|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程编号 1502760001-01  题目类型 实验2 | |  |  |  | | --- | --- | --- | | **得分** | **教师签名** | **批改日期** | |  | 冯禹洪 |  | |

**深圳大学实验报告**

**课程名称： 计算机系统(2)**

**实验项目名称： 数据表示实验**

**学院： 计算机与软件学院**

**专业： 腾班**

**指导教师： 冯禹洪**

**报告人： 叶茂林 学号： 2021155015 班级： 腾班**

**实验时间： 2023年4月6日**

**实验报告提交时间： 2023年4月9日**

**教务处制**

|  |
| --- |
| **实验目的与要求：**   1. 了解各种数据类型在计算机中的表示方法 2. 掌握C语言数据类型的位级表示及操作 |
| **方法、步骤：**   1. 安装gcc-multilib：   2、根据bits.c中的要求补全以下的函数：  int bitXor(int x, int y);  int tmin(void);  int isTmax(int x);  int allOddBits(int x);  int negate(int x);  int isAsciiDigit(int x);  int conditional(int x, int y, int z);  int isLessOrEqual(int x, int y);  int logicalNeg(int x);  int howManyBits(int x);  unsigned float\_twice(unsigned uf);  unsigned float\_i2f(int x);  int float\_f2i(unsigned uf);  3、在Linux下测试以上函数是否正确，指令如下：  \*编译：./dlcbits.c  \*测试：makebtest  ./btest |
| **实验过程及内容：**  1、安装gcc-multilib：  切换清华源更新sources.list，如图1所示。    图1  安装make gcc-multilib，如图2所示。    图2  Make编译程序生成btest，如图3所示。    图3  运行btext测试一下，结果如图4所示。    图4  2、根据bits.c中的要求补全以下的函数：  （1）int bitXor(int x, int y);  由离散数学可知，X和Y异或可以表示为(~X&Y)|(X&~Y)，或不能用，有德摩根律等价于~(~(~X&Y)&~(X&~Y))，如图5所示。    图5  （2）int tmin(void);  32位整数最小值是0X8000 0000，即将1左移31位可以得到，如图6所示。    图6  （3）int isTmax(int x);  32位最大的二进制补码是0X7FFF FFFF，加一之后是32位最小值0X8000 0000，再加上自己是0，即对2\*(x+1)进行逻辑非运算可以得到1，由此可以判读出是否是最大值，但是0加上自己也是0，所以需要把0的情况排除，即对x+1进行两次逻辑非运算，最后取与结果，如图7所示。    图7  （4）int allOddBits(int x);  先把所有奇数位提取出来，将x和0XAAAA AAAA相与即可，首先通过0XAA左移相加的方式构造出0XAAAA AAAA，再令x为按位与的结果，如果所有奇数位都是1，那么x此时应该是0XAAAA AAAA，按位取反后是0X5555 5555，再左移一位后就变回0XAAAA AAAA，所以我们可以通过将x和~x+~x异或的结果来判断，如图8所示。    图8  （5）int negate(int x);  要求其负数的补码，计算上可以直接对原码进行按位取反后加一即可，如图9所示。    图9  （6）int isAsciiDigit(int x);  即要判断x是否大于0X30并且小于0X3A，对二者按位取反加一后得到负数与x相加，将结果右移31位即可判断符号位，如图10所示。    图10  （7）int conditional(int x, int y, int z);  即当x为0时返回z，否则返回y，即当!x为0时返回y，当!x为1时返回z，考虑按位与运算，需要返回的值与0XFFFF FFFF（即-1）相与，不返回的值与0相与，首先将1按位取反加一得到-1，加上0就是需要返回，加上-1就是不返回，如图11所示。    图11  （8）int isLessOrEqual(int x, int y);  一开始我想到的是直接用y-x判断是否是负数来操作，但是测试了一下发现异号的时候有可能会出现溢出的情况，所以要先判断是否同号，通过右移31位取符号位进行异或可以判断是否同号，同号的情况就进行y-x操作，异号的情况就判断y是否是非负数，如图12所示。    图12  （9）int logicalNeg(int x);  只有x是0才返回1，取x相反数-x，那么x和-x的符号位应该都是0，右移31位和1按位与提取出符号位进行判断，如图13所示。    图13  （10）int howManyBits(int x);  如果是正数的话，我们先右移16位，如果为0，那说明高16位都是0，那么问题变成了判断低16位的；如果不是0，那么说明高16位不都是0，那么问题变成了判断高16位的。  即是将问题规模变小，每次二分。  负数的话直接取反就行，通过和符号位异或就可以实现，因为任何数和0异或是其本身，和1异或就是取反。  0和-1非常特殊，它们均只需要1位就能表示，事实上，由于进行了负数取反，-1实际变成了0，如图14所示。    图14  （11）unsigned float\_twice(unsigned uf);  浮点数IEEE标准是用符号+阶码+尾数来表示的，对于规格化的数，只需要将阶码加一即是加倍，对于非规格化的数，需要将尾数左移一位，对于无穷大直接返回本身。  首先提取符号位、阶码和尾数出来，判断是否为非规格化的数，再判断是否是规格化的数，分别进行操作即可，如图15所示。    图15  （12）unsigned float\_i2f(int x);  浮点数是由符号位s+阶码exp+尾数frac三部分组成的，首先通过判断正负确定符号位，然后把负数变成正数，循环左移数一下阶码s，然后确定尾数frac。  如果阶码exp小于等于24，那么尾数frac直接就是x左移24-exp位；如果阶码exp大于24，那么小数部分就需要舍入，由于是偶数舍入，先判断被舍弃的部分是否是中间值，即首位是否是1，如果是的话，再判断尾数frac最后一位是否是1，是就frac加一，如果被舍弃的部分大于中间值，那么frac直接加一，最后判断一下frac有无溢出，即判断后面23位是不是变成0了，如果有，exp加一，frac右移一位，如图16所示。    图16  （13）int float\_f2i(unsigned uf);  先提取符号位、阶码和尾数出来，判断阶码看看在不在范围，然后根据阶码左移右移尾数，最后如果是负数的话，补码需要取反加一，如图17所示。    图17  3、在Linux下测试以上函数是否正确，指令如下：  \*编译：./dlc bits.c  编译出现问题，如图18所示。    图18  将变量的声明放在代码块前面，并优化运算符的使用数目后，编译无问题，如图19所示。    图19  \*测试：make btest  ./btest  最后测试代码，全部正确，如图20所示。    图20 |
| **实验结论、心得体会：**  本次实验可以说是非常的艰辛，远没有我想象中那么简单，短短的十三个函数实现，却让人把各种资料都查了个遍，其解题思路之巧妙、数据表示之精准让人敬畏计算机科学。  这绝对是一次深度烧脑的实验，实验中我遇到了两个卡壳点，一个是实现函数int isLessOrEqual(int x, int y)，我在进行调试的时候，进行打印输出变量的值测试，发现居然加入printf函数输入时成功通过了所有测试样例，而去掉之后神奇般再次出错，网罗资料后了解到printf有可能会改变程序运行结果，但不知是何原因，于是我把函数代码单独拎出来用gdb调试，确实是正确的结果，于是上报老师，联系助教学长，最后发现是O优化的问题，程序设置了O1优化，改为O优化后测试通过，为什么O1优化会改变了程序的行为，这个问题的思考已经超出了我的能力范围了。  第二个卡壳点是在实现整数转浮点数函数中，这个函数用了我一天的时间才搞定，网上代码泥沙俱下不如靠自己，我在反复翻阅浮点数表示的章节，不断思考向偶数舍入的过程，终于是把转换过程搞的明明白白，后来出错调试的时候，是溢出的问题，包括frac溢出和负数取相反数的溢出，一开始我是让Chat GPT3.5帮我调试的，但是它还没有那么聪明，调试过程中有时候它也把0x80000001当作INT\_MIN，还好我看得出来它不对，还得让我来，开始内存表示级别的调试，看着一堆十六进制的内存数据表示，不断调试，终于通过全部测试样例了。  没想到这个实验会写这么久，确实这个实验之后，数据表示已经刻到骨子了，写完是如释重负、如沐春风、一身轻松啊，不过却可以深深地体会到计算机科学的美妙，敬畏计算机科学。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字： 冯禹洪  2023年 月 日 |
| 备注： |