计算机系统基础
实验报告

Lab 1

学生姓名： -

学　　号： -

日　　期： 2022.9.29

一：实验内容

同学们需要解出若干程序谜题，编写代码并通过正确性测试，最后提交代码和报告。希望同学们多加思考，在解题过程中能学到的远不止二进制本身，还能加深对位运算的理解，以及学到一些算法知识。

二：实验结果

./dlc -e bits.c 运行结果：

```
qzdc@LAPTOP-II37PGUL:/mnt/d/my_project/ICS/datalab-handout$  ./dlc -e bits.c
dlc:bits.c:150:bitNor: 3 operators
dlc:bits.c:161:tmax: 2 operators
dlc:bits.c:176:isTmin: 6 operators
dlc:bits.c:187:minusOne: 1 operators
dlc:bits.c:201:absVal: 5 operators
dlc:bits.c:215:leastBitPos: 3 operators
dlc:bits.c:239:byteSwap: 18 operators
dlc:bits.c:256:logicalShift: 7 operators
dlc:bits.c:271:isLessOrEqual: 12 operators
dlc:bits.c:285:multFiveEighths: 8 operators
dlc:bits.c:326:bitCount: 36 operators
dlc:bits.c:359:greatestBitPos: 39 operators
dlc:bits.c:371:bang: 6 operators
dlc:bits.c:415:bitReverse: 39 operators
dlc:bits.c:475:mod3: 78 operators
dlc:bits.c:497:float_neg: 7 operators
dlc:bits.c:536:float_i2f: 29 operators
dlc:bits.c:560:float_twice: 11 operators
qzdc@LAPTOP-II37PGUL:/mnt/d/my_project/ICS/datalab-handout$
```

./btest 运行结果：

```
qzdc@LAPTOP-II37PGUL:/mnt/d/my_project/ICS/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
qzdc@LAPTOP-II37PGUL:/mnt/d/my_project/ICS/datalab-handout$
```

三：实验内容

1. bitNor：根据公式可以得到：

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

2. tmax：将 1 右移 31 位可得到：

```c
int tmax(void)
{
  return ~(0x1 << 31);
}
```

3. isTmin：最大的数取反加一是自己，同时排除也有这个性质的 0：

```c
int isTmin(int x)
{
  int y = x;
  x = ((~x + 1) ^ y);
  y = (!y);
  return !(x + y);
}
```

4. minusOne：返回负一，即 0 的反码：

```c
int minusOne(void)
{
  return ~(0x0);
}
```

5. absVal：先判断正负，正数右移 31 得 0x0，负数得 0xffffffff，记为有，然后原数异或 y 在减去 y，得到绝对值：

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

6. leastBitpos：假设最右一个 1 的位置是 a，a 以右全为 0，取反后 a 为 0，以右全为 1. 再加一个 1，a 为 1，以右变为 0. 再与自己，将除了最右的 1，其它都置为 0：

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

7. byteSwap：将 m 和 n 都乘以 8，再将 0xff 左移 m 和 n 位，取得需要换的字节，然后通过右移左移交换 mm 和 nn 取得的字节，再把原来的数这两字节置为 0，再或上 mm 和 nn，完成交换：

```c
int byteSwap(int x, int n, int m)
{
  int nn, mm, xn, xm;
  int n_ = n << 3, m_ = m << 3;
  nn = 0x000000ff << n_;
  mm = 0x000000ff << m_;
  xn = x & nn;
  xm = x & mm;
  xn = (xn >> n_ << m_) & mm;
  xm = (xm >> m_ << n_) & nn;
  x = (x & (~nn)) & (~mm);
  x = (x | xn) | xm;
  return x;
}
```

8. logicalShift：先获得符号位，只取 1bit，然后左移 31 位，再右移 n-1 位，记为 tool，再异或 x，如果某位为 1 就取反，为 0 就不变：

```c
int logicalShift(int x, int n)
{
  int tool, sign;
  sign = (x >> 31) & 0x1;
  tool = (sign << 31) >> n << 1;
  x = x >> n;
  x = x ^ tool;
  return x;
}
```

9. isLessOrEqual：先比较符号位 sign，再看相减之后的符号位 cmp，然后返回 sign 或 cmp：

```c
int isLessOrEqual(int x, int y)
{
  int sign, cmp;
  sign = (x >> 31) & (!(y >> 31)) & 0x1;
  cmp = !(((y + (~x + 1)) >> 31) & 0x1);
  return sign | cmp;
}
```

10. mulFiveEighrhs：5/8 拆成 1/8 和 1/2，右移后相加，然后判断是否忽略了进位，进行修正：

```c
int multFiveEighths(int x)
{
  int rem = ((x >> 2) & 0x1) & ((x & 0x1));

  return (x >> 3) + (x >> 1) + rem;
}
```

11. bitCount：先 2 位 2 位加，再 4 位 4 位加，再 8 位 8 位加，再 16 位 16 位加，得到结果：

```
int bitCount(int x)
{
  int tool, t1, t2;
  tool = (0x55 << 8) | 0x55;
  tool = (tool << 16) | tool;
  t1 = x & tool;
  t2 = (x >> 1) & tool;
  x = t1 + t2;

  tool = (0x33 << 8) | 0x33;
  tool = (tool << 16) | tool;
  t1 = x & tool;
  t2 = (x >> 2) & tool;
  x = t1 + t2;

  tool = (0xf) | (0xf << 8);
  tool = tool | (tool << 16);
  t1 = x & tool;
  t2 = (x >> 4) & tool;
  x = t1 + t2;

  tool = (0xff) | (0xff << 16);
  t1 = x & tool;
  t2 = (x >> 8) & tool;
  x = t1 + t2;

  tool = (0xff) | (0xff << 8);
  t1 = x & tool;
  t2 = (x >> 16) & tool;
  x = t1 + t2;

  return x;
}
```

12. greatestBitPos：先判断再左边 16 位还是右边 16 位出现第一个 1，再判断是这 16 位的左 8 位还是后 8 位，然后是左 4 位还是右 4 位，然后是左 2 位还是右 2 位，然后判断具体哪一位。因为当 1 第一次再左边出现时，与上 1 不为零，再两次取反，就可以得到 1，来判断是左还是右。

```c
int greatestBitPos(int x)
{
  int tool, pos;
  tool = (~0) << 16;
  pos = 0;
  pos = !!(x & tool) << 4;

  tool = (~0) << (pos + 8);
  pos = pos + (!!(x & tool) << 3);

  tool = (~0) << (pos + 4);
  pos = pos + (!!(x & tool) << 2);

  tool = (~0) << (pos + 2);
  pos = pos + (!!(x & tool) << 1);

  tool = (~0) << (pos + 1);
  pos = pos + (!!(x & tool));

  pos = (1 << pos) & x;

  return pos;
}
```

13. bang：利用 0=-0=+0 的性质，当一个数及其取反加一后都是 0，那么就返回 1，其他情况返回 0：

```c
int bang(int x)
{
  return ((~(x | (~x + 1))) >> 31) & 0x1;
}
```

14. bitReverse：先 2 位 2 位交换，再 4 位 4 位交换，再 8 位 8 位，16 位 16 位，得到结果。用 4 位 4 位交换的工具变量 tool2 生成 2 位 2 位交换的工具变量 tool1，以优化至 40 步以内。

```c
int bitReverse(int x)
{
  int tool, x1, x2;
  int tool1, tool2;

  tool2 = (0x33 << 8) | 0x33;
  tool2 = (tool2 << 16) | tool2;

  tool1 = tool2 ^ (tool2 << 1);

  x1 = (x >> 1) & tool1;
  x2 = (x & tool1) << 1;
  x = x1 | x2;

  x1 = (x >> 2) & tool2;
  x2 = (x & tool2) << 2;
  x = x1 | x2;

  tool = (0xf << 8) | 0xf;
  tool = (tool << 16) | tool;
  x1 = (x >> 4) & tool;
  x2 = (x & tool) << 4;
  x = x1 | x2;

  tool = 0xff << 16 | 0xff;
  x1 = (x >> 8) & tool;
  x2 = (x & tool) << 8;
  x = x1 | x2;

  tool = 0xff << 8 | 0xff;
  x1 = (x >> 16) & tool;
  x2 = (x & tool) << 16;
  x = x1 | x2;
  return x;
}
```

15. mod3：先求绝对值，将 x 每 2 位分为一组，与 3 得到余数，求和更新 x，然后多次反复此操作，直至 x 只有 2 位。然后用数字逻辑的方法得到余数，再相或得到最后结果，最后还原结果的符号：

```c
int mod3(int x)
{
  int a0, a1, a2, a3, a4, a5, a6, a7, a8, a9,
      a10, a11, a12, a13, a14, a15, sign, x1, x2;
  sign = x >> 31;
  x = (x ^ sign) + (~sign + 1);

  a0 = x & 0x3;
  a1 = (x >> 2) & 0x3;
  a2 = (x >> 4) & 0x3;
  a3 = (x >> 6) & 0x3;
  a4 = (x >> 8) & 0x3;
  a5 = (x >> 10) & 0x3;
  a6 = (x >> 12) & 0x3;
  a7 = (x >> 14) & 0x3;
  a8 = (x >> 16) & 0x3;
  a9 = (x >> 18) & 0x3;
  a10 = (x >> 20) & 0x3;
  a11 = (x >> 22) & 0x3;
  a12 = (x >> 24) & 0x3;
  a13 = (x >> 26) & 0x3;
  a14 = (x >> 28) & 0x3;
  a15 = (x >> 30) & 0x3;

  x = a0 + a1 + a2 + a3 + a4 + a5 + a6 +
      a7 + a8 + a9 + a10 + a11 + a12 + a13 + a14 + a15;

  a0 = x & 0x3;
  a1 = (x >> 2) & 0x3;
  a2 = (x >> 4) & 0x3;
  /// a3=(x>>6)&0x3;
```

```
    /// a3=(x>>6)&0x3;

    x = a0 + a1 + a2;

    a0 = x & 0x3;
    a1 = (x >> 2) & 0x3;

    x = a0 + a1;

    a0 = x & 0x3;
    a1 = (x >> 2) & 0x3;

    x = a0 + a1;

    a1 = x >> 1;
    a2 = x & 0x1;
    x1 = a1 & (~a2);
    x2 = a2 & (~a1);

    x = (x1 << 1) | x2;
    x = (x ^ sign) + (~sign + 1);

    return x;
}
```

16. float_neg：先取 x 的非符号位，然后判断是否是非正常情况下的 NaN，如果是就直接返回，否则对其首字母取反并返回：

```
unsigned float_neg(unsigned uf)
{
  int a, b, x;
  x = uf & 0x7fffffff;
  a = !((x >> 23) ^ 0xff);
  b = x & 0x7fffff;
  if (a && b)
    return uf;
  return uf ^ (0x80000000);
}
```

17. float_i2f：先保存符号位，然后判断是否取反加一。如果 x=0，直接返回 0，如果 x=0x80000000，返回 0xcf000000。然后从左数第二位开始向右遍历，直到遇见第一个 1，保存其位置并记录数据

长度。然后由长度加上 bias 得到指数位的数据，再操作原始数据对齐数据位。然后判断是否需要进位，当小数部分大于 5 时或小数部分等于 5 且尾数为 1 则进位：

```
unsigned float_i2f(int x)
{
  int t = 0x40000000, len = 31, sign, pow, num, f, judge;
  if (!x)
    return 0;
  if (x == 0x80000000)
    return 0xcf000000;
  sign = x & 0x80000000;
  if (sign)
  {
    x = (~x) + 1;
  }
  while (!(x & t))
  {
    t = t >> 1;
    len = len - 1;
  }
  pow = (len + 126);
  if (len >= 24)
  {
    num = (x >> (len - 24)) & 0x7FFFFF;
    judge = (x << (31 - len)) & 0x7f;
    if ((judge > 0x40) || (judge == 0x40 && ((num & 0x1))))
    {
      num++;
    }
  }
  else
  {
    num = (x << (24 - len)) & 0x7FFFFF;
  }
  pow = pow << 23;
  f = (sign | pow) + num;
  return f;
}
```

18. float_twice：分别取得 uf 的符号位和指数位。然后如果 uf 为 0，直接返回 uf，如果 uf 的指数位全为 0 时，直接返回原数乘以二与符号位，如果 uf 指数位全为 1，则直接返回 uf。当正常情况下，

就返回原数的指数位+1 的数代表乘以 2：

```c
unsigned float_twice(unsigned uf)
{
  int pow, sign;
  sign = uf & 0x80000000;
  pow = (uf & 0x7f800000) >> 23;
  if (!(uf & 0x7fffffff))
    return uf;
  if (!pow)
    return sign | (uf << 1);
  if (pow == 0xff)
    return uf;
  return uf + (0x1 << 23);
}
```