位操作

实验介绍

- 本实验总共包括有关位操作的15个编程题
- 你们的目标就是实现这15个编程题
- 所有要实现的代码都在bits.c里面

实验步骤

- tar xvf datalab-handout.tar解压代码，包含如下文件
- Bits.c:唯一需要修改的文件
- Btest.c :该文件的作用是对我们实现的bits.c功能的正确性行评估，
- README:关于btest的一些说明。
- Dlc：语法检查

实验步骤

- 打开bits.c文件，该文件包含了一个结构体team，我们需要首先补充该team中的数据
- Bits.c中包含需要实现的15个函数，文件中规定了实现每个函数需要的逻辑和算术操作符（规定数量）。
  - 只能使用规定的操作符! ~ & ^ | + << >>
  - 不能使用循环或者条件语句
  - 不能使用超过8位的常数（ff)

- 完成后用./dlc bits.c检查bits.c的语法是否正确，就是是否按照要求使用规定数量的操作符

实验步骤

- 如果语法检查无误，那么使用make btest，生成btest可执行文件，该文件检查bits.c中实现的函数功能是否与要求的一致，具体用法如下./btest
- 如果还需要修改bits.c那么需要make clean；make btest重新生成btest文件
- ./btest -f isPositive单独测试某一个函数

位操作

- 见table 1
- rating ---困难等级
- max ops ---最多可使用的操作符

- 左移 就是: 丢弃最高位,0补最低位 （算术和逻辑都是这样）
- 再说右移,明白了左移的道理,那么右移就比较好理解了.

- 右移的概念和左移相反,就是往右边挪动若干位,运算符是>>.

- 右移对符号位的处理和左移不同,对于有符号整数来说,比如int类型,右移会保持符号位不变,例如:

- int i = 0x80000000;
- i = i >> 1; //i的值不会变成0x40000000,而会变成0xc0000000

- 就是说,符号位向右移动后,正数的话补0,负数补1,也就是汇编语言中的算术右移.同样当移动的位数超过类型的长度时,会取余数,然后移动余数个位.

- 负数10100110 >>5(假设字长为8位)，则得到的是 11111101

- 总之,在C中,左移是逻辑/算术左移(两者完全相同),右移是算术右移,会保持符号位不变 . 实际应用中可以根据情况用左/右移做快速的乘 /除运算,这样会比循环效率高很多.