Setting a bit of a variable to 1.

- use bitwise OR with a 1

$$\begin{array}{r} ? \\ \text{OR} \quad 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ |\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ |\ 0 \\ \hline 1 \end{array}$$

OR'ing with a zero doesn't change anything

Example: set the rightmost bit of x to 1.

```
int x = ...;
x = x | 1;
```

$$\begin{array}{rr} x: & 0\ldots10100 \\ |\ 1: & 0\ldots00001 \\ \hline & 0\ldots10101 \end{array}$$

0…1010 remain the same; last bit changed to 1

Set the m^{th} bit of x to 1:

```
x = x | (1 << m);  // mask has a 1
                   // in the mth position
```

(1 << m): mask

"Clear" a bit to 0.

- use bitwise AND with a 0.

$$\begin{array}{r} ? \\ \&\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ \&\ 1 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ \&\ 1 \\ \hline 1 \end{array}$$

AND'ing anything with 1 has no change.

Clearing the rightmost bit of x (to 0).

- mask would have to be 11...110

How to create that mask?

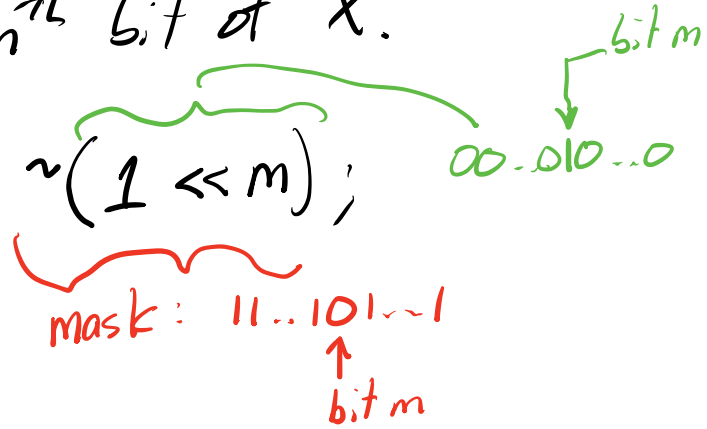
① #define MYMASK 0xFFFFFFFE

generally better → ② #define MYMASK ~1 // complement of
// 000...01

```
x = x & MYMASK;
```

Clear the m^{th} bit of x.

```
x = x & ~(1 << m);
```



Binary Arithmetic

- already seen addition.
- subtraction:

$$x - y \equiv x + (-y)$$

(flip the bits of y and add 1.

- multiplication
  - same way as humans, except in binary not decimal.

Decimal:

```
  ³ ³
   345
    27
  ----
  2415
  690
  ----
  9315
```

Binary:

```
   11010   ← 26 decimal
     110   ← 6 decimal
  ------
       0
   11010
  11010
--------
10011100   ← 156 decimal
```

Division:

- Done the same way as humans do.
  - binary not decimal

Decimal:

```
        129
    _______
56 | 7253
     56
     ---
     165
     112
     ---
      533
      504
      ---
       29
```

129 remainder = 29

Binary:

```
            11110          ← 30 decimal
      __________
101 | 10011010             ← 154 decimal
      101
      -----
      01001
        101
      -----
       01000
         101
         ---
           111
           101
           ---
           100             remainder  ← 4 decimal
```

5 decimal → 101

IEEE Floating Point Numbers

- approximation to real numbers
- 32 bits "float"
- 64 bits "double"

32 bit IEEE floating point numbers:

- specified using scientific notation in binary.

In decimal: 6.02×10^{23}

mantissa

exponent

"normalized" - one digit before the point.

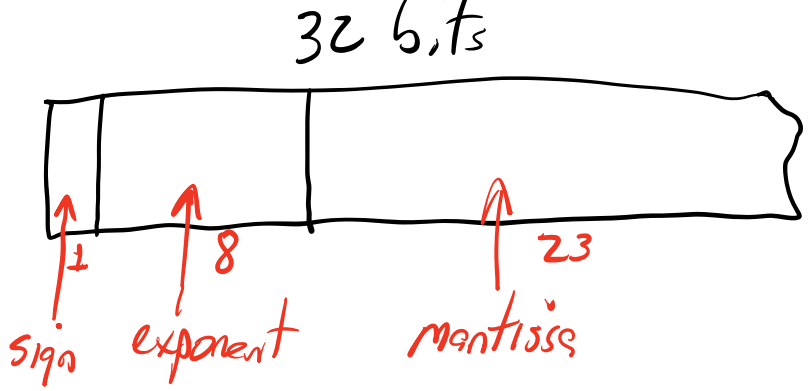
-3.15×10^{15}

2.01×10^{-7}

In binary: 1.0101×2^{10}

sign → -1.11×2^{-101}

The sign, exponent, and the mantissa are represented as fields in the 32-bit number.



Sign: 1 = negative, 0 = positive

Exponent: A bias of 127 is used.

- the number stored in the exponent field is the actual exponent plus 127.
  - so actual exponent is the stored exponent - 127.

If stored exponent is 13, then actual exponent is -114.

Mantissa:

- Must be normalized, so one non-zero digit before the point.

1010.1×2^5

1.0101×2^8

0.1×2^{-3}

1.0×2^{-4}

- there will always be a 1 before the decimal point.
  - except for the number 0.0

- in the stored mantissa, the leading 1 is implicit.

- only the digits after the point are stored.

1.01011010...0

.01011010...0

Zero is represented specially as all 0's (all 32 bits)

000...0

32 bits