ICS LAB1-数据位表示

宁晨然 17307130178

一、实验准备

本次实验内容是完成 bits.c 中的数据位运算、熟悉计算机内部数据存放形式，运用 c 语言实现各种操作。在实验前，我先复习了 float、int、signed、unsigned 的机内表示，复习 ppt 和书本知识，重新巩固了一遍。然后阅读 README 和介绍 pdf，重点看了看 btest 和 dlc 的使用方法，比较困惑的是 dlc 的作用，先使用了一下这两个工具发现报错。

```
chty627@ubuntu:~/Documents/lab1/datalab-handout$ make btest
gcc -O -Wall -m32 -lm -o btest bits.c btest.c decl.c tests.c
In file included from btest.c:16:0:
/usr/include/stdio.h:27:36: fatal error: bits/libc-header-start.h: No such file
 or directory
 #include <bits/libc-header-start.h>
                                    ^
compilation terminated.
In file included from decl.c:1:0:
/usr/include/stdio.h:27:36: fatal error: bits/libc-header-start.h: No such file
 or directory
 #include <bits/libc-header-start.h>
                                    ^
compilation terminated.
In file included from /usr/lib/gcc/x86_64-linux-gnu/4.8/include-fixed/limits.h:
168:0,
                 from /usr/lib/gcc/x86_64-linux-gnu/4.8/include-fixed/syslimits
.h:7,
                 from /usr/lib/gcc/x86_64-linux-gnu/4.8/include-fixed/limits.h:
34,
                 from tests.c:3:
/usr/include/limits.h:26:36: fatal error: bits/libc-header-start.h: No such fil
e or directory
 #include <bits/libc-header-start.h>
                                    ^
compilation terminated.
```

```
chty627@ubuntu:~/Documents/lab1/datalab-handout$ sudo apt-get install gcc-4.8-m
ultilib
```

百度一下后发现应该是 gcc 降级之后某个安装包缺失，所以我尝试了一下 sudoapt-get install gcc-multilib 命令，发现还是会报部分错误，再咨询一下发现需要安装具体版本的 multilib，便使用了 sudo apt-get install gcc-4.8-multilib。最后测试成功。

先学会评分工具的使用，一步一步来，做好实验准备，才能在之后 bits.c 中测试不会出错、倒回来解决，这样很容易错以为是自己修改错了什么东西。

二、实验结果

btest 结果 41/41，dlc 结果没有超出限制。

```
chty627@ubuntu:~/Documents/lab1/datalab-handout$ ./btest
Score   Rating  Errors  Function
 1      1       0       bitAnd
 2      2       0       getByte
 3      3       0       logicalShift
 4      4       0       bitCount
 4      4       0       bang
 1      1       0       tmin
 2      2       0       fitsBits
 2      2       0       divpwr2
 2      2       0       negate
 3      3       0       isPositive
 3      3       0       isLessOrEqual
 4      4       0       ilog2
 2      2       0       float_neg
 4      4       0       float_i2f
 4      4       0       float_twice
Total points: 41/41
chty627@ubuntu:~/Documents/lab1/datalab-handout$ ./dlc -e bits.c
/usr/include/stdc-predef.h:1: Warning: Non-includable file <command-line> included from includable file /usr/include/stdc-predef.h.
dlc:bits.c:143:bitAnd: 4 operators
dlc:bits.c:157:getByte: 3 operators
dlc:bits.c:169:logicalShift: 6 operators
dlc:bits.c:193:bitCount: 33 operators
dlc:bits.c:203:bang: 5 operators
dlc:bits.c:212:tmin: 1 operators
dlc:bits.c:225:fitsBits: 6 operators
dlc:bits.c:238:divpwr2: 7 operators
dlc:bits.c:248:negate: 2 operators
dlc:bits.c:258:isPositive: 4 operators
dlc:bits.c:272:isLessOrEqual: 14 operators
dlc:bits.c:288:ilog2: 30 operators
dlc:bits.c:305:float_neg: 3 operators
dlc:bits.c:337:float_i2f: 18 operators
dlc:bits.c:355:float_twice: 9 operators

Compilation Successful (1 warning)
```

三、实验过程

(1)、bits.c 解题思路（带*的题目参考了解题思路后独立完成）

1.bitAnd
由狄摩根定律显然

2. getByte
取 n 字节只需将右边 n 字节右移，然后用位与操作取末字节（左移 3 相当于乘 8）

3. Logicalshift *
逻辑右移与算术右移只差在高位填充的数字，则只需算术右移后将高位置 0 即可
用 s 构造前面 n 个 0，后面全为 1 的数，再与 x>>n 求位与运算

4. bitcount * （花费时间很长，参考许多种方法才明白）
从 00、01、10、11 两位的分法计数，与 01 取与运算后得到低位计数，再将数右移一位，与 01 取与运算后得到高位计数，相加即两位数的 count
之后操作相似，四位数的分发计数，与 0011 取与，同理类推
再推八位数，与 00001111 取与
再推 16 位数，与 0000000011111111 取与
再推 32 位数……
最终得到 32 位数的 count

5. bang *
0 与其他数的区别是 0 的相反数是其本身，所以用该数与该数的相反数取或运算，得到的符号位若为 1 即非 0，若为 0 即 0，所以先取反后右移取出符号位

6. tmin
求 int 最小，即负数中 10000……0

7. fitsbits
若 n 位能表示 x，则去掉后面 n-1 位时，正数中无 1，负数+1 后>>1 也不存在 1。

8.divpwr2 *
负数与正数的除法有区别，若为负数需要加 2^n-1 的偏置量，若为正数则不需要，所以用符号位右移 31 位再位与运算

9.negate
相反数只需要取反加一即可

10. isPositive
只需要判断符号位是否为 0，然后排除正 0 的情况即可

11. isLessOrEqual *
分两种情况，第一种是 x y 异号且 x 小于 0，y 大于 0

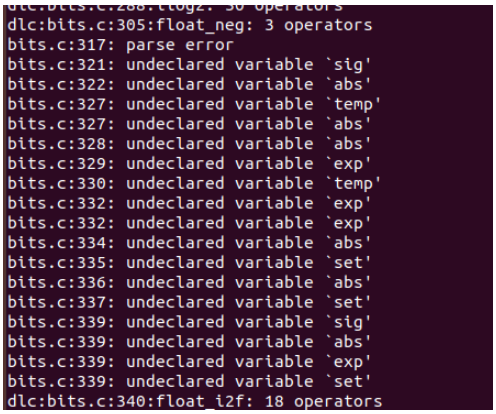
第二种是符号相同（相减不会溢出），x-y-1 小于 0

12. ilog2
相当于求 1 右边有几个 0，则采用 2^n 的方法右移，如果右移结果为 0 则不右移，结果为 1 则右移，列举 16、8、4、2、1 后可以构成 31 内所有数字。这道题后来查询了他人做法，发现这种方法叫做分治法。

13. float_neg *
分两种情况，第一种，当为 NaN，即 exp 全 1，且 uf 大于 0x7f800000
第二种，直接将符号位取反

14. float_i2f *
第一种情况，x=0，返回 0
若为负数，将符号位改为正，所有操作改成正数的操作
第二种情况（正常数），直接左移，直到所有位数移完，记录左移次数，再转换为 exp，abs 右移 9 即 frac 尾数
第三种情况 如果 int 数超过了 23 位的精度，需要对后面的精度进行舍去，运用四舍五入的方法
最终的 unsigned 结果为 sig + (exp<<23) + (abs>>9) + set

这道题做了很长时间，一部分时间是考虑情况较少，只好翻书看 ppt 找 float 的机内表示；后来发现自己写的代码太复杂，又在缩减；最后参考了附录网站中这道题的简易解题思路，才尝试写了出来。写出来之后发现特殊情况 0x80000000 的问题，又把 int 改成了 unsigned，最后由于 dlc 的判断问题，还花了些时间解决 dlc 报错，后来只需要把变量名声名放在开头。



15. float_twice *
第一种情况，denorms 或者 0，exp 全 0，只需要保证符号位不变，尾数左移一位
第二种情况，infinity 或者 NaN，返回本身
第三种情况，norms，exp 加 1 即可

(2)、遇到的困难

主要是在 bitcount/ilog2/float_i2f 三个函数中花费较多时间，这几个函数我都在网上查找了很多种解法，的确各有各的优势、想法，自己也思考了很多他们为什么可以这样思考出解法，也找到了原因，有些是从机内表示本身的特点出发，有些是二进制算数的巧妙操

作。在看完别人的思路后，我也自己重新做了一遍，当然这样我还是会做错，很多细节需要打磨，特别是 `float_i2f` 让我纠结了好久好久……总的来说，花费时间比我预想的要多些。

大部分题目是自己想出来的，打*号的有些是自己的思路总是做错…重点应该放在熟悉机内表示和二进制操作!

四、实验感受

一方面是在二进制的运算中不熟练导致不清楚解题思路过程，另一方面是知道解题思路后实现过程中遇到很多小细节无法 debug。实际操作中其实查询了很多资料，参考了很多算法逻辑才慢慢摸索出一种方法。这次学到了很多关于二进制的操作方法和 int float 的机内表示！希望下次能够更加快速的学会使用新的工具更快速的完成作业。

Reference:

//reference[1] https://blog.csdn.net/a794767404/article/details/79251430
//reference[2] https://zhuanlan.zhihu.com/p/28335741
//reference[3] https://www.cnblogs.com/tenlee/p/4951639.html
//reference[4] http://www.cnblogs.com/jarviswhj/p/9502612.html