ics-lab1

```
/*
 * 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+1)&(~x))>>31)&1;
}

/*
 * bitCount - returns count of number of 1's in word
 *   Examples: bitCount(5) = 2, bitCount(7) = 3
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 40
 *   Rating: 4
 */
int bitCount(int x) {
  int m1 = 0x11|(0x11<<8);
  int m2 = m1|(m1<<16);
  int bitcount = x & m2;
  bitcount += x>>1 & m2;
  bitcount += x>>2 & m2;
  bitcount += x>>3 & m2;

  bitcount = bitcount + (bitcount>>16);
  m2 = 0xF|(0xF<<8);
  bitcount = (bitcount & m2) + ((bitcount>>4) & m2);
  return (bitcount + (bitcount>>8)) & 0x3F;
}

/*
 * copyLSB - set all bits of result to least significant bit of x
 *   Example: copyLSB(5) = 0xFFFFFFFF, copyLSB(6) = 0x00000000
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 5
 *   Rating: 2
 */
int copyLSB(int x) {
  return (~(x&1)+1);
}

/*
 * 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 j = x>>31;
  int m = (1<<n) + (~0);
  return (x + (j & m))>>n;
}

/*
 * evenBits - return word with all even-numbered bits set to 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 8
 *   Rating: 2
 */
int evenBits(void) {
  int mask = 0x55|(0x55<<8);
  mask = mask|(mask<<16);
  return mask;
}

/*
 * 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) {
  return !((((x<<(33+~n))>>(33+~n)))^x);
}

/*
 * getByte - Extract byte n from word x
 *   Bytes numbered from 0 (LSB) to 3 (MSB)
 *   Examples: getByte(0x12345678,1) = 0x56
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 *   Rating: 2
 */
int getByte(int x, int n) {
  return (x>>(n<<3))&0xFF;
}

/*
 * 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 flag = ((x>>31)^(y>>31))&1;
  int tmp = ((y+(~x+1))>>31)&1;
  return ((!flag)&tmp)|(flag&(y>>31));
}

/*
 * isNonNegative - return 1 if x >= 0, return 0 otherwise
 *   Example: isNonNegative(-1) = 0.  isNonNegative(0) = 1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 *   Rating: 3
 */
int isNonNegative(int x) {
  return !(x>>31);
}

/*
 * isNotEqual - return 0 if x == y, and 1 otherwise
 *   Examples: isNotEqual(5,5) = 0, isNotEqual(4,5) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 *   Rating: 2
 */
int isNotEqual(int x, int y) {
  return !(!(x^y));
}

/*
 * isPower2 - returns 1 if x is a power of 2, and 0 otherwise
 *   Examples: isPower2(5) = 0, isPower2(8) = 1, isPower2(0) = 0
 *   Note that no negative number is a power of 2.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 60
 *   Rating: 4
 */
int isPower2(int x) {
  return !((x>>31)|(!x)|((x&(~x+1))^x));
}

/*
 * leastBitPos - return a mask that marks the position of the
 *               least significant 1 bit. If x == 0, return 0
 *   Example: leastBitPos(96) = 0x20
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 *   Rating: 4
 */
int leastBitPos(int x) {
  return (~x+1)&x;
}

/*
 * logicalShift - shift x to the right by n, using a logical shift
```

```
 *   Can assume that 1 <= n <= 31
 *   Examples: logicalShift(0x87654321,4) = 0x08765432
 *   Legal ops: ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int logicalShift(int x, int n) {
  return (~((x>>31)<<31)&(x>>1))>>(n+(~0));
}

/*
 * satAdd - adds two numbers but when positive overflow occurs, returns
 *          maximum possible value, and when negative overflow occurs,
 *          it returns minimum positive value.
 *   Examples: satAdd(0x40000000,0x40000000) = 0x7fffffff
 *             satAdd(0x80000000,0xffffffff) = 0x80000000
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 30
 *   Rating: 4
 */
int satAdd(int x, int y) {
  int sum = x + y;
  int overflow = ((sum^x)&(sum^y))>>31;
  int sum1 = sum&(~overflow);
  int sum2 = overflow&((1<<31)+(sum>>31));
  return sum1|sum2;
}

/*
 * tc2sm - Convert from two's complement to sign-magnitude
 *   where the MSB is the sign bit
 *   You can assume that x > TMin
 *   Example: tc2sm(-5) = 0x80000005.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 4
 */
int tc2sm(int x) {
  return (~(x>>31)&x)|((x>>31)&((1<<31)+(~x+1)));
}
```