# 计算机系统第二次实验——DataLab

黄郭斌 计科1502 201507010206

1. 实验内容

1.int bitAnd(int x, int y) {

 return ~((~x)|(~y));

}

运用了德摩定律，~((~x)|(~y))= (~ (~x))& (~ (~y))=x&y

2. int getByte(int x, int n) {

return (x>>(n<<3))&255;

}

要从x中提取一个字节，而字节编号为0~3。一个字节为8位2进制。n<<3即为n\*8位。x>>(n<<3)即为x只保留下除去最后n\*8位剩下的部分。再&255则只保留剩下的最右一个字节。得出的结果便为编号指定要提取的那个字节。

3. int logicalShift(int x, int n) {

   int mask=~(((1<<31)>>n)<<1);   return mask&(x>>n);

}

~(((1<<31)>>n)<<1)即为2^32-n-1，也就是mask=(000…011111…1)2（n个0，31-n个1）。再mask&(x>>n)即为将x算术右移n位后并上mask这个前n位为0的掩码，使当x为负数时位移补1换为补0。

4. int bitCount(int x) {         int result;

        int tmp\_mask1=(0x55)|(0x55<<8);

        int mask1=(tmp\_mask1)|(tmp\_mask1<<16);

 int tmp\_mask2=(0x33)|(0x33<<8);

        int mask2=(tmp\_mask2)|(tmp\_mask2<<16);

int tmp\_mask3=(0x0f)|(0x0f<<8);

        int mask3=(tmp\_mask3)|(tmp\_mask3<<16);

int mask4=(0xff)|(0xff<<16);

 int mask5=(0xff)|(0xff<<8);

        result=(x&mask1)+((x>>1)&mask1);

        result=(result&mask2)+((result>>2)&mask2);

result=(result+(result>>4))&mask3;

result=(result+(result>>8))&mask4;

result=(result+(result>>16))&mask5;

return result;

}

本题采用二分法，先计算x每两位中1的个数，并用对应的两位来储存这个个数。然后计算每四位1的个数，再用对应的四位进行储存。依次类推，最后整合得到16位中1的个数，即为x中1的个数并输出。

5. int bang(int x) {

  return (~((x|(~x+1))>>31))&1; }

(x|(~x+1)即当x为0时，结果为（00…0)2(31个0)。其余情况最首位均为1。因此右移31位后再取反只有x=0时最后一位为1。再&1取最后一位。因此当x=0时得到1，其余情况得到0。

6. int tmin(void) {   return 1<<31; }

1<<31即(100…0)2(31个0)。其中1是符号位。即为负零，以表示最小的整数。

7. int fitsBits(int x, int n) {

int shiftNumber=~n+33;

  return !(x^((x<<shiftNumber)>>shiftNumber));

}

shiftNumber=~n+33即为-1-n+33=32-n，((x<<shiftNumber)>>shiftNumber)即先左移32-n位，再右移32-n位，即保留最后n位数。在与x按位异或并逻辑取反，若两者两同即x可被表示为一个n位整数，结果为!0，即返回为1。两者不同则不可表示为n位整数，结果为!(一个非零数)，返回0。

8. int divpwr2(int x, int n) {

int signx=x>>31;

int mask=(1<<n)+(~0);

  int bias=signx&mask;

 return (x+bias)>>n;

}

signx=x>>31为取x的符号位，mask=(1<<n)+(~0)即mask等于2n-1，bias=signx&mask即当x为正数，signx=0，bias为0，x为负数，signx=11111111111111，bias=mask。(x+bias)>>n当x为正数即x>>n，得到结果符合要求。当x为负数时，需要加上偏置量2n-1得到预期结果，因此加上signx，得到结果符合要求。

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

~x+1即-1-x+1=-x。得到结果正确。

10. int isPositive(int x) {   return !((x>>31)|(!x)); }

(x>>31)即取x的符号位，!x为逻辑取反，当x=0时!x=1，其他情况均为0. (x>>31)|(!x)即当x=0时得到1，x为负数得到1，x为正数得到0。再进行逻辑取反，即当x为正数时返回1，x为0或负数时返回0。

11. int isLessOrEqual(int x, int y) {

 int signx=x>>31;

int signy=y>>31;

int signSame=((x+(~y))>>31)|(!(signx^signy));

int signDiffer=signx&(!signy);

return signDiffer|signSame;

}

前两步取x、y的符号位，第三步中(x+(~y))>>31为当x-y-1为负数时取1，!(signx^signy)为两者相同时取1，两者相与即使x<=y时signSame为1，第四步即当x<0，x>0时signDiffer为1。最后一步即当x<0，x>0时必返回1，其余情况则x<=y返回1。效果即为如果x <= y，则返回1，否则返回0。

12. int ilog2(int x) {

        int bitsNumber=0;

        bitsNumber=(!!(x>>16))<<4;

        bitsNumber=bitsNumber+((!!(x>>(bitsNumber+8)))<<3);

bitsNumber=bitsNumber+((!!(x>>(bitsNumber+4)))<<2);

bitsNumber=bitsNumber+((!!(x>>(bitsNumber+2)))<<1);

bitsNumber=bitsNumber+(!!(x>>(bitsNumber+1)));

bitsNumber=bitsNumber+(!!bitsNumber)+(~0)+(!(1^x));

 return bitsNumber;

}

本题与bitcout的方法相似，也为二分法。

bitsNumber=(!!(x>>16))<<4即x向右移16位后若若大于0即得到(10000)2=16，否则得到0。这是判断最高位是否不为0，若不为0则包含2的16次方。即得到最高位的log数。

bitsNumber=bitsNumber+((!!(x>>(bitsNumber+8)))<<3);         bitsNumber=bitsNumber+((!!(x>>(bitsNumber+4)))<<2);         bitsNumber=bitsNumber+((!!(x>>(bitsNumber+2)))<<1);         bitsNumber=bitsNumber+(!!(x>>(bitsNumber+1)));

为判断从高到底各位的log情况，这个方法将每位分开求log并相加。         bitsNumber=bitsNumber+(!!bitsNumber)+(~0)+(!(1^x)); 这一句是当x为零时，还需要减去1才能得到正确的数值。

13. unsigned float\_neg(unsigned uf) {

unsigned result;

unsigned tmp;

  tmp=uf&(0x7fffffff);

 result=uf^0x80000000;

i(tmp>0x7f800000)

result=uf;

return result; }

  tmp=uf&(0x7fffffff) 为将uf的符号位改为0。     result=uf^0x80000000

即当uf不是NAN，通过加0x80000000来改变符号位。   if(tmp>0x7f800000)        result=uf;

0x7f800000即无穷大，如果tmp的值比无穷大还大，那就是NAN，则返回uf。

14. unsigned float\_i2f(int x) { //check

unsigned shiftLeft=0;

unsigned afterShift, tmp, flag;

unsigned absX=x;

unsigned sign=0;

//special case

if (x==0) return 0;

//if x < 0, sign = 1000...,abs\_x = -x

if (x<0)

{

sign=0x80000000;

absX=-x;

}

afterShift=absX;

//count shift\_left and after\_shift

while (1)

{

tmp=afterShift;

afterShift<<=1;

shiftLeft++;

if (tmp & 0x80000000) break;

}

if ((afterShift & 0x01ff)>0x0100)

flag=1;

else if ((afterShift & 0x03ff)==0x0300)

flag=1;

else

flag=0;

return sign + (afterShift>>9) + ((159-shiftLeft)<<23) + flag;

}

全0的时候，int和float均代表0

Sign记录符号位，通过

while (1)

{

tmp=afterShift;

afterShift<<=1;

shiftLeft++;

if (tmp & 0x80000000) break;

}

可以获得aftershift的前23位即为尾数位，aftershift通过转换即可获得阶码位。但是尾数位后面还要根据浮点数的舍入规则进行舍入,具体就是:当被舍去的大于0.1则向上进位,小于0.1则抹去,等于0.1时,如果前面(保留的最后一位尾数位)为1则向上进位,为0则抹去。具体则通过这段代码实现：

if ((afterShift & 0x01ff)>0x0100)

flag=1;

else if ((afterShift & 0x03ff)==0x0300)

flag=1;

else

flag=0;

15. unsigned float\_twice(unsigned uf) {

unsigned f=uf;

     if ((f & 0x7F800000) == 0){

                f = ((f & 0x007FFFFF)<<1) | (0x80000000 & f);

  }

     else if ((f & 0x7F800000) != 0x7F800000){

 f =f+0x00800000;

 }

return f;

}

第一个if语句判断非规格化的数。其中(f&0x007FFFFF)<<1作用为令符号位和阶码被屏蔽，令尾数左移。(0x80000000 & f)是将符号位恢复。

第二个else if语句判断即为规格化数。f=f+0x00800000即若是规格化数，对它的阶码加1。如果都不满足的话最后会返回uf原来的值。

二．实验结果

