Datalab 实验报告

姓名：郭丹琪 学号：2018202067

1. int bitXor(int x, int y) {
   return (~ (x & y)) & (~(~x & ~y));
   }

   解题思路：因为在 xy 相等时需要结果为 0，xy 不等时结果为 1.，且只能用~和&，所以尝试了 x&y , x&~y , ~x&y , ~x&~y。发现当 x&y 和~x&~y 组合起来时，xy 相同和 xy 相异时的结果是相反的，所以只要再考虑怎样表示出对应的结果。因为只能用~和&，所以在（x&y）和（~x&~y）前尝试分别加~，再把两个部分用&处理看结果。最后尝试得(~ (x & y)) & (~(~x & ~y))可以得到结果。

2. int evenBits(void) {
   int result=85;
   result+=(result<<8);
   result+=(result<<16);
   return result;
   }

   解题思路：在范围内能设置的最大的偶数位为 1，奇数位为 0 的数是 85。此时从右数 8 位都已设置完了。所以往左移 8 位后再相加就设置完了 16 位，再左移 16 位并相加即可。

3. int fitsShort(int x) {
   return !(((x<<16)>>16)^x);
   }

   解题思路：要能将二进制位 32 位的数当成 16 位的数，则将 x 来回移位后值不发生变化即可。将 x 左移 16 位后再右移 16 位，如果在移位后能满足与原 x 相等，则 x 就可以当成 16 位的数。

4. int isTmax(int x) {
   return (!((x+1)^(~x)))&(!!(x+1));
   }

   解题思路：要判断 x 是否为最大值，即判断 x 是否为 0x7FFFFFFF。发现 0x7FFFFFFF+1 后等于~0x7FFFFFFF，所以利用这一点，即判断!((x+1)^(~x))。但是在测试的时候发现，如果 x=-1，这个式子也成立。所以另外设置了对-1 的判断，即判断 x+1，若 x=-1，则 x+1=0。取两次！是为了让其余数的结果为 1，以免最后返回值为 0、1 以外的数。

5. int fitsBits(int x, int n) {
   int mask=~n+1;
   return !(((x<<mask)>>mask)^x);
   }

   解题思路：本题其实与第 3 题想法一样，所以直接改变第 3 题中的值。但是因为不能用减号，所以在表示移位 32-n 时用的是移位-n，表示成~n+1。

6. int upperBits(int n) {

```
int x=1;
x=x<<31;
x=x>>(n+31);
return x+(!n);
}
```

解题思路：要将最前面的 n 个数设置为 1，最先想到的是用移位，先将最前面的第一个数设为 1，接下来再向右移 n-1 位，即表示为移 n+31 位，在移位的时候会自动把左边的位补为 1。但是在测试的时候发现，如果 n=0，移 n+31 位就会变成把所有位都设为 1。所以考虑在结果返回的时候对 n=0 的情况进行处理，发现若所有位都被设为 1，则再+1 就会变成 0，而！0 恰好等于 1。并且若 n 非 0，则 ！ n=0，对结果没有影响。所以在最后返回时加上了！ n。

7. int allOddBits(int x) {

```
int n=170;
n+=(n<<8);
n+=(n<<16);
return !((x&n)^n);
}
```

解题思路：只要判断奇数位是否都为 1，偶数位无所谓。所以先设置 n 为一个奇数位都为 1，偶数位都为 0 的数，然后将 x&n。x 的偶数位无论是什么，在&n 后都会被设成 0，而 x 的奇数位如果是 1，在&n 后还是 1，如果是 0，在&n 后就还是 0。所以若 x 的奇数位都为 1，在 x&n 后，就会变成一个奇数位都为 1，偶数位都为 0 的数，即 n。所以再判断是否与 n 相等即可。

8. int byteSwap(int x, int n, int m) {

```
int mask1=255;
int m8=(m<<3),n8=(n<<3);
int mask3=~((mask1<<n8)^(mask1<<m8));
int x1=((x>>m8)<<n8)&(mask1<<n8);
int x2=((x>>n8)<<m8)&(mask1<<m8);
return (x&mask3)+(x1^x2);
}
```

解题思路：将 mask1 设置为一个字节中每一位都为 1 的掩码，mask1<<n8、mask1<<m8 分别为第 n,m 字节中每一位都为 1 的掩码。再设置一个第 m 个字节和第 n 个字节中每一位都为 0，其余都为 1 的掩码 mask3，目的是将第 m 个字节、第 n 个字节和其余字节的内容分别保留在不同地方。因为不知道 m , n 的关系，所以先将第 m，n 个字节的内容分别移到第 0 个字节，然后再移到对方的字节，再使用相应的掩码，分别保留住 m 移到 n 的部分和 n 移到 m 的部分。再将 x&mask3，清空 m , n 上的内容。原本的思路是直接再加上原本保留的 m 移到 n 的部分和 n 移到 m 的部分即可，但是在测试的时候发现，如果 mn 相同，则会出现重复叠加的情况。所以在最后要再进行（x1^x2）的操作，来排除移位结果相同的影响。

9.
```
int absVal(int x) {
return (( (~x)+1 ) & (x>>31)) + ( x& (~ (x>>31)));
}
```
解题思路：负数取绝对值可以直接按位取反再+1，但是正数这样操作就相当于取负数，所以要在结果中把正负数的操作区分开。考虑到正负数的二进制的区别是左边第一位，负数二进制的左边第一位是 1，而正数是 0。所以将 x 右移 31 位后，负数变为 0XFFFFFFFF，正数变为 0。在按位取反+1 后，先与移位结果&，负数结果还是绝对值，而正数被设为 0。再加上 x&移位结果的按位取反，使正数的结果变成 0 加上正数本身，而负数为的负数的绝对值+0。

10.
```
int divpwr2(int x, int n) {
int bias=(1<<n)+(~0);
x=x+((x>>31)&bias);
return x>>n;
}
```
解题思路：因为需要向 0 取整，而直接用 x>>n 来表示 x/(2^n)的话，正数会符合向 0 取整，但是负数就会变成向另一个方向取整，所以要对负数的结果进行处理。了解到给负数加上偏差值(2^n)-1 后就可以向 0 取整。所以给 x 加上一个偏差值（2^n)-1，即用（1<<n）+(~0)来表示，但是正数不需要加偏差值，所以要把 bias&（x>>31），使正数加上的偏差值为 0。加完偏差值后再 x>>n 就可以求出 x/(2^n)向 0 取整的结果。

11.
```
int leastBitPos(int x) {
return ((~x)+1)&x;
}
```
解题思路：考虑到要将最后一个 1 留下并把其他位都设为 0，则想到用~x。在~x 后，x 中所有原本是 1 的地方都会变成 0。而再+1 后，x 的最后几位 0 在按位取反并+1 后会进位到 x 的最后一位 1 的位置，所以原本 x 中为 1 的地方现在只有最后一位还是 1。所以用（(~x）+1）&x 就可以在消去其余 1 的同时留下最后一位 1。

12.
```
int logicalNeg(int x) {
int m=(~x)+1;
return (~((x | m)>>31))&1;
}
```
解题思路：用 x 的相反数来与 x 比较来判断 x 是否为 0。若 x 不等于 0 或-2147483648，x 与(~x)+1 的左边第一位一定是相反的，所以(x | m)>>31 的结果一定为 0XFFFFFFFF。而虽然-2147483648 的(~x)+1 的结果为本身，但(x | m)>>31 得结果也是 0XFFFFFFFF。只有 0 的(x | m)>>31 结果是 0。但是要在结果是 0 的时候返回 1，所以要对(x | m)>>31 进行按位取反的操作，再&1。

13.
```
int bitMask(int highbit, int lowbit) {
int mask=~0;
return ((mask<<lowbit)^((mask<<highbit)<<1))&(mask<<lowbit);
}
```
解题思路：先设置一个所有位都是 1 的 mask，将 mask 左移 lowbit 位后，后 lowbit-1 位就

全是 0，从右数第 lowbit 位到第 32 位都是 1。同理，将 mask 左移 highbit 位，再左移 1 位后，从右数第 highbit+1 位到第 32 位都是 1，从右第 1 位到第 highbit 位都是 0。将两个结果按位异或后就会变成从右第 lowbit 位到第 highbit 位都是 1，其余为 0。但是要考虑 lowbit>highbit 的情况，所以将按位异或的结果再与 mask 左移 lowbit 位的结果进行&操作，就能把 lowbit>highbit 时结果中的 1 全消掉，而不改变 lowbit<highbit 时的结果。

14.
```
int isLess(int x, int y) {
int m=(~y)+1;
	return ((!((x^y)>>31))&(!!((x+m)>>31))) | ((!!(x>>31))&(!(y>>31)));
}
```
解题思路：分成两种情况来考虑。当 xy 符号相同时，即!((x^y)>>31)的结果为 1 时，x-y<0 时结果成立，用!!((x+m)>>31)来表示 x-y 的结果<0。当 xy 符号不同时，只有 x<0 且 y>0 时结果成立，所以用((!!(x>>31))&(!(y>>31))来表示该情况。这两种情况满足一种即成立，所以在两种判断情况间用 | 。

15.
```
int logicalShift(int x, int n) {
int mask1=(1<<31);
mask1=mask1>>n;
mask1=mask1+((!n)<<31);
mask1=mask1<<1;
mask1=~mask1;
return mask1&(x>>n);
}
```
解题思路：考虑到在向右移位的时候，算术移位负数会变成左边几位都是 1，而逻辑移位是 0。所以设置左边几位对应为 0 的掩码。设置掩码的操作与第 6 题相似，先设置掩码左边几位都为 1，再按位取反。要额外处理 n=0 的情况，所以得先 mask1=mask1+((!n)<<31)，将 n=0 位去掉后，才能将 mask1<<1。最后的 mask1 为左边前 n 位都是 0，后面都是 1 的掩码。再将 mask1 与 x 右移后的结果作&运算即可。

16.
```
int satMul2(int x) {
int n=x<<1;
return ( n & (~((n^x)>>31)) ) | (((n^x)>>31)&(~((x>>31)^(1<<31))));
}
```
解题思路：判断是否溢出可以通过判断计算后的符号。先设置 n 为 x*2，如果 n 与 x 符号不同，则是溢出。用(~((n^x)>>31))判断 nx 符号是否相同，如果相同则结果为 0XFFFFFFFF，不同则为 0，再与 n 进行&操作，保留没有溢出的结果，把溢出的结果设为 0。在溢出的情况下，x>0 时要把值设为 0x7FFFFFFF，x<0 时设为 0x80000000。所以用(~((x>>31)^(1<<31)))来设置掩码，把这 2 种情况下的值表示出来，再&((n^x)>>31)来使没溢出的情况下的掩码为 0。最后再通过计算结果和掩码间的 | 操作来设置溢出的结果。

17.
```
int subOK(int x, int y) {
int s=x+(~y)+1;
```

```
return !(((x^y)&(s^x))>>31);
}
```

解题思路：在 xy 符号相同时一定不会溢出，在 xy 符号不同时才可能溢出。在 xy 符号不同时，判断是否溢出要判断 x-y 的结果是否与 x 的符号相同。所以要判断是否溢出，要判断 xy 的符号和 x-y 与 x 的符号。判断符号用按位异或再右移 31 位来判断。在 x^y 为 0 时，s^x 为 0 或 1 都不会溢出。在 x^y 为 1 时，只有 s^x 为 0 才不会溢出，而想 x^y 和是 x^s 都为 1 时才会溢出。所以选择对(x^y)和(s^x)进行&操作，右移后再进行！操作。

18.
```
int bang(int x) {
int m=(~x)+1;
return (~((x | m)>>31))&1;
}
```

解题思路：本题与第 12 题相同。

19.
```
int bitParity(int x) {
x=x^(x>>16);
x=x^(x>>8);
x=x^(x>>4);
x=x^(x>>2);
x=x^(x>>1);
return x&1;
}
```

解题思路：要判断是否有奇数个 0，即判断是否有奇数个 1，所以该题与判断是否有偶数个 1 的题是一样的。将 x 移位后与自身亦或，即将 x 的前一半与后一半亦或，如果刚好两个对应的数都是 1，则两个 1 都会被消掉，即把 x 中的 1 成对消除。以此类推，消到最后如果最后一个数是 1，则 x 中有奇数个 1，也就是有奇数个 0。

20.
```
int isPower2(int x) {
return (!(x>>31))&(~(!x))&(!(x & (x + (~0))));
}
```

解题思路：首先要判断 x 为负数和 0 的情况，所以先在结果中设置与!(x>>31)进行&的操作来把负数的结果设为 0，再与(~(!x))进行&的操作来把 0 的结果设为 0。然后再判断正数的情况。经计算判断得若要 x 为 2 的幂，则 x 的二进制表示的位数中只能有一个为 1。而 x-1 的操作为减去 x 的最后一位 1，所以将 x & (x + (~0))，若 x 中只有一个 1，则结果为 0，否则为其他不同的数。所以可以用!(x & (x + (~0)))来判断正数是否为 2 的幂的情况。