1. **bitXor**

- 代码

```
/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
  return ~(~(~x & y) & ~(x & ~y));
}
```

- 分析

$$a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$$

由摩根公式有

$$\neg(P \wedge Q) \Leftrightarrow (\neg P) \vee (\neg Q)$$
$$\neg(P \vee Q) \Leftrightarrow (\neg P) \wedge (\neg Q)$$

则

$$\begin{aligned} a \oplus b &= (\neg a \wedge b) \vee (a \wedge \neg b) \\ &= \neg\neg((\neg a \wedge b) \vee (a \wedge \neg b)) \\ &= \neg(\neg(\neg a \wedge b) \wedge \neg(a \wedge \neg b)) \end{aligned}$$

2. **tmin**

- 代码

```
/*
 * tmin - return minimum two's complement integer
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 4
 *   Rating: 1
 */
int tmin(void) {
  int a = (1 << 31);
  return a;

}
```

- 分析

  补码（two's-complement）：在补码的定义中，字的最高有效位权值为负。设现在有二进制序列 $\overrightarrow{x}$，长度为 w，则其补码表示为（B2T, Binary to Two's-complement）：

  $$B2T_w(\overrightarrow{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

  int 类型有 32 位，只需令最高位为 1 即可（`1 << 31`）。

3. isTmax

- 代码

```
/*
 * 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 a = x + 1;
  int b = ~x;
  int c = x + 2;

  // 若 x + 1 = ~x, 则 !(a ^ b) = 1, 否则为 0.
  // 若 x = -1, 则 c = x + 2 = 1.
  // 若 c = 1 则 !(!(c ^ 1)) = 0, 否则为 1.
  return !(a ^ b) & !(!(c ^ 1));
}
```

- 分析

  记 int 的最大值为 `MAX_INT (= 0x7FFFFFFF)`，它具有如下性质：

  `MAX_INT + 1 == ~MAX_INT`

  除了 MAX_INT 外，只有 `-1 (= 0xFFFFFFFF)` 具有同样的性质，故还需要排除掉 `x = -1` 的情况。

- 模板

```
!(a ^ b) //若 a == b, 返回 1, 否则返回 0.
```

4. **allOddBits**

- 代码

```
/*
 * allOddBits - return 1 if all odd-numbered bits in
word set to 1
 *   where bits are numbered from 0 (least
significant) to 31 (most significant)
 *   Examples allOddBits(0xFFFFFFFD) = 0,
allOddBits(0xAAAAAAAA) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 2
 */
int allOddBits(int x) {
/* 这种做法用到的操作数超出限制，所以后面精简了一下
  int a = !((x & 170) ^ 170);
  int b = !(((x>>8) & 170) ^ 170);
  int c = !(((x>>16) & 170) ^ 170);
  int d = !(((x>>24) & 170) ^ 170);

  return a & b & c & d;
*/
  int a = 170;
  a = (a << 8) + 170;
  a = (a << 8) + 170;
  a = (a << 8) + 170;

  return !((x & a) ^ a);
}
```

- 分析

  把 32-bits 分成 4 块（每块 8-bits）去判断就可以了。

5. **negate**

- 代码

```
/*
 * negate - return -x
 *   Example: negate(1) = -1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 5
 *   Rating: 2
 */
int negate(int x) {
  return (~x + 1);
}
```

- 分析

  补码求相反数：取反加 1。

6. isAsciiDigit

- 代码

```
/*
 * isAsciiDigit - return 1 if 0x30 <= x <= 0x39
(ASCII codes for characters '0' to '9')
 *   Example: isAsciiDigit(0x35) = 1.
 *            isAsciiDigit(0x3a) = 0.
 *            isAsciiDigit(0x05) = 0.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 3
 */
int isAsciiDigit(int x) {
  int nx = (~x + 1); // 取反, 详情见上文.
  int a = 47 + nx;
  int b = 57 + nx;
  return (a >> 31) & (!(b >> 31));
}
```

- 分析

$$0x30 \le x \le 0x39 \rightarrow 0x29 < x \le 0x39 \rightarrow \begin{cases} 0x29 - x < 0 \\ 0x39 - x \ge 0 \end{cases}$$

7. conditional

- 代码

```
/*
 * conditional - same as x ? y : z
 *   Example: conditional(2,4,5) = 4
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int conditional(int x, int y, int z) {
  int a = !x;   // x = 0, a = 1; x != 0, a = 0.

  return ((~a+1)&z) + ((~(!a)+1)&y);
}
```

- 模板

```
int a = 0 or 1;
if(a == 0)
    ~a + 1 = 0;
else if(a == 1)
    ~a + 1 = 0xFFFFFFFF;
```

8. **isLessOrEqual**

- 代码

```
/*
 * 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 a = (x>>31) ^ (y>>31);    // x,y 同号, a = 0;
x,y 异号, a = 1.

  // x,y 异号时的特判: x<0, y>=0, b=1; x>=0, y<0, b=0
  int b = (x>>31) & ((y>>31) ^ 1);
```

```
  int nx = ~x + 1;  // -x
  int c = !((y + nx) >> 31);    // 判断 y - x 的正负

  // a ? b : c
  int ta = !a;
  return ((~ta+1)&c) + ((~(!ta)+1)&b);
}
```

- 分析

  如果 `x, y` 同号，则判断 `y + (-x)` 是否大于等于 0 即可；

  如果 `x, y` 异号，直接 `y - x` 可能会溢出，所以需要特判。

9. **logicalNeg**

- 代码

```
/*
 * 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
 */
int logicalNeg(int x) {
  int nx = ~x + 1;

  return ((x>>31)^1) & ((nx>>31)^(x>>31)^1) & 1;
}
```

- 分析

  若 x 的最高位为 1，则代表它是复数，直接返回 0，`((x>>31)^1)`；

  若 x 取反加 1 后最高位不变，则代表它是 `0` 或 `0x80000000`，其实加入前一条判断就是为了排除掉 `0x80000000` 这种情况。

  还有一个点，C语言右移后会自动在高位补 1，但我们关注的只有最低的那一位，故还需要在最后 `&1` 。

10. **howManyBits**

- 代码

```
/* 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) {
  // 若 x 为负数, 则将其各位取反; 否则不变.
  x = (~((x >> 31) & 1) + 1) ^ x;

  int temp, ans = 0;

  // 分治
  // 考虑 x 的高 16 位. 若 x 的高 16 位不为 0, 则 ans+16,
x>>16; 否则不动.
  temp = (!!(x >> 16)) << 4;
  ans = ans + temp;
  x = x >> temp;

  // 考虑 x 的高 8 位...
  temp = (!!(x >> 8)) << 3;
  ans = ans + temp;
  x = x >> temp;

  temp = (!!(x >> 4)) << 2;
  ans = ans + temp;
  x = x >> temp;

  temp = (!!(x >> 2)) << 1;
  ans = ans + temp;
  x = x >> temp;

  temp = !!(x >> 1);
  ans = ans + temp;
```

```
  x = x >> temp;

  ans = ans + x;

  return ans + 1;
}
```

- 分析

  对于正数，找最高位的 1。假设最高位的 1 位于第 n 位，则答案为 `n+1`（还需要在前面加个 '0'）。

  对于负数，找最高位的 0。假设最高位的 0 位于第 n 位，则答案为 `n+1`（还需要在前面加个 '1'）。事实上，任何负数的补码都可以表示成 $0b10.....$ 的形式，无论多少个前导 1 都可以合并成 1 个，即 $0b1110..... = 0b10.....$。那么我们只需要把负数的各位取反，便可与正数一致处理了。

- 参考

  https://zhuanlan.zhihu.com/p/59534845?utm_source=qq

11. **floatScale2**

- 提示

  下面的内容请结合 "csapp 2.4 浮点数" 食用。

- 代码

```
/*
 * floatScale2 - 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 floatScale2(unsigned uf) {
  // 先把 s, exp, frac 字段解析出来
```

```
  unsigned mask_exp = 255;
  unsigned mask_frac = (1<<23) - 1;
  unsigned s = (uf>>31) & 1;
  unsigned exp = (uf>>23) & mask_exp;
  unsigned frac = uf & mask_frac;

  // 阶码域全 0，直接将尾数乘 2
  if(!exp) {
    frac = frac << 1;

    return (s<<31) | (frac);
  }

  // 阶码域全 1，该浮点数为 NaN 或无穷大，直接返回
  if(exp == mask_exp)
    return uf;

  // 上面两种情况都是非规格化的值，现在考虑规格化的值
  // 直接将阶码域加 1 即可
  exp = (exp + 1) & mask_exp;
  uf = (s<<31) | (exp<<23) | (frac);

  // 如果阶码域变成全 1，则要把它当成无穷大返回，将尾数置 0
  if(exp == mask_exp)
    uf = uf & (~mask_frac);

  return uf;
}
```

12. **floatFloat2Int**

- 代码

```
/*
 * floatFloat2Int - Return bit-level equivalent of
expression (int) f
 *   for floating point argument f.
 *   Argument is passed as unsigned int, but
 *   it is to be interpreted as the bit-level
representation of a
 *   single-precision floating point value.
```

```
 *   Anything out of range (including NaN and
infinity) should return
 *   0x80000000u.
 *   Legal ops: Any integer/unsigned operations
incl. ||, &&. also if, while
 *   Max ops: 30
 *   Rating: 4
 */
int floatFloat2Int(unsigned uf) {
  // 先把 s, exp, frac 字段解析出来
  int mask_exp = 255;
  int mask_frac = (1<<23) - 1;
  int s = (uf>>31) & 1;
  int exp = (uf>>23) & mask_exp;
  // 阶码 E
  int exp_ = exp - 127;
  int frac = uf & mask_frac;

  // 若阶码大于等于 31, 说明该数大于等于 2^31, 超出了 int 的
范围
  // 这里顺便处理了阶码域全为 1 的情况
  if(exp_ >= 31)
    return 0x80000000u;

  // 若阶码小于 0, 说明该数大于 -1 且小于 1, 转成 int 型为
0
  if(exp_ < 0)
    return 0;

  // 若阶码等于 0, 则返回 1(注意正负)
  if(exp_ == 0) {
    if(s)
      return -1;
    return 1;
  }

  // frac 域占 23 位
  // 若阶码大于等于 23, 则说明转成 int 型后 frac 上所有的位
都在小数点的左边
  if(exp_ >= 23)
    frac <<= (exp_ - 23); // 可能还需要左移
  else
    frac >>= (23 - exp_); // 否则需要右移
```

```
  // 超出 int 的范围
  if(frac >> 31)
    return 0x80000000u;

  // 负数
  if(s)
    frac = ~frac + 1;
  return frac;
}
```

13. **floatPower2**

- 代码

```
/*
 * floatPower2 - Return bit-level equivalent of the
expression 2.0^x
 *   (2.0 raised to the power x) for any 32-bit
integer x.
 *
 *   The unsigned value that is returned should have
the identical bit
 *   representation as the single-precision
floating-point number 2.0^x.
 *   If the result is too small to be represented as
a denorm, return
 *   0. If too large, return +INF.
 *
 *   Legal ops: Any integer/unsigned operations
incl. ||, &&. Also if, while
 *   Max ops: 30
 *   Rating: 4
 */
unsigned floatPower2(int x) {
    unsigned ans;

    // 规格化的值，直接操作阶码域即可
    if(x <= 127 && x >= -126) {
        ans = (x+127)<<23;
        return ans;
    }
```

```
    // If too large, return +INF.
    if(x > 127) {
        ans = 255<<23;
        return ans;
    }

    // If the result is too small to be represented
as a denorm, return 0.
    return 0;
}
```

- 注意

  这一题的数据太大了，有的电脑可以过，有的电脑可能会超时，如果超时可以 `./btest -T 20` 手工增大中断时限。