12 位运算

前面介绍的各种运算都是以字节作为最基本位进行的。但在很多系统程序中常要求在位(bit)一级进行运算或处理。C语言提供了位运算的功能，这使得C语言也能像汇编语言一样用来编写系统程序。

1.1 位运算符C语言提供了六种位运算符：

&	按位与
|	按位或
^	按位异或
~	取反
<<	左移
>>	右移

1.1.1按位与运算

按位与运算符"&"是双目运算符。其功能是参与运算的两数各对应的二进位相与。只有对应的两个二进位均为1时，结果位才为1，否则为0。参与运算的数以补码方式出现。例如：9&5可写算式如下：

```
  00001001        (9的二进制补码)
 &00000101        (5的二进制补码)
  00000001        (1的二进制补码)
```

可见9&5=1。

按位与运算通常用来对某些位清0或保留某些位。例如把a 的高八位清 0 ，保留低八位，可作a&255运算(255 的二进制数为0000000011111111)。

【例12.1】

```
main(){
    int a=9,b=5,c;
    c=a&b;
```

```
    printf("a=%d\nb=%d\nc=%d\n",a,b,c);
}
```

1.1.1按位或运算

按位或运算符“|”是双目运算符。其功能是参与运算的两数各对应的二进位相或。只要对应的二个二进位有一个为 1 时，结果位就为 1。参与运算的两个数均以补码出现。

例如：9|5 可写算式如下：

```
    00001001
   |00000101
    00001101        (十进制为 13)可见 9|5=13
```

【例 12.2】

```
main(){
    int a=9,b=5,c;
    c=a|b;
    printf("a=%d\nb=%d\nc=%d\n",a,b,c);
}
```

1.1.1按位异或运算

按位异或运算符“^”是双目运算符。其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为 1。参与运算数仍以补码出现，例如 9^5 可写成算式如下：

```
    00001001
   ^00000101
    00001100        (十进制为 12)
```

【例 12.3】

```
main(){
    int a=9;
    a=a^5;
    printf("a=%d\n",a);
}
```

1.1.1求反运算

求反运算符～为单目运算符，具有右结合性。其功能是对参与运算的数的各二进位按位

求反。
例如～9 的运算为：
~(0000000000001001)结果为：1111111111110110

1.1.1左移运算

左移运算符“<<”是双目运算符。其功能把“<< ”左边的运算数的各二进位全部左移若干位，由“<<”右边的数指定移动的位数，高位丢弃，低位补 0。
例如：
a<<4
指把 a 的各二进位向左移动 4 位。如 a=00000011(十进制 3)，左移 4 位后为 00110000(十进制 48)。

1.1.1右移运算

右移运算符“>>”是双目运算符。其功能是把“>> ”左边的运算数的各二进位全部右移若干位，“>>”右边的数指定移动的位数。
例如：
设 a=15，
a>>2
表示把 000001111 右移为 00000011(十进制 3)。
应该说明的是，对于有符号数，在右移时，符号位将随同移动。当为正数时，最高位补0，而为负数时，符号位为 1，最高位是补 0 或是补 1 取决于编译系统的规定。Turbo C 和很多系统规定为补 1。
【例 12.4】

```
main(){
    unsigned a,b;
    printf("input a number:   ");
    scanf("%d",&a);
    b=a>>5;
    b=b&15;
    printf("a=%d\tb=%d\n",a,b);
}
```


请再看一例!
【例 12.5】

```
main(){
    char a='a',b='b';
    int p,c,d;
    p=a;
    p=(p<<8)|b;
    d=p&0xff;
```

```
    c=(p&0xff00)>>8;
    printf("a=%d\nb=%d\nc=%d\nd=%d\n",a,b,c,d);
}
```

1.1 位域（位段）

有些信息在存储时，并不需要占用一个完整的字节，而只需占几个或一个二进制位。例如在存放一个开关量时，只有 0 和 1 两种状态，用一位二进位即可。为了节省存储空间，并使处理简便，C 语言又提供了一种数据结构，称为"位域"或"位段"。

所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。这样就可以把几个不同的对象用一个字节的二进制位域来表示。

1. 位域的定义和位域变量的说明

位域定义与结构定义相仿，其形式为：

```
struct 位域结构名
    { 位域列表 };
```

其中位域列表的形式为：

```
类型说明符 位域名：位域长度
```

例如：

```
struct bs
 {
   int a:8;
   int b:2;
   int c:6;
 };
```

位域变量的说明与结构变量说明的方式相同。 可采用先定义后说明，同时定义说明或者直接说明这三种方式。

例如：

```
struct bs
 {
   int a:8;
   int b:2;
   int c:6;
 }data;
```

说明 data 为 bs 变量，共占两个字节。其中位域 a 占 8 位，位域 b 占 2 位，位域 c 占 6 位。

对于位域的定义尚有以下几点说明：

1) 一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。

例如：

```
struct bs
 {
```

```
    unsigned a:4
    unsigned :0          /*空域*/
    unsigned b:4         /*从下一单元开始存放*/
    unsigned c:4
  }
```

在这个位域定义中，a 占第一字节的 4 位，后 4 位填 0 表示不使用，b 从第二字节开始，占用 4 位，c 占用 4 位。

2) 由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过 8 位二进位。

3) 位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：

```
struct k
 {
  int a:1
  int  :2            /*该 2 位不能使用*/
  int b:3
  int c:2
 };
```

从以上分析可以看出，位域在本质上就是一种结构类型，不过其成员是按二进位分配的。

2. 位域的使用

位域的使用和结构成员的使用相同，其一般形式为：

位域变量名·位域名

位域允许用各种格式输出。

【例 12.6】

```
main(){
    struct bs
    {
      unsigned a:1;
      unsigned b:3;
      unsigned c:4;
    } bit,*pbit;
    bit.a=1;
    bit.b=7;
    bit.c=15;
    printf("%d,%d,%d\n",bit.a,bit.b,bit.c);
    pbit=&bit;
    pbit->a=0;
    pbit->b&=3;
    pbit->c|=1;
    printf("%d,%d,%d\n",pbit->a,pbit->b,pbit->c);
}
```

上例程序中定义了位域结构 bs，三个位域为 a,b,c。说明了 bs 类型的变量 bit 和指向

bs 类型的指针变量 pbit。这表示位域也是可以使用指针的。程序的 9、10、11 三行分别给三个位域赋值(应注意赋值不能超过该位域的允许范围)。程序第 12 行以整型量格式输出三个域的内容。第 13 行把位域变量 bit 的地址送给指针变量 pbit。第 14 行用指针方式给位域 a 重新赋值，赋为 0。第 15 行使用了复合的位运算符"&="，该行相当于：

```
pbit->b=pbit->b&3
```

位域 b 中原有值为 7，与 3 作按位与运算的结果为 3(111&011=011,十进制值为 3)。同样，程序第 16 行中使用了复合位运算符"|="，相当于：

```
pbit->c=pbit->c|1
```

其结果为 15。程序第 17 行用指针方式输出了这三个域的值。

1.1 本章小结

1. 位运算是C语言的一种特殊运算功能， 它是以二进制位为单位进行运算的。位运算符只有逻辑运算和移位运算两类。位运算符可以与赋值符一起组成复合赋值符。如&=,|=,^=,>>=,<<=等。
2. 利用位运算可以完成汇编语言的某些功能，如置位，位清零，移位等。还可进行数据的压缩存储和并行运算。
3. 位域在本质上也是结构类型，不过它的成员按二进制位分配内存。其定义、说明及使用的方式都与结构相同。
4. 位域提供了一种手段，使得可在高级语言中实现数据的压缩，节省了存储空间，同时也提高了程序的效率。