bitXor 函数

要求：

```
/*
 * 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 2;
}
```

思路：

按题目给的例子来说，首先上方例子的二进制表示如下：

```
0x4 <x>   :   00000100
0x5 <y>   :   00000101
0x1 <x^y> :   00000001
```

其次，做**按位异或**运算，也就是说同位置上相同为 0，不同为 1

而且我们只能使用**按位取反运算符**和**按位与运算符**，下方直接放步骤了

```
x & y                  :   00000100
~x                     :   11111011
~y                     :   11111010
~x & ~y                :   11111010
~(x & y)               :   11111011
~(~x & ~y)             :   00000101
~(x & y) & ~(~x & ~y)  :   00000001
```

这时可以看到只要最后一位都是 1 的话，那么进行按位与运算就行算出来两个数的按位异或运算后的结果了

也就是说，利用原数间的按位与运算来标记出两数都为 1 的位置；利用原数取反值的按位与运算来标记出两数都为 0 的位置

那么最后剩下不同的位就是两个数的不同位，且剩下的不同位上的值肯定是 0

所以再分别做一次取反，就能把两个运算结果上都是 0 的地方变成 1

再做按位与运算就行算出按位异或运算的值了

题解：

```
int bitXor(int x, int y) {
  return ~(x & y) & ~(~x & ~y);
}
```

tmin 函数

要求：

求最小补码值

```
/*
 * tmin - return minimum two's complement integer
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 4
 *   Rating: 1
 */
int tmin(void) {
  return 2;
}
```

思路：

在 int 类型下，补码最小值的二进制表示为 1000 0000 0000 0000 0000 0000 0000 0000

对应的十六进制为 0x8000 0000，也就是 -2147483648

题解：

```
int tmin(void) {
  return 1 << 31;
}
```

isTmax 函数

要求：

求参数是否为最大补码值

```
/*
 * 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) {
  return 2;
}
```

思路：

int 类型的最大补码值是 0x7fffffff

如果可以使用 << 位运算符的话，就可以这么写 `return !(x ^ ~(1 << 31));`

但是这里限制了，因为 `0x7fffffff + 1 == 0x80000000` 且 `0x80000000 * 2 == 0`

那么我们可以判断 `x + x + 2` 的结果是不是 0：如果是 0，那么就满足 x 的值为 0x7fffffff

但是这样又有一个特例 `-1`，我们可以靠 `!(~x)` 来检测这个数是否为 -1

因为 `~(-1) == 0`，所以如果写成 `!(x + x + 2 | !(~x))` 就可以检测 x 是否为 -1 了

那么这个式子就是完整的式子了，不过鉴于 gcc 给的代码样式的建议：

```
bits.c:166:18: warning: suggest parentheses around arithmetic in operand of
‘|’ [-Wparentheses]
   return !(x + x + 2 | !(~x));
```

最后就多加了一下括号

题解：

```
int isTmax(int x) {
  return !((x + x + 2) | !(~x));
}
```

allOddBits 函数

要求：

如果所有奇数位都为 1 则返回 1；否则返回 0

```
/*
 * 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) {
  return 2;
}
```

思路：

```
0xFFFFFFFD <x>       :   11111111111111111111111111111101
0x55555555           :   01010101010101010101010101010101
x | 0x55555555       :   11111111111111111111111111111101
~(x | 0x55555555)    :   00000000000000000000000000000010
!(~(x | 0x55555555)) :   00000000000000000000000000000000
```

既然 0xAAAAAAAA 是所有奇数位都为 1 且偶数位都为 0 的数，且我们就要求测试一个数是不是奇数位都为 1

那么我们可以用 0xAAAAAAAA 的冤家：0x55555555

这个数的奇数位都为 0 且偶数位都为 1，那么我们可以和 x 做按位或运算来得到一个二进制序列

因为 0x55555555 的奇数位上都为 0，那么这个方法就可以测试 x 的奇数位上是不是都为 1

如果 x 上有一个奇数位不为 1，那么对应结果上的位上面的值就是 0

如果 x 上的奇数位都为 1，那么对应的结果必定是 -1

为了满足函数要求，x 上的奇数位都为 1 的话，就返回 1

那么就需要先用按位异或，如果最终结果为 -1 (x 上的奇数位都为 1)，那么结果就变成 0；反之结果就不是 0

最后再用逻辑异或将返回值改为 1 和 0：x 上的奇数位都为 1 的时候返回值为 1

按道理这样就能完整的表达这个函数了，但是实验要求常量不能大于 0xff，所以要改一下

题解：

```
int allOddBits(int x) {
  return !(~(x | (0x55 + (0x55 << 8) + (0x55 << 16) + (0x55 << 24))));
}
```

negate 函数

要求：

求一个数的负数

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

思路：

这个比较简单，一个数的取反数等于它本身的负数 -1

题解：

```
int negate(int x) {
  return ~x + 1;
}
```

isAsciiDigit 函数

要求：

求 x 的值转化为字符时是否等于 0 ~ 9

```
/*
 * 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) {
  return 2;
}
```

思路：

可以用 `x - 0x30 >= 0` 和 `x - 0x3a < 0` 来计算

这里就以 `x - 0x30 >= 0` 来举例：

`~0x30 + 1` 的意思就是取 0x30 的负数

`(x + ~0x30 + 1) >> 31` 的意思就是看 x 会不会小于 0x30：如果小于 0x30，则结果一直等于 -1；反之一直等于 0

我们要的结果是 x 大于 0x30 的时候返回 1，所以在前面需要加个逻辑取反运算符

`x - 0x3a < 0` 同理，就是最后要多一次逻辑取反运算

题解：

```
int isAsciiDigit(int x) {
  return !((x + ~0x30 + 1) >> 31) & !!((x + ~0x3a + 1) >> 31);
}
```

conditional 函数

要求：

实现 `x ? y : z` 三目运算符

```
/*
 * 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) {
  return 2;
}
```

思路：

当 x 不为 0 时，返回值为 y，也就是说式子要为 `0xffffffff & y | 0x0`

当 x 为 0 时，返回值为 z，也就是说式子要为 `0x0 | 0xffffffff & z`

`~!x + 1` 的作用是在 x 不为 0 的时候，将 x 转化为 0；反之将 x 转化为 0xffffffff

`!x - 1` 的作用是在 x 为 0 的时候，将 x 转化为 0xffffffff；反之将 x 转化为 0

因为不能用 `-` 号，所以这里用前面的 negate 函数里的内容来进行负数的转换

最终 `!x - 1` 变为 `!x + ~1 + 1`

题解：

```
int conditional(int x, int y, int z) {
  return ((!x + ~1 + 1) & y) | ((~!x + 1) & z);
}
```

isLessOrEqual 函数

要求：

如果 x<=y 返回 1，否则返回 0

```
/*
 * 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) {
  return 2;
}
```

思路：

~~跟 isAsciiDigit 函数差不多~~是我一开始错误的想法

`(x + ~y + 1) >> 31` 这个式子的意思就是：如果 x >= y，返回 0；反之返回 -1

那么前面加个逻辑取反运算符就能满足题目要求，一开始我是这么想的然后

```
ERROR: Test isLessOrEqual(-2147483648[0x80000000],0[0x0]) failed...
...Gives 0[0x0]. Should be 1[0x1]
```

看来这种方法并不能检测，直接用 `y - x` 可能会超出 int 的表示范围

所以当 x 与 y 同号时，将其转换为 `p = y - x >= 0`，然后 p 符号位 `(p >> 31)` 为 0 就返回 1；反之则返回 0

异号时，只要 `x >= 0`，就要返回 0，否则返回 1，由 `!!(x >> 31)` 能达到该效果

`(x >> 31) ^ (y >> 31)` 可作为 x y 同号异号的判断，异号的时候值为 1；反之同号时值为 0

题解：

```
int isLessOrEqual(int x, int y) {
  return (!((x >> 31) ^ (y >> 31)) & !((y + ~x + 1) >> 31)) | (!!((x >> 31)
^ (y >> 31)) & !!(x >> 31));
}
```

logicalNeg 函数

要求：

实现逻辑取反运算符

```
/*
 * 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) {
  return 2;
}
```

思路：

题目要求 `x == 0` 的时候返回 1，否则返回 0

因为没有 `!` 供我们使用，所以返回值要变成 1 或者是 0 就必须借助其他运算符

这里我采取利用 `-1` 和 `0` 来分别代表 `0` 和 `1`，也就是说：主要利用 `~` 运算符

利用 `(x | (~x + 1)` 式子可以在 x 不为 0 使，使结果的符号位为 1，也就是说，向右移 31 位后的结果必定是 -1

那么最后再加上 1 就可以使得结果返回 0 了

如果 x 为 0 的话，经过这两个式子结果仍然是 0，加上 1 那么返回值就变为了 1

题解：

```
int logicalNeg(int x) {
  return ((x | (~x + 1)) >> 31) + 1;
}
```

xxx 函数

要求：

```

```

思路：

题解：

```

```

xxx 函数

要求：

```
1
```

思路：

题解：

```
1
```

xxx 函数

要求：

```
1
```

思路：

题解：

```
1
```

xxx 函数

要求：

```
1
```

思路：

题解：

```
1
```

####