**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： 计算机科学与技术(创新班)**

**指导教师： 马晨琳**

**报告人： 李文俊 学号： 2023150001 班级： 高性能班**

**实验时间： 2025年4月2日 至 4月16日**

**实验报告提交时间： 2025年 4 月 6 日**

**教务处制**

|  |
| --- |
| **一、实验目的：**   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 |
| **二、实验内容：**   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 floatScale2(unsigned uf) ;  int floatFloat2Int(unsigned uf);  unsigned floatPower2(int x);  3、在Linux下测试以上函数是否正确，指令如下（详见README文件）：  \*编译：./dlc bits.c  \*测试：make btest  ./btest |
| **三、实验思路及求解过程：**  1. int bitXor(int x, int y);  目的：x^y using only ~ and &  思路：  摩根定律可以把A Xor B = (A&~B)U(~A&B)，而AUB又可以等价为AUB= ~(~A&~B)，  因此**A Xor B=~(~(A&~B)&~(~A&B))**    图1  2.int tmin(void);  目的：return minimum two's complement integer  思路：  Tmin=1000000，可以直接移位    图2  3. int isTmax(int x);  目的：returns 1 if x is the maximum, two's complement number,  思路：  利用Tmax 0111111..的特性 **Tmax+1=~Tmax**，得到**第一个判断条件!(~x^(x+1))**  判断两个数是否相等：可以通过异或为0则相等，再取！，则相等返回1。  测试发现 -1(0111111) Failed, 发现-1同样满足 ~x^(x+1) =0，因此需要将-1排除，  得到**第二个判断条件：！！((x+1)^0)**，如果是-1，返回0。    图3  4. int allOddBits(int x);  目的：如果所有奇数位都为1则返回1;否则返回0  思路1：  因为能使用的立即数不能超过0xff，所以一次能检测8位，可以通过 x&0xaa(10101010),得到低八位的奇数位，再判断是否与0xaa相等（异或操作再取！）。  32位需要比较 8 16 24 32(x本身)，所以移位操作只需要3个  如果分开检测，操作符个数必然超过12个，这里用到的一个技巧是所有数先&再判断，只有&的结果都是0xaa才满足要求。    图4.1  思路2：  另一个思路就是不改变x，而是**通过移位操作得到一个掩码**    图4.2  5. int negate(int x);  目的：返回-x  思路：  **补码的本质 -x=~x+1**    图5  6. int isAsciiDigit(int x);  目的：如果x是ascii码中的0~9，返回1;否则返回0  思路1：  **1.** 0-9就是0x30 <= x <= 0x39，先确定高四位是0011，**第一个判断条件 (x&0xf0)^0x30**  **2.** 低四位通过观察得到 不能出现 1x1x和11xx，得到**第二个判断条件**  **!((x&0xa)^0xa) 和 !((x&0xc)^0xc)**  **3.** 最后高24位的0需要全为0，因此**得到第三个判断条件 (x>>8)^0**  **4. 将这几个条件按逻辑结合在一起，减少操作数**    图6.1  思路2：  **将问题转换为 x-0x30>=0 & 0x39-x>=0，直接通过运算然后判断符号位，比较直观、**  **为什么不需要考虑溢出呢：因为有两个条件同时限制，即使x-0x30溢出，右边也不会同时满足。**    图6.2  7. int conditional(int x, int y, int z);  目的：实现x?y:z  思路：  开始时只想到了第一步就是 **(y& !x) | (z& x); 根据!x来筛选y/z**，但是&会破坏原本的y和z，  后来了解到可以通过 **~!x+1,将0换成0x000,1换成 0xffff,真是妙啊。**    图7  8. int isLessOrEqual(int x, int y);  目的：如果x<=y返回1否则返回0  思路：  **1.** 将问题转化为 x-y<0和x=y，**涉及运算必然要考虑溢出，因此将运算符分为同号和异号，**  **判断相等：!(x^y) 同号判断方式：!( (x^y)>>31) 同号返回1**  **2. 同号不需要考虑溢出，判断 x+(~y+1) 是否小于0即可**  **3. 异号则根据符号位直接判断 , x>0 ,y<0 返回0， x<0,y>0返回1，(x>>31)&!(y>>31)可同时满足。**    图8  9. int logicalNeg(int x);  目的：实现！运算符的功能  思路：  **1**. 抓住0的一个特性，**正负相等，得到 x^(~x+1) ==0**  **2. 怎么处理其他值呢？ 通过观察 x^(~x+1) ，如果不是0和Tmin，最高位都会变成0，所以可以判断符号位+1**  **3. 特殊情况 Tmin（100000）特殊处理 （x>>31）&0x1^0x1, 如果是Tmin返回0**    图9.1  **优化：**上面两个条件都需要x>>31,不如融合起来**：不用异或操作，而用或操作，x | (~x+1) , 除了0之外，其他数最高位都是1，这样就不用单独处理Tmin，真是妙啊！**    图9.2  10. int howManyBits(int x);  目的：返回将X表示为补码所需的最小有效位数  思路：  **1. 这个题目最关键的地方在于如果处理正数和负数，如果是正数，需要找到它的最高位为1位置，然后加上符号位，如果是负数，在于找到最高位为1的位置，不需要加上符号位，**  **这是一个误区，如果这样想，问题就会变得很复杂，实际上正数和负数是统一的，对于负数来说，需要找到最高位为0的位置，然后加上符号位(比如-5(1011))，这样正数和负数本质上就相同**  **因此，可以把负数，转换为正数，将问题转换为找到最高位为1的位置**  **2.怎么找？顺序找需要>> , & , + ，三个运算符\*30已经超过最大操作符数，可以巧妙的采用二分法：**  **因为现在我们只需要找到最高的1：**  **如果高16位有（不全为0），那就只需要看高16位，result+=16，将高16位移到低16位，然后看高8位，这样不断往下走到第一位**  **如果高16位没有：那就不需要移位，只看低16位**  **无论有还是没有，都只需要看低16位。**  **3.特殊情况：0和-1都只需要1位，0可以通过二分结果为0 +1=1，而-1 符号位和数值位重叠，它的位级表示就是 1，在负数转成正数时变成了 ~1 🡪 0，因此，-1和0其实相同。**    图10  11. unsigned floatScale2(unsigned uf);  目的：以unsinged表示的二进制浮点数f的二倍2f,返回对应flaoat的二进制unsigned型  思路：  **1. 这道题考察的是对于IEEE浮点数编码格式的理解**，**unsigned uf 分为 sign(1) exp(8) frac(23)**  **Sign=uf&(1<<31) exp=(uf>>23)&0xff frac=(uf<<9)>>9**  **2.** 直观上好像只需要将exp+1即可，开始我也这么认为，然后报错 0x1,应该返回0x2  **找到误区：直接对非规格化数指数+1的错误，对于非规格化，必须通过左移尾数处理，否则就改变了非规格化的性质, 即只能通过 frac<<1处理**  **3.处理特殊情况：exp无穷大，返回inf，exp+1==0xff，返回无穷大,最后返回浮点数编码的结果。**    图11  12. int floatFloat2Int(unsigned uf);  目的：将单精度浮点数f转换为 int 类型,超过范围返回0x80000000u  思路：考察浮点数转整数 (int)f  **1.** 将浮点数分为**sign(1) exp(8) frac(23)**  **2. 是否需要区分规格化和非规格化呢？不需要！因为非规格化表示的接近于零的数，应该返回0**  **3. 规格化：得到 E=exp-127 , M=0x80000|frac**  **4. M按照E移位，需要注意的是：对于int，除去符号位和隐含标志位，M只有30位，E不能大于30和小于0(小数)**  **5.因为得到exp的时候已经>>23,所以需要根据E>23分类讨论继续左移还是截断小数**  **6.产生问题：怎么处理符号位? M是否会溢出？现在M有24位：00000000 1frac(M)，最多再左移(30-23=7)，所以M不可能超过01111111（Tmax),所以最后不需要再根据符号位判断是否溢出**  **7.unsigned 转 int ，负数=~x+1,不会溢出，正数直接返回结果。**    图12  13. unsigned floatPower2(int x);  目的：将int x转换为单精度浮点数表示2^x，如果太小返回0，太多返回+inf  思路：  **1.** 直观上exp=x<<23解决问题，注意需要**exp=E+bias=x+127**  **2. 特殊情况：x太小，返回0，x太大，返回+INF（exp全1）**  **3. 边界确定：在于exp范围是 unsigned 8位 ，即 0-255,边界为0/255**  **4. 坑：需要设置frac=0x1，使结果表示为 1\*2^exp吗？实际上是不需要的，做的时候遗忘了浮点数规格化已经隐含了一个1**    图13  测试结果  1.编译：    图14  2. 测试：make btest    图15  3. 运行测试程序    图16 |
| **四、实验结论及问题：**  **结果显示36/36，说明编写的函数全部通过。**  通过本次实验，我深刻体会到了位级运算的重要性，并且对位级运算的理解更加深入。位级运算是一种基础的运算方法，它可以高效地执行各种逻辑运算。在实验中，我学习到了位级运算的基本概念，例如与、或、异或、取反等操作。通过这些操作可以对二进制数进行各种逻辑运算，这对于理解计算机底层原理有很大的帮助。 在实验过程中，我也锻炼了使用位级运算的能力，学会了如何使用位级运算对二进制数进行各种操作。例如使用位掩码来提取二进制数的特定位，使用位移操作来将二进制数向左或向右移动，使用逻辑运算来进行位级运算等。这些操作不仅能够在实验中使用，也可以在编写实际的程序时使用，从而提高程序的效率和性能。 通过datalab实验，我还学习了很多其他的知识。例如IEEE浮点数的存储方式和对浮点数进行位级操作、使用gdb调试程序和makefile进行编译等。  同时在实验中，我也发现了自己在学习第二章的诸多问题：常见问题包括位操作逻辑设计复杂（如用有限运算符实现条件判断）、特殊值处理不当（如0、全1或非规格化数的边界情况）、溢出判断错误（未正确处理整数运算的符号位变化或浮点数阶码超限），以及浮点数精度丢失（如规格化移位时隐含位处理错误或舍入偏差），需结合位模式分析逐层调试解决。  **在不断调试的过程中，也在不断地感受着CSAPP实验的魅力。** |

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2025年4月 日 |
| 备注： |