

课程实验报告

课 程 名 称： 计算机组成与系统

实验项目名称： datalab-handout实验

专 业 班 级： 软件工程1605

姓 名： 吴多智

学 号： 201626010520

指 导 教 师：

完 成 时 间： 2018 年 5 月 9 日

信息科学与工程学院

|  |  |
| --- | --- |
| 实验题目：datalab-handout | |
| 实验目的：填写 bits.c里面的函数，使其按照规定的要求（比如只能使用有限且规定的操作符和数据类型，不能使用控制语句等等）实现函数的功能。 | |
| 实验环境：Ubuntu16.04.4 x86系统 | |
| 实验内容及操作步骤：  将各函数修改如下：  1. bitAnd函数  int bitAnd(int x, int y) {   return ~((~x)|(~y)); }  //运用了德摩定律，~((~x)|(~y))= (~ (~x))& (~ (~y))=x&y。  2.getbyte函数  int getByte(int x, int n) {   return (x>>(n<<3))&255; }  //要从x中提取一个字节，而字节编号为0~3。一个字节为8位2进制。n<<3即为n\*8位。x>>(n<<3)即为x只保留下除去最后n\*8位剩下的部分。再&255则只保留剩下的最右一个字节。得出的结果便为编号指定要提取的那个字节。  3.logicalshift函数  int logicalShift(int x, int n) {    int mask=~(((1<<31)>>n)<<1);  return mask&(x>>n); }  //~(((1<<31)>>n)<<1)即为232-n-1，也就是mask=(000…011111…1)2（n个0，31-n个1）。再mask&(x>>n)即为将x算术右移n位后并上mask这个前n位为0的掩码，使当x为负数时位移补1换为补0。   1. bitCount函数   int bitCount(int x) {  int count;  int tmpMask1 = (0x55)|(0x55<<8);  int mask1 = (tmpMask1)|(tmpMask1<<16);  int tmpMask2 = (0x33)|(0x33<<8);  int mask2 = (tmpMask2)|(tmpMask2<<16);  int tmpMask3 = (0x0f)|(0x0f<<8);  int mask3 = (tmpMask3)|(tmpMask3<<16);  int mask4 = (0xff)|(0xff<<16);  int mask5 = (0xff)|(0xff<<8);  count = (x&mask1)+((x>>1)&mask1);  count = (count&mask2)+((count>>2)&mask2);  count = (count + (count >> 4)) & mask3;  count = (count + (count >> 8)) & mask4;  count = (count + (count >> 16)) & mask5;  return count;  }  本题采用二分法，先计算x每两位中1的个数，并用对应的两位来储存这个个数。然后计算每四位1的个数，再用对应的四位进行储存。依次类推，最后整合得到16位中1的个数，即为x中1的个数并输出。  5.bang(int x) 函数  /\*  \* bang - Compute !x without using !  \* Examples: bang(3) = 0, bang(0) = 1  \* Legal ops: ~ & ^ | + << >>  \* Max ops: 12  \* Rating: 4  \*/  int bang(int x) {  return (~((x|(~x+1))>>31))&1;  }  6.tmin(void)函数  /\*  \* tmin - return minimum two's complement integer  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 4  \* Rating: 1  \*/  int tmin(void) {  return 1<<31;  }  7.fitsBits(int x, int n) 函数  /\*  \* fitsBits - return 1 if x can be represented as an  \* n-bit, two's complement integer.  \* 1 <= n <= 32  \* Examples: fitsBits(5,3) = 0, fitsBits(-4,3) = 1  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 15  Rating: 2  \*/  int fitsBits(int x, int n) {  int shiftNumber= 32 + (~n + 1);// 32 - n  return !(x^((x<<shiftNumber)>>shiftNumber));  }  8.divpwr2(int x, int n)函数  /\*  \* divpwr2 - Compute x/(2^n), for 0 <= n <= 30  \* Round toward zero  \* Examples: divpwr2(15,1) = 7, divpwr2(-33,4) = -2  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 15  \* Rating: 2  \*/  int divpwr2(int x, int n) {  int signx = x >> 31;  int mask = (1 << n) + (~0);  int bias = signx & mask;  return (x + bias) >> n;  }  9.negate(int x)函数  /\*  \* negate - return -x  \* Example: negate(1) = -1.  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 5  \* Rating: 2  \*/  int negate(int x) {  return (~x) + 1;  }  //~x+1=即-1-x+1=-x。正确。  10.isPositive(int x)函数  /\*  \* isPositive - return 1 if x > 0, return 0 otherwise  \* Example: isPositive(-1) = 0.  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 8  \* Rating: 3  \*/  int isPositive(int x) {  return !((x >> 31) | (!x));  }  11.isLessOrEqual(int x, int y)函数  /\*  \* isLessOrEqual - if x <= y then return 1, else return 0  \* Example: isLessOrEqual(4,5) = 1.  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 24  \* Rating: 3  \*/  int isLessOrEqual(int x, int y) {  int singx = (x >> 31) & 1;  int singy = (y >> 31) & 1; //比较符号位 1 0 = 1, 0 1 = 0;  int sing = (singx ^ singy) & singx; //保证singx和singy异号  int tmp = x + ((~y) + 1); // x - y, 同号情况下,异号情况下会越界 0 0 = , 1 1 =  tmp = ((tmp>>31)&1) & (!(singx ^ singy));// 保证singx 和 singy 同号  //int t = (!(x ^ y)); //判断相等  //printf("sing =%d, tmp = %d\n", sing, tmp);  return (sing | tmp | ((!(x ^ y)))); //  }  12.ilog2(int x)函数  /\*  \* ilog2 - return floor(log base 2 of x), where x > 0  \* Example: ilog2(16) = 4  \* Legal ops: ! ~ & ^ | + << >>  \* Max ops: 90  \* Rating: 4  \*/  int ilog2(int x) {  int bitsNumber=0;  bitsNumber=(!!(x>>16))<<4;//  bitsNumber=bitsNumber+((!!(x>>(bitsNumber+8)))<<3);  bitsNumber=bitsNumber+((!!(x>>(bitsNumber+4)))<<2);  bitsNumber=bitsNumber+((!!(x>>(bitsNumber+2)))<<1);  bitsNumber=bitsNumber+(!!(x>>(bitsNumber+1)));  bitsNumber=bitsNumber+(!!bitsNumber)+(~0)+(!(1^x));  return bitsNumber;  }  本题与bitcout的方法相似，也为二分法。  bitsNumber=(!!(x>>16))<<4即x右移16位后若大于0即得到（10000)2=16,否则得到0，判断最高位是否为0，若不为0，则包含2的16次方。即得到最高位的数.其他同理。  13.unsigned float\_neg(unsigned uf)函数  /\*  \* float\_neg - Return bit-level equivalent of expression -f for  \* floating point argument f.  \* Both the argument and result are passed as unsigned int's, but  \* they are to be interpreted as the bit-level representations of  \* single-precision floating point values.  \* When argument is NaN, return argument.  \* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while  \* Max ops: 10  \* Rating: 2  \*/  unsigned float\_neg(unsigned uf) {  unsigned result;  unsigned tmp;  result=uf ^ 0x80000000; //当uf不是NAN，改变符号位  tmp=uf & (0x7fffffff);//将uf符号位改为0.  if(tmp > 0x7f800000)//比无穷大还大，即NAN。  result = uf;  return result;  }  14.unsigned float\_i2f(int x) 函数  /\*  \* float\_i2f - Return bit-level equivalent of expression (float) x  \* Result is returned as unsigned int, but  \* it is to be interpreted as the bit-level representation of a  \* single-precision floating point values.  \* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while  \* Max ops: 30  \* Rating: 4  \*/  unsigned float\_i2f(int x) {  unsigned shiftLeft=0;  unsigned afterShift, tmp, flag;  unsigned absX=x;  unsigned sign=0;  //special case  if (x==0) return 0;  //if x < 0, sign = 1000...,abs\_x = -x  if (x<0)  {  sign=0x80000000;  absX=-x;  }  afterShift=absX;  //count shift\_left and after\_shift  while (1)  {  tmp=afterShift;  afterShift<<=1;  shiftLeft++;  if (tmp & 0x80000000) break;  }  if ((afterShift & 0x01ff)>0x0100)  flag=1;  else if ((afterShift & 0x03ff)==0x0300)  flag=1;  else  flag=0;    return sign + (afterShift>>9) + ((159-shiftLeft)<<23) + flag;  }  15.unsigned float\_twice(unsigned uf) 函数  /\*  \* float\_twice - Return bit-level equivalent of expression 2\*f for  \* floating point argument f.  \* Both the argument and result are passed as unsigned int's, but  \* they are to be interpreted as the bit-level representation of  \* single-precision floating point values.  \* When argument is NaN, return argument  \* Legal ops: Any integer/unsigned operations incl. ||, &&. also if, while  \* Max ops: 30  \* Rating: 4  \*/  unsigned float\_twice(unsigned uf) {  unsigned f = uf;  if ((f & 0x7F800000) == 0)  {  f = ((f & 0x007FFFFF) << 1) | (0x80000000 & f);  }  else if ((f & 0x7F800000) != 0x7F800000)  {  f =f + 0x00800000;  }  return f;  }  实验结果及分析：    IMG_256  编译顺利通过了，当然最后两个确实不是很懂，借鉴了网上的代码。检验通过了，具体操作步骤、rating等都低于上限。  收获与体会：这个实验花费时间挺多的，也很有趣，本来以为挺简单的一个c语言嘛，没想到能弄出这么多花样。一门语言果然想要学到精通掌握还需要大量的练习。 | |
| 实  验成绩 |  |