# 1、bitAnd函数

（1） 函数描述及操作要求 ① 函数功能：实现两个int型数据x和y的与运算，并返回结果，结果为int型数据

② 可用操作：~ |

③ 最大操作数：8

（2） 函数实现

① 函数实现代码

```
int bitAnd(int x, int y) {
  return ~((~x)|(~y));
}
```

② 函数实现思路

因为只有~ |两个可用操作，但是要实现的是&操作，因此可以考虑用德摩根定律将&运算变为~ |结合的表达式。

# 2、getByte函数

（1） 函数描述及操作要求

① 函数功能：int数据x从低位到高位的4个字节依次编号为0~3，要求取出该数据中的第n个字节，并返回结果，结果为int型数据

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：6

（2） 函数实现

① 函数实现代码

```
int getByte(int x, int n) {
  return (x>>(n<<3)&0xff);
}
```

② 函数实现思路

要获取数据x中的第n个字节，可以考虑先将要得到的字节通过移位操作移到第0个字节的位置，然后将其与0xff进行与运算，这样就得到了x的第n个字节，并且同时保证了高位的3个字节都为0。

##3、logicalShift函数 （1） 函数描述及操作要求

① 函数功能：将int型数据x逻辑右移n位，0 <= n <= 31，并返回结果，结果为int型数据

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：20

（2） 函数实现

① 函数实现代码

```
int logicalShift(int x, int n) {
  int mask=~(((1<<31)>>n)<<1);
  return ((x>>n)&mask);
}
```

② 函数实现思路

要实现将int型数据逻辑右移n位，可以先将其进行算数右移，然后将其与数据（最高的n位均为0，其他位均为1）进行与运算使得最高的n位清0，且其他位保持不变。

# 4、bitCount函数

（1） 函数描述及操作要求

① 函数功能：计算int型数据x的二进制串中1的个数，并返回结果，结果为int型

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：40

（2） 函数实现

① 函数实现代码

```
int bitCount(int x) {
 /*unsigned int tmp;
 tmp = (x &1090785345)
 +((x>>1)&010101010101)
 +((x>>2)&010101010101)
 +((x>>3)&010101010101)
 +((x>>4)&010101010101)
 +((x>>5)&010101010101);  //其中010101010101等为八进制
 return (tmp%63);*/

  x=(x&0x55555555)+((x>>1)&0x55555555);
  x=(x&0x33333333)+((x>>2)&0x33333333);
  x=(x+(x>>4))&0x0f0f0f0f;//相加进位?
  x=(x+(x>>8))&0x00ff00ff;
  x=(x+(x>>16))&0x0000ffff;
  return x;
}
```

② 函数实现思路

此函数的功能是返回数的二进制表示中1的个数，思路是使用二分法，先得到0x55555555对应32位的0101010101…，0x33333333对应32位的00110011…，0x0f0f0f0f对应32位的00001111…，0x00ff00ff对应32位的0000000011111111…，0x0000ffff对应32位的低位全为1。先计算x中每两位中1的个数，用num记录1的个数，以此类推，计算每4位、8位、16位中1的个数，最后整合的结果即为x中1的个数。每次错位就相当于一次相加。

##5、bang函数 （1） 函数描述及操作要求

① 函数功能：不使用!运算符实现!x，x为int型数据，并返回结果，结果为int型

② 可用操作：~ & ^ | + << >>

③ 最大操作数：12

（2） 函数实现

① 函数实现代码

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

② 函数实现思路

此函数的功能是实现逻辑取反操作，对于非零数取反输出0，零取反输出1.思路是先将数与其相反数相或，如果参数是零则这里得到的结果是全0，如果参数是非零数得到的结果只有两种情况，要么是全1，要么是0x80000000。再将得到的数向右移31位后按位取反，对于0得到的结果是全0，而对于非零数得到的结果是全1，最后返回与1相与的结果即可。

##6、tmin函数 （1） 函数描述及操作要求

① 函数功能：返回最小二进制补码整数，结果为int型数据

② 可用操作：! ~ & ^ | + << >>

③ 最大操作数：4

（2） 函数实现

① 函数实现代码

```
int tmin(void) {
  return (1<<31);
}
```

② 函数实现思路

最小二进制补码整数即符号位为1，其他位全为0。直接返回1左移31位的结果即可。

##7、fitsBits函数 （1） 函数描述及操作要求

① 函数功能：如果int型数据x可以表示为n位二进制补码整数（其中1 <= n <= 32）,则返回1，否则返回0。

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：15

（2） 函数实现

① 函数实现代码

```
int fitsBits(int x, int n) {
  int shift=n+(~0);//右移n-1位，正0负1
  return !(((x>>shift)+1)>>1);//-2^(n-1)右移n-1为-1
}
```

② 函数实现思路

此函数的功能是判断参数x能否用n位的二进制补码进行表示，即等同于判断x是否在-2^(n-1)与2^(n-1)-1之间。考虑将其分为两种情况，如果在范围内，将其向右移动n-1位后，加1再移位得到的结果应该是0，而如果不在范围内那么按照上述的操作得到的结果就应该是1，再将所得结果取反即可。

##8、divpwr2函数 （1） 函数描述及操作要求

① 函数功能：对于0 <= n <= 30，计算x / （2^n），向零舍入，返回计算结果，结果为int型。

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：15

（2） 函数实现

① 函数实现代码

```
int divpwr2(int x, int n) {
  int sign=0, var=0;
  sign=x>>31;
  var=(1<<n)+(~0);
  return (x+(sign&var))>>n;//注意负数右移，转换为补码再右移，再还原
}
```

② 函数实现思路

进行除法运算时，对于非负数来说，是默认向0取整的，而对于负数来说，则需要在移位之前加一个偏置量进行处理。

##9、negate函数 （1） 函数描述及操作要求

① 函数功能：返回int型数据x的相反数-x。

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：5

（2） 函数实现

① 函数实现代码

```
int negate(int x) {
  return (~x)+1;
}
```

② 函数实现思路

x的相反数即为x取反加1

##10、isPositive函数 （1） 函数描述及操作要求

① 函数功能：对于int型数据x，如果x > 0，返回1，否则返回0。

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：8

（2） 函数实现

① 函数实现代码

```
int isPositive(int x) {
  int t=(x>>31)&1;//正0负1零0
  return !((!x)|t);
}
```

② 函数实现思路

此函数的功能是判断x是否为正。思路是将数分为两种情况，一种是为0，另一种情况是不为0。当不为0时，将数右移31位，如果是正数这样会得到全0，如果是负数得到全1；当为0，函数返回0

##11、isLessOrEqual函数 （1） 函数描述及操作要求

① 函数功能：对于int型数据x和y，如果x <= y，则返回1，否则返回0。

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：24

（2） 函数实现

① 函数实现代码

```
int isLessOrEqual(int x, int y) {
  int signx=x>>31&1;
  int signy=y>>31&1;
  int signD=(!signy)&signx;//异号
  int signr=!(signx^signy);
  int result=x+(~y)+1;
  int signS=(result>>31)&signr;//同号
  return signD|signS|!(x^y);
}
```

② 函数实现思路

x <= y可以分两种情况来考虑：x和y同号，x和y异号。异号时要满足x <= y那么x一定为负数；同号时要满足x <= y，只能是两者相减为负数或者为0。

##12、ilog2函数 （1） 函数描述及操作要求

① 函数功能：返回log2x的值，返回结果为int型。

② 可用操作：！~ & ^ | + << >>

③ 最大操作数：90

（2） 函数实现

① 函数实现代码

```
int ilog2(int x) {
  int ans=0;
  ans=!!(x>>16)<<4;//从16位开始两边计算，有1向高位，无1向低位
  ans=ans+(!!(x>>(ans+8))<<3);
  ans=ans+(!!(x>>(ans+4))<<2);
  ans=ans+(!!(x>>(ans+2))<<1);
  ans=ans+(!!(x>>(ans+1)));
  return ans;
}
```

② 函数实现思路

该函数实际上就是要找到最接近一个数n使得2n最接近x，且满足2n <= x。因此可以先将int型数据x右移16位，并进行两次取反操作，如果得到的值为1，则说明x的高16位中存在至少一个1，那么ans应加上16；如果得到的值为0，则说明高16位中不存在1。然后再将x右移(ans+8)位，同样进行两次取反操作，如果得到的值为1，则说明在(ans+8)和(ans+15)这8位中至少有一个1，那么ans应加上8；如果得到的值为0，则说明这8位中不存在1。依次类推，继续将x右移(ans+4)，(ans+2)，(ans+1)位，并进行同样的操作即可得到最终结果。

##13、float_neg函数 （1） 函数描述及操作要求

① 函数功能：返回浮点参数f的表达式-f的位等效项。参数和结果都作为无符号int传递，但是它们将被解释为单精度浮点值的位级表示。 当参数为NaN时，返回参数。

② 可用操作：任何整数/无符号运算，包括 ||, &&. also if, while

③ 最大操作数：10

（2） 函数实现

① 函数实现代码

```
unsigned float_neg(unsigned uf) {
 unsigned t=uf&0x7fffffff;
 if(t>0x7f800000)return uf;//大于无穷大
 else return uf^(1<<31);//浮点数的相反数只需要改变符号位就可以
}
```

② 函数实现步骤

函数的参数可能为NaN(Not a Number)因此需要进行判断。如果参数uf的第23位到第30位全为1，而且uf的低23位不为0，这说明uf解释为单精度浮点值的位级表示时是一个NaN，所以应该直接返回，否则应直接将uf的最高位（符号位）取反即可得到-f。

##14、float_i2f函数 （1） 函数描述及操作要求

① 函数功能：返回表达式（浮点数）x的等价位。 结果以unsigned int形式返回，但是将其解释为单精度浮点值的位级表示。

② 可用操作：任何整数/无符号运算，包括 ||, &&. also if, while

③ 最大操作数：30

（2） 函数实现

① 函数实现代码 unsigned float_i2f(int x) { unsigned sign=0, ufrac=0, exp=0, absx=x, frac=0, shiftleft=0, pos=1<<31; unsigned result=0; if(x==0)return 0; else if(x<0) { absx=-x; sign=pos; } while((pos&absx)==0)//找到除符号位的第一个1 { absx<<=1; shiftleft+=1; } exp=127+31-shiftleft;//阶码 ufrac=absx&0xff;//舍去的尾数 frac=(absx>>8)&(~(pos>>8));//尾数 result=sign|(exp<<23)|frac; if(ufrac>0x80)result++; else if(ufrac==0x80) { if(frac&1)result++;//向偶舍入 } return result; } ② 函数实现思路

该函数主要考察int型数据到float型数据转化的过程，因此按照步骤一步一步判断执行即可。首先对x进行判断，如果等于0则可以直接返回，如果小于0则进入循环，找到x除符号位外的最高位的1。找到1后就可以得到左移的值，进而就可以得到阶码值，将被舍去的尾数ufrac，和将获得的尾数。之后再将符号位，阶码，尾数这三者相或就得到了result。当然最后还要对尾数ufrac进行判断，如果大于0x80，则result加1；如果等于0x80，则应向偶舍入。最后返回结果即可。

##15、float_twice函数 （1） 函数描述及操作要求

① 函数功能：返回浮点参数f的表达式2 * f的位等效项。 参数和结果都作为unsigned int传递，但是它们将被解释为的位级表示。单精度浮点值。当参数为NaN时，返回参数

② 可用操作：任何整数/无符号运算，包括 ||, &&. also if, while

③ 最大操作数：30

（2） 函数实现

① 函数实现代码

```
unsigned float_twice(unsigned uf) {
  unsigned exp=0, sign=0, frac=0, result=0, pos=1<<31;
  if(uf==0)return 0;
  else if(uf==pos)return uf;//正0负0返回自己
  sign=pos&uf;
  exp=(uf>>23)&0xff;
  frac=uf&((1<<23)-1);
  if(exp==0)//阶码为0,尾数x2,判断进位
  {
    frac<<=1;
    if(frac&(1<<23))
    {
        frac=frac&((1<<23)-1);
        exp++;
    }
  }
  else if(exp==0xff)return uf;//阶码全1,nan或无穷大
  else //其他阶码x2
  {
    exp++;
  }
  result=sign|(exp<<23)|frac;
  return result;
}
```

② 函数实现思路

首先对uf进行判断，如果uf为正0或负0，则直接返回uf，因为他们的两倍还是它本身。然后对阶码值进行判断，如果阶码值等于0xff，则说明uf为无穷大或者NaN，那么应直接返回uf；如果阶码值为0，说明uf为非规格化值，则应先将尾数乘2。然后判断尾数变化后的值是否存在进位，如果是则应该将阶码值加1，且要将尾数的值进行更新；如果阶码值不为0，则直接将其加1即可。最后将符号位，阶码，尾数三者相或就得到了2*uf，最后返回该值即可。