



$11101_2 = 29$  in base 10  
 $\swarrow \downarrow \downarrow \downarrow \downarrow$   
 $16 \ 8 \ 4 \ 2 \ 1 = 2^0$   
 $2^4 \ 2^3 \ 2^2 \ 2^1$

Binary arithmetic :

$$\begin{array}{r}
 11 \\
 1010 \xrightarrow{10} \\
 + 11011 \xrightarrow{27(?)} \\
 \hline
 100101 (=37)
 \end{array}$$

Decimals:

base 10:  $1/10 = 0.1$

1010  $\overline{)11.0}$

base 2:  $0.0001100_2$  (long division)

$\pi, e, \sqrt{2}$  can only be approximated by decimals.

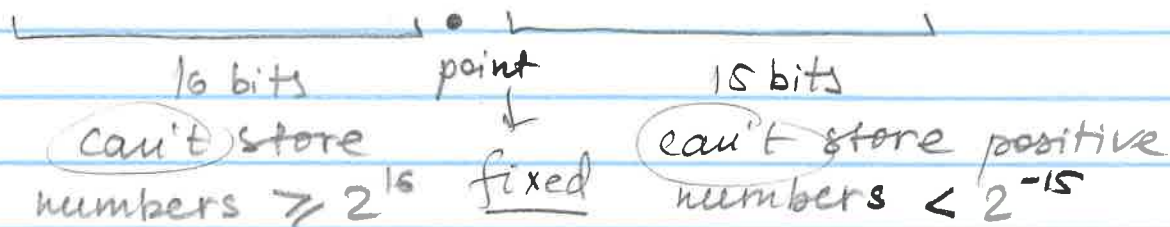
**Q:** How do we represent numbers on our computers?

- Bit: 1 or 0
- Word: a certain number of bits

Early computers used fixed-point representation :

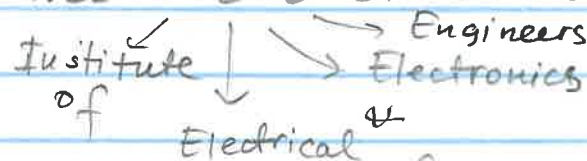
one bit for a sign and a certain number of bits to store parts of a binary number to the left of the binary point, and the remaining bits — to store the part to the right of the point.

The system is limited: for example,



More flexible: floating-point representation.

Adopted standard is the IEEE standard. (80's)

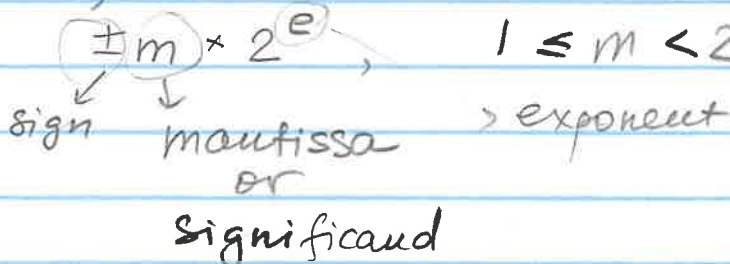


- Consistent representation of floating-point numbers across machines.
- Correctly rounded arithmetic.
- Consistent treatment of exceptional situation, e.g., division by 0.

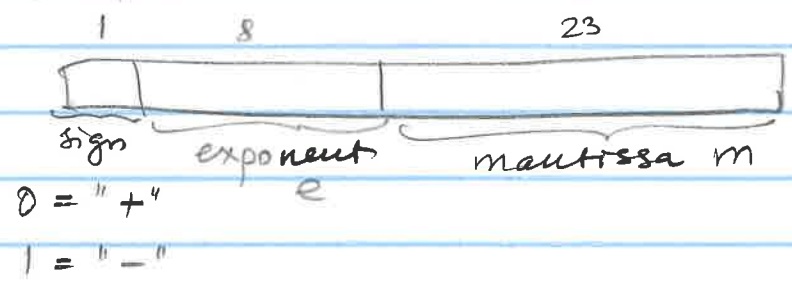
There are three IEEE precisions:

- single-precision word has 32 bits
- double precision word has 64 bits
- extended-precision: 80 bits.

So, a binary number is written in the form  $(\pm m) \times 2^e$ ,  $1 \leq m < 2$



• Single-precision word: 32 bits



Ex:  $10 = 1010_2 = 1.010_2 \times 2^3 \rightarrow$ 

1	8	23
0	e=3	1.0100... 0

If a number can be stored exactly using this setting  $\Rightarrow$  we call it a floating-point number. Otherwise, a number is rounded to a floating-point number.

e.g.,  $1/10 = 1.1001100_2 \times 2^{-4}$  must be rounded.

Improvement:  $\pm m \times 2^e, \quad 1 \leq m < 2$   
 $m = \underbrace{b_0.b_1b_2\dots b_n}_{\text{always 1}} \Rightarrow$  do not need to store!

We only store  $b_1\dots b_n$  knowing that  $b_0=1$

E.g.,  $10 = 1010_2 = 1.010_2 \times 2^3 \rightarrow$ 

0	e=3	0100... 0
---	-----	-----------

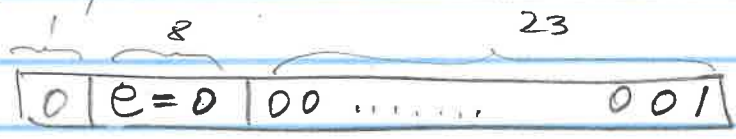
  
 "hidden-bit representation"

There are special numbers, like 0, that need a special way to be represented:

$0 \neq (1.b_1b_2\dots) \times 2^e$

- The gap between 1 & the next larger floating-point number is called the machine precision and is denoted by  $\epsilon$  (MATLAB: "eps").

Single precision: after 1, comes  $1 + 2^{-23}$  ;



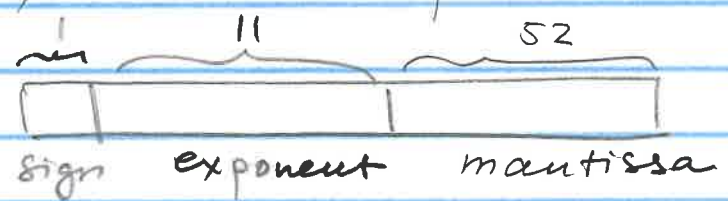
So,  $\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$

(Note:  $1_{10} = (1.0)_2 \times 2^0 \rightarrow$ 

0	e=0	00 ... 00
---	-----	-----------

)

MATLAB: the default precision is double precision, i.e. 64 bit per word:



$1 + 2^{-52}$  is the closest to 1 larger number:  
 $\epsilon = 2^{-52} \times 2.2 \times 10^{-16}$  (type "eps" in MATLAB)

Note: gap between 0 & the smallest positive number can be filled using subnormal numbers.

# §§ 5.4, 5.5. IEEE Floating-Point Arithmetic & Rounding.

Recall: - consistency across machines  
 - correct rounding  
 - consistent treatment of exceptional situations (NaN: %)

IEEE standard.

Consider IEEE double precision:

If: e field (11 bits) then Number is:  $\pm (0.b_1 b_2 \dots b_{52})_2 \times 2^{-1022}$  Type  $\rightarrow$  0 or subnormal

$(1_{10} =)$	00 ... 01	$\pm (1.b_1 b_2 \dots b_{52})_2 \times 2^{-1022}$	} normalized numbers: e field = actual exp. + 1023
$(2_{10} =)$	00 ... 10	$\pm (1.b_1 b_2 \dots b_{52})_2 \times 2^{-1021}$	
$\vdots$	$\vdots$	$\vdots$	
$(1023_{10} =)$	011 ... 11	$\pm (1.b_1 b_2 \dots b_{52})_2 \times 2^0$	
$\vdots$	$\vdots$	$\vdots$	
$(2046_{10} =)$	11 ... 10	$\pm (1.b_1 \dots b_2)_2 \times 2^{1023}$	

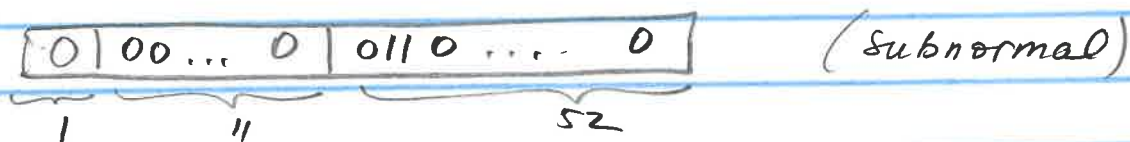
11 ... 11 }  $\pm \infty$  if  $b_1 = \dots = b_2 = 0 \rightarrow$  Exception  
 NaN otherwise

Smallest number:  $(1.0)_2 \times 2^{-1022} \approx 2.2 \times 10^{-308}$   
 Largest  $\leftarrow$  :  $(1.1 \dots 1)_2 \times 2^{1023} \approx 1.8 \times 10^{308}$   
 $> 0$

Exponent field = actual  $e + 1023$ , in Single precision: actual  $e + 127$ .

So, why is that? E.g., 8 bits in single precision for  $e$  can only give values from 0 to 255. To cover  $e < 0$ , the exponent is 127 greater than the real one, e.g. for  $(1.01011101)_2 \times 2^5$ , the eight-bit exp. field is  $5 + 127 = 132 = (10000100)_2$ .

Special exponent fields: all 0's or 1's.  
Number  $(1.1)_2 \times 2^{-1024} = (0.011)_2 \times 2^{-1022}$ :



less precision than normalized numbers.

0: all 0's in exp. field & mantissa.

All 1's: exceptions.

MATLAB:  $0/0 \rightarrow \text{NaN}$

$a/0 (a \neq 0) \rightarrow \text{Inf} (\pm \infty)$

Previously:  $1/0 - 2/0 = 0$

Now:  $1/0 - 2/0 = \text{NaN}$

Also:  $\infty + a = \infty$

$a - \infty = -\infty$

$\infty \times 0 = \text{NaN}$ , etc.

$2/\text{Inf} = 0$

$\text{Inf}/2 = \text{Inf}$

$2/0 = \text{Inf}$

Rounding: 4 modes

• Round down:  $\text{round}(x) \leq x$

• Round up:  $\text{round}(x) \geq x$

- Round towards 0 :  $\text{round}(x)$  is either  $\text{round-down}(x)$  or  $\text{round-up}(x)$ , whichever lies between 0 &  $x$ . If  $x > 0 \Rightarrow \text{round}(x) = \text{round-down}(x)$ , if  $x < 0 \Rightarrow \text{round}(x) = \text{round-up}(x)$ .

- Round to the nearest : either  $\text{round-down}$  or  $\text{round-up}$ , whichever is closer.

→ Default:

Ex:  $\frac{1}{15} = 1.1001100_2 \times 2^{-4}$  in double-precision



$(+1023) - 4 = 1019_{10}$

↑ (repeats →

↓  
Round-up (nearest)

- Absolute rounding error is

$$|\text{round}(x) - x|$$

In double precision,  $|\text{round}(x) - x| < \underbrace{2^{-52}}_{\epsilon} \times 2^e$

- Relative rounding error is

$$\frac{|\text{round}(x) - x|}{|x|} < \frac{\epsilon \times 2^e}{m \times 2^e} < \epsilon$$

We can write:  $\text{round}(x) = x(1 + \delta)$  w/  $|\delta| < \epsilon$



IEEE standard: if  $a$  &  $b$  are fl.-point numbers then

$$a \oplus b = \text{round}(a+b) = (a+b)(1+\delta_1)$$

$$a \ominus b = \text{round}(a-b) = (a-b)(1+\delta_2)$$

$$a \otimes b = \text{round}(ab) = (ab)(1+\delta_3)$$

$$a \oslash b = \text{round}(a/b) = (a/b)(1+\delta_4)$$

$$w/ |\delta_i| < \epsilon, \quad i=1,2,3,4.$$

→ Read §5.6: examples on rounding.

§5.7: Exceptions.

$\pm\infty$ , NaN

Other:

- 1) overflow: true result is greater than the largest fl.-pt. number  
 $((1.1\dots 1)_2 \times 2^{1023} \approx 1.8 \times 10^{308}$  for double precis.)

Can be set to  $\infty$  or the largest number.

- 2) Underflow: true result is less than the smallest fl.-pt. number.  
 Stored as a subnormal number or set to 0.