Lab1-DataLab

姓名:黄昊
学号：21300240011

```
hh@hh-virtual-machine:~/VScode Projects/datalab-handout$ ./btest
Score   Rating  Errors  Function
 1       1       0       bitNor
 1       1       0       tmax
 1       1       0       isTmin
 1       1       0       minusOne
 2       2       0       absVal
 2       2       0       leastBitPos
 2       2       0       byteSwap
 3       3       0       logicalShift
 3       3       0       isLessOrEqual
 3       3       0       multFiveEighths
 4       4       0       bitCount
 4       4       0       greatestBitPos
 4       4       0       bang
 4       4       0       bitReverse
 4       4       0       mod3
 2       2       0       float_neg
 4       4       0       float_i2f
 4       4       0       float_twice
Total points: 49/49
```

```
hh@hh-virtual-machine:~/VScode Projects/datalab-handout$ ./dlc -e bits.c
dlc:bits.c:149:bitNor: 3 operators
dlc:bits.c:160:tmax: 2 operators
dlc:bits.c:172:isTmin: 9 operators
dlc:bits.c:183:minusOne: 1 operators
dlc:bits.c:197:absVal: 7 operators
dlc:bits.c:214:leastBitPos: 5 operators
dlc:bits.c:234:byteSwap: 20 operators
dlc:bits.c:249:logicalShift: 10 operators
dlc:bits.c:267:isLessOrEqual: 16 operators
dlc:bits.c:283:multFiveEighths: 10 operators
dlc:bits.c:329:bitCount: 34 operators
dlc:bits.c:349:greatestBitPos: 15 operators
dlc:bits.c:363:bang: 6 operators
dlc:bits.c:389:bitReverse: Warning: 41 operators exceeds max of 40
dlc:bits.c:414:mod3: 29 operators
dlc:bits.c:432:float_neg: 7 operators
dlc:bits.c:471:float_i2f: 16 operators
dlc:bits.c:489:float_twice: 9 operators
hh@hh-virtual-machine:~/VScode Projects/datalab-handout$
```

P1

```
int bitNor(int x, int y) {
   return ~x & ~y;
}
```

由摩根定理，~x & ~y = ~（x | y）, 即异或运算

P2

```
int tmax(void) {
   return ~(1 << 31);
```

```
}
```

二补码的最大值为 0111_1111_1111_1111_1111_1111_1111_1111,即为 1 的补码 0000_0000_0000_0000_0000_0000_0000_0001 左移 31 位后（1000_0000_0000_0000_0000_0000_0000_0000）再取反

P3

```
int isTmin(int x) {
   return !(x ^ ~(x + (~1 + 1))) & !!(x);
}
```

二补码的最小值为 1000_0000_0000_0000_0000_0000_0000_0000，这个数减 1 后再取反为它本身，即~(x + (~1 + 1))，一个数和自己取异或等于 0，0 取非等于 1，特殊情况 0 减 1 后取反也等于零，用!!（x）来排除 0 的情况

P4

```
int minusOne(void){
   return ~0;
}
```

-1 的补码为 1111_1111_1111_1111_1111_1111_1111_1111，即 0 取反

P5

```
int absVal(int x){
```

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

先把 x 右移 31 将 x 的位全部变为符号位的数值，如果 x 是正数，则 sign 为 0，x & ~sign = x，如果 x 为负数，则 sign 的位全是 1，~x + 1 & sign = ~x + 1，即 x 的相反数

P6

```
int leastBitPos(int x){
   return  (x & (x + ~1 + 1)) ^ x;
}
```

X - 1 后其最小位的 1 后面的 0 都会变成 1，再与 x 取与就可以将 x 后面的 0 都变成 1，得到的结果再与原来的 x 取异或，原来最后一位 1 前面的位不变，异或后都变成零，后面的 0 变成 1 异或后也都变成 0，只有最后一位 1 保持不变，即要返回的掩码

P7

```
int byteSwap(int x, int n, int m){
  int a = x >> (n << 3) & 0xff;
   int b = x >> (m << 3) & 0xff;
  x &= ~(0xff << (n << 3));
   x |= b << (n << 3);
```

```
    x &= ~(0xff << (m << 3));
    x |= a << (m << 3);
  return x;
}
```

a,b 记录 n，m 位置上的 byte，x &= ~(0xff<<(n<<3))把 x 在 n 位上的 byte 变成 0， x |= b<<(n<<3)再把 b 记录的 m 位上的 byte 移动到 x 的 n 位上；m 位上的交换同理

P8

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

先取 x 的符号位 sign，再将 x 加上符号位左移 31 位的结果，即 sign 的最高位为 x 的符号位；x 的符号位是 1 加 sign 后进位溢出最高位变成 0，如果是 0 则不变,、; 将 x 右移 n 位后再把原来的符号位变回来，即加上 sign << (31 + (~n) + 1)

P9

```
int isLessOrEqual(int x, int y) {
```

```
  int fx = x >> 31 & 1;
  int fy = y >> 31 & 1;
  int a = y + (~x + 1);
  int flag = (a >> 31) & 1;
  int b = !flag;
  int c = (!fy) & fx;
  return ((b & !(fx ^ fy)) | c);
}
```

x <= y 即 y-x>=0

flag 为 y-x 的符号位，如果 y-x 大于 0，则符号位为 0，即 b=1；c=1 时 fy=0，fx=1，即 y 大于 0，x 小于 0，这种情况也返回 1；!（fx^fy）如果符号相同则为 1，不同则为 0，所有两个符号相同的数相减最后结果为正数，即 b=1，则表示 y 大于等于 x，并且返回 1，否则返回 0；最后再和 c 取或则包含了所有情况

P10

```
int multFiveEighths(int x) {
  int l1 = x & 1;
  int l3 = (x & 4) >> 2;
  int of = ((l3 + l1) & 2) >> 1;
  x = (x >> 3) + (x >> 1) + of;
  return x;
```

}

直接计算乘 5 除 8 会造成溢出，所有改变运算顺序先除 8 再乘 5，乘 5 即 x 左移 2 位再加上 x，除 8 即 x 右移 3 位；先除 8 则 x 的后 3 位会变成余数，再乘 5 即将除 8 的结果左移 2 位，此时余数变为最后 1 位，再加上原来除于 8 后的结果，此时需要判断 x 右移 3 位和右移 3 位再左移 2 位的余数相加会不会产生溢出，即原来 x 的最后一位和倒数第 3 位相加会不会产生进位；11 和 13 分别取这两位，of 位两位相加的进位，最后三个结果相加即为答案。

P11

```
int bitCount(int x) {
  int temp, res, t1, t2, t3, t4, t5;
  temp = 0x55 | (0x55 << 8);
  t1 = temp | (temp << 16);
  temp = 0x33 | (0x33 << 8);
  t2 = temp | (temp << 16);
  temp = 0x0f | (0x0f << 8);
   t3 = temp | (temp << 16);
   t4 = 0xff | (0xff << 16);
   t5 = 0xff | (0xff << 8);
   res = (x & t1) + ((x >> 1) & t1);
   res = (res & t2) + ((res >> 2) & t2);
```

```
   res = (res & t3) + ((res >> 4) & t3);
   res = (res + (res >> 8)) & t4;
  res = (res + (res >> 16)) & t5;
  return res;
}
```

P12

```
int greatestBitPos(int x) {
  x |= x >> 1;
   x |= x >> 2;
   x |= x >> 4;
   x |= x >> 8;
   x |= x >> 16;
  x = ((~x >> 1) | (1 << 31)) & x;
   return x;
}
```

先将 x 位中为 1 的最高位后面的位都变成 1，再将 x 取反后右移 1 位的值（同时要让最高位为 1 来避免原来 x 最高位是 1 的情况取反再取与后最高位变成 0）和 x 取与产生错位使得只有最高位的 1 保留下来

P13

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

除了 0 以外，其他的数与他的相反数取或的结果其最高位符号位一定是 1，再右移 31 位后所有位都变成 1，再取反都变成 0 和 1 取与结果也是 0；如果 x 是 0 则取反后位全变成 1，再和 1 取与结果也是 1

P14

```
int bitReverse(int x){
  int temp, t1, t2, t3, t4, t5;
  temp = 0x55 | (0x55 << 8);
  t1 = temp | (temp << 16);
  temp = 0x33 | (0x33 << 8);
  t2 = temp | (temp << 16);
  temp = 0x0f | (0x0f << 8);
   t3 = temp | (temp << 16);
   t4 = 0xff | (0xff << 16);
   t5 = 0xff | (0xff << 8);
  x = ((x >> 1) & t1) | ((x & t1) << 1);
  x = ((x >> 2) & t2) | ((x & t2) << 2);
  x = ((x >> 4) & t3) | ((x & t3) << 4);
```

```
  x = ((x >> 8) & t4) | ((x & t4) << 8);
  x = ((x >> 16) & t5) | ((x & t5) << 16);
  return x;
}
```

T1 = 0101_0101_0101_0101_0101_0101_0101_0101

T2 = 0011_0011_0011_0011_0011_0011_0011_0011

T3 = 0000_1111_0000_1111_0000_1111_0000_1111

T4 = 0000_0000_1111_1111_0000_0000_1111_1111

T5 = 0000_0000_0000_0000_1111_1111_1111_1111

将 x 左移 1 位和 t1 取与的结果和 x 右移一位和 t1 取与的结果取或可以将 x 中每两位的位置交换；同理，t2 交换每 4 位中的前两位和后两位；t3 交换每 8 位中的前 4 位和后 4 位；t4 交换每 16 位中的前 8 位和后 8 位；t5 交换前 16 位和后 16 位

P15

```
int mod3(int x){
  int sign = x & (1 << 31);
  int y = x ^ sign;
  int t1 = (y >> 16) + (y & ((0xFF << 8) + 0xFF));
  int t2 = (t1 >> 8) + (t1 & 0xFF);
  int t3 = (t2 >> 4) + (t2 & 0xF);
  int t4 = (t3 >> 2) + (t3 & 3);
```

```
  int t5 = (t4 >> 2) + (t4 & 3);
  int t6 = (t5 >> 2) + (t5 & 3);
  int mod = (!(t6 ^ 3) + 7) & t6;
  return mod + (sign >> 30);
}
```

先把 x 的符号位变成 0 方便右移操作，并保留 x 的符号位 sign；因为 3 只比 2 大 1，所以对 x 每次除于 2 的幂次再加上其余数乘以 2 的幂次，直到余数为个位数就能得到结果；根据 int 有 32 位的特性，依次右移 16、8、4、2 位，即除于 2 的 16 次方、2 的 8 次方……，并依次加上它们的余数左移 16、8、4、2 位即乘以 2 的 16 次方、2 的 8 次方……；t5，t6 处理余数相加产生的进位，使得最后余数只有最后两位即个位数；mod = (!(t6 ^ 3) + 7) & t6，如果 t6 = 3，则!(t6 ^ 3) = 1 + 7 = 8，即 0100 和 t6 取与后等于 0，即 mod = 0，其他情况为 1 或 2，得到结果还是 t6，最后因为余数是 0，1 或 2，最多占两位，结果只要再加上 sign 右移 30 位即可。

P16

```
unsigned float_neg(unsigned uf) {
  if ((((uf & 0x7fffffff) >> 23) == 255) && (uf &
0x7fffff))
    return uf;
```

```
  else return uf + (1 << 31);
}
```

If 语句判断 uf 是否是为 NaN，(uf & 0x7fffffff) >> 23 把 uf 的符号位变为 0 并右移 23 位取其阶码位，如果等于 255 并且尾数非 0 时返回 uf 表示为 NaN；否则 uf 的符号位加 1，如果为正数则符号位变为 1，负数则符号位加 1 溢出变为 0

P17

```
unsigned float_i2f(int x) {
  int count = 0;
  unsigned temp, flag;
  unsigned a;
  unsigned abs = x;
  int sign = 0;
  if (x == 0) return 0;
  if (x < 0)
  {
  sign = 0x80000000;
  abs = -x;
  }
  a = abs;
  while (1)
```

```
  {
  temp = a;
  a <<= 1;
  count++;
  if (temp & 0x80000000) break;
  }
  if ((a & 0x01ff) > 0x0100)
    flag = 1;
  else if ((a & 0x03ff) == 0x0300)
    flag = 1;
  else
    flag = 0;
  return sign + (a >> 9) + ((159 - count) << 23) + flag;
}
```

除 0 以外最小的整数就是 1，所有可以用规格化表示，如果 x=0，则直接返回 0；如果 x 为负数则取它的相反数并把 sign 的最高位记为 1 表示负数;a 左移 1 位即除于 2 后的结果，a 记录左移的次数，重复此操作直到 a 的第一位不是 0，此时 32-a 就等于指数部分大小，其阶码值为 127+32-a；先判断 a 的后 9 位是否大于 0x0100 或者后 10 位是否大于 0x300 来判断是否需要进位，然后将 a 右移 9 位到浮点数的小数部分最后将符号位 sign 加上小数位 a>>9 加上阶码位 (159-a)<<23 加上进位 flag 得到结果

P18

```
unsigned float_twice(unsigned uf) {
  if ((uf & 0x7f800000) == 0) return ((uf & 0x007FFFFF) <<
1) | (uf & 0x80000000); // e = 00000000
  if ((uf & 0x7f800000) != 0x7f800000) return uf +
0x00800000;  //e != 11111111
  return uf; //e = 11111111
}
```

如果 uf 的阶码位都是 0，则为非规格化表示，将其小数左移一位就表示乘 2，并保留其符号位；如果 uf 阶码不全为 0，则位规格化表示，只要将阶码数加 1 即可；如果阶码全为 1 则为 NaN，返回 uf