计算机系统基础

实验报告

Lab-1-DataLab

姓名：马颢宸

学号：23307110426

一、实验目的

加深对整数和浮点数二进制表示的认识。

需要解出若干程序谜题，编写代码并通过正确性测试。

二、实验结果

1、终端中执行 ./dlc -e bits.c 后的截图

```
root@Ranxiaoxiao:~/icslab/lab1#/lab1-datalab-ranxiaoxiao-mmm/datalab# ./dlc -e bits.c
dlc:bits.c:149:tmax: 2 operators
dlc:bits.c:162:bitNor: 3 operators
dlc:bits.c:178:getByte: 3 operators
dlc:bits.c:191:roundUp: 6 operators
dlc:bits.c:207:absVal: 5 operators
dlc:bits.c:227:isLessOrEqual: 15 operators
dlc:bits.c:244:logicalShift: 6 operators
dlc:bits.c:263:swapOddandEven: 15 operators
dlc:bits.c:281:secondLowBit: 8 operators
dlc:bits.c:300:rotateNBits: 15 operators
dlc:bits.c:317:fractions: 8 operators
dlc:bits.c:343:overflowCalc: 29 operators
dlc:bits.c:365:mul3: 22 operators
dlc:bits.c:393:float_abs: 12 operators
dlc:bits.c:431:float_half: 28 operators
dlc:bits.c:479:float_i2f: 39 operators
dlc:bits.c:519:oddParity: 12 operators
dlc:bits.c:541:bitCount: 35 operators
```

2、终端中执行 ./btest 后的截图

```
root@Ranxiaoxiao:~/icslab/lab1#/lab1-datalab-ranxiaoxiao-mmm/datalab# make clean
rm -f *.o btest fshow ishow *~
root@Ranxiaoxiao:~/icslab/lab1#/lab1-datalab-ranxiaoxiao-mmm/datalab# make all
gcc -O0 -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
gcc -O0 -Wall -m32 -o fshow fshow.c
gcc -O0 -Wall -m32 -o ishow ishow.c
root@Ranxiaoxiao:~/icslab/lab1#/lab1-datalab-ranxiaoxiao-mmm/datalab# ./btest
Score   Rating  Errors  Function
 1      1       0       tmax
 2      2       0       bitNor
 2      2       0       bitCount
 3      3       0       absVal
 4      4       0       logicalShift
 4      4       0       isLessOrEqual
 2      2       0       getByte
 3      3       0       roundUp
 4      4       0       swapOddandEven
 4      4       0       secondLowBit
 5      5       0       rotateNBits
 5      5       0       fractions
 7      7       0       overflowCalc
 7      7       0       mul3
 3      3       0       float_abs
 7      7       0       float_i2f
 4      4       0       float_half
 2      2       0       oddParity
Total points: 69/69
```

三、实验内容

P1 tmax
在 32 位二进制中，最大正数表示为 0x7fffffff，即最高位符号位为 0，其余为 1。
通过将 1 算术左移 31 位得到 0x80000000，再取反得到 0x7fffffff。

P2 bitNor
由摩根定律可知，~(x|y) = (~x)&(~y)

P3 getByte
1 byte = 8 bits。由于算术右移以 bit 为单位，故将 n 算术左移 3 位，即 n*8。
再将 x 算术右移 n*8 位得到 m，m 的最后两位 byte 即为所求。
将 m 与 0x000000ff 做与运算，消去除最后两位 byte 外的其余值，即得所求返回值。

P4 roundUp
256 = 2^8。因为返回数必须是 256 的倍数，所以第 0~7 位必须为 0。
将 x 算术右移 8 位，以消去后 8 位的值，同时方便后续舍入操作。
判断后 8 位数字的值。如果为 0，则第 8 位加 0；如果>0，则第 8 位加 1。从而使返回值不小于 x。
将上述所得值算术左移 8 位，使其成为 256 的倍数，即为所求值。

P5 absVal
通过将 x 算术右移 31 位并与 1 做与运算取得 x 的符号位 p。
当 p=1 时，应返回 x 的相反数（即对 x 取反加 1）。x 与 0xffffffff 异或即对 x 取反。
当 p=0 时，返回 x。x 与 0x00000000 异或即为 x。

P6 isLessOrEqual
当 x、y 异号时，当且仅当 x 为负数 y 为正数返回 1。
当 x、y 同号时，当且仅当 y−x 大于等于 0 返回 1。由于 0 的符号位也为 0，故当 y−x 的符号位为 0 时，返回 1。

P7 logicalShift
X>>n 为算术右移，会导致第 31 位到第 31−n 位与符号位相同。而逻辑右移要求第 31 位到第 31−n 位为 0。
设置掩码。先将 1 左移 31 位得到 0x80000000，再将该值右移 n−1 位得到掩码。由于不能使用−运算符，故先右移 n 位再左移 1 位。

P8 swapOddandEven
设置奇数位与偶数位的掩码。由于 0xAA = 10101010，为前 8 位的奇数位掩码，故通过依次将 0xAA 左移 24、16、8 位得到 32 位二进制中奇数位的掩码。取反得偶数位的掩码。
分别通过掩码获取 x 的奇数位与偶数位，再将奇数位右移 1 位，偶数位左移 1 位并相加，得到奇数位与偶数位互换后的值。
其中，在将奇数位右移时，需将第 31 位置为 0。

P9 secondLowBit
x 与~x+1 在最后一位 1 及其往后的值均相等，往前的值均相反。
将 x 与~x+1 做与运算，得到仅含最后一位 1 的 last。
x 与~last 做与运算，将最后一位 1 置为 0。
重复上述操作，即得倒数第二位 1。

P10 rotateNbits
由于算术右移会导致最前 n 位消失以及最后 n 位为 0。需在算术右移后将前 n 位补在最后 n 位。
分别设置左右 n 位掩码。先用左掩码获取 x 前 n 位，将其右移 32-n 位后，考虑算术右移会受符号位影响，故使用右掩码消除该影响。从而将前 n 位移至最后 n 位。
将 x 算术左移 n 位并与上述最后 n 位相加，即得所求值。

P11 fractions
在二进制数中，左移 n 位等同于乘以 2^n，右移 n 位等同于除 2^n。故 7x 可表示为 4x+2x+x（其中 4x 通过左移两位实现，2x 通过左移 1 位实现），x/16 可通过右移 4 位实现。
其中，由于算术右移会导致前 4 位受符号位影响，而题中强调 x <=（1<<28）。故可设置掩码，将前 4 位设置为 0，消除符号位的影响。

P12 overflowCalc
由于三个数在相加中会产生溢出，且无法直接通过 x+y+z 记录溢出项，故先求两数之和。
两数求和，若产生溢出，溢出项一定为 1。
可通过比较加数与结果的最高项判断是否产生溢出。若两个加数最高项均为 1，则一定发生溢出；若两个加数最高项均为 0，则一定不溢出；若两个加数一 0 一 1，则当结果最高项为 1 时无溢出，结果最高项为 0 时溢出。
分别对 x+y 以及（x+y）+z 做溢出分析，返回值为两结果之和。

P13 mul13
由于最高位为符号位，且进行乘法运算，故若产生溢出，则最高项改变，否则，最高项不变。
可通过对最高项进行异或检查是否发生溢出。
判断 x 的符号。当 x 为负数时，若产生溢出，则返回 max；当 x 为正数时，若产生溢出则返回 min。
其中，由于 x*3=2x+x，两处运算任意一处发生溢出都将导致结果溢出。故对两处操作进行两次溢出分析。

P14 float_abs
在 IEEE 浮点数表示中，只需将最高位符号位设置为 0，即可得浮点数的绝对值。
其中，当 x 为 NaN 时（即指数项全为 1，尾数不为 0），返回 x。
通过分别设置符号位（第 31 位）、指数项（第 30 位至第 23 位）及尾数（第 22 位至第 0 位）的掩码提取浮点数的三个部分进行操作。

P15 float_half
在 IEEE 浮点数表示中，可通过两种方法得到 x/2。当指数不为 0 且不为 1 时，指数减 1 即可得 x/2；当指数为 0 时，尾数右移 1 位即可得 x/2。特别地，当指数为 1 时，若直接指数

减 1，x 则转变为非规范数，会导致精度改变，故将指数与尾数一同右移 1 位，通过尾数的右移得到 x/2，同时指数右移在尾数上补 1。
其中，当尾数右移时，由于最后一位被消除，故需进行舍入操作，判断最后两位是否为 0。若不为 0，则在右移后的尾数上加 1。
同上题，当 x 为 NaN 时，返回 x。

P16 float_i2f
将整数转换为浮点数时，先特殊判断 x 的特殊情况。当 x=0，返回 0；当 x=0x80000000 时，由于后续 x 取相反数会出错，故单独操作其返回 x 的浮点数形式 0xcf000000。
判断 x 的符号位。当 x 为负数时，取 x 相反数。
循环获取指数。因为在二进制中，左移 n 位为 2^n，故通过循环搜索 x 的最高位的 1，得到其索引，再加上偏移值 127，即得指数。
掩码获取尾数。通过设置尾数的掩码，将 x 平移后再与掩码做与运算得到尾数。其中，由于右移过程中会导致最后 8 位消失，故此处也需进行舍入操作。又因为在进行舍入操作时，可能导致尾数溢出至指数位，故需再判断尾数的溢出。若溢出，利用掩码获取尾数，并对指数加 1。

P17 oddParity
由于总数为 32 位，为偶数，可考虑两两配对比较进行求解。
通过分段求异，先用前 16 位与后 16 位求异，保留单独的 1；再在 16 位中分 8 位求异；再分 4 位求异……最后两位进行求异。若为奇数，此时保留下的为 1，按要求求反则为 0；若为偶数，则求反为 1。

P18 bitCount
利用分步思想。首先两两配对，通过掩码获取右移偶数位，并与原数相减，从而使每两位的值可以表示原数中这两位 1 的个数。（即 11->10，10->10，01->01，00->00）
再以两位为一组，每两组相加，可得四位相邻位的 1 的个数；再每四组相加……以此类推，最后返回 x 的后 6 位（max=32，即 2^5）即可代表原数 x 中 1 的个数。将 x 与 0x3f 进行与运算，得到所求值。

四、参考资料

https://blog.csdn.net/Jackiezhang1993/article/details/129132362

五、对实验的感受

第一次做 ics 的 lab 实验【lab0 不算（bushi 】，感觉还是很有难度，很多题都有非常多需要注意的细节，照着./btest 改了很多次才能改对，但是做完之后确实收获非常大。感觉对位运算以及 IEEE 浮点数的掌握都更深刻了【感觉比数逻学完更深刻（bushi】。

六、对助教的建议

希望后面的实验不要太难 www。设置 lab 请手下留情 QAQ。