//1

/\*

\* bitXor - x^y using only ~ and &

\* Example: bitXor(4, 5) = 1

\* Legal ops: ~ &

\* Max ops: 14

\* Rating: 1

\*/

int bitXor(int x, int y) {

return (~(x&y))&(~(~x&~y));

}

**先使用非和与实现或，再用或和与实现异或。**

/\*

\* tmin - return minimum two's complement integer

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 4

\* Rating: 1

\*/

int tmin(void) {

return 0x1<<31;

}

**返回2进制int的最小值，所以我们只需要把1 11左移$3$1位，就能得到min。**

//2

/\*

\* isTmax - returns 1 if x is the maximum, two's complement number,

\* and 0 otherwise

\* Legal ops: ! ~ & ^ | +

\* Max ops: 10

\* Rating: 1

\*/

int isTmax(int x) {

int i = x+1;// 这里我们取极端情况，当x就是最大值的时候，加一就变为最小值，看上面的图得出

x = x + i;// 最大值+最小值得到全一，0xFFFF FFFF

x = ~x;// 取反得全零。0x0000 0000

i = !i;// 给i取反，上面结论说了，一个非零的布尔值都是true，也就是1

x = x + i;// 所以当我们给x+i的时候，要么加0要么就加1，它就会影响到下面的结果

return !x;

}

**题目中规定不能使用移位符号，明确32位整形最大值为0x7fffffff即可**

/\*

\* allOddBits - return 1 if all odd-numbered bits in word set to 1

\* where bits are numbered from 0 (least significant) to 31 (most significant)

\* Examples allOddBits(0xFFFFFFFD) = 0, allOddBits(0xAAAAAAAA) = 1

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 12

\* Rating: 2

\*/

int allOddBits(int x) {

int y=0xaa,z;

y=y+(y<<8);

y=y+(y<<16);

z=~y;

return !~((x&y)+z);

}

**首先要获取奇数位的信息,则有mask:0xAAAAAAAA.(十六进制A的二进制表示为1010).当奇数位全为1时,(x>>1 | x)则表示0xffffffff,反之,则不成立.**

/\*

\* negate - return -x

\* Example: negate(1) = -1.

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 5

\* Rating: 2

\*/

int negate(int x) {

return ~x+1;

}

**对补码求负，各位取反再加一即可。**

//3

/\*

\* isAsciiDigit - return 1 if 0x30 <= x <= 0x39 (ASCII codes for characters '0' to '9')

\* Example: isAsciiDigit(0x35) = 1.

\* isAsciiDigit(0x3a) = 0.

\* isAsciiDigit(0x05) = 0.

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 15

\* Rating: 3

\*/

int isAsciiDigit(int x) {

return !((0x39 + (~x+1))>>31|(x+(~0x30+1))>>31);

}

**首先要获取奇数位的信息,则有mask:0xAAAAAAAA.(十六进制A的二进制表示为1010).当奇数位全为1时,(x>>1 | x)则表示0xffffffff,反之,则不成立.**

/\*

\* conditional - same as x ? y : z

\* Example: conditional(2,4,5) = 4

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 16

\* Rating: 3

\*/

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

x = !!x;

x = ~x + 1;

return (x & y) | (~x & z);

}

/\*

\* isLessOrEqual - if x <= y then return 1, else return 0

\* Example: isLessOrEqual(4,5) = 1.

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 24

\* Rating: 3

\*/

int isLessOrEqual(int x, int y) {

int \_x = ~x + 1;

int ans = \_x + y;

int sign = (ans >> 31) & 1;

int x\_sign = (x >> 31) & 1;

int y\_sign = (y >> 31) & 1;

return ((x\_sign ^ y\_sign) & ((!y\_sign) & x\_sign)) | (!(x\_sign ^ y\_sign) & (!sign));

}

**首先要获取奇数位的信息,则有mask:0xAAAAAAAA.(十六进制A的二进制表示为1010).当奇数位全为1时,(x>>1 | x)则表示0xffffffff,反之,则不成立.**

//4

/\*

\* logicalNeg - implement the ! operator, using all of

\* the legal operators except !

\* Examples: logicalNeg(3) = 0, logicalNeg(0) = 1

\* Legal ops: ~ & ^ | + << >>

\* Max ops: 12

\* Rating: 4

\*/

int logicalNeg(int x) {

return ((x|(~x+1))>>31)+1;

}

**一个数的相反数等于自身的只有零，利用0 00的逆元仍为0 00，所以0 00与它的逆元的符号位都是0，而其他的数和它的逆元的符号位至少有一个是1 11。然后把符号位右移31 3131位，如果符号位为1 11则为0 x f f f f f f f f 0xffffffff0xffffffff，符号位为0 00则为0 x 00000000 0x000000000x00000000，然后加1 11就是最终结果。**

/\* howManyBits - return the minimum number of bits required to represent x in

\* two's complement

\* Examples: howManyBits(12) = 5

\* howManyBits(298) = 10

\* howManyBits(-5) = 4

\* howManyBits(0) = 1

\* howManyBits(-1) = 1

\* howManyBits(0x80000000) = 32

\* Legal ops: ! ~ & ^ | + << >>

\* Max ops: 90

\* Rating: 4

\*/

int howManyBits(int x) {

int b16,b8,b4,b2,b1,b0;

int sign=x>>31;

x = (sign&~x)|(~sign&x);//如果x为正则不变，否则按位取反（这样好找最高位为1的，原来是最高位为0的，这样也将符号位去掉了）

// 不断缩小范围

b16 = !!(x>>16)<<4;//高十六位是否有1

x = x>>b16;//如果有（至少需要16位），则将原数右移16位

b8 = !!(x>>8)<<3;//剩余位高8位是否有1

x = x>>b8;//如果有（至少需要16+8=24位），则右移8位

b4 = !!(x>>4)<<2;//同理

x = x>>b4;

b2 = !!(x>>2)<<1;

x = x>>b2;

b1 = !!(x>>1);

x = x>>b1;

b0 = x;

return b16+b8+b4+b2+b1+b0+1;//+1表示加上符号位

}

**忽略符号位,把最高位的1右侧所有位都置为1,再查询1的个数,最中的结果再加1.**

//float

/\*

\* floatScale2 - Return bit-level equivalent of expression 2\*f for

\* floating point argument f.

\* Both the argument and result are passed as unsigned int's, but

\* they are to be interpreted as the bit-level representation of

\* single-precision floating point values.

\* When argument is NaN, return argument

\* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

\* Max ops: 30

\* Rating: 4

\*/

unsigned floatScale2(unsigned uf) {

int exp = (uf & 0x7f800000) >> 23;

int sign = uf & (1 << 31);

if(exp == 255) return uf;

if(exp == 0) return (uf<<1) | sign;

exp++;

if(exp == 255) return 0x7f800000 | sign;

else return (uf & 0x807fffff) | (exp << 23);

}

/\*

\* floatFloat2Int - Return bit-level equivalent of expression (int) f

\* for floating point argument f.

\* Argument is passed as unsigned int, but

\* it is to be interpreted as the bit-level representation of a

\* single-precision floating point value.

\* Anything out of range (including NaN and infinity) should return

\* 0x80000000u.

\* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while

\* Max ops: 30

\* Rating: 4

\*/

int floatFloat2Int(unsigned uf) {

unsigned INF = 0x80000000;

//提取符号位

int s = (uf >>31) & 0x1;

//提取阶码

int E = uf >> 23 & 0xff;

//提取阶数

int e = E -127;

if (uf == 0) return 0;

//因为输入是规格化浮点数,转化为整数时第23位需要为1

uf &= 0x00ffffff;

uf |= 0x00800000;

//浮点数中0~22位的数字逻辑上位小数,当看作整数时相当于乘以了2^23

//阶码为255或阶数大于等于32时,视为溢出,输出INF.因为int为32bit,超出即溢出,且考虑1bit符号位

if ((uf & 0x7f80000) == 0x7f80000 || e>= 32) return INF;

if (e<0) return 0;//若为小数,返回零

//无符号数的移位运算都是逻辑移位

if (e <= 23) uf >>= 23 - e;//因为浮点数尾数宽度为23bit,位数小于等于23,尾数位右移.这是一种舍入方式

else uf <<= e-23;//位数大于23,尾数左移

//当符号位为负数,uf要取值为它的相反数

if(s) uf = ~uf + 1;

return uf;

}

**首先要先提取出float的exp和frac的值，注意这里的frac要在前面添上1，因为对于int\_32，其表示范围为-232 . . . 2 32 − 1，所以如果exp−127>31，则说明会发生溢出，根据题目要求返回infinityinfinity，如果e x p − 127 < 0 ∣ exp\_frac ==0返回0。**

**否则对根据exp对frac进行左移或者右移，再判断frac的符号是否与原来的float相同，相同返回frac，如果发生了正溢出（原来为正的，变成负的了）则返回infinityinfinity，如果发生负溢出，则返回相反数，两种溢出的处理方式不同，主要是因为，发生正溢出，说明int有一位被当成符号位了，也就是说这个int的真值部分就需要32位，所以超过了表示的范围。**

/\*

\* floatPower2 - Return bit-level equivalent of the expression 2.0^x

\* (2.0 raised to the power x) for any 32-bit integer x.

\*

\* The unsigned value that is returned should have the identical bit

\* representation as the single-precision floating-point number 2.0^x.

\* If the result is too small to be represented as a denorm, return

\* 0. If too large, return +INF.

\*

\* Legal ops: Any integer/unsigned operations incl. ||, &&. Also if, while

\* Max ops: 30

\* Rating: 4

\*/

unsigned floatPower2(int x) {

int exp = x + 127;

int INF = 0xff<<23;

if(exp <= 0) return 0;

if(exp >= 255) return INF;

return exp << 23;

}

**因为V =（− 1） s i g n \*M ∗2 E ，要得到2.0x我们只需要改变E的值，也就是改变e x p的值，而M = 0，如果exp≥255，返回infinity(0x7f800000)，如果exp≤0返回0，否则返回exp<<23**