# data lab

李心玥，PB22000197

# 实验目的

熟练运用C语言的位运算实现各种运算函数，深入理解数据在计算机中的表示方式。

---

# 实验过程

## 环境准备

执行`make btest`报错：`fatal error: bits/libc-header-start.h: No such file or directory`，于是执行如下命令安装缺少的库：

```
$ sudo apt-get install gcc-multilib
```

之后再执行`make btest`即可以正常编译。

## bitXor

由逻辑代数的知识，有

$$A \oplus B = \overline{A}B + A\overline{B}$$

而我们不能使用或运算，故可以使用德摩根定律将其展开：

$$X + Y = \overline{\overline{X + Y}} = \overline{\overline{X}\,\overline{Y}}$$

代入异或的运算式即可，最终代码为：

```
return ~(~(~x & y) & ~(x & ~y));
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f bitXor
Score   Rating  Errors  Function
 1      1       0       bitXor
Total points: 1/1
```

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./dlc -e bits.c
dlc:bits.c:146:bitXor: 8 operators
```

## tmin

最小的32位补码表示的整数应该是`x80000000`，即将1左移31次，代码如下：

```
return 1 << 31;
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f tmin
Score   Rating  Errors  Function
 1      1       0       tmin
Total points: 1/1
```

```
dlc:bits.c:155:tmin: 1 operators
```

## isTmax

最大的32位补码表示的整数为`x7fffffff`，这个数的特点是它取反后等于自身加1，但另一个数`xffffffff`也具有此特点，故需要排除（可以判断取反后是否为0），最终代码为：

```
return !((x + 1) ^ (~x)) & !!(~x);
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f isTmax
Score   Rating  Errors  Function
 1      1       0       isTmax
Total points: 1/1
```

```
dlc:bits.c:166:isTmax: 8 operators
```

## allOddBits

构造`xAAAAAAAA`作为`mask`：

```
int mask = 0xAA + (0xAA << 8);
mask = mask + (mask << 16);
```

然后用这个`mask`将`x`的奇数位都提取出来，最后与`mask`做异或运算，判断结果是否为0即可：

```
return !((x & mask) ^ mask);
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f allOddBits
Score   Rating  Errors  Function
 2      2       0       allOddBits
Total points: 2/2
```

```
dlc:bits.c:179:allOddBits: 7 operators
```

## negate

直接取反+1即可：

```
return ~x + 1;
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f negate
Score   Rating  Errors  Function
 2      2       0       negate
Total points: 2/2
```

```
dlc:bits.c:189:negate: 2 operators
```

## isAsciiDigit

分别与`x30`和`x39`相减，判断是否小于0（即符号位是否为1）即可：

```
int sign = (1 << 31);
int less_than_zero = (x + ~0x30 + 1) & sign;  // x - 0x30 < 0?
int greater_than_nine = (0x39 + ~x + 1) & sign;  // 0x39 - x < 0?
return !(less_than_zero | greater_than_nine);
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f isAsciiDigit
Score   Rating  Errors  Function
 3      3       0       isAsciiDigit
Total points: 3/3
```

```
dlc:bits.c:205:isAsciiDigit: 11 operators
```

## conditional

类似2选1MUX，结果应该形如

$$Ay + \overline{A}z$$

我们希望这个$A$在`x`不为0时为全1（即-1），在`x`为0时为0，可以如下设置：

```
int mask = ~(!!x) + 1
```

再将`mask`作为选择信号即可：

```
return (mask & y) + (~mask & z);
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f conditional
Score   Rating  Errors  Function
 3      3       0       conditional
Total points: 3/3
```

```
dlc:bits.c:216:conditional: 8 operators
```

## isLessOrEqual

若`x`与`y`同号，则直接相减判断符号位即可；若`x`与`y`异号，则直接相减可能溢出，此时应直接通过两者的符号来判断，当`y`非负时返回1，否则返回0。代码如下：

```
int oppo_sign = !(y >> 31);  // y >= 0?
int same_sign = !((y + ~x + 1) >> 31);  // y - x > 0?
int is_same_sign = !((y >> 31) ^ (x >> 31));  // if x and y has the same sign?
return (is_same_sign & same_sign) | (!is_same_sign & oppo_sign);
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f isLessOrEqual
Score   Rating  Errors  Function
 3      3       0       isLessOrEqual
Total points: 3/3
```

```
dlc:bits.c:229:isLessOrEqual: 15 operators
```

## logicalNeg

当`x`不为0时，其与其相反数的符号位必有一个为1，基于此可判断，代码如下：

```
return ~((x | (~x + 1)) >> 31) & 1;
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f logicalNeg
Score   Rating  Errors  Function
 4      4       0       logicalNeg
Total points: 4/4
```

```
dlc:bits.c:241:logicalNeg: 6 operators
```

## howManyBits

对于正数，问题可转化为求最高的一位1在哪里；对于负数，问题可转化为求最高的一位0在哪里；为统一问题，将负数取反，从而问题转化为求最高的一位1在哪里：

```
x = x ^ (x >> 31);
```

接下来采用二分法，先判断高16位中是否有1，若有则在高16位中寻找最高位的1；否则在低16位中继续。然后判断高8位、高4位、高2位、高1位：

```
int msb16 = !!(x >> 16) << 4;  // 高16位是否有1?
x = x >> msb16;  // 如果有，接下来考虑高16位
int msb8 = !!(x >> 8) << 3;  // 高8位是否有1?
x = x >> msb8;  // 如果有，接下来考虑高8位
```

```
int msb4 = !!(x >> 4) << 2;  // 高4位是否有1?
x = x >> msb4;  // 如果有，接下来考虑高4位
int msb2 = !!(x >> 2) << 1;  // 高2位是否有1?
x = x >> msb2;  // 如果有，接下来考虑高2位
int msb1 = !!(x >> 1);  // 高1位是否有1?
x = x >> msb1;  // 如果有，接下来考虑高1位
```

最后将全部加起来，再加上一位符号位：

```
return msb16 + msb8 + msb4 + msb2 + msb1 + x + 1;
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f howManyBits
Score   Rating  Errors  Function
 4      4       0       howManyBits
Total points: 4/4
```

```
dlc:bits.c:275:howManyBits: 32 operators
```

## floatScale2

先提取出符号位与指数域：

```
int sign = uf & (1 << 31);
int exp = (uf >> 23) & 0xff;
```

若指数域为0，则为非规约数，直接乘2并保留符号位即可：

```
if (exp == 0) {  // subnormal
  return (uf << 1) | sign;
}
```

若指数域为255，则为无穷或NAN，直接返回：

```
if (exp == 0xff) {  // inf or nan
  return uf;
}
```

其余情况下，将指数域加1：

```
exp = exp + 1;  // increment exp
```

若指数域加1后变成255，即溢出，则将fraction域置为全0，变成无穷数：

```
if (exp == 0xff) {  // overflow
  return (exp << 23) | sign;
```

```
}
```

否则，将原数的指数域换成递增后的指数域：

```
return (exp << 23) + (uf & ~(0xff << 23));
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f floatScale2
Score   Rating  Errors  Function
 4      4       0       floatScale2
Total points: 4/4
```

```
dlc:bits.c:306:floatScale2: 17 operators
```

## floatFloat2Int

先提取符号域、指数域（减去127后），以及1.F：

```
int sign = uf & (1 << 31);
int exp = ((uf >> 23) & 0xff) - 127;
int frac = (1 << 23) + (uf & (~(0xff << 23) + ~(1 << 31)));
```

若指数域超过31，则无法用32位整数表示，溢出：

```
if (exp > 31) {  // overflow
    return 1 << 31;
}
```

若指数域小于0，则结果的绝对值一定小于1，直接返回0：

```
if (exp < 0) {
    return 0;
}
```

其他情况下，对`frac`右移即可：

```
if (exp > 23) {
    ret = (frac >> (exp - 23));
} else {
    ret = (frac >> (23 - exp));
}
```

最后，若为负数，还要返回其相反数：

```
if (sign) {
        return -ret;
```

```
}
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f floatFloat2Int
Score   Rating  Errors  Function
 4      4       0       floatFloat2Int
Total points: 4/4
```

```
dlc:bits.c:342:floatFloat2Int: 22 operators
```

## floatPower2

先给`x`加上偏移127，若小于0则直接返回0，若大于等于255则直接返回+inf，其他情况正常将`x`左移23位：

```
x = x + 127;
if (x < 0) {  // to small
    return 0;
}
if (x >= 255) {  // too large
        return 0xff << 23;
}
return x << 23;
```

测试如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest -f floatPower2
Score   Rating  Errors  Function
 4      4       0       floatPower2
Total points: 4/4
```

```
dlc:bits.c:365:floatPower2: 5 operators
```

---

# 实验测试

分别执行 `./btest` 命令测试正确性，以及 `./dlc bits.c` 命令测试程序规范性如下：

```
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./btest
Score   Rating  Errors  Function
 1      1       0       bitXor
 1      1       0       tmin
 1      1       0       isTmax
 2      2       0       allOddBits
 2      2       0       negate
 3      3       0       isAsciiDigit
 3      3       0       conditional
 3      3       0       isLessOrEqual
 4      4       0       logicalNeg
 4      4       0       howManyBits
 4      4       0       floatScale2
 4      4       0       floatFloat2Int
 4      4       0       floatPower2
Total points: 36/36
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$ ./dlc bits.c
lrel7@OK:/mnt/f/ustc/大二下/计算机系统详解/datalab$
```

可见程序完全正确。

---

# 实验收获

通过本次实验，我锻炼了使用位运算的能力，并深化了对计算机中数据表示方式的理解。