datalab思路分析

第一题bitXor

由题意，要通过`~、&`实现`^`操作，可以进行如下分析：

对于两个字串`1100`和`1010`，只要通过`~、&`能够得到`0110`即可。对任意一位异或，情况只能是如上四种中的一种。

通过不断尝试得到以下可行方案：

```
int bitXor(int x, int y) {
      int x1=~x;
      int y1=~y;
      int t1=~(x&y);
      int t2=~(x1&y1);
      x=t1&t2;
      return x;
}
```

第二题thirdBits

初始化的整型变量限制在八位以内，则可以用`01001001`来进行初始化，然后向右移再`|`本身，用类似课上讲的判断"有多少个1"办法来得到结果：

```
int thirdBits(void) {
    int b=73;
    b=(b<<9)^b;
    b=(b<<18)^b;
  return b;
}
```

第三题fitsShort

先将x左移16位，再右移16位回来。

如果高位有数字，移动后会变成全1或全0。

那么再与原x异或，如果得到非0值就证明发生了改变。

故再进行取反操作即可。

```
int fitsShort(int x) {
  int y=x<<16;
  y=y>>16;
  return !(x^y);
}
```

第四题isTmax

Tmax+1再乘二可以得到0，考虑利用这个性质来写。

但因为有-1的干扰，所以得排除是-1的情况，正好（-1+1）==0，所以可以用！来排除。

以及乘二如果用左移1位会出问题，所以得用加法。

```
int isTmax(int x) {
  int y=x+1;
  return !(y+y+!y);
}
```

第五题fitBits

有点像fitShort，但是换一种方法，因为向左移再向右的话会多一个步骤。

于是选择向右移动，同时利用模32的特性，n+31。

右移之后，与全符号位异或，如果说移动后为全0或全1，那么就可以表示，否则肯定不可以。

```
int fitsBits(int x, int n) {
  int t=n+31;
  return !((x>>t)^(x>>31));
}
```

第六题upperbits

由题意，n有33个取值。

想到用1右移到最左边，然后再向右移动n-1位，利用它会取模的性质，则可以变成32+n来替换33+~n：

```
int upperBits(int n) {
    int y=(1<<31<<(!n))>>(n+31);
    return y;
}
```

第七题anyOddBit

思路类似于`thirdBits`

先初始化一个`10101010`的串，按照第三题的操作得到一个32位的10串`a`，然后取和再`!!`调到0或1即可：

```
int anyOddBit(int x) {
    int a=170;
    a=a<<8|a;
    a=a<<16|a;
    x=x&a;
    x=!!x;
        return x;
}
```

第八题byteSwap

首先得得到那两个字节的位置。利用255正好是八位整数来获得。

把需要换的byte调到后八位，然后用255&一下清除掉高位。

重点是利用课上讲过的a^(a^b)=b的性质来换位置。

```
int byteSwap(int x, int n, int m)
{
  int temp = 255;
  n = n << 3;
  m = m << 3;
  int change = ((x >> n) ^ (x >> m)) & temp;
  x = x ^ ((change << m) | (change << n));
  return x;
}
```

第九题absVal

本来有个绝佳的四步解法,利用异或的性质来写

```
int absVal(int x) {
  int y=x>>31;
  return (y^x)+(y&1);
}
```

后来重新思考第二步，考虑到~(x-1) =(~x)+1，于是简化为：

```
int absVal(int x) {
  int y=x>>31;
  return (y+x)^y;
}
```

第十题divpwr2

按照课本，增加一个”尾数“可以实现特定舍入。

由于要向0舍入，所以正数向下舍，负数向上。

故用`x>>31<<n`可以得到全0或前为1后为0的一个串，再与y取异或可以得到n位1或者0.加上再用位运算除即可。

```
int divpwr2(int x, int n) {
    int y=x>>31;
    int z=y^(y<<n);
    x=(x+z)>>n;
    return x;
}
```

第十一题floatNeg

将符号位取反可用^0x80000000来做，重点是非数的返回。

所以用if来判断，由于可以用任意初值，所以用0x7fffffff掉符号位之后来进行大小比较，判断是否返回本身。

```
unsigned float_neg(unsigned uf) {
  if((uf&0x7fffffff)>0x7f800000)
  {
    return uf;
  }
 return uf^0x80000000;
}
```

第十二题logicalNeg

本题只需要区分0和别的数，但是没有`!`实现`!`真的很困难。

想到`0xffffffff+1=0,0+1=1`，考虑通过这个来入手。

对于除了0以外的数，他的负值`|`上它本身都能使得最高位置为1。

再右移31位，此时只会得到0xffffffff或0。通过这个串加1可以实现区分。

```
int logicalNeg(int x) {
  int y=1+~x;
  int z=(y|x)>>31;
  return z+1;
}
```

第十三题bitMask

想法是得到一个低位全是1的串，去并一个高位全是1的串，来得到结果。

为了节省操作数，在得到低位全是1的串时用了点技巧。因为得到高位全是1的，会用到0xffffffff，又由于这个值正好是-1,所以考虑借助它，用$2^n - 1$得到的是n-1位全是1的串的性质来写。

```
int bitMask(int highbit, int lowbit) {
  int x=~0;
  int y=(2<<highbit)+x;
  int z=x<<lowbit;
  return y&z;
}
```

第十四题isGreater

首先用`^`得到符号位，大小比较肯定要作差，同号肯定不溢出，而异号可能溢出。

假设不同号，那么x符号位是0的话就大于，此时`~(t|y)`为0，没有影响。反之，则会得到全1的串。

如果同号，那么就相当于x-y+1，考虑到我们要的是大于，当x=y的时候刚好符号位还是1，所以发现好像1可以不加了，就直接用`!`即可。

```
int isGreater(int x, int y) {
  int t=((x^y)>>31);
  return !((x+~(t|y))>>31);
}
```

第十五题logiclaShift

通过左右移先拿到一个符号位的串，然后移动，得到一个前几位是1，后面是0的或者全是0的串。

再异或一下，就可以了。

```
int logicalShift(int x, int n) {
  int t=x>>31<<31;
  t=t>>n<<1;
  x=x>>n;
  return x^t;
}
```

但这个多了一步，于是想到一个新方法，利用

(0+1）^1==0,(1+1)^1=1。

刚好n位相加有个进位为1，能把前面全部变成1111，然后异或得到全是0.

```
int logicalShift(int x, int n) {
int m=1<<31>>n;
  return ((x>>n)+m)^m;
}
```

第十六题satMul2

溢出的情况，乘二之后符号位异或一定是1，通过这个可以判断出是否溢出。

同时根据溢出情况来看，最大正和最小负正好互相取反就是。

考虑用`^`实现取反。

然后得右移31位，发现刚好flag在溢出时为-1，那么根据模32的性质再用一下就可以了。

```
int satMul2(int x) {
  int a=x+x;
  int flag=(x^a)>>31;
  int ans=(a>>flag)^(flag<<31);
  return ans;
}
```

第十七题subOK

首先判断异号不，同号肯定不溢出。

然后判断异号情况，如果溢出了，那么符号位会和x相反。

通过右移符号位再加1就可判断了。

```
int subOK(int x, int y) {
  int a=(x^y);
  int b=(x+(~y)+1);
  int c=x^b;
  return 1+((a&c)>>31);
}
```

第十八题trueThreeFourths

首先x乘3/4就等于x-1/4，这样不会溢出，且精确度也有保障。

重点就是看向0舍入怎么实现。

我们发现其实乘3/4后的尾数实际上只取决于最后两位，类似于除以2时候的余数的概念。

同时，正数和负数的策略应该要有所区别。

然后再分析一下余数的情况，得到最终结果。

```
int trueThreeFourths(int x) {
  int t=~(x>>2);
  int ans=t+x;
  int mod=x&3;
  return !(mod&(t>>31))+ans;
}
```

第十九题isPower2

二进制表示时，如果x是2的n次方，那么x&（x-1）=0，于是利用这个性质来写。

但还需要排除两个，0和0x80000000。

有如下版本：

```
int isPower2(int x) {
  int t=~0;
  return !((x&(x+~(x>>31))|(!x)));
}
```

为了再省一个操作符，考虑两个串的共同性质，以及可以利用的-1。

于是考虑到减一后再右移30位，只有这两个值会仍然还有1，所以可以借此排除。

```
int isPower2(int x) {
  int t=~0;
  int y=x+t;
  int z=y>>30;
  return !((x&y)|z);
}
```

第二十题float_i2f

首先，要对0进行判断，不然无法处理。

然后，取一下符号位。为了方便处理exp，所以取绝对值转成正数。

然后用一个while循环找到最高位，得到exp。

最后判断向偶数舍入。

然后把各个段加起来。

理论上找最高位可以用二分法优化，但是肯定也优化不到十多步，就不想了。

```
unsigned float_i2f(int x) {。
  int signbit,highbit,exp,fracbits,flag;
  unsigned temp,result;
  if(!x)
  return x;
  signbit=x&0x80000000;
  if(signbit)
  x=1+~x;
  highbit=0;
  temp=x;
  while(!(temp&0x80000000))
  {
     temp<<=1;
     highbit++;
  }
  temp=temp<<1;
  exp=158-highbit;
  flag=0;
  if((temp&0x1ff)>0x100)
  flag=1;
  if((temp&0x3ff)==0x300)
  flag=1;
  result=(signbit|(exp<<23)|(temp>>9))+flag;
  return result;
}
```

第二十一题howManyBits

利用`x^(x<<1)`找到最高位的1，然后来找1在哪里，用一个二分法减少一点步骤。

```
int howManyBits(int x) {
    int temp = x ^ (x << 1);
    int bit_16,bit_8,bit_4,bit_2,bit_1;
    bit_16 = !!(temp >> 16) << 4;
    temp = temp >> bit_16;
    bit_8 = !!(temp >> 8) << 3;
    temp = temp >> bit_8;
    bit_4 = !!(temp >> 4) << 2;
    temp = temp >> bit_4;
    bit_2 = !!(temp >> 2) << 1;
    temp = temp >> bit_2;
    bit_1 = !!(temp >> 1);
    return 1 + bit_1 + bit_2 + bit_4 + bit_8 + bit_16;
}
```

第二十二题float_half

首先是特判0.

对于非边界条件，直接exp-1就可以。

对于边界条件判断。

如果exp是0，那么就要动frac。

如果exp是1，会从规格化到非规格化。

```
unsigned float_half(unsigned uf) {
unsigned sign = uf & 0x80000000;
unsigned abs = 0x7fffffff & uf;
unsigned half = abs_uf >> 1;
if(abs<= 0xffffff)//非规格化。
return sign + (half+ (uf & half & 1)) ;
if(abs>= 0x7f800000)
return uf;
else
return uf - 0x800000;
}
```