can be found with:
  \[(a_7a_6a_5a_4a_3a_2a_1a_0)_2 = (a_7 \cdot 2^7) + (a_6 \cdot 2^6) + ... + (a_1 \cdot 2^1) + (a_0 \cdot 2^0)\]

- \((10111011)_2 = -69\) (If the number is signed)
  \((10111011)_2 = 187\) (If the number is unsigned)

- \((11001101)_2 = -51\) (If the number is signed)
  \((11001101)_2 = 205\) (If the number is unsigned)

- \((01101101)_2 = 109\) (If the number is signed)
  \((01101101)_2 = 109\) (If the number is unsigned)
BITWISE OPERATORS

- Operations on bits at individual levels can be carried out using Bitwise operations in C.
- Bits come together to form a byte which is the lowest form of data that can be accessed in digital hardware.
- The whole representation of a number is considered while applying a bitwise operator.
- Each bit can have the value 0 or the value 1.
## BITWISE OPERATORS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Bitwise Exclusive OR</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Left shift</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Right shift</td>
</tr>
<tr>
<td>~</td>
<td>Ones's complement (unary)</td>
</tr>
</tbody>
</table>
Bitwise AND &

- The bitwise AND operator is a single ampersand: &.
- It is just a representation of AND and does its work on bits and not on bytes, chars, integers, etc.
- So basically a binary AND does the logical AND of the bits in each position of a number in its binary form.
  - 11001110 & 10011000 = 10001000
  - 5 & 3 = 1 ( 101 & 011 = 001)
Bitwise OR |  

- Bitwise OR works in the same way as bitwise AND.
- Its result is a 1 if one of the either bits is 1 and zero only when both bits are 0.
- Its symbol is '|' which can be called a pipe.
- $11001110 \ | \ 10011000 = 11011110$
- $5 \ | \ 3 = 7 \ (101 \ | \ 011 = 111)$
Bitwise Exclusive OR ^

- The Bitwise EX-OR performs a logical EX-OR function or in simple term adds the two bits discarding the carry.
- Thus result is zero only when we have 2 zeroes or 2 ones to perform on.
- Sometimes EX-OR might just be used to toggle the bits between 1 and 0.
- Thus: \( i = i ^ 1 \) when used in a loop toggles its values between 1 and 0.
- \( 5 \ ^ 3 = 6 \ ( 101 \ ^ 011 = 110 ) \)
# Bitwise Exclusive OR ^

<table>
<thead>
<tr>
<th>bit a</th>
<th>bit b</th>
<th>a &amp; b (a AND b)</th>
<th>a</th>
<th>b (a OR b)</th>
<th>a ^ b (a XOR b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
The symbol of right shift operator is `>>`.

For its operation, it requires two operands.

It shifts each bit in its left operand to the right. The number following the operator decides the number of places the bits are shifted (i.e. the right operand).

Thus by doing `number >> 3` all the bits will be shifted to the right by three places and so on.

Blank spaces generated on the left most bits are filled up by zeroes.

Right shift can be used to divide a bit pattern by 2 as shown:

- `10 >> 1 = 5`  \((1010) >> 1 = (0101)\)
Right Shift $>>$

- If the number is signed, then **sign extension** is done in right shift operation.
- Sign extension puts the highest bit’s value of the number into the blank spaces on the left most bits generated.

In this sample, as the original number’s highest bit is 1, new generated bits are also 1 after right shift.
Left Shift <<

- The symbol of left shift operator is `<<`.
- It shifts each bit in its left operand to the left. It works opposite to that of right shift operator.
- Blank spaces which is generated on the right most bits are filled up by zeroes.
- Left shift can be used to multiply an integer in multiples of 2 as in:
  - $5 \ll 1 = 10$  
  - $(101) \ll 1 = (1010)$
Unary Operator ~ One's Complement

- The one's complement (~) or the bitwise complement gets us the complement of a given number.
- Thus we get the bits inverted, for every bit 1 the result is bit 0 and conversely for every bit 0 we have a bit 1.
- \( \sim 5 = 2 \) (\( \sim 101 = 010 \))
PRACTISE on BITWISE OPERATIONS

- It is better to know how bitwise operations take place while we write programs.
- OR operator is the **union of bits** of two numbers having the value 1.
PRACTISE on BITWISE OPERATIONS

• AND operator is *intersection of bits* of two numbers having the value 1.

  
  ![Bitwise Operations Example]

  
  • In this sample there is no bits both have 1. So the intersection of all bits are 0.
PRACTISE on BITWISE OPERATIONS

• OR operator can be used to make a number’s bits 1.

  Before : 00000000111111110000000011111111
  Bits to be 1 : 00000000000000000000000000000000
  After : 000100001111111111110001000011111111

• AND operator can be used to check if a bit is 1 or not.

  0000011101011011
  1100110100010101
  00000000000000001000000000000000000
  → Mask
When the data which shows the states of keys information read from memory, the meaning of every bit is:

<table>
<thead>
<tr>
<th>Bit number</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Right shift pressed/not</td>
</tr>
<tr>
<td>1</td>
<td>Left shift pressed/not</td>
</tr>
<tr>
<td>2</td>
<td>Ctrl pressed/not</td>
</tr>
<tr>
<td>3</td>
<td>Alt pressed/not</td>
</tr>
<tr>
<td>4</td>
<td>Scroll on/off</td>
</tr>
<tr>
<td>5</td>
<td>Num Lock on/off</td>
</tr>
<tr>
<td>6</td>
<td>Caps Lock On/off</td>
</tr>
</tbody>
</table>
EXAMPLE: Keyboard Codes

- For checking whether numlock is on or off, we need to check bit number 5 of the key information data x.
- For this purpose we can perform binary AND operation with x and 32 operands.
- For example, if the key information data is 01101011, then we can use (00100000=32) to check is bit number 5 is 1 or 0.

  01101011 &
  00100000 → Mask

- As the bit number 5 is 1 in key information data the result is 32, otherwise result would be 0.
EXAMPLE: Ipv4 Address

• IPv4 addresses are stored in network packages in 32 bit form.
• Each 8 bits correspond to a segment of ip number which is separated by point.
  • For example: 192.168.1.2 is 0xc0a80102 in hexadecimal format.
• Lets write a program that reads 32 bits IPv4 adress and writes each segment separated with points.
EXAMPLE: Ipv4 Address

- For this we need to take each 8 bits from 32 bit IPv4 address using & bitwise operator with a suitable mask.
- For example if we want to take lowest 8 bits we have to use a mask 0x000000ff which will preserve the lowest 8 bits of the data.
EXAMPLE: Ipv4 Address

- If the preserved bits is not the lowest 8 we have to right shift the obtained number to the lowest 8 bit.
  - Value: 11000000101010000000000100000010 c0a80102 3232235778
  - Mask: 11111111000000000000000000000000 ff000000 4278190080
  - Result: 11000000000000000000000000000000 c0000000 3221225472
- The result we get here is 3221225472 and not 192 as we expected.
- The reason is that the obtained number is not in the lowest 8 bit. We need to shift the number 24 times to the right. (>> 24)
  - Value: 1100000001010100000000000100000010 c0a80102 3232235778
  - Mask: 11111111100000000000000000000000 ff000000 4278190080
  - Result: 00000000000000000000000000000000 c0000000 192
EXAMPLE: Ipv4 Address

```c
#include <stdio.h>
int main(void)
{
    unsigned int ipAdres = 0xc0a80102;
    unsigned maske = 0xfff00000;
    int segment1,segment2,segment3,segment4;
    int i, bit=32;
    unsigned tmp;
    for(i=1;i<=4;i++)
    {
        tmp = ipAdres & maske;
        if(i!=4){
            maske = maske >> 8;
            tmp = tmp >> (bit-i*8);
            printf("%d.",tmp);
        }
        else printf("%d",tmp);
    }
    getchar();
    return 0;
}
```
Binary Addition

```c
#include <stdio.h>
#include <stdlib.h>

//binary addition
int main()
{
    unsigned int x=3, y=1, sum, carry;
    sum = x ^ y;
    carry = x & y;
    while(carry!=0)
    {
        carry = carry << 1;
        x = sum;
        y = carry;
        sum = x ^ y;
        carry = x & y;
    }
    printf("%d", sum);
    getchar();
    return 0;
}
```
SORULAR

* M uzunluğundaki bir sayı dizisinde en az X kere tekrar eden sayıları ve en fazla tekrar eden sayıyı bulup ekranı yazdıran algoritmayı çiziniz. X, M ve sayı dizisi kullanıcı tarafından girilecektir.

* 10 tabanında verilen bir sayıyı kullanıcıının verdiği tabana dönüştüren ve ekranı yazdıran algoritmayı çiziniz. Yeni tabandaki sayı bir dizide saklanmalıdır.

* Çarpımlarla elde edilen sayının basamaklarının toplamı, çarpılan iki sayının birine eşit olan iki basamaklı tüm sayı ikililerini bulup ekranı yazdıran algoritmayı çiziniz.

Ör:  12*43=516  
     5+1+6=12

* En büyük elemanı 100 olan bir dizinin içindeki farklı eleman sayısını bulan algoritmayı çiziniz.

* Bir dizinin içinde arka arka en çok geçen sayı ikilisini bulan algoritmayı çiziniz.

Ör: 1 2 3 6 4 3 1 2 3 6 2 3 dizisi için cevap “2 3” ikilisi olacaktır.
Robin Williams  Matt Damon

GOOD WILL HUNTING

Ben Affleck  Minnie Driver  Stellan Skarsgård

The New Film by Gus Van Sant

www.moviegoods.com