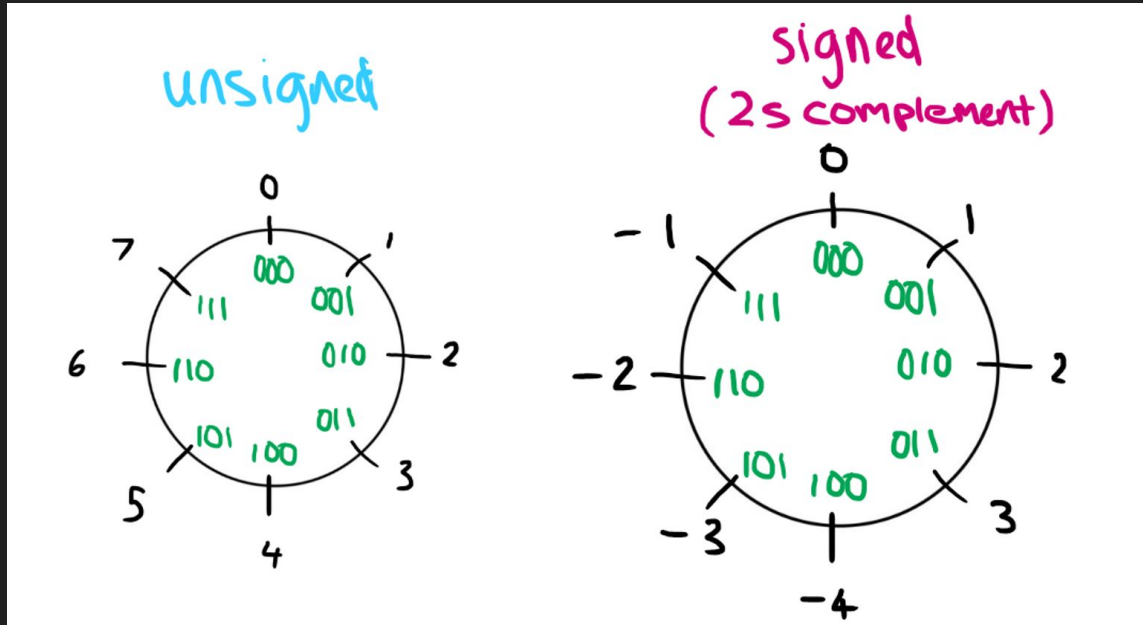
Week 7 ⭕

Negative & floating-point numbers

Negative numbers & two's-complement

- Last week: **unsigned** integers
- This week: **signed** integers
- **Two's complement** is the standard representation for signed integers

An example of a hypothetical 3-bit two's-complement system:



Negative numbers & two’s-complement

Let’s try converting:

- 8-bit two’s-complement → decimal
  - For positive numbers (leftmost bit = 0): as usual
  - For negative numbers (leftmost bit = 1): convert using two’s-complement, then apply the strategy for positive values
  - Spreadsheet
  - What is the range of values you can achieve? (with 8-bits)

```
-  0x13
b: 0001 0011

-  0x64
b: 0110 0100

-  0xff
b: 1111 1111

-  0x80
b: 1000 0000
```

Negative numbers & two’s-complement

Let’s try converting:

- Decimal → 8-bit two’s-complement
  - For positive numbers (leftmost bit = 0): as usual
  - For negative numbers (leftmost bit = 1):
    - We write the positive value as binary
    - We flip the bits
    - We add one to the number
- Convert:
  - 1
  - 127
  - -1
  - -128

Negative numbers & two’s-complement

Positive:

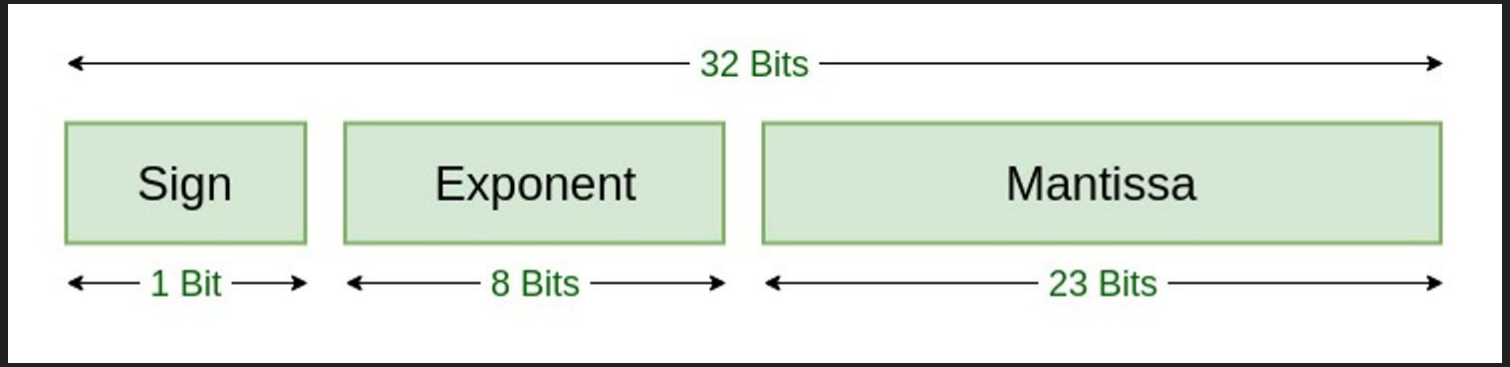
```
1
If positive, it is just the usual binary.
0000 0001

127
If positive, it is just the usual binary.
0111 1111
```

Negative:

```
-128
If negative, we write the positive value as binary.
1000 0000 = 128

Then, we flip the bits of the number.
0111 1111

To finish, we add one to the number to get our solution.
1000 0000

-1
If negative, we write the positive value as binary.
0000 0001

Then, we flip the bits of the number.
1111 1110

To finish, we add one to the binary to get our solution
1111 1111
```

IEEE 754 Floating Point Numbers

- Floating point numbers are decimals
- Represented essentially using scientific notation



$$\text{sign} \times (1 + \text{frac}) \times 2^{\text{exp}-127}$$

IEEE 754 Floating Point Numbers

All values are computed by the formula

$$\textbf{sign} \times (1 + \textbf{frac}) \times 2^{\textbf{exp} - 127}$$

where

- **sign** is 1 if the most significant bit (m.s.b) is 0, or -1 if the m.s.b is 1
- **exp** is determined by the 8 bits following the sign bit (as a value in the range 0..255)
- **frac** is determined by the least significant 23 bits ($\textbf{bit}_{22} \times 2^{-1} + \textbf{bit}_{21} \times 2^{-2} + \cdots + \textbf{bit}_{1} \times 2^{-22} + \textbf{bit}_{0} \times 2^{-23}$)

IEEE 754 Floating Point Numbers

There are some special cases (important for today's lab) - https://en.wikipedia.org/wiki/IEEE_754-1985

- zero:
  - has an exponent of: 0 (all bits in the exponent are 0)
  - has a mantissa of: 0 (all bits in the mantissa are 0)
  - could be positive zero (sign=0) or negative zero (=sign=1)

- nan (not a number)
  - used to indicate a value we cant represent
    - 0/0, sqrt of negative numbers
  - sign could be 0 or 1
  - has an exponent with all bits set to 1 (1111 1111)
  - mantissa can be any non-zero value
  - we can have positive nan (sign=0) or negative nan (sign=1)

IEEE 754 Floating Point Numbers

There are some special cases (important for today's lab) - https://en.wikipedia.org/wiki/IEEE_754-1985

- infinity
  - has an exponent with all bits set to 1
  - the difference is that mantissa for infinity is 0
  - could be positive infinity (sign=) or negative infinity (sign=)

IEEE 754 Floating Point Numbers

4. What decimal numbers do the following single-precision IEEE 754-encoded bit-strings represent?

a. `0 00000000 00000000000000000000000`
b. `1 00000000 00000000000000000000000`
c. `0 01111111 10000000000000000000000`
d. `0 01111110 00000000000000000000000`
e. `0 01111110 11111111111111111111111`
f. `0 10000000 01100000000000000000000`
g. `0 10010100 10000000000000000000000`
h. `0 01101110 10100000101000001010000`

Each of the above is a single 32-bit bit-string, but partitioned to show the sign, exponent and fraction parts.

IEEE 754 Floating Point Numbers

4. What decimal numbers do the following single-precision IEEE 754-encoded bit-strings represent?

a. `0 00000000 00000000000000000000000`
b. `1 00000000 00000000000000000000000`
c. `0 01111111 10000000000000000000000`
d. `0 01111110 00000000000000000000000`
e. `0 01111110 11111111111111111111111`
f. `0 10000000 01100000000000000000000`
g. `0 10010100 10000000000000000000000`
h. `0 01101110 10100000101000001010000`

Each of the above is a single 32-bit bit-string, but partitioned to show the sign, exponent and fraction parts.

Answers:

a. $0.0000 = (1 + 0.0) \times 2^{0-127} = 1.0 \times 2^{-127}$ (... which is close to zero)

b. -0.0000 ... same as above, but the sign bit is 1, so -ve

c. $1.5 = (1 + 0.5) \times 2^{127-127} = 1.5 \times 2^{0}$

d. $0.5 = (1 + 0.0) \times 2^{126-127} = 1.0 \times 2^{-1}$

e. $0.999999 = (1 + 0.999999) \times 2^{126-127} = 1.999999 \times 2^{-1}$

(... but we're limiting this to the approximately 7 digits we can represent here)

f. $2.75 = (1 + 0.375) \times 2^{128-127} = 1.375 \times 2^{1}$

g. $3145728.00 = (1 + 0.5) \times 2^{148-127} = 1.5 \times 2^{21}$

h. $0.0000124165 = (1 + 0.627451) \times 2^{110-127} = 1.627451 \times 2^{-17}$

(... again, limited to approximately 7 digits)

IEEE 754 Floating Point Numbers

5. Convert the following decimal numbers into IEEE 754-encoded bit-strings:

a. 2.5
b. 0.375
c. 27.0
d. 100.0

IEEE 754 Floating Point Numbers

5. Convert the following decimal numbers into IEEE 754-encoded bit-strings:

a. 2.5

b. 0.375

c. 27.0

d. 100.0

Answers:

a. `0 10000000 01000000000000000000000`

$= 1.25 \times 2^1 = (1 + 0.25) \times 2^{128-127}$

b. `0 01111101 10000000000000000000000`

$= 1.5 \times 2^{-2} = (1 + 0.5) \times 2^{125-127}$

c. `0 10000011 10110000000000000000000`

$= 1.6875 \times 2^4 = (1 + 0.6875) \times 2^{131-127}$ where $0.6875 = 2^{-1} + 2^{-3} + 2^{-4}$

d. `0 10000101 10010000000000000000000`

$= 1.5625 \times 2^6 = (1 + 0.5625) \times 2^{133-127}$ where $0.5625 = 2^{-1} + 2^{-4}$