ICS2023-datalab

1. bitCount

```c
/*
 * 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) {
}
```

最简单：`x & 1` + `(x >> 1) & 1` + ... + `(x >> 31) & 1`

如何减少操作数？分治的想法！

```c
int bitCount(int x) {
    int mask, s;
    // mask = 0001 0001 0001 0001 0001 0001 0001 0001
    mask = 0x11 | (0x11<<8);
    mask = mask | (mask<<16);
    // 计算每四位的 sum
    s =  x & mask;
    s += x>>1 & mask;
    s += x>>2 & mask;
    s += x>>3 & mask;
    // 现在, s = sum(28,31),...,sum(0,3)
    s = s + (s>>16);
    // s = ...,  sum(12,15) + sum(28,31), ..., sum(0,3) + sum(16,19)
    mask = 0xf | (0xf<<8);
    // mask = 0000 1111 0000 1111
    s = (s & mask) + ((s>>4) & mask);
    return (s + (s>>8)) & 0x3f;
}
```

2.copyLSB

```c
/*
 * 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) {
    // get least significant bit of x
    x = x & 0x1;
    // 有很多种方式将所有bit全设置
    // (1) << 后 >>
    return x << 31 >> 31;
    // (2) 小trick: -0 = 全0, -1=全1
    return ~x + 1;
}
```

3.evenBits

```c
/*
 * evenBits - return word with all even-numbered bits set to 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 8
 *   Rating: 2
 */
int evenBits(void) {
    // return 0x55555555
    int x = 0x55;
    return x+(x<<8)+(x<<16)+(x<<24);
}
```

4. fitBits

```c
/*
 * 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) {
}
```

思路1： x >> (n - 1) 前面应该是全 0 or 全 1

```
int fitsBits(int x, int n) {
  // x >> (n - 1) 并检查是否是全 0 or 全 1
  n = x >> (n+~0);
  // n - 1 =
  //  (1) ~0
  //  (2) n + (-1) = n + (~1 + 1)
  // 全0或全1, !(n) ^ !(~n)才会返回true
  return !(n) ^ !(~n);
}
```

思路2：比较x最后n位的32位扩展的值是不是和x一样即可

```
int fitsBits(int x, int n) {
  // 比较x最后n位的32位扩展的值是不是和x一样即可
  int y, z;
  y = 32 + ~n + 1; // 32 - n
  z = x << y >> y;
  return !(z ^ x);
}
```

5. getByte

```
/*
 * 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) {
    /*x >> (8*n) to move the targeted byte to the last byte, then use a mask to get it*/
```

```
    x = x >> (n << 3);
    return (x & 0xff);
}
```

6. isGreater

```
/*
 * 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) {
    /*judge whether y - x < 0, then handle cases where s overflows*/
    int signx = x >> 31;
    int signy = y >> 31;
    // s = y - x = y + (~x + 1)
    int s = y + (~x + 1);
    // case1: x >= 0, y < 0 则 c1为全f，否则为0
    int c1 = (~signx & signy);
    // case2: other, x<0 并且 y>=0时，判断y-x的符号
    int c2 = (s >> 31) & (~signx | signy);
    return (c1 | c2) & 1;
}
```

7. isNonNegative

```
/*
 * 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) {
    /*get the opposite of the sign*/
    return (~(x >> 31) & 1);
}
```

8. isNotEqual

```c
/*
 * 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) {
    /*x ^ y == 0 only when x == y, then convert other results to 1*/
    return !!(x ^ y);
}
```

9. leastBitPos

```c
/*
 * 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) {
  // 二进制数x为1的最低位在第n位，[0,n-1]为全0，做取反操作[0,n-1]为全1，n为0
  // 再加1，[0,n-1]为全0，n为1，[n+1,63]为和原数一一对应相反
  // 与原数做and即可得到mask
  return x & (~x+1);
}
```

10. logicalShift

```c
/*
 * 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) {
  // first shift x to the right by 1
```

```c
  // then use the mask 0x7fffffff to change the most significant bit of x to 0
  // finally shift x to the right by (n - 1)
  int mask;
  mask = ~(1 << 31);
  x = x >> 1;
  x = x & mask;
  return (x >> (n + ~0));
}
```

11. satAdd

```c
/*
 * satAdd - adds two numbers but when positive overflow occurs, returns
 *          the maximum value, and when negative overflow occurs,
 *          it returns the minimum value.
 *   Examples: satAdd(0x40000000,0x40000000) = 0x7fffffff
 *             satAdd(0x80000000,0xffffffff) = 0x80000000
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 30
 *   Rating: 4
 */
int satAdd(int x, int y) {
  // if (x > 0 && y > 0 && x + y < 0) || (x < 0 && y < 0 && x + y > 0), overflow
  // - if notOverflow = 0xffffffff, then return s;
  // - if notOverflow is 0, then check sign of x;
  //   + if signx = 0xffffffff, then return 0x80000000(1 << 31)
  //   + if signx = 0, then return 0x7fffffff(~(1 << 31))
  int signx, signy, sum, signs, notOverflow;
  signx = x >> 31;
  signy = y >> 31;
  sum = x + y;
  signs = sum >> 31;
  // 如果overflow，那么signx, signy, ~signs三个符号是相同的
  notOverflow = ~((signx & signy & ~signs) | (~signx & ~signy & signs));
  sum = (sum & notOverflow) | (~notOverflow & ((~signx & ~(1 << 31)) | (signx & (1 << 31))));
  return s;
}
```

12. howManyBits

```c
/* howManyBits - return the minimum number of bits required to represent x in
```

```
 *               two's complement
 *  Examples: howManyBits(12) = 5
 *            howManyBits(298) = 10
 *            howManyBits(-5) = 4
 *            howManyBits(0)  = 1
 *            howManyBits(-1) = 1
 *            howManyBits(0x80000000) = 32
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 90
 *  Rating: 4
 */
int howManyBits(int x) {
  int b16, b8, b4, b2, b1, b0, sign;
  sign = x >> 31;
  x =  (~sign & x) | (sign & ~x);
  // 如果x为正则不变，否则按位取反（这样好找最高位为1的，原来是最高位为0的，这样也将符号位去掉了）
  // 转化为寻找最高为位为1的
  b16 = !!(x >> 16) << 4; //当前数（32位）高16位是否有1
  x = x >> b16; //如果有（至少需要16位），则将原数右移16位；否则原数不变（b16=0）
  b8 = !!(x >> 8) << 3; //当前数（16位）的高8位是否有1
  x = x >> b8; //如果有（当前数至少需要8位），则右移8位；否则当前数不变（b8=0）
  b4 = !!(x >> 4) << 2; //当前数（8位）的高4位是否有1
  x = x >> b4;
  b2 = !!(x >> 2) << 1; //当前数（4位）的高2位是否有1
  x = x >> b2;
  b1 = !!(x >> 1); // 当前数（2位）的高位是否为1
  x = x >> b1;
  b0 = x; // 当前数（1位）是否为1
  return b16 + b8 + b4 + b2 + b1 + b0 + 1;//+1表示加上符号位
}
```

13. logicalNeg

```
/*
 * logicalNeg - implement the ! operator, using all of
 *              the legal operators except !
 *   Examples: logicalNeg(3) = 0, logicalNeg(0) = 1
 *   Legal ops: ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 4
 */
// 注意到 0 == -0，除 0 外的所有数与其相反数至少有一个负数（-2147483648 特殊考虑一下，是差不多的）
int logicalNeg(int x) {
```

```c
        return ((x | (~x + 1)) >> 31) + 1;
}
```

14. dividePower2

```c
/*
 * dividePower2 - Compute x/(2^n), for 0 <= n <= 30
 *  Round toward zero
 *   Examples: dividePower2(15,1) = 7, dividePower2(-33,4) = -2
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 3
 */
// 由于 >> 是向下取整而并非向 0 取整，正数向下取整即是向 0 取整，负数向上取整才是向 0 取整
// 所以对于负数要加上 2^n-1 再 >>
int dividePower2(int x, int n) {
        int sign = x >> 31;
        int bias = sign & ((1 << n) + (~0));
        return (x + bias) >> n;
}
```

15. bang

```c
/*
 * bang - Convert from two's complement to sign-magnitude
 *   where the MSB is the sign bit
 *   You can assume that x > TMin
 *   Example: bang(-5) = 0x80000005.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 4
 */
// 正数什么都不用做，负数需要得到绝对值（取反加一）和符号位（1<<31）
int bang(int x) {
    int sign = x >> 31;
    return (x ^ sign) + (((1 << 31) + 1) & sign);
}
```

原码(Sign-Magnitude)：最高有效位是符号位，用来确定剩下的位应该取负权还是正权：

$$B2S_w(\vec{x}) \doteq (-1)^{x_{w-1}} \cdot \left(\sum_{i=0}^{w-2} x_i 2^i \right)$$