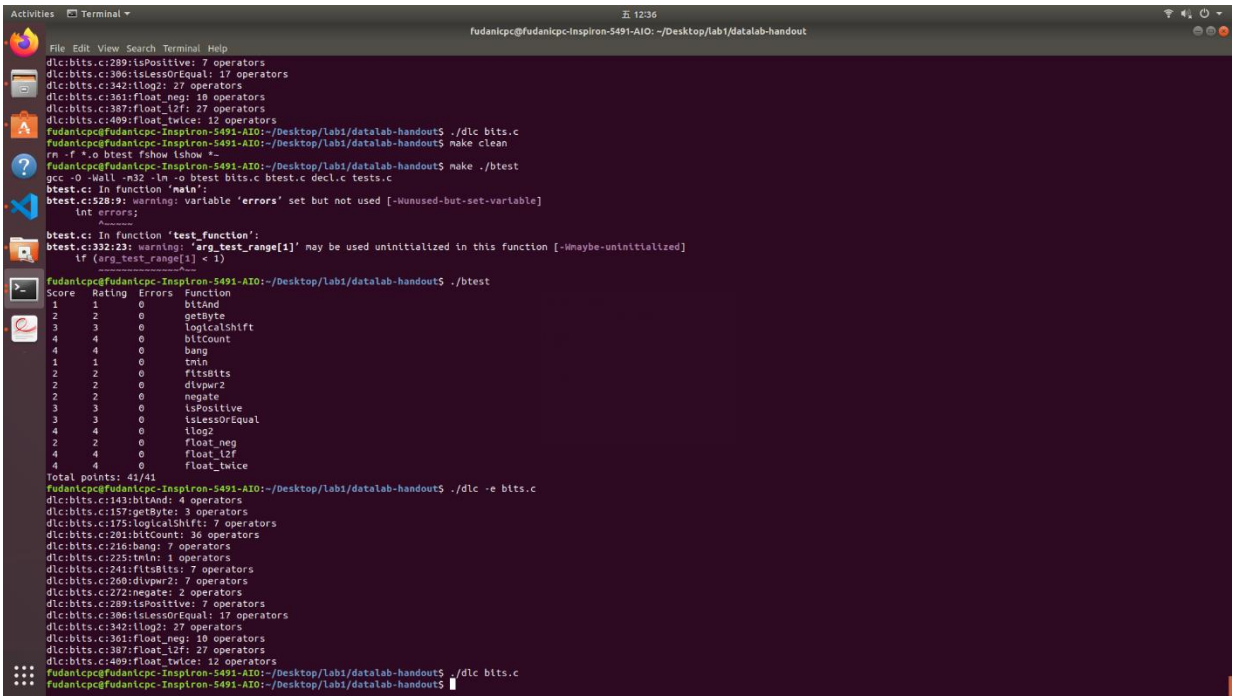
“计算机系统基础（上）Lab1”报告

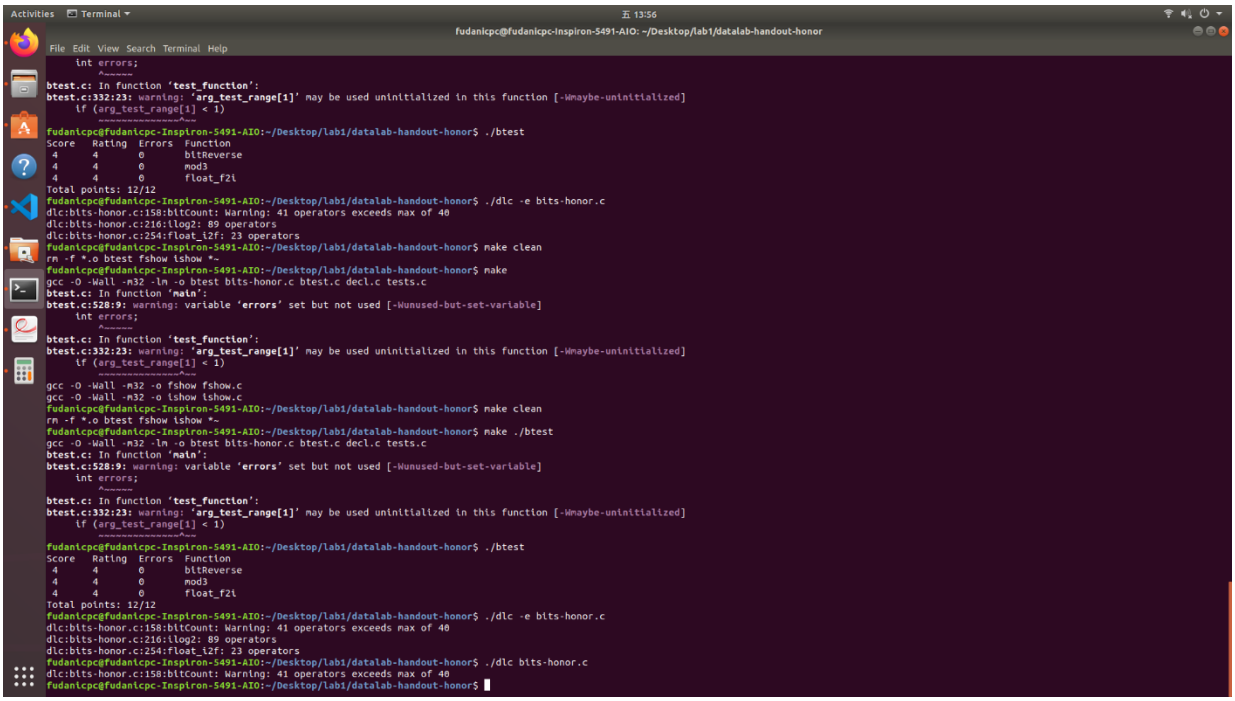
计算机科学与技术

19307130296

孙若诗

一、 运行截图





注：$bits-honor.c$中$bitreverse$未满足操作数限制。

二、 程序解析

1) $bits.c$

1．$bitAnd$

描述：用~和|实现&。

操作：4/8

解答：观察真值表，发现取反后或的结果刚好与&相反。

2．$getByte$

描述：取出 x 从低位起的第 n 个 byte。

操作：3/6

解答：一个 byte 为 8 个 bit，所以先将 n 左移 3 位，得到 x 需要右移的位数。x 右移后与 255，即 8 个 1，即可得到第 n 个 byte 上的数。

3．$logicalShift$

描述：逻辑右移。

操作：7/20

解答：移出来的前 n 位必须为 0，所以求一个只有后 32-n 位为 1 的数与上即可（似乎这种工具数被称为掩码）。可以将 31 个 1 右移 n-1 位，但是 n 为 0 时右移-1 位会出问题，先右移 n 位再左移 1 位可以避开这个坑。

备注：之前一直以为逻辑右移需要把负数补码中前导 1 都消去，做这道题时才意识到只需要处理移出的空位，至于原先就存在的前导 1 不必在意。

4．$bitCount$

描述：数 int 中二进制 1 的个数。

操作：36/40

解答：采用分治算法，先每相邻 2 位 1 的个数相加，再每相邻 4 位，直到前 16 位和后 16 位相加得到全部 1 的个数。关于取每相邻 n 位的操作还是用掩码，例如取相邻的 1 位使用 010101……0101，2 位使用 00110011……0011，以此类推。

5．$bang$

描述：实现!x。

操作：7/12

解答：先用~x+1 求出相反数，非 0 数和它的相反数之间至少有一个最高位为 1，取或之后把最高位移下来，然后-1 变+1（也就是再取一次相反数），再异或 1 即可。

备注：注意到-2147483648 的相反数在 int 范围内无法表示，~x+1 仍旧是它本身，但是所幸它最高位本来就为 1，并不影响结果。

6．$tmin$

描述：最小的 2 补码。

操作：1/4

解答：1<<31。

7．$fitsBits$

描述：查询 x 是否能用 n 位二补码表示。

操作：7/15

解答：注意到 int 本身就是补码形式，所以 n 位够用代表 32-n 位都是多余的，左移 32-n 位再移回来，如果数不变说明可以。如果能用 n 位表示，左移时失去的和右移时补回来的都是符号位上的数，所以结果不变。

备注：有了相反数操作就很容易实现减法了。但是-2147483648 仍不能实现。^操作可替代==。

8．$divpwr2$

描述：x/(2^n)，靠近 0 取整。

操作：7/15

解答：直接>>n 位，负数会出问题，这是因为负数的默认向下取整相当于远离 0 取整。我们在正数向上取整的时候经常用到一个操作，(a+b-1)/b 相当于 a/b 向上取整，因为如果刚好整除，没有影响；只要有余数，都能向上进一位。这里我们就是相当于除数为负时改成向上取整，所以要给负数加 2^n-1。

9．$negate$

描述：求相反数。

操作：2/5

解答：~x+1。

10．$isPositive$

描述：判断是否为正数。

操作：7/8

解答：只有 0 是符号位为 0 的非正数，特判一下即可，其他数只需要看符号位是否为 0。

11．$isLessOrEqual$

描述：x 是否小于等于 y。

操作：17/24

解答：直观的想法是判 x-y 是否为小于等于 0，但是 x 为-inf，y 为 inf 时可能会溢出，因此先判一下它们是否异号，异号则可以直接判断大小。

12. $ilog2$

描述：log 以 2 为底 x，向下取整，x>0。

操作：27/90

解答：显然答案在 0~31 之间，显然任何 0~31 的整数都可以由 16、8、4、2、1 从大到小拼凑得到，并且如果当前数为 x，可加数为 y，目标数为 z，x+y<=z 时给 x 加上 y 一定不会出错。因此按从高到低顺序检查它是否大于等于 2 的这些幂次，是则累加这个指数，并减去这个幂次检查下一个即可。

13. $float\_neg$

描述：给一个浮点数取相反数。

操作：10/10

解答：nan 的定义为指数位全为 1，分数值非 0。除去 nan，余下的数符号位取反即可。

备注：可以发现 inf 也是分正负的。

14. $float\_i2f$

描述：将 int 转换为 IEEE 标准下的 float。

操作：27/30

解答：模拟题，首先把符号位取出来。然后看一看绝对值一共有多少位。左移到最高位，右移 8 位作为分数值，后面的 8 位要被舍去。如果最后一位为 1 或者后 8 位不为空，需要进 1。指数需要加 127，最后结果把符号位、指数位和分数值依次拼起来即可。

备注：-2147483648 和 0 是特殊情况，-2147482648 在 int 内无法取绝对值，0 没有 1，因此特判掉这两个数。

15. $float\_twice$

描述：给一个浮点数乘 2。

操作：12/30

解答：分别取出指数位和分数值，若指数位为 0，给分数值左移 1，加上符号位。若指数位不为 0 也未满，直接给指数值+1。

备注：验证了课上所讲的原理，若分数值超出 23 位，多出的一位恰好加给指数位，无需特别处理。

2) $bits-honor.c$

1. $bitReverse$

描述：左右翻转二进制位。

操作：41/40

解答：类似 bitcount，依次交换相邻 2 位、4 位……32 位。

备注：以上思路需要 41 次操作，并未达到题目要求。

2. $mod3$

描述：模 3。

操作：89/90

解答：观察易知 2 的偶数幂模 3 均为 1，2 的奇数幂模 3 均为 2，因此类似 bitcount 地分治统计有多少个偶数幂和奇数幂，可以把被模数降到 16*1+15*2=46 及以下规模，这时只剩下 6 位，可以每两位&3 相加，得到的结果仍然模 3 同余。如此迭代下去，3 次后只剩下 2 位，只要判断一下剩下的是否恰好为 3，是则减去 3 即可。

备注：原本对后期一位位右移统计，只能卡到 103 次操作，每两位一处理的思路得到了高庆麾同学的指导。关于符号的讨论详见总结。

3. $float\_f2i$

描述：将 float 转化为 int。

操作：23/30

解答：取出符号位、指数位和分数值。分数值需要左移指数位、右移 23 位，因此先用指数值减去 127 再减 23，和分数值位数比较，若超过 int 范围就直接返回结果（-inf 或者 0），确定未超过限制再移动分数值。

备注：int 右移超过位数会错误，并不是因为前面补 0 就没有影响。

三、 总结

1. 质疑 mod3 的 check 程序是否有负数情况，程序完全未处理符号位时也通过了 btest 测试。最后手动调整了负数的偏差，负数时结果-1，因为-2147483648%3=-2=1，但当作第 32 位处理，偶数位的贡献为 2，多了 1。但-2 时输出为 1，在模运算下等价。

   题目描述中的 mod3(-100)=-1 本身也不太妥当，一般数学定义的整数模运算结果应为[0,n-1]区间内整数，若出现负数被模也应当加到正数域内，因此 mod3(-100)=2 似乎更规范。但是使用 C++测试，（-100）%3 确实为-1，可能是绝对值取模后再加符号。
2. 第一次写 C 语言程序，了解到函数要先定义所有变量，不能中途定义的规则。
3. Linux 系统的 programming mode calculator，Windows 系统的程序员计算器都有较为直观的数制转换、同步显示功能，对于这次 lab 有很大帮助。
4. 熟悉了一些常用的替代关系：(~0)=1，^可替代==，|对 bool 型可替代+，~x+1 可替代-，|可替代||，&可替代&&，x 为 0 或 1 时&xx……x 可部分替代 if(x)，!!

可以将一个 int 转为 bool。

5. 认识到了自己对于计算机系统的无知。完成 lab 过程中虚拟机无法开机，尝试了网络上搜索到的 netsh winsock reset、关闭 3D 优化等各种方法，均未奏效。解决方法似乎只剩下重装，深感困扰，又暂时不愿意陷入安装双系统的麻烦。最终还是在一台主系统为 Ubuntu 的机器上完成了这次 lab。

四、 参考文献

1. https://zh.wikipedia.org/wiki/NaN
2. https://zh.wikipedia.org/wiki/IEEE_754
3. https://zh.wikipedia.org/wiki/模除
4. https://zh.wikipedia.org/wiki/C 语言