```
/*
 * CS:APP Data Lab
 *
 * <Please put your name and userid here>
 *
 * bits.c - Source file with your solutions to the Lab.
 *          This is the file you will hand in to your instructor.
 *
 * WARNING: Do not include the <stdio.h> header; it confuses the dlc
 * compiler. You can still use printf for debugging without including
 * <stdio.h>, although you might get a compiler warning. In general,
 * it's not good practice to ignore compiler warnings, but in this
 * case it's OK.
 */

#if 0
/*
 * Instructions to Students:
 *
 * STEP 1: Read the following instructions carefully.
 */

You will provide your solution to the Data Lab by
editing the collection of functions in this source file.

INTEGER CODING RULES:

  Replace the "return" statement in each function with one
  or more lines of C code that implements the function. Your code
  must conform to the following style:

  int Funct(arg1, arg2, ...) {
      /* brief description of how your implementation works */
      int var1 = Expr1;
      ...
      int varM = ExprM;

      varJ = ExprJ;
      ...
      varN = ExprN;
      return ExprR;
  }

  Each "Expr" is an expression using ONLY the following:
  1. Integer constants 0 through 255 (0xFF), inclusive. You are
      not allowed to use big constants such as 0xffffffff.
  2. Function arguments and local variables (no global variables).
  3. Unary integer operations ! ~
  4. Binary integer operations & ^ | + << >>

  Some of the problems restrict the set of allowed operators even further.
  Each "Expr" may consist of multiple operators. You are not restricted to
  one operator per line.

  You are expressly forbidden to:
  1. Use any control constructs such as if, do, while, for, switch, etc.
  2. Define or use any macros.
  3. Define any additional functions in this file.
  4. Call any functions.
  5. Use any other operations, such as &&, ||, -, or ?:
  6. Use any form of casting.
  7. Use any data type other than int.  This implies that you
     cannot use arrays, structs, or unions.


  You may assume that your machine:
  1. Uses 2s complement, 32-bit representations of integers.
  2. Performs right shifts arithmetically.
  3. Has unpredictable behavior when shifting an integer by more
     than the word size.

EXAMPLES OF ACCEPTABLE CODING STYLE:
  /*
   * pow2plus1 - returns 2^x + 1, where 0 <= x <= 31
   */
  int pow2plus1(int x) {
     /* exploit ability of shifts to compute powers of 2 */
     return (1 << x) + 1;
  }

  /*
   * pow2plus4 - returns 2^x + 4, where 0 <= x <= 31
   */
  int pow2plus4(int x) {
     /* exploit ability of shifts to compute powers of 2 */
     int result = (1 << x);
     result += 4;
     return result;
  }

FLOATING POINT CODING RULES

For the problems that require you to implent floating-point operations,
the coding rules are less strict.  You are allowed to use looping and
conditional control.  You are allowed to use both ints and unsigneds.
You can use arbitrary integer and unsigned constants.
You are expressly forbidden to:
  1. Define or use any macros.
  2. Define any additional functions in this file.
  3. Call any functions.
  4. Use any form of casting.
  5. Use any data type other than int or unsigned.  This means that you
     cannot use arrays, structs, or unions.
  6. Use any floating point data types, operations, or constants.


NOTES:
  1. Use the dlc (data lab checker) compiler (described in the handout) to
     check the legality of your solutions.
  2. Each function has a maximum number of operators (! ~ & ^ | + << >>)
     that you are allowed to use for your implementation of the function.
     The max operator count is checked by dlc. Note that '=' is not
     counted; you may use as many of these as you want without penalty.
  3. Use the btest test harness to check your functions for correctness.
  4. Use the BDD checker to formally verify your functions
  5. The maximum number of ops for each function is given in the
     header comment for each function. If there are any inconsistencies
     between the maximum ops in the writeup and in this file, consider
     this file the authoritative source.

/*
 * STEP 2: Modify the following functions according the coding rules.
 *
 *   IMPORTANT. TO AVOID GRADING SURPRISES:
 *   1. Use the dlc compiler to check that your solutions conform
 *      to the coding rules.
 *   2. Use the BDD checker to formally verify that your solutions produce
 *      the correct answers.
 */


#endif
/*
 * bitParity - returns 1 if x contains an odd number of 0's
 *   Examples: bitParity(5) = 0, bitParity(7) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 20
 *   Rating: 4
 */
int bitParity(int x) {
  /* Work things down.  At any time, upper part of words will
     contain junk.  Mask this off at the very end
  */
 int wd16 = x ^ x>>16; /* Combine into 16 bits */
 int wd8 = wd16 ^ wd16>>8; /* Combine into 8 bits */
```

```
  int wd4 = wd8 ^ wd8>>4;
  int wd2 = wd4 ^ wd4>>2;
  int bit = (wd2 ^ wd2>>1) & 0x1;
  return bit;
 }
 /*
  * rotateRight - Rotate x to the right by n
  *   Can assume that 0 <= n <= 31
  *   Examples: rotateRight(0x87654321,4) = 0x76543218
  *   Legal ops: ~ & ^ | + << >> !
  *   Max ops: 25
  *   Rating: 3
  */
 int rotateRight(int x, int n) {
     /* Create mask for n = 0 */
     int zmask = (~!n)+1;
     int lsval = 33+~n;
     /* Shift left by 32-n */
     int left = x << lsval;
     /* Arithmetic shift right by n */
     int right = x >> n;
     /* Mask off upper 1's */
     int lmask = ~0 << lsval;
     right &= ~lmask;
     return (zmask&x) | (~zmask&(left|right));
 }
 /*
  * byteSwap - swaps the nth byte and the mth byte
  *  Examples: byteSwap(0x12345678, 1, 3) = 0x56341278
  *            byteSwap(0xDEADBEEF, 0, 2) = 0xDEEFBEAD
  *  You may assume that 0 <= n <= 3, 0 <= m <= 3
  *  Legal ops: ! ~ & ^ | + << >>
  *  Max ops: 25
  *  Rating: 2
  */
 int byteSwap(int x, int n, int m) {
     int n8 = n << 3;
     int m8 = m << 3;
     int n_mask = 0xff << n8;
     int m_mask = 0xff << m8;
     int n_byte = (( x & n_mask ) >> n8) & 0xff;
     int m_byte = (( x & m_mask ) >> m8) & 0xff;
     int bytes_mask = n_mask | m_mask;
     int left_over = x & ~bytes_mask;
     return left_over | (n_byte << m8) | (m_byte << n8);
 }
 /*
  * fitsShort - return 1 if x can be represented as a
  *   16-bit, two's complement integer.
  *   Examples: fitsShort(33000) = 0, fitsShort(-32768) = 1
  *   Legal ops: ! ~ & ^ | + << >>
  *   Max ops: 8
  *   Rating: 1
  */
 int fitsShort(int x) {
   int shift1 = x >> 15;
   int shift2 = x >> 16;
   return !(shift1 ^ shift2);
 }
 /*
  * bitAnd - x&y using only ~ and |
  *   Example: bitAnd(6, 5) = 4
  *   Legal ops: ~ |
  *   Max ops: 8
  *   Rating: 1
  */
 int bitAnd(int x, int y) {
   return ~(~x | ~y);
 }
 /*
  * subOK - Determine if can compute x-y without overflow
  *   Example: subOK(0x80000000,0x80000000) = 1,
  *            subOK(0x80000000,0x70000000) = 0,
  *   Legal ops: ! ~ & ^ | + << >>
  *   Max ops: 20
  *   Rating: 3
  */
 int subOK(int x, int y) {
   int diff = x+~y+1;
   int x_neg = x>>31;
   int y_neg = y>>31;
   int d_neg = diff>>31;
   /* Overflow when x and y have opposite sign, and d different from x */
   return !(~(x_neg ^ ~y_neg) & (x_neg ^ d_neg));
 }
 /*
  * isGreater - if x > y  then return 1, else return 0
  *   Example: isGreater(4,5) = 0, isGreater(5,4) = 1
  *   Legal ops: ! ~ & ^ | + << >>
  *   Max ops: 24
  *   Rating: 3
  */
 int isGreater(int x, int y) {
   int x_neg = x>>31;
   int y_neg = y>>31;
   return !((x_neg & !y_neg) | (!(x_neg ^ y_neg) & (~y+x)>>31));
 }
 /*
```

```
 * 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 shift = 32 + ~n + 1;
  int move = (x << shift) >> shift;
  return !(x ^ move);
}
/*
 * negate - return -x
 *   Example: negate(1) = -1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 5
 *   Rating: 2
 */
int negate(int x) {
  return ~x+1;
}
/*
 * isTmax - returns 1 if x is the maximum, two's complement number,
 *     and 0 otherwise
 *   Legal ops: ! ~ & ^ | +
 *   Max ops: 10
 *   Rating: 1
 */
int isTmax(int x) {
    int nx = ~x;
    int nxnz = !!nx;
    int nxovf = !(nx+nx);
    return nxnz & nxovf;
}
```