Recall: $fl: \mathbb{R} \to \mathbb{F}$

This will be the double precision floating point numbers.
Store in 8 bytes or 64 bits
The binary precision is $d = 52$

$$\frac{1}{2}\varepsilon_{mach} = \frac{1}{2} \cdot \frac{1}{2^{52}}$$

$$-\log_{10}\left(\frac{1}{2}\varepsilon_{mach}\right) \approx$$

```
julia> -log10(1/2*emach)
15.954589770191003
```

almost 16 decimal digit of precision for each floating-point number..

For comparison: Single precision takes 4 bytes to store and $d = 23$

$\varepsilon_{mach} = \frac{1}{2^{23}}$ for single precision

$$-\log_{10}\left(\frac{1}{2}\varepsilon_{mach}\right) \approx$$

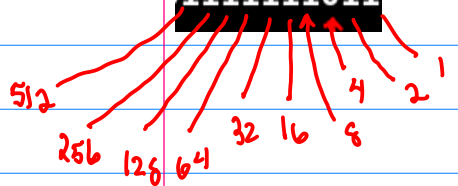
```
julia> emach=1/2.0^23
1.1920928955078125e-7

julia> -log10(1/2*emach)
7.224719895935548
```

a little more than 7 decimal digits of precision..

```
julia> bitstring(0.1)
"0011111110111001100110011001100110011001100110011001100110011010"
```

$1111111011 \approx 1019$ $\qquad n \approx 1019 - 1023 \approx -4$




```
julia> 1019-1023
-4
```

b_i

$f \approx$ `1001100110011001100110011001100110011001100110011001100110011010`

$$\left(1 + \sum b_i 2^{-i}\right) \times 2^{-4}$$

$$0 < \left| \frac{fl(0.1) - 0.1}{0.1} \right| \leq \frac{1}{2}\varepsilon_{mach} = \frac{1}{2^{53}}$$

- Note that .1 means 10 cents in accounting and when computers are used in banking they work in base 10 to avoid not being able to accurately store 10 cents in memory...

Arithmetic:

Let $x, y \in \mathbb{F}$

addition done by computer circuits..

$x \oplus y \approx fl(x+y)$

$x \ominus y = fl(x-y)$

built into hardware

result of exact addition rounded to the nearest floating point number that can be stored back in memory.

may be errors storing x and y in the computer but we're in the next section about the arithmetic...

$$x \circledast y = fl(x \cdot y)$$

$$x \circledast y = fl(x / y)$$

$$\text{sqrt}(x) \approx fl(\sqrt{x})$$

$$\sin(x) = fl(\sin x)$$

$$\log(x) = fl(\ln x)$$

} may or may not be built in to the hardware, but if not exist as common subroutines in the programming environment.

↑ for us Julia.

Sometimes the functions in the library don't quite round to the nearest representable value of the correct result.

↑ Sometimes they are off by a bit or two (least significant) for speed...

Computer arithmetic for $\oplus$ and $\circledast$ is not associative because of the rounding.

Example:

Add .46, .33, .83 — rounding to 2 significant (decimal) digits after each addition.

In order

$$.46 + .33 = .79$$

$$.79 + .83 = 1.62 \rightarrow \text{round to } 1.6$$

The other way

$$\begin{array}{r} .83 \\ .33 \\ \hline 1.16 \end{array} \xrightarrow{\text{round}} 1.2 \qquad \begin{array}{r} 1.2 \\ .46 \\ \hline 1.66 \end{array} \xrightarrow{\text{round}} 1.7$$

The answers depends on the order. The most accurate answer is given by adding the numbers starting the smallest ones first...

Related idea loss of Precision:

$$\begin{array}{r} .52 \\ -\ .51 \\ \hline .01? \end{array}$$

.52 ← 2 digits of precision

.51 ← 2 digits of precision

.01? ← 1 digit of precision

↑ don't know what goes here because the third place after the decimal wasn't stored in the original numbers...

Subtraction of nearly equal numbers happens naturally when approximating a derivative

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}$$

if h is small the difference between $f(x_0+h)$ and $f(x_0)$ is about the same.

Need to work around this when solving differential equations...

Another example $f(x) = x + 1$ when x is small

On computer $fl(x) \oplus 1 = fl(fl(x) + 1)$

$$fl(x) = x + \varepsilon x \quad \text{where} \quad |\varepsilon| \leq \frac{1}{2} \varepsilon_{mach}$$

Finish this next time...