Computer Limitations and Representation

- Recall the Universal Computer
  - There is a limited tape size to perform calculation
- Recall the von Neumann and Harvard architecture
  - There is a predefined width to registers and memory
- Abstract representations with limited sizes for:
  - Natural Numbers & Zero: unsigned char, unsigned int
  - Integers: short int, int, long int
  - Rational/Real:
    - Fix Point ---
    - Floating Point float, double

- An encoding of each will include one or more of the following:

sign	whole	fractional	expon sign	exponent

+4.225 x10^ +2
+1.010101 x2^ +101

Scientific Notation

$$
\begin{array}{r}
14.3 \times 10^7 \\
- \quad 9.2 \times 10^7 \\
\hline
5.1 \times 10^7
\end{array}
$$

- All numbers represented as: $m \times 10^N$
- Simplifies operations on large and small numbers.
  - Distance between sun and earth: 92,000,000 = 9.2×10^7
  - Distance between sun and mars: 143,000,000 = 1.43×10^8
- Floating point representation
  - a representation of scientific notation
  - introduces the notion of infinity, and NaN (0 / 0 = ?, 0 x infinity = ?)
- Representation of: $-1.1011101 \times 2^{-1001}$
  - Assume a size of 16
  - Note the whole part is alway "1", so I left it out!

-	-	1	0	0	1	1	0	1	1	1	0	1	0	0	0
sign	exponent part with sign					fractional part									

Floating Point Encoding

always 1: so we don't store it

Original number: 2# - 0.000100101

Recall Scientific Notation: $\boxed{-}\,\boxed{1}.\boxed{0100101} \times 2^{\boxed{-100}\,(4)}$

- Components to Encode
  - sign: negative
  - significant or the mantissa: 0100101
  - exponent: - 100
    - Issue: negative exponents
    - Solution: store values with a bias
- Bias:
  - Shift all numbers along the number line by N
  - Typically it is half the range:
    - `3 bits ->              011 == 3`
    - `5 bits ->           0 1111 == 15`
    - `8 bits ->        0111 1111 == 127`
    - `11 bits ->  011 1111 1111 == 1023`

Symbol	Encoding
+	0
-	1

Number		Encoding (Bias: 4)
-4		000
-3		001
-2		010
-1		011
0	000	100
1	001	101
2	010	110
3	011	111

1	e	e	e	0	1	0	0

Floating Point Encoding

https://en.wikipedia.org/wiki/Single-precision_floating-point_format

Recall Scientific Notation: $- 1.0100101 \times 2^{-100\ (-4)}$

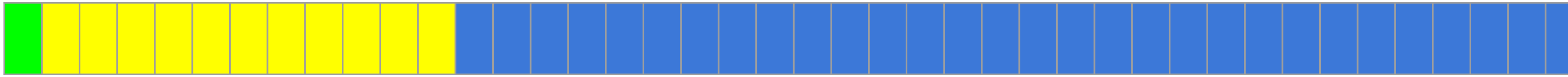
- Formats:
    - binary16 (half): `1 + 5 + 10 = 16,    0 1111 = 15`

s	e	e	e	e	e	m	m	m	m	m	m	m	m	m	m

    - binary32 (single): `1 + 8 + 23 = 32,    0111 1111 = 127`

s	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

    - binary64 (double): `1 + 11 + 52 = 64,  01 1111 1111 = 1023`



Floating Point Encoding

Recall Scientific Notation: $- 1.0100101 \times 2^{-100\ (4)}$

- Consider a new format: c122f8 (quarter)
  - c122f8 (quarter): `1 + 3 + 4 = 8,` `011 = 3`

- `Components`
  - `sign: 1`
  - `mantissa: 010`~~`10`~~ `;` `Drop the extra bits.`
  - `expon: -4 + 3 = -1` `Opps, number is two small.`

1	e	e	e	0	1	0	0

Floating Point Encoding

Recall Scientific Notation: - 1.0100101 x 2 $^{-100\ (-4)}$

- Half Precision
  - float16 (half): `1 + 5 + 10 = 16,` `0 1111 = 15`

- `Components`
  - `sign: 1`
  - `mantissa: 0100101 ; fill in least significant bits with zero (0)`
  - `expon: -4 + 15 = 11 → 1011`

1	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0

Floating Point Encoding

Recall Scientific Notation: $- 1.0100101 \times 2^{-100\ (-4)}$

- Single Precision
  - float32 (single): `1 + 8 + 23 = 32,      0111 1111 = 127`
- Components
  - `sign: 1`
  - `mantissa: 010010 ; fill in least significant bits with zero (0)`
  - `expon:  -4 + 127 = 123 → 0111 1011`

1	0	1	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	m	m	m	m	m	m	m	m	m	m	m	m	m	m