**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： 计算机与软件学院所有专业**

**指导教师： 刘 刚**

**报告人：XXX 学号：XXXXXXXXXX 班级： 计算机科学与技术（高性能班）**

**实验时间： 2024年4月 15 日 至 4月 26日**

**实验报告提交时间： 2024年4月 21 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：     图表 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 min(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)  1）题目描述：   * bitXor – 只使用有~和&来计算x XOR y * 示例: bitXor(4, 5) = 1 * 合法操作: ~ & * 最大操作数: 14 * 评级: 1   2）思路：  由离散数学知识可知，，由于题目限制只能使用和，所以考虑德摩根定律，用和凑出，即。  3）代码实现：    图表 2 bitXor函数代码  2、int tmin(void)  1）题目描述：   * tmin - 返回最小的补码整数 * 合法操作: ! ~ & ^ | + << >> * 最大操作数: 4 * 评级: 1   2）思路：  对于32位补码整数，最小值即为0x80000000，相当于0x80左移24位。  3）代码实现：    图表 3 tmin函数代码  3、int isTmax(int x)  1）题目描述：   * 如果x是最大的二进制补码，返回1；否则，返回0 * 允许的操作符: ! ~ & ^ | + * 最多操作符数目: 10 * 分值: 2   2）思路：  对于32位补码整数，最大值即为0x7FFFFFFF，相当于0x7F左移24位加上0xFF左移16加上0xFF左移8加上0xFF。  为了判断x是否为最大值，采用x与最大值0x7FFFFFFF相减的方式，若相减结果为0，说明x等于0x7FFFFFFF，则用!进行布尔值取反，返回1；若相减结果不为0，说明x不等于0x7FFFFFFF，则用!进行布尔值取反，返回0。x加上的结果即为x与0x7FFFFFFF相减的结果。  3）代码实现：    图表 4 isTmax函数代码  4、int allOddBits(int x)  1）题目描述：   * 如果所有奇数位都为1则返回1;否则返回0 * 例子： allOddBits(0xFFFFFFFD) = 0，allOddBits(0xAAAAAAAA) = 1 * 允许的操作符: ! ~ & ^ | + << >> * 最多操作符数目: 12   2）思路：  对于32位补码整数，奇数位都为1的数为0xAAAAAAAA，则令x = x & 0xAAAAAAAA提取出x奇数位上的所有数，再令x与0xAAAAAAAA相减，若相减结果为0，则说明x奇数位上全为1；否则，说明x奇数位不全为1。  3）代码实现：    图表 5 allOddBits函数代码  5、int negate(int x)  1）题目描述：   * 返回x的相反数 * 例子: negate(1) = -1. * 允许的操作符: ! ~ & ^ | + << >> * 最多操作符数目: 5   2）思路：  对于补码整数，取反加一即为该数的相反数。  3）代码实现：    图表 6 negate函数代码  6、int isAsciiDigit(int x)  1）题目描述：   * 如果x是ascii码中的0~9，返回1;否则返回0 * 例子: isAsciiDigit(0x35) = 1，isAsciiDigit(0x3a) = 0，isAsciiDigit(0x05) = 0。 * 允许的操作符: ! ~ & ^ | + << >> * 最多操作符数目: 15   2）思路：  对于每个输入的x，需要满足x>=‘0’且x<=‘9’。判断x是否在该范围内，可以将x与48（即‘0’）相减，判断相减结果的符号位，若符号位为1，表示x< ‘0’，不满足条件，返回0；若符号位为0，表示x>=‘0’，满足条件，返回1。同理，将57（即‘9’）与x相减，判断符号位，若符号位为1，表示x>‘9’，不满足条件，返回0；若符号位为0，表示x<=‘9’，满足条件，返回1。将两个判断结果相与，若结果为1，则说明x满足x>=‘0’且x<=‘9’，函数返回1即可。  判断符号位是否为1，只需要令相减结果与0x80000000相与，若结果不为0，说明符号位为1；相反，则符号位为0。  3）代码实现：    图表 7 isAsciiDigit函数代码  7、int conditional(int x, int y, int z)  1）题目描述：   * 实现x？y：z * 例子: conditional(2,4,5) = 4 * 允许的操作符: ! ~ & ^ | + << >> * 最多操作符数目: 16   2）思路：  可以将结果用表达式( a &y)|( b &z)表示，若x != 0，则a=0xFFFFFFFF，b=0；若x==0，则a=0，b=0xFFFFFFFF。  对于a，令a=!x + ~1 + 1（~1+1相当于0xFFFFFFFF），当x != 0时，!x=0，a=0+0xFFFFFFFF=0xFFFFFFFF;当x==0时，!x=1，a=1+0xFFFFFFFF=0。  对于b，令b=~(!x) + 1，当x != 0时，!x=0，b=~0+1=0xFFFFFFFF+1=0;当x==0时，!x=1，b=~1+1=0xFFFFFFFF。  3）代码实现：    图表 8 conditional函数代码  8、int isLessOrEqual(int x, int y)  1）题目描述：   * 如果x<=y返回1否则返回0 * 例子: isLessOrEqual(4,5) = 1. * 允许的操作符: ! ~ & ^ | + << >> * 最多操作符数目: 24   2）思路：  令y与x相减，检查相减结果的符号位，若符号位为1，说明y-x<0即x>y，不满足条件，返回0；若符号位为0，说明y-x>=0即x<=y，满足条件，返回1。  判断符号位是否为1，只需要令相减结果与0x80000000相与，若结果不为0，说明符号位为1；相反，则符号位为0。  3）代码实现：    图表 9 isLessOrEqual函数代码  9、int logicalNeg(int x)  1）题目描述：   * 实现！运算符的功能 * 例子: logicalNeg(3) = 0, logicalNeg(0) = 1. * 允许的操作符: ~ & ^ | + << >> * 最多操作符数目: 12   2）思路：  通过取相反数进行非零判断。令y=~x+1（y=-x）并讨论x与y的符号位，有如下几种情况：  a. 当x为0时，两者符号位都为0。  b. 当x=0x8000 0000时，两者符号位都为1。  c. 当x既不为0也不为0x8000 0000时，两者符号位为01或10。  不难发现，令，若x为0，的符号位为0，则z的符号位为1；其他情况下，的符号位为1，则z的符号位为0。因此，若z符号位为1，将z左移31位，取第0位返回，结果为1；若z符号位为0，将z左移31位，取第0位返回，结果为0。  3）代码实现：    图表 10 logicalNeg函数代码  10、int howManyBits(int x)  1）题目描述：   * 返回将X表示为补码所需的最小有效位数。 * 例子: howManyBits(12) = 5 * howManyBits(298) = 10 * howManyBits(-5) = 4 * howManyBits(0) = 1 * howManyBits(-1) = 1 * howManyBits(0x80000000) = 32 * 允许的操作符: ! ~ & ^ | + << >> * 最多操作符数目: 90   2）思路：  可以通过对x取值范围判断其最小有效位数。假设最小有效位数为n，则会有以下几种情况：   1. 当n=1时，x>=-1且x<=0。 2. 当n>1且n<31时，x>且x<=或x>且x<=。 3. 当n=32时， x>或x>。   对于以上a、b和c三种情况，通过x与临界值相减检查符号位来判断x是否位于该范围内，若位于范围内，则令结果等于该范围对应的n；  在代码实现中，采用n逐个检验的方式，即n从1开始直到32对x所处范围进行判断，若x位于该范围内，则令结果加上当前的n。  3）代码实现：    图表 11 howManyBits函数代码  11、unsigned floatScale2(unsigned uf)  1）题目描述：   * 以unsinged表示的浮点数二进制的二倍的二进制unsigned型 * 参数和结果都会被作为unsigned返回，但是会表示为二进制的单精度浮点值。 * 允许的操作符: 任何整数或者无符号数操作符包括： ||, &&. also if, while * 最多操作符数目: 30   2）思路：  假设exp表示单精度浮点数uf的23到30位（即浮点数的阶码），frac表示单精度浮点数uf的0到22位（即浮点数的尾数），则会有以下几种情况：   1. 当exp=0xff时，表示uf为无穷大或NAN，直接返回uf即可； 2. 当exp=0时，表示uf为非规格化浮点数。若uf[22]=0，则将frac左移一位即可，即尾数乘二且不产生溢出；若uf[22]=1时，则将exp自增1，然后再将frac左移一位即可，相当于浮点数（假设frac为1XX……XX）从0.1XX……XX×转换为1. XX……X0×。 3. 对于其他情况，将exp自增1，特殊的，当exp==0xff，说明浮点数溢出，令frac=0，返回无穷大。   3）代码实现：    图表 12 floatScale2函数代码  12、int floatFloat2Int(unsigned uf)  1）题目描述：   * floatFloat2Int - 返回浮点参数f的整数部分的位级等效形式（int f）。 * 参数以无符号整数形式传递，但应被解释为单精度浮点值的位级表示。 * 任何超出范围的值（包括NaN和无穷大）应返回0x80000000u。 * 合法操作: 任何整数/无符号操作，包括||、&&。也可以使用if、while * 最大操作数: 30 * 评级: 4   2）思路：  假设exp表示单精度浮点数uf的23到30位（即浮点数的阶码），frac表示单精度浮点数uf的0到22位（即浮点数的尾数），则会有以下几种情况：   1. 当exp-127<0时，即浮点数的大小小于0，则返回0。 2. 当exp-127>=31时，浮点数的大小超出了补码整数的表示范围，于是返回0x80000000u。 3. 当exp-127>=24时，浮点数的整数部分为，再根据浮点数符号位返回带符号的整数部分。 4. 当exp-127<=23时，浮点数的整数部分为，再根据浮点数符号位返回带符号的整数部分。   3）代码实现：    图表 13 floatFloat2Int函数代码  13、unsigned floatPower2(int x)  1）题目描述：   * floatPower2 - 返回表达式2.0^x（2的x次方）的位级等效形式，其中x是任意的32位整数。 * 返回的无符号值应具有与单精度浮点数2.0^x相同的位表示。如果结果太小而无法表示为非规格化数，则返回0。如果太大，则返回+INF。 * 合法操作：任何整数/无符号操作，包括||、&&。也可以使用if、while * 最大操作数：30 * 评级：4   2）思路：  假设exp = x+127表示单精度浮点数的阶码，2.0^x=(-1)^0×(1.0)×2^(exp-127)，答案有以下几种情况：   1. 当exp >= 127，表示阶码超出了浮点数阶码的表示范围，返回INF。 2. 当exp <=0 且 exp >= -22，需要用非规格化浮点数来表示结果，需要凑出其尾数，即0x1 << (22 + exp)。 3. 当exp < -22，结果太小而无法表示为非规格化数，则返回0。 4. 其余情况，exp均在规格化浮点数阶码的表示范围内，则根据阶码返回结果即可，即exp << 23。   3）代码实现：    图表 14 floatPower2函数代码 |
| **四、实验结论及问题：**  **最终测试结果：**    图表 15 最终测试结果  所有函数以及所有测试点测试均通过，说明所有解法均正确。  **实验结论：**  通过位操作对数值进行操作相对于高级语言而言更具挑战性。这种方式需要更深入地理解底层二进制表示和逻辑运算，并且在运算过程中需要考虑更多细节。位操作涉及到底层的逻辑运算，需要对计算机底层的运行机制有更清晰的理解，同时也需要考虑相关的离散数学的数学定理以正确地处理各种情况。  对浮点数进行操作与对补码整数的操作也有很大的不同。在操作浮点数时，需要严格考虑其格式，包括符号位、指数部分和尾数部分，以及不同数值下的表示意义。因此，在处理浮点数时，需要更加小心谨慎，确保操作的正确性和精确性，以避免产生错误的结果。 |

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2024年4月 日 |
| 备注： |