计算机程序设计

Computer Programming

位运算

主讲：李卫海

目录
CONTENTS

◎ 位运算的概念

- 很多算法是按二进制位进行运算的
- 位运算速度快，效率高

- 位运算的运算对象是二进制的位
- 只能对整型数据(包括字符型)进行位运算
- 负数以补码形式参与运算

◎ 位运算符

- 注意位运算符与逻辑运算区别

运算符	运算	举例	优先级从高到低
			(逻辑非运算符)
~	按位取反	~flag	
			(算术运算符)
<<	左移	a << 2	
>>	右移	b >> 3	
			(关系运算符)
&	按位与	flag & 0x37	
^	按位异或	flag ^ 0xC4	
|	按位或	flag | 0x5A	
			(赋值运算符)

◎ 按位与 (Bitwise AND) &

- 运算规则
  - 0 & 0 = 0;
  - 0 & 1 = 0;
  - 1 & 0 = 0;
  - 1 & 1 = 1;
- 特殊用法
  - 特定位清零
  - 保留其它位

```
  1010,1101 (0xAD)
& 0110,1001 (0x69)
  0010,1001 (0x29)
```

```
  xxxx,xxxx
& 0110,0010 (0x62)
  0xx0,00x0
```

◎ 按位或 (Bitwise Inclusive OR) |

- 运算规则
  - 0 | 0 = 0;
  - 0 | 1 = 1;
  - 1 | 0 = 1;
  - 1 | 1 = 1;
- 特殊用法
  - 特定位置一
  - 保留其它位

```
  1010,1101 (0xAD)
| 0110,1001 (0x69)
-----------------
  1110,1101 (0xED)
```

```
  xxxx,xxxx
| 0110,0010 (0x62)
-----------------
  x11x,xx1x
```

◎ 按位异或 (Bitwise Exclusive OR) ^

- 运算规则
  - 0 ^ 0 = 0;
  - 0 ^ 1 = 1;
  - 1 ^ 0 = 1;
  - 1 ^ 1 = 0;
- 特殊用法
  - 特定位取反
  - 保留其它位

```
   1010,1101 (0xAD)
^  0110,1001 (0x69)
-------------------
   1100,0100 (0xC4)
```

```
   xxxx,xxxx
^  0110,0010 (0x62)
-------------------
   x̄x̄xx,xx̄x̄xx
```

◎ 按位取反 (Bitwise NOT) ~

- 运算规则
  - ~0 = 1;
  - ~1= 0;

```
~ 0110,1001 (0x69)
  1001,0110 (0x96)
```

◎ 左移 (Left Shift) <<

- 运算规则
  - `i << n`
  - 把i各位全部向左移动n位
  - 最左端的n位被移出丢弃
  - 最右端的n位用0补齐

```
5 << 3 = 40
00000101 (0x05) << 3 → 00101000 (0x28)
```

- 用法
  - 若没有溢出，则左移n位相当于乘上2^n
  - 运算速度比真正的乘法和幂运算快得多

◎ 右移 (Right Shift) >>

- 运算规则
  - `i >> n`
  - 把i各位全部向右移动n位
  - 最右端的n位被移出丢弃
  - 最左端的n位用0补齐(逻辑右移)
  - 或最左端的n位用符号位补齐(算术右移)

由编译系统的实现者决定。
为了可移植性，最好仅对无符号数进行移位运算

```
5 >> 2 = 1
00000101 (0x05) >> 2 → 00000001 (0x01)
```

- 用法
  - 右移n位相当于除以2^n，并舍去小数部分
  - 运算速度比真正的除法和幂运算快得多

◎ 位运算

- 按位运算时，两个操作数长度应相等，
- 否则先扩展，再运算
  - 两个操作数右端对齐
  - 短的数据左端用符号位补齐
    - 正数或无符号数左端用0补满
    - 负数左端用1补满

- 复合赋值运算符

  &=，^=，|=，~=，<<=，>>=

◎ 位运算

- 例，将16进制短整数按二进制打印输出
  - 输入：F1E2
  - 输出：1111000111100010
  - 输入：13A5
  - 输出：0001001110100101

```
include <stdio.h>
void main()
{   int i;
    short a;
    scanf("%X", &a);
    for (i=15;i>=0;i--)
        printf("%1d", a&1<<i ? 1 : 0);
}
```

◎ 结构体中的位

- 位段是以位为单位定义长度的结构体类型成员
- 定义形式：

  struct 位域结构名 {

  位段成员列表;

  };

  位段成员的说明形式为：

  类型说明符 位段名:位段长度;

```
struct packed_data {
    unsigned a:2;
    unsigned b:6;
    unsigned c:4;
    unsigned d:4;
    unsigned :0;
    int i;
}
```

位段成员

普通结构体成员

- 注意：
  - 一个位段分配在同一个存储单元之中，不能跨单元
    - 可以用 unsigned :0; 表示从下一个存储单元开始存放
    - 可以用未命名的位段来填充或调整位置，例如 unsigned :3;

◎ 结构体中的位

- 使用中应注意成员所占的位及其由长度限定的存取值域
  - 溢出了会怎样？自行编程测试一下
- 不能对位段应用取地址运算
- 不能用指针指向位段
- 位段数组不被允许

◎ 位运算举例

- 例，求char型数据二进制表示中1的个数
- 方法1：模拟进制转换，反复除2，累加余数

```
int count1(unsigned char v)
{   int num=0;
    while (v)
    {   if (v%2==1) num++;
        v/=2;
    }
    return(num);
}
```

◎ 位运算举例

- 例，求char型数据二进制表示中1的个数
- 方法2：将方法1中运算改为位运算

```
int count2(unsigned char v)
{   int num=0;
    while (v)
    {   num+=v &0x01;
        v>>=1;
    }
    return(num);
}
```

◎ 位运算举例

- 例，求char型数据二进制表示中1的个数
- 方法3：将循环次数降为1的个数

思路：设法使得每次循环都能减少一个1

```
int count3(unsigned char v)
{   int num=0;
    while (v)
    {   v&=(v-1);
        num++;
    }
    return(num);
}
```

减1会使得最后一个1变为0，与之后1就少了一个

◎ 位运算举例

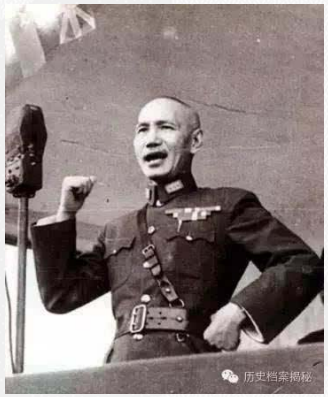
- 例，求char型数据二进制表示中1的个数
- 方法4：不用循环用判断分支

```
int count4(unsigned char v)
{   int num=0;
    switch (v)
    {   case 0x00: num=0; break;
        ......
    }
    return(num);
}
```

程序员：敲得累死了！敲错了算谁的？
存储器：代码不占空间啊？空间不是钱啊？
CPU：分支不花时间啊？

◎ 位运算举例

- 例，求char型数据二进制表示中1的个数
- 方法5：“以空间换时间”




```
int count5(unsigned char v)
{   static int countTable[256]={0,1,...};
    return countTable[v];
}
```

程序员：认真输入比绞尽脑汁还是轻松些的
存储器：我真的很便宜
CPU：引用很轻松

程序员黄金法则：不需要展示技巧的时候就用最直接的方法实现；不值得优化的东西就不要费脑筋。

——鲁迅

预防为主，治疗为辅(An ounce of prevention is worth a pound of cure）磨刀不误砍柴工

程序员经常错误地认为高效率编码就是快速堆砌代码，很多程序员不经思索和设计就直接编写代码。很不幸地是，这种鲁莽做法将会编写出草率的、脆弱的代码，然后不得不时常进行调试和打补丁，甚至将这些代码整段替换掉。因此，编码效率不仅包括编码的时间，而且还有调试代码的时间。