**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： 数学与统计学院**

**专业： 信息与计算科学（数学与计算机实验班）**

**指导教师： 罗胜**

**报告人： 王曦 学号： 2021192010 班级： 21数计**

**实验时间： 2023 年 04 月 01 日**

**实验报告提交时间： 2023 年 04 月 21 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：     或者：    2、根据bits.c中的要求补全以下的函数：  intbitXor(int x, int y);  inttmin(void);  intisTmax(int x);  ntallOddBits(int x);  int negate(int x);  intisAsciiDigit(int x);  int conditional(int x, int y, int z);  intisLessOrEqual(int x, int y);  intlogicalNeg(int x);  inthowManyBits(int x);  unsignedfloat\_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. bitXor – x^y using only – and & 2. 思路: , 而. 3. 代码:   int bitXor(int x, int y) {      return ~(~(x & ~y) & ~(y & ~x));  }  图1: bitXor()函数的实现   1. tmin - return minimum two's complement integer 2. 思路: int的最小值为, 其补码的MSB为1, 其余位为0. 3. 代码:   int tmin(void) {      return 1 << 31;  }  图2: tmin()函数的实现   1. isTmax - returns 1 if x is the maximum, two's complement number, and 0 otherwise 2. 思路: 若x是int的最大值, 则其补码的MSB为0, 其余位为1. 考虑将其转化为0来判断.    1. y = x + 1为int的最小值, 其补码的MSB为1, 其余位为0.    2. x = x + y的补码全为1, 此时对x按位取反即为0.    3. 注意x = -1, 即x的补码全为1时, 会被①②误判为int的最大值. 此时y = x + 1的补码全为0, 则可通过检查!y是否为0判断. 3. 代码:   int isTmax(int x) {      int y = x + 1;      x = x + y;      x = ~x;      y = !y;      x = x + y;      return !x;  }  图3: isTmax()函数的实现   1. allOddBits - return 1 if all odd-numbered bits in word set to 1 2. 思路: 取掩码mask = 0xAAAAAAAA, 则它的二进制表示的奇数位为1, 偶数位为0, 检查x和x & mask是否相等即可. 因不允许使用大常数, 可将AA平移到对应数位后相加. 3. 代码:   int allOddBits(int x) {      int mask = (0xAA << 24) | (0xAA << 16) | (0xAA << 8) | 0xAA;      x = x & mask;      return !(x ^ mask);  }  图4: addOddBits()函数的实现   1. negate - return -x 2. 思路: 负数的补码是对应的正数补码取反 + 1. 3. 代码:   int negate(int x) {      return ~x + 1;  }  图5: negate()函数的实现   1. isAsciiDigit - return 1 if 0x30 <= x <= 0x39 (ASCII codes for characters '0' to '9') 2. 思路:    1. 判断x >= 0x30, 可判断(x – 0x30)的符号位是否为0 .    2. 判断x <= 0x39, 可判断(x – 0x3A)的符号位是否为1, 此处不选减0x39是因为减完后符号位仍为0. 3. 代码:   int isAsciiDigit(int x) {      int lower = ~0x30 + 1;      int upper = ~0x3a + 1;      int sgn1 = x + lower >> 31;      int sgn2 = x + upper >> 31;      return !sgn1 & sgn2;  }  图6: isAsciiDigit()函数的实现   1. conditional - same as x ? y : z 2. 思路:    1. 判断x是否为0, 可用!x将x先变为0或1. 考虑将x变为掩码: x = 0时, mask = !x – 1 = 0xffffffff; x != 0时, mask = !x – 1 = 0x00000000.    2. 返回y时, 需将z置0; 返回z时, 需将y置0. 上述操作可用mask、~mask分别与y和z相与实现. 3. 代码:   int conditional(int x, int y, int z) {      int neg\_one = ~1 + 1;      int mask = !x + neg\_one;      return (mask & y) | (~mask & z);  }  图7: conditional()函数的实现   1. isLessOrEqual - if x <= y then return 1, else return 0 2. 思路: 按x与y的符号位不同分为两种情况.    1. x与y异号: 负数小, 返回x是否为负数即可.    2. x与y同号: 返回(y -x)的符号位取反即可. 3. 代码:   int isLessOrEqual(int x, int y) {      int sgn\_x = x >> 31 & 1;      int sgn\_y = y >> 31 & 1;      int sgn\_xor = sgn\_x ^ sgn\_y;      int neg\_x = ~x + 1;      int y\_minus\_x = y + neg\_x;      int sgn\_y\_minus\_x = y\_minus\_x >> 31 & 1;      return (sgn\_xor & sgn\_x) | (!sgn\_xor & !sgn\_y\_minus\_x);  }  图8: isLessOrEqual()函数的实现   1. logicalNeg - implement the !s operator, using all of the legal operators except ! 2. 思路:    1. 若x != 0且x != 0x8000, 则!x和x的符号位相反.    2. 若x == 0, 则!x和x的符号位都为0.    3. 若x == 0x8000, 则!x和x的符号位都为1.   综上, 只需!x和x的符号位至少有一个为1即可保证x非零.   1. 代码:   int logicalNeg(int x) {      int neg\_x = ~x + 1;      x = x | neg\_x;      return (x >> 31) + 1;  }  图9: logicalNeg()函数的实现   1. howManyBits - return the minimum number of bits required to represent x in two's complement 2. 思路:    1. 若x == 0, 则只需要1 bit.    2. 若x > 0, 设其MSB为第n位, 则只需再加上符号位, 即用(n + 1)可表示.    3. 若x < 0, 需找到其最高的位0的位置. 为简化, 将x取反. 将x的二进制表示按2的幂次的长度分段, 依次检查每一段中是否有1. 如第一次检查x的低16位中是否有1, 若有则x至少需16 bits才可表示, 移除其低16位, 检查接下来的8位. 依次检查16、8、4、2、1位是否有1, 最终答案为各部分的1的个数加上符号位. 3. 代码:   int howManyBits(int x) {      // if x < 0, then nothing happens, else, flip      int sgn = x >> 31;  x = (sgn & ~x) | (~sgn & x);      // check if there are bits in every sections      int b16, b8, b4, b2, b1, b0;      // check the lowest 16 places  b16 = !!(x >> 16) << 4;  x = x >> b16;      // check the lowest 8 places      b8 = !!(x >> 8) << 3;  x = x >> b8;      // check the lowest 4 places      b4 = !!(x >> 4) << 2;  x = x >> b4;      // check the lowest 2 places      b2 = !!(x >> 2) << 1;  x = x >> b2;      // check the lowest place      b1 = !!(x >> 1);  x = x >> b1;      b0 = x;      return b16 + b8 + b4 + b2 + b1 + b0 + 1;  // + 1 for the sign bit  }  图10: howManyBits()函数的实现   1. float\_twice - Return bit-level equivalent of expression 2\*f for floating point argument f 2. 思路: 先按IEEE-754标准定义的浮点数, 分别截取出uf的符号sgn、阶码exp、尾数frac. 按uf是规格化或非规格化分类:    1. uf是非规格化浮点数, 即exp = 0时, frac乘2即可.    2. exp != 0时:       1. 若exp != 255, 则uf是规格化浮点数, exp加1即可. 注意若exp加1后exp变为全1, 则应返回Infinity, 即将frac置为0.       2. 若exp == 255, 则返回NaN即可. 3. 代码:   unsigned float\_twice(unsigned uf) {      unsigned sgn = uf & (0x80 << 24);      unsigned exp = uf & ((0x7f << 24) + (0x80 << 16));      unsigned frac = uf & ((0x7f << 16) + (0xff << 8) + 0xff);      if (!exp) frac = frac << 1;  // Denormalized      else {  // uf < 0          int mask = (0x7f << 24) + (0x80 << 16);          if (exp ^ mask) {  // not all bits are 1 in exp              exp += 0x80 << 16;  // (exp + 1) means (x 2) for Normalized              if (!(exp ^ mask))  // all bits are 1 in exp, then set frac to 0 and return inf                  frac = 0;          }      }      return sgn | exp | frac;  }  图11: float\_twice()函数的实现   1. float\_i2f - Return bit-level equivalent of expression (float) x 2. 思路:    1. 先求得x的符号sgn. 求阶码exp和尾数frac前, 可特判x取特殊值的情况. 若x == 0, 因0的非规格化表示即全0, 故返回x自身即可; 若x == -infinity, 则将exp置为0x9e后返回即可.    2. 若x为负数, 则将其变为正数.    3. 确定小数点的位置pos后, 将x截断至剩下小数位, 并根据e = E + bias求得exp.    4. 求得小数位frac, 注意特判舍入的情况, 若舍入后发生进位, 则更新exp, 并将frac截断. 3. 代码:   unsigned float\_i2f(int x) {      int sgn = x >> 31 & 1;      int exp = 0;      int frac = 0;      if (!x) return x;  // the float expression for 0 (Denormalized) is itself      else if (x == (1 << 31)) exp = 0x9e;      else {          if (sgn) x = -x;  // transform x into a positive number          int pos;  // binary point          for (pos = 30; !(x >> pos); pos--);          x = x << (31 - pos);          exp = pos + 127;  // e = E + bias          int mask = (0x7f << 16) | (0xff << 8) | 0xff;          frac = (x >> 8) & mask;          x = x & 0xff;  // lowest 8 places          int flag = x > 128 || (x == 128 && (frac & 1));  // round to even numbers            frac = frac + flag;          if (frac >> 23) {  // carry              frac = frac & mask;              exp = exp + 1;          }      }      return (sgn << 31) | (exp << 23) | frac;  }  图12: float\_i2f()函数的实现   1. float\_f2i - Return bit-level equivalent of expression (int) f for floating point argument f 2. 思路:    1. 从uf中截取出符号sgn、阶码exp和尾数frac.    2. 判断是否规格化, 是否为特殊值, 是否溢出.    3. 将exp和frac转化为整型的补码, 或上符号位后返回即可. 3. 代码:   int float\_f2i(unsigned uf) {      unsigned sgn = uf & (0x80 << 24);      unsigned exp = (uf & ((0x7f << 24) + (0x80 << 16))) >> 23;      unsigned frac = (uf & ((0x7f << 16) + (0xff << 8) + 0xff)) | (1 << 23);      if (!exp || exp < 127) return 0;  // zero or Denormalized less than 1  else if (exp >= 31 + 127) return 1 << 31;  // out of range      // transform exp and frac into complement      int E = exp - 127;      frac = frac >> 23 - E;  unsigned neg\_frac = ~frac + 1;      unsigned neg\_one = ~1 + 1;      unsigned mask = !sgn + neg\_one;      unsigned neg\_not\_s = ~(!sgn) + 1;      frac = (mask & neg\_frac) | (neg\_not\_s & frac);      return sgn | frac;  }  图13: float\_f2i()函数的实现 |
| **实验结论：**  在root用户下执行下面的命令, 编译并运行test.c.    图14: 编译并运行test.c    图15: 检查结果    观察到各函数的测试都通过, 实验完成. |
| **心得体会：**   1. 本次实验稍有难度, 考察对位运算的熟练掌握和对特殊情况、边界情况的处理. 2. 经过本次实验, 我提高了自身的位运算水平, 加深了对定点数、IEEE754浮点数的理解, 对数值表示带来的程序bug有更清晰的认识. |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  2018年 月 日 |
| 备注： |