**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： 吴嘉楷 学号： 2022150168 班级： 国际班**

**实验时间： 2024年4月1日 至 4月20日**

**实验报告提交时间： 2024年4月15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：     或者：    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. **在 Linux 虚拟机中⽤ Firefox 打开 webvpn.szu.edu.cn，进⼊ Blackboard，下载 .zip⽂件** 2. **解压文件**   将压缩包文件转移到Desktop目录下，然后cd进入Desktop目录，使用unzip datalab-handout-2019-12-16.zip解压指令解压.zip文件，生成应该.tar文件。  使用tar -xvf ./datalab-handout-2019-12-16.tar指令解压缩.tar文件。   1. **换 Ubuntu apt 源 & 安装 make 和 gcc-multilib** 2. cd进入apt所在文件目录，使用vi sources.list命令编辑文件，替换成清华源。 3. 输入apt-get update命令更新apt源 4. 输入$ apt-get install make gcc-multilib命令安装make 5. **进入bits.c ，根据bits.c中的具体要求补全函数** 6. int bitXor(int x,int y)   思路：  异或操作的结果，是让两个操作数数值不同的二进制位变为1。由已有知识可得，A^B = ~A & B | A & ~B。可是，在本题中，我们不能使用或运算，于是，需要利用 与 和 非 运算转化得到 或 运算。由摩根定律可得，a | b = ~(~a & ~b)，于是，我们代入a、b可得：A^B = ~（~（~A & B）& ~（A & ~B））。  代码：  1. int bitXor(int x, int y) {  2. int op1 = ~x & y;  3. int op2 = x & ~y;  4. /\* or \*/  5. return ~(~op1 & ~op2);  6. }   1. int tmin(void)   思路：  当最高位为1，其余位均为0时，将得到一个最小的二进制补码整数。由于编程运行环境视为32-bit操作系统，于是，只需将1左移31位即可。  代码：  1. int tmin(void) {  2. return 1<<31;  3. }   1. int isTmax(int x)   思路：  在32-bit操作系统下，最大的二进制补码数是01111…11，可以看成是最小的二进制补码数10000…00按位取反得到的，也可以看成是其减1得到的。而10000…00自加，结果是0，其他二进制数自加则结果不为0。于是，我们可以通过判断 ~x + ~x 是否为0，来判断x是不是最大的二进制补码数。  此外，在C语言中，！逻辑非运算符，对于任意m不等于0，！m = 0，当m等于0是，！m = 1。因此，当x为最大的二进制补码数时，！（~x + ~x）等于1，否则为0.  但是，0自加后也为0，因此，0xffffffff取反后自加也为0，需要排除此情况！  源代码：  1. int isTmax(int x) {  2. unsigned op = ~x;  3. return !(op+op) & (!!op);  4. }   1. int allOddBits(int x)   思路：  如果一个二进制数奇数位上全为1，那么它与一个偶数位上全为1的二进制数相或，必能得到一个所有数位均为1的二进制数。  而1111…111取反后值为0，且在C语言中，！逻辑非运算符，对于任意m不等于0，！m = 0，当m等于0是，！m = 1。于是，若n为一个数位上全为1的二进制数，则！（~n）= 1，否则！（~n）= 0。  因此，可以让x与0x55555555相或，将结果进行取反~，再进行逻辑非！后的值返回即可。  源代码：  1. int allOddBits(int x) {  2. /\* get 0x55555555 \*/  3. int op = 85 + (85<<8) + (85<<16) + (85<<24);  4. int or\_res = x | op;  5. return !(~or\_res);  6. }  e. int negate(int x)  思路：  要想得到一个数x的相反数，我们可以从补码的角度出发，将x按位取反，再加1即可。  源代码：  1. int negate(int x) {  2. return ~x+1;  3. }  f. int isAsciiDigit(int x)  思路：  由于0x30和0x39在十进制下表示为48、57，因此判断x是否大于等于0x30，可以转化为判断x-48是否大于等于0，判断x是否小于等于0x39，可以转化为判断57-x是否大于等于0。  两数相减操作，可以将减去减数转化为加上减数的相反数，即减数按位取反再加1。如：x-y = x + （-y） = x + （~y+1）。  同时，判断一个数是否大于等于0，可以由二进制数的最高位（符号位）来判断，若最高位为0，则该数为非负数，即大于等于0，若最高位为1，则该数小于0.  源代码：  1. int isAsciiDigit(int x) {  2. int a = (x + (~48+1))>>31;  3. int b = (57 + (~x+1))>>31;  4. return (!a)&(!b);  5. }  g.int conditional(int x,int y,int z)  思路：  条件选择一般与 或 运算有关，x？y：z可以看成当x非0时取y，当x为0时取z。于是，我们需要利用x构造两个操作数a、b，构成（a&y）|（b&z），使得当x非0时，a&y = y，b&z = 0，当x为0时，a&y = 0，b&z = z。  因为0xffffffff & 任意的数m = m，因此，当x非0时，a = 0xffffffff，b = 0，当x为0时，a = 0，b = 0xffffffff。据此，我们构造出a = ！x+（~1+1），b = ~（！x）+1  代码：  1. int conditional(int x, int y, int z) {  2. int negX = !x;  3. return ((negX+(~1+1))&y) | ((~negX + 1)&z);  4. }  h. int isLessOrEqual(int x, int y)  思路：  当x<=y时，y-x>=0，我们可以根据y-x的二进制值的符号位来判断结果。若最高位为0，说明y>=x，否则，y<x。  因此，当我们对最高位进行取反后，可以得到：若y>=x，则取反结果为1，否则，结果为0，直接将此结果作为返回值即可。  而y-x可以转化为y+（-x），进而转化为y+（~x+1）。但是，两数相减需要考虑数据溢出的问题，当x与y符号位不同时，若x为非负数，则y为为负数，此时应返回0，若y为非负数，则x为负数，此时应返回1，即返回值根据x的符号位而定，其余情况不会出现数据溢出问题，按照符号位取反的结果返回即可。  代码：  1. int isLessOrEqual(int x, int y) {  2. int x\_sign = x>>31;  3. int y\_sigh = y>>31;  4. int choice = （x^y）&1;  5. /\* y-x \*/  6. int minus = y+(~x+1);  7. int highest = minus>>31;  8. return choice&x\_sign | (~choice)&(!highest);  9. }  i. int logicalNeg(int x)  思路：  等于！取反操作符，只有当x=0时，！x才能为1，其余情况均为0。那么，我们只需判断x是否为0即可，当x等于0时，返回1，否则返回0。对于0而言，有一个特性：0的相反数等于它本身。  但是，对于0x80000000而言，经过取反再加一后得到的相反数也是它本身。  因此，我们需判断x和x的相反数的符号位是否均为0，若均为0，则x = 0。  代码：  1. int logicalNeg(int x) {  2. int neg = ~x+1;  3. int x\_sign = x>>31;  4. int neg\_sign = neg>>31;  5. return ~(x\_sign|neg\_sign)&1;  6. }  j. int howManyBits(int x)  思路：  要找一个二进制补码所需的最少数位，其实只需要看其最高的有效位即可。分正负数情况讨论：若为正数，找到权值最大的1在第几位；若为负数，找到权值最大的0在第几位，再加上1个符号位即可得到结果。  然而，负数按位取反后，等价与找1的最高位置，从而减少分类操作。  已知双重取反！！可以让一个非0值变为1，让一个0不发生改变。因此，我们可以通过！！来判断一个数是否含有1。  我们每次通过！！判断前半数位是否有1，有则继续再取前半数位的前半高位判断，无则取后半数位的前半高位进行判断。  （注意，记得判断最后一次移位操作之后的数，若为0，说明该二进制数本身为0，无需加1，否则需要加1）  代码：  1. int howManyBits(int x) {  2. int sign = x>>31;  3. /\* negate the negation \*/  4. x = (~sign)&x | sign&(~x);  5. int top16 = (!!(x>>16))<< 4;  6. x = x>>top16;  7. int top8 = (!!(x>>8)) << 3;  8. x = x>>top8;  9. int top4 = (!!(x>>4)) << 2;  10. x = x>>top4;  11. int top2 = (!!(x>>2)) << 1;  12. x = x>>top2;  13. int top1 = !!(x>>1) ;  14. x = x>>top1;  15. int end = !!x;  16. return top16+top8+top4+top2+top1+end + 1;  17. }  k. unsigned floatScale2(unsigned uf)  思路：  在计算机中，32位浮点数的存储主要是按照IEEE 754标准来的，它将32个二进制位划分成3部分，最高位是符号位S，次高8位是偏移后的指数位E，低23位是尾数位M。  于是，我们对uf进行乘2操作，主要是对其中的E指数位进行操作。  在IEEE 754标准中，规定了一些特殊值：  当阶码E全为0时，若尾数M全为0，表示真值0，若尾数M不全为0，表示非规格数；当阶码E全为1，若尾数M全为0，表示无穷大，若尾数M不全为0，表示“NAN”。  我们可以利用类似掩码的方式，提取出符号位S、指数位E、尾数M各自的信息。  当阶码E全为1时，直接原样返回uf即可；  当阶码E全为0时，若尾数M全为0，此时表示真值0，因为0\*2 = 2，所以也是直接返回uf；  当阶码E全为0时，若尾数M不全为0，此时表示非规格数，尾数部分仅仅表示二进制小数部分，不会在前面补1，此时对浮点数进行\*2操作主要是靠尾数左移1位实现的。但是，若尾数最高位为1，则移位后M全为0，存在数据溢出。因此，需要同时让阶码E+1，让32位二进制表示一个规格化数。  当阶码E为其他情况时，先让E加1，然后判断E+1后是否等于255，若是，需要让尾数M置为0，使二进制数表示无穷大。  代码：  1. unsigned floatScale2(unsigned uf) {  2. int mask1 = 0x80000000; //1000 0000 0000...  3. int mask2 = 0x7f800000; //0111 1111 1000...  4. int mask3 = 0x007fffff; //0000 0000 0111...  5. unsigned S = uf & mask1;  6. unsigned E = uf & mask2;  7. unsigned M = uf & mask3;  8. /\* exp = 11111111 \*/  9. if(!(E^mask2))return uf;  10. if(!E){  11. if(!!(M&0x00400000))E = E + 0x00800000;  12. M = M<<1;  13. }  14. else{  15. E = E + 0x00800000;  16. if(!(E^mask2))M = 0;  17. }  18. return S|E|M;  19. }  l. int floatFloat2Int(unsigned uf)  思路：  我们可以利用类似掩码的方式，提取出符号位S、指数位E、尾数M各自的信息。  在IEEE 754标准中，规定了一些特殊值：  当阶码E全为0时，若尾数M全为0，表示真值0，若尾数M不全为0，表示非规格数；当阶码E全为1，若尾数M全为0，表示无穷大，若尾数M不全为0，表示“NAN”。  由于int类型的表示范围十分有限，仅能表示 -231 ~ 231-1，于是，  当阶码E < 127时，浮点数的真值小于1，此时直接返回0；  当阶码E >157时，指数部分大于等于31，此时超出int类型的最大表示范围，于是返回 0x80000000u；  由于尾数部分有23位，规格数情况下转化为浮点数会在前面补1，因此，当十进制指数为23，即E = 150时，此时二进制数无需移位即可表示为int类型二进制数。  当阶码127=< E < 150 时，二进制数需要进行右移才能正确表示int类型的二进制数；  当阶码150< E <= 157时，二进制数需要进行左移。  最后，需要根据符号位S决定是否返回相反数。  代码：  1. int floatFloat2Int(unsigned uf) {  2. int S = uf>>31;  3. int E = (uf>>23) & 0xff;  4. int M = uf & 0x007fffff;  5. int intRes = 0;  6. if(E<127)return 0;  7. if(E>157)return 0x80000000u;  8. if(E<150)intRes = (0x00800000+M)>>(150-E);  9. else intRes = (0x00800000+M)<<(E-150);  10. if(S)return ~intRes + 1;//negate  11. return intRes;  12. }  m. unsigned floatPower2(int x)  思路：  在浮点数中，指数的十进制表示范围为-126 ~ 127，阶码全为1，尾数全为0表示INF无穷大。  于是，当x小于-126时，我们直接返回0，当x大于127时，我们直接返回0x7f800000。  对于其它情况，若要以IEEE 754标准去表示2.0x，需要令23个尾数位全为0，阶数E的值为实际指数值x + 偏移量127。  因此，其他情况下，将x+127的值左移23位留出尾数位，然后返回即可。  代码：  1. unsigned floatPower2(int x) {  2. if(x < -126)return 0;  3. else if(x > 127)return 0x7f800000;  4. else return (x+127)<<23;  5. }  **5. 成绩截图：**    图1 最终成绩  由图可见，在不断的调试、修改之后，终于拿到了满分！ |
| **四、实验结论及问题：**  1. int bitXor(int x, int y)编写错误  解决方案：检查函数得知，在对y进行取反操作时，~y写成了-y，导致函数出错。  2. int isTmax(int x)失分  解决方案：函数实现过程中涉及到加法运算，需要考虑有符号加法还是无符号加法，两者对溢出处理是不同的，此处该使用无符号加法（溢出时对结果取模）。  3. int isLessOrEqual(int x, int y)失分  解决方案：对int类型进行右移操作是算术右移，高位会补符号位，于是对右移的结果异或后可能导致高位上存在1。此时，需要利用&1操作，只保留最低位的数字。  4. int logicalNeg(int x)失分  解决方案：此问题原因与第3个问题类似，是由于算术右移补符号位，导致我们取到的所谓的符号位可能并不是一个有效位为1的数。因此，需要利用&1操作，只保留最低位的数字。  5. unsigned floatScale2(unsigned uf)失分  解决方案：当阶码全为0，M不全为0是，此时让浮点数翻倍需要依靠尾数M左移一位，而不是依靠改变阶码E。但是，当阶码E全为0，尾数M最高位为1时，需要同时做移位和阶码加1的操作。  6. unsigned floatPower2(int x)出错  解决方案：debug发现，结果大多返回了0，说明x在-126到127范围下返回值太小，可能是移位操作有误。检查后发现，其中的左移23位操作写成了右移23位，导致错误产生。 |

|  |
| --- |
| 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： |