Csapp

Lab1实验报告

10224507041姚凯文

2023/3/19

文件：Datalab——bits.c

题目：

1.bitXor

本题题目要求利用按位取反（~）和按位与（&）实现按位异或（^）运算，即输入的两个数同一位上相同则为0，不同则输出1。

设输入的两个数为a，b，可以通过~a^b在a为1的每个位置将b对应的位置设置为0，可通过~（~a&~b）实现将a为1的位置上，b的对应位置的值设为1，即按位运算OR，（即为课本上p39习题2.13中的bis，bic操作），由此，可以通过x^y=(x&~y)|(~x&y)来实现按位异或。

int bitXor(int x, int y) {

int result=~((~(~x&y))&(~(x&~y)));

return result;

}

2.tmin

本题要求返回TMIN由二进制补码结构可知TMIN=1000……000（共有31个0）故仅需将1左移31位即可。

int tmin(void) {

return 1<<31;

}

3.isTmax

本题要求返回x是否与Tmax相等的布尔值，Tmax为0x7fffffff，故若x+1与x按位取反结果相等，，则只有可能x=Tmax或x=-1，再用!(~x)排除掉-1即可。

int isTmax(int x) {

int a=x+1;

int b=a^~x;

return !(b|!(~x));

}

4.allOddBits

本题要求获得所有奇数位是否为1的结构作为布尔量输出，由于存在偶数位干扰，直接在原数中检查奇数位难度较大，故使用掩码将该数的奇数位取出，又由于题目对使用常数的范围有明确限制，故只能通过移位运算来获得相应掩码，即0xAAAAAAAA,再利用掩码取出奇数位上的数，使用异或判断是否全为1即可。

int allOddBits(int x) {

int m=0xAA+(0xAA<<8)+(0xAA<<16)+(0xAA<<24);

return !((x&m)^m);

}

5.negate

一个数的相反数等于它的反码加一。

int negate(int x) {

return (~x)+1;

}

6. isAsciiDigit

题目要求确定输入是否在0x30到0x39之间，利用上一题计算出这两个数的相反数后相加，再取符号位判断大小即可。

int isAsciiDigit(int x) {

int a=(~0x30)+1;

int b=(~0x39)+1;

int digit\_a=!((x+a)>>31);

int digit\_b=((x+b+(~1)+1)>>31);

return digit\_a&digit\_b;

}

7. conditional

本题要求利用位运算符实现三目运算符，首先要将x转换为布尔量，若x的布尔量为零则返回y，否则返回z，因为只能返回一个值，所以必须将返回值与布尔量结合起来，利用某个方法返回，观察到布尔量为1或0，且当！x为1时！！x为0，故可以试图通过将！x与！！x（！x与！！x取值为1或0）分别与0xffffffff（-1）相加的方式得到0x00000000和0xffffffff，从而利用&运算将不想返回的数变为0从而屏蔽掉，最终得到正确结果。

int conditional(int x, int y, int z) {

int a=(~((!!x)+(~(1)+1)))&y;

int b=(~((!x)+(~(1)+1)))&z;

return a+b;

}

8.isLessOrEqual

题目要求比较x是否大于等于y，可利用第五题negate的函数方法，将x转化为x的相反数，再将-x与y相加，对结果判断符号位（任何大于等于0的数的符号位均为零，故符号位必须为0）即可。

int isLessOrEqual(int x, int y) {

int a=~(x)+1;

int sum=y+a;

int mostsig=sum>>31;

return !mostsig;

}

9. logicalNeg

题目要求输出一个数的布尔值，因为只有0的布尔值是0，所以我们可以找一些仅有0才有的性质，比如只有0的相反数还是0，除去Tmin以外其它数的相反数都与原数符号相反，为了使用这条性质，可以通过一种特殊的方法来取符号位，任意一个数进行31位算数右移后必定为0x00000000或0xffffffff， 对两个右移结果都再加上1得到1和0，不难发现除了0和Tmin，其余数x与-x右移结果都满足这种情况且必定一个为0，另一个为1.而x=0时两种情况右移加一均为1，x=Tmin时两种情况均为0，故对上述结果分析可知，对两个结果使用&进行按位与便可获得题目要求的结果。

int logicalNeg(int x) {

int negx=~(x)+1;

int digit1=(x>>31)+1;

int digit2=(negx>>31)+1;

return digit1&digit2;

}

10. howManyBits

题目要求返回一个数用补码表示最少需要几位，由于题目中数字均为补码表示，故最高表示位前面的所有数字应该都是全0(正数)或是全1(负数)。以298为例，298表示为二进制数为0000 0000 0000 0000 0000 0001 0010 1010。找到第一个与前面所有数不同的数的前一位（即符号位）即可，为了找到第一个不同的数字的前一位，首先要将这个数左移一位使位数错开，利用^运算可以得到一个数字这个数字开头n位均为0，第n+1位为1（即用最小长度表示数字时的符号位的位置）。这个1相对于最右边的数字的位置就是要返回的最小长度。仍以298为例：

298： 0000 0000 0000 0000 , 0000 0001 0010 1010

298<<1: 0000 0000 0000 0000 , 0000 0010 0101 0100

298^(298<<1): 0000 0000 0000 0000 , 0000 0011 0111 1110

接下来只需要找到298^(298<<1)中从左往右数第一个1的位置就可以了。

由于32位数字位数较多，考虑从中间开始逐渐缩小范围的查找方法，由于第n+1个数左边全为0，便于判断终止条件，故首先将得到的数字右移16位得到：

0000 0000 0000 0000 , 0000 0000 0000 0000 bit16=0

利用！！（flag<<16）若这个数每一位都是0，则返回0，说明原数字前16位均为0，否则返回1，说明前16位有1。再将结果左移4位（已知int只有32位故可以用5位二进制来表示0到31，最后运算时再将结果+1——因为数字至少有一位。这样就可以表示0到32的所有值了）若结果为1表示最少需要从左往右数的16位来表示，若结果为0则表示不需要从右往左的16为来表示，同理继续从中点查找，由于第一遍已经找过了从右往左前16位，故若前16位有1则将该数左移16去掉无关位继续查找，若前16位均为0则不变，说明第一个1在从左往右数第16位之后，故继续查找。仍以298为例：

假设使用bitn来表示单次查找在发现在当次查找的数中从右往左数已知需要多少位来表示原数

-> 0000 0011 0111 1110 bit8=8

-> 0000 0011 bit4=0

-> 0011 bit2=0

-> 11 bit1=1

-> 1

最后再将1+bit+bit2+bit4+bit8+bit16的结果返回即可。

int howManyBits(int x) {

int flag=x^(x<<1);

int bit16,bit8,bit4,bit2,bit;

bit16=!!(flag>>16)<<4;

flag=flag>>bit16;

bit8=!!(flag>>8)<<3;

flag=flag>>bit8;

bit4=!!(flag>>4)<<2;

flag=flag>>bit4;

bit2=!!(flag>>2)<<1;

flag=flag>>bit2;

bit=!!(flag>>1);

return 1+bit+bit2+bit4+bit8+bit16;

}

11. floatScale2

题目要求输出2\*f，f乘以二的倍数可以通过按一定规则左移f得到。不考虑无穷，IEEE浮点数格式一共分为三种形式，非规格化，NAN以及规格化。若输入为NAN则直接返回即可。需要注意的是，对于非规格化数的左移，IEEE标准规定了一个特殊的处理方式。具体而言，由于非规格化数尾数没有隐式的“1”，因此在左移时需要将小数点前的第一位作为隐含1，然后向左移动一位，这样可以保证左移后的数值不变。所以非规格化数直接左移1位但要保留符号位。规格化的话尾数不变，只移动指数位并保持符号位不变即可。

unsigned floatScale2(unsigned uf) {

unsigned exp=(uf>>23)&0xff;

unsigned most\_sig=uf>>31<< 31;

int t=uf;

if (exp==0xff)

return uf;

else if (exp==0)

{

t=uf<<1;

t=t|most\_sig;

}

else

{

t=t+(1<<23);

t=t|most\_sig;

}

return t;

}

12. floatFloat2Int

题目要求返回一个浮点数的整数部分，和上一题思路类似，可以通过讨论输入的浮点数x的符号位和阶数来分类实现浮点数到整数的转换，当x为非规格化的浮点数或其阶数小于127时（bias=2^(8-1)-1=127,真正的指数E=exp-bias=exp-127）该数为0到1之间的数，取整返回0，当其阶数大于127而小于127+31（E=exp-bias=exp-127，而int型数据最多只有32位表示2的31次方，该数的二进制表示为1后面跟了31个0，如果浮点数的指数大于等于 31，那么无论尾数如何，它都无法表示这个数。因此如果浮点数的阶数字段大于等于 158，那么它就无法转换成整数，应该返回 0x80000000 表示错误。若可以正常表示，由于IEEE浮点数表示类似科学计数法，尾数均大于1小于2，故返回2的指数次方2^E并根据符号位加上符号即可

int floatFloat2Int(unsigned uf) {

unsigned exp=(uf>>23)&0xff;

unsigned most\_sig=(uf>>31)&0x1;

int t=1<<(exp-127);

if(exp >=127+31)

return 0x80000000;

if(exp<127)

{

return 0;

}

if(most\_sig)

return -t;

return t;

}。

运行结果：

