A. 第一题是计算某个有符号整形的绝对值。有符号数使用的是补码编码，补码编码的特点是

$$abs(x)=\begin{cases} x & (x\geq 0)\\ \sim x+1 & (x<0)\end{cases}$$

为此，先判断x的符号位是 0 还是 1。如果是 0，则返回x。如果是 1，则返回$\sim x+1$。这里使用了判断逻辑，但是题目不允许 if 分支语句的出现，这里我们可以用"掩码"+"加法"的方式绕开这一阻碍。

$$abs(x)=(x\gg 31)\&(\sim x+1)+(\sim x\gg 31)\&(x)$$

B. 第二题要求使用或和非两个运算符实现与运算，实际上是狄摩根定理的运用。

$$\sim(x\&y) = (\sim x)\,|\,(\sim y)$$

$$x\&y = \sim((\sim x)|(\sim y))$$

C. 第三题要求实现能指定某个给定连续区间内的位为 1 的生成函数，如 bitMask(5,3)=0x38。

很容易想到，这个掩码可以由两段区间取交集得到。以 8 位码为例，我们有

$$bitMask(5,3)=00111000b=11111000b\&00111111b=bitMask(31,lowbit)\&(\sim bitMask(31,highbit+1))$$

这样我们就把任务转变为了计算$bitMask(31,lowbit)$和$bitMask(31,highbit+1)$的过程，这可以用左移操作实现，$bitMask(31,x)=(\sim 0)\ll x$。由此，我们有了计算本题的算式。

$$ones=(\sim 0)$$

$$bitMask(highbit,lowbit)=(ones\ll lowbit)\&(\sim(ones\ll highbit\ll 1))$$

这里没有使用$(ones\ll(highbit+1))$而是使用了$ones\ll highbit\ll 1$，原因是移位运算符只会取$highbit$的低 5 位，即当$highbit=31$时，编译器会认为$(ones\ll(highbit+1))$是左移 0 位，这就导致了错误的结果。

D. 第四题要求使用非和或两个运算符实现异或操作。异或操作可以拆解为

$$bitXor(x,y)=(\sim(x\&y))\,\&\,(x|y)$$

参考类似 B 的狄摩根定理，可以把或运算改成仅含非和与的运算。

$$(\sim(x\&y))\,\&\,(x|y)=(\,\sim(x\&y)\,)\,\&\,(\sim((\sim x)\&(\sim y)))$$

E. 第五题要求实现条件表达式(三目运算符)。本题允许使用!运算符，它可以直接告诉我们整数的布尔性(是否为 0)，因此使用$!x$来生成全 0 或全 1 的掩码是非常明智的。

$$mask=(!x)\ll 31\gg 31$$

基于掩码和加法就能实现类似条件分支的功能。

$$conditional(x,y,z)=(mask\&z)+(\sim mask\&y)$$

F. 第六题要求我们计算出一个偶数位全为 1 的掩码。因为只能使用 8 位以内的常数，所以我们需要移位再取或。

$$even8 = 0x55$$

$$even16 = even8\,|\,(even8\ll 8)$$

$$even32=even16\,|\,(even16\ll 16)$$

G. 实现一个判等功能。因为允许使用!运算符，所以可以取异或再调用!运算符。

$$isEqual(x,y)=!\,(x\hat{}y)$$

为什么不用减法？减法不但需要更多的运算符，而且会导致溢出。

H. 实现一个小于号运算符。这里先做减法，再判断减法结果的符号位。

$$isLess(x,y)=\begin{cases}1 & (sign(x-y)=1)\\ 0 & (sign(x-y)=0)\end{cases}$$

和 G 中提到的一样，要考虑减法的溢出问题。为此，我们考虑 x、y 的符号情况。

$$isLess(x,y)=\begin{cases}1 & (sign(x)=sign(y),sign(x-y)=1)\\ 1 & (sign(x)=1,sign(y)=0)\\ 0 & else\end{cases}$$

所以，使用移位先提取 x、y 的符号掩码，然后使用判断上式的前两个条件。

$$negx = x\gg 31$$

$$negy = y\gg 31$$

$$isLess(x,y)=(\,(negx\&(\sim negy))\,|\,((\sim(negx\hat{}negy))\&\Big(\Big(x+\big((\sim y)+1\big)\Big)\gg 31\Big))\,)\,\&\,1$$

I. 判负函数，即提取符号位

$$isNegative(x)=x\gg 31\&1$$

J. 判 0 函数，即起到!的作用。0 的补码表示有一个特殊的，其他整数没有的性质：即$sign(x)=0,sign(x-1)=1$

从这一点下手，写出表达式。

$$isZero(x) = \left(\left(\left((\sim x)\&(x-1)\right)\right) \gg 31\right)\&1 = \left(\left(\left((\sim x)\&(x+(\sim 0))\right)\right) \gg 31\right)\&1$$

$$isNonZero = \sim\left(\left(\left((\sim x)\&(x+(\sim 0))\right)\right) \gg 31\right)\&1$$

K. 判断某个数是否是 2 的幂，这个问题需要技巧。2 的幂有哪些？包括 1,2,4,8,16…。它们有一个特点，第一是正数，第二，它们的二进制表示仅有 1 个 1。由此，我们使用$x\&(x-1)$去掉x最右侧的一个 1，再判断$x\&(x-1)$是不是全零即可。

特别的，0 和 0x80000000 同样让上面的条件满足，因此特别考虑 x 是否为 0 及是否为负。

$$not\_zero \;=\; !(!x)$$
$$positive \;=\; (\sim x) \gg 31$$
$$isPower2(x) = (not\_zero)\,\&\,(positive)\,\&\,(!\,(x\,\&\,(x+(\sim 0))))$$

L. 提取最右侧的 1。一样是技巧，注意到$\sim x+1$置最右的 1 为 1，其右置 0，其左取反。所以

$$LeastBitPos(x) = x\,\&\,(\sim x+1)$$

M. 实现逻辑反，同 J。

N. 逆转字节，依次提取 4 个字节，并做适当移位即可。

```
int reverseBytes(int x) {
    // 依次取4个字节，移位相加
    // 注意，b4是符号填充，所以最后要额外操作。
    int mask = 0xff;
    int b1 = (x&mask)<<24;
    int b2 = (x&(mask<<8))<<8;
    int b3 = (x&(mask<<16))>>8;
    int b4 = ((x&(mask<<24))>>24) & mask;
    return b1+b2+b3+b4;
}
```

O. 一个加号实现三数之和。实际上相当于分解异或和进位，再做加法。

```
static int sum(int x, int y) {
  return x+y;
}
int sum3(int x, int y, int z) {
  int word1 = 0;
  int word2 = 0;
  /**************************************************************
     Fill in code below that computes values for word1 and word2
     without using any '+' operations
  ***************************************************************/
  word1 = x ^ y ^ z;
  word2 = ((x & y) | ((x ^ y) & z)) << 1;
  /**************************************************************
     Don't change anything below here
  ***************************************************************/
  return sum(word1,word2);
}
```