Bit Hack #1. Check if the integer is even or odd.

```
if ((x & 1) == 0) {
  x is even
}
else {
  x is odd
}
```

Bit Hack #2. Test if the n-th bit is set = = 1

```
if (x & (1<<n)) {
  n-th bit is set
}
else {
  n-th bit is not set
}
```

Bit Hack #3. Set the n-th bit.

```
y = x | (1<<n)
```

Bit Hack #4. Unset the n-th bit.

```
y = x & ~(1<<n)
```

Bit Hack #5. Toggle the n-th bit. **(if n = = 1 => n = 0 else if n = = 0 => n = 1)**

```
y = x ^ (1<<n)
```

Bit Hack #6. Turn off the rightmost 1-bit. (00001**1**- > 00001**0**)

```
y = x & (x-1)
```

Bit Hack #7. Isolate the rightmost 1-bit.

```
    //  10111100  (x)
  &     01000100  (-x)
        ----------
        00000100
```

```
y = x & (-x)
```

Bit Hack #8. Right propagate the rightmost 1-bit. // 10111100 (x)
| 10111011 (x-1)

10111111

y = x | (x-1)

Bit Hack #9. Isolate the rightmost 0-bit. // 10111100 (x)

01000011 (~x)
& 10111101 (x+1)

00000001

y = ~x & (x+1)

Bit Hack #10. Turn on the rightmost 0-bit. // 10111100 (x)
| 10111101 (x+1)

10111101

y = x | (x+1)

Bit Hack #11. Multiply/ devide by power of 2

0000 0111 << 3 = 0011 1000 –-> 7 * (2^3) = 56
0000 1001 << 4 = 1001 0000 –-> 9 * (2^4) = 144
0001 0100 >> 2 = 0000 0101 –-> 20 / (2^2) = 5
1111 0111 << 2 = 1101 1100 –-> -9 * (2^2) = –36

//**right** -> x * 2
//**left** - > x / 2

Bit Hack #12. Is number a power of 2?

if ((x & (x – 1) = = 0) // true
else // false

Bit Hack #13. Are numbers opposed?

int x, y;

if ((x ^ y) < 0) // true
else // false