Real Numbers

Real Number

↓

Can't be represented precisely

Only approximations

Fixed point numbers | Floating Point numbers

- Before we discuss the representation of fixed and floating point numbers, we look at how fractions are converted between decimal & binary number systems.

Converting fractions

- **Binary → Decimal**

$$\begin{array}{ccccccc} 0 & -1 & -2 & -3 & -4 & -5 \\ 0. & 0 & 1 & 1 & 0 & 1 \end{array}$$

$$(0.01101)_2 = 0.01101$$

$$2^{-2}\ 2^{-3}\quad 2^{-5}$$

$$\frac{1}{4} + \frac{1}{8} + \frac{1}{32} = \frac{8+4+1}{32} = \frac{13}{32}$$

$$= (0.40625)_{10}$$

- **Decimal → Binary** : multiply by 2 & note integer part.

$$(0.6875)_{10} = (\ldots)_2.$$

x2	0.6875	1
	1.3750	
take remainder	0.3750	
x2		0
	0.750	
take remainder		
x2	0.750	1
take remainder	1.5	
	0.5	
x2		1.
	1.0	
take rem.		
	0	done.

read.

answer.

$$(0.1011)_2$$

$$(0.1)_{10} = (\ldots)_2$$

	0.1	
x2		0
	0.2	
x2		0
	0.4	
x2		0
	0.8	
x2		1
	1.6	
	0.6	
x2		1
	1.2	
	0.2	
x2		0
	0.4	

repeat

ans: $0.0\underbrace{110}\underbrace{0110}\underbrace{0110}\ldots$

repeats.

- Decimal → octal   multiply by 8, note integer part.

$(0.12)_{10} = (\ldots)_8. = (0.0753\ldots)_8.$

$$\begin{array}{r} 0.12 \\ \times 8 \\ \hline 0.96 \end{array} \qquad 0$$

$$\begin{array}{r} 0.96 \\ 8 \\ \hline 7.68 \end{array} \qquad 7.$$

$$\begin{array}{r} 0.68 \\ 8 \\ \hline 5.44 \end{array} \qquad 5.$$

$$\begin{array}{r} 0.44 \\ \cdot 8 \\ \hline 3.52 \end{array} \qquad 3$$

⋮ etc.

Fixed point ~~real~~ numbers

- integer :
$$\begin{array}{c}7\,6\,5\,4\,3\,2\,1\,0\\(10101010)_2\end{array} = 2^7+2^5+2^3+2^1$$
$$= 128+32+8+2 = \underline{170}$$

It is assumed that the decimal point lies at the right most digit :

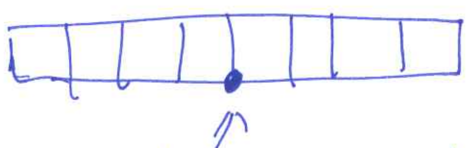
10101010.

- Fixed point number:

  is very similar to integer representation, except the decimal point is assumed to lie elsewhere.

Note: computer CANNOT place a decimal point explicitly. It must assume ~~the local~~ ~~...~~ a fix place for it.

eg: Assume decimal point lies at :



imaginary point !!!

Then the ~~number~~ bit pattern

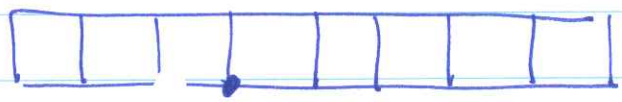
10101010

means:
$$\begin{array}{c}3\,2\,1\,0\;\;{-1}\,{-2}\,{-3}\,{-4}\\1010.1010\end{array}$$

$$8+2\,.\,\frac{1}{2}+\frac{1}{8} = 10.\frac{5}{8} = \underline{10.625}.$$

Converting Fixed Point Binary numbers.

Assume binary fixed point representation is:



Binary to decimal conversion:

Pattern: $\boxed{01101100}$

This implies (assume decimal point):

$$011.01100$$

$$2^1\ 2^0\ \ 2^{-1}\ 2^{-2}\ 2^{-3}$$

$$\text{Value} = 2+1+\frac{1}{4}+\frac{1}{8}.$$

$$= 3\frac{3}{8}$$

$$= 3.375$$

$\boxed{\text{Decimal to binary (fixed point) conversion:}}$

$(3.14)_{10} = ?$

Split 3.14 into integer part → 3
and fraction part → 0.14.

Integer part converted as follows:

$$
\begin{array}{r|l}
2 & 3 \quad 1 \\
\hline
2 & 1 \quad 1 \\
\hline
 & 0
\end{array}
$$

$(3)_{10} = (11)_2$.

Fraction part converted as follows:

$(0.14)_{10} = (0.001000 \ldots)$

$$
\begin{array}{ll}
 & 0.14 \\
\times 2 & \overline{} \rightarrow 0 \\
 & 0.28 \\
\times 2 & \overline{} \rightarrow 0 \\
 & 0.54 \\
\times 2 & \overline{} \rightarrow 1 \\
 & 1.12 \\
 & \\
 & 0.12 \\
\times 2 & \overline{} \rightarrow 0 \\
 & 0.24 \\
\times 2 & \overline{} \quad 0 \\
 & 0.48 \\
\times 2 & \overline{} \quad 0 \\
 & 0.96
\end{array}
$$

So:

$$(3.14)_{10} = (11.001000\ldots$$

Fit in 8 bits and line up the decimal point:

	1	1	.0	0	1	0	0

answer.

Conclusion: ~~Real numbers are NOT represented precise~~ly.

- Integer numbers can be represented precisely by computers.

- Non-integer numbers CANNOT.
  There will be a round off / truncation error for non-integer numbers.
  The magnitude of the error depends on the number of bits used to represent the fraction.

Representing Floating Point numbers

- Floating point $\neq$ Real !

  Real number can have infinite number of digits.

- Floating point number n:

$$n = \underset{\text{mantissa}}{m} \times 10^{\overset{\text{exponent}}{e}} \qquad \text{(for humans).}$$

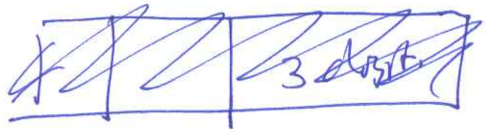
eg: $19.47 = 0.1947 \times 10^{2}$
$= 1947.0 \times 10^{-2}$. etc.

- ~~Not every possible number~~

- Mantissa & exponent are of finite length.

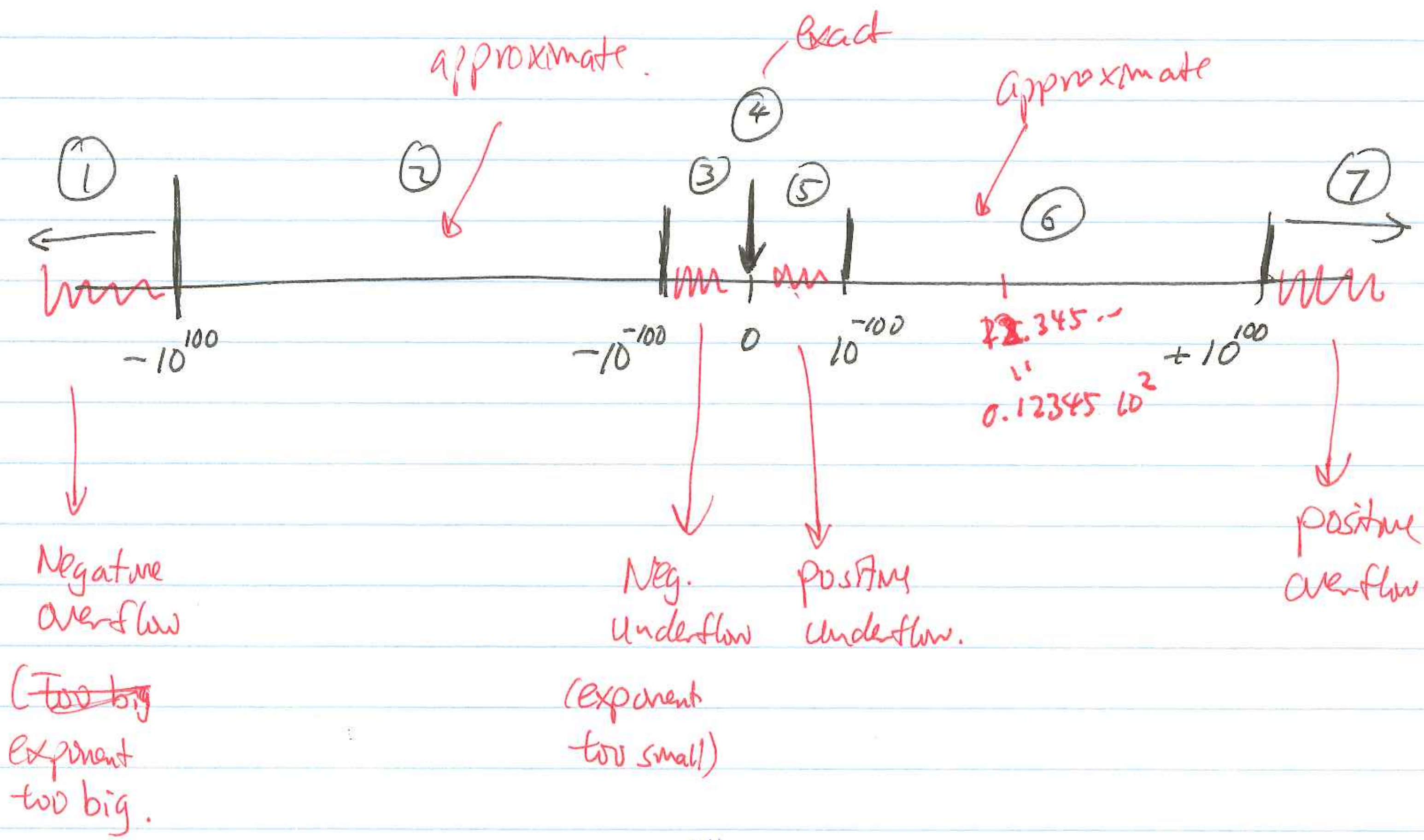
- Not every possible numerical value can be expressed.

eg: S1



Example : 3 $\underset{\text{digit}}{\overset{\text{decimal}}{}}$ mantissa & |exponent| $\leq$ 99 (decimal)

The real numbers ~~than~~ can be divided into 7 regions:



Any number between $(-10^{100}, -10^{-10})$.

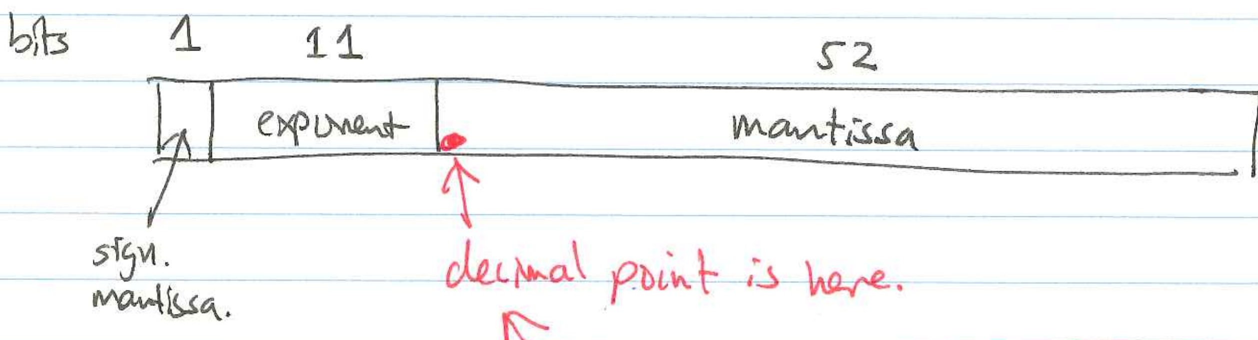
IEEE floating-point standard 754

- IEEE 754 defines 3 standard formats for floating point numbers.

  (Most computers now uses these formats).

  - single precision. (32 bits)
  - double precision (64 bits)
  - Extended precision (80 bits)

- Double precision format: (others are similar)



- the mantissa is between 1 & 2.:

  $1 \leq m < 2.$ (Decimal point assumed HERE)

- exponent is (2^e) in excess 2^{10}.

(Thus: the mantissa is signed magnitude.

the exponent is excess 2^{10}).

Operations with floating point numbers:

- multiply & divide : (1) multiply / divide mantissa
  (2) add / subtract exponent

  (3) Normalize !
  ↓
  bring result back to a form that $1 \leq$ mantissa < 2

- add & subtract : (1) bring both operands to same exponent value.

  (2) add / subtract mantissa (exponent = exponent of one operand)

  (3) Normalize.

Floating Point Number

- Short coming of fixed point number : bound by range
- A better way to represent real number is the "scientific" notation:

$$\text{real number} = \text{fraction} \times 10^{\text{exponent}}.$$

fraction is also called mantissa

- The computer version of this way of representing real numbers is called "floating point".
- Example :

$$314 = 3.14 \times 10^{2} = 0.314 \times 10^{3}$$
$$0.000012 = 1.2 \quad 10^{-5} = 0.12 \times 10^{-4}$$

- ~~One~~ There are many ways to represent the same number :

$$3.14 = \underbrace{3.14 \times 10^{0}}_{\text{form 1.}} = 0.314 \times 10^{1} \quad \text{etc.}$$

One form is usually chosen ~~as~~ the standard. (ie: where to put the decimal point).

- Standard forms typically chosen are:

(1) mantissa consists of 1 digit ($\neq \emptyset$)

eg:		number stored as:
$314 =$	3.14×10^2	31400 +02
$31.4 =$	3.14×10^1	31400 +01
$1255 =$	1.255×10^3	12550 +03
$0.00012 =$	1.2×10^{-4}	12000 −04

(2) mantissa is $\emptyset$ and leading digit of fraction must not be $\emptyset$.

eg:		number stored as
$314 =$	0.314×10^3	31400 +03
$31.4 =$	0.314×10^2	31400 +02
$1255 =$	0.1255×10^4	12550 +04
$0.00012 =$	0.12×10^{-3}.	12000 −03

- Now do this in binary:

Note that:

$$101 = 10.1 \times 2^1$$
$$= 1.01 \times 2^2$$
$$= 0.101 \times 2^3$$

↗ "5"

Here also, there are many way to represent the same floating point number.

One way is picked as the standard.

- Eg: mantissa consists of 1 digit ($\neq 0$)

Then:

number	standard repr	internally stored as:
101	$= 1.01 \times 2^2$	10100000 0010
0.101	$= 1.01 \times 2^{-1}$	10100000 "-0001"
-101	$= -1.01 \times 2^2$	"-10100000" 0010"
-0.101	$= -1.01 \times 2^{-1}$	"-10100000" "-0001"

So: mantissa AND exponent can be pos. & neg. numbers!!!

- Single precision floating point number: 4 bytes.

Standard form: 1 digit mantissa ($\neq 0$).

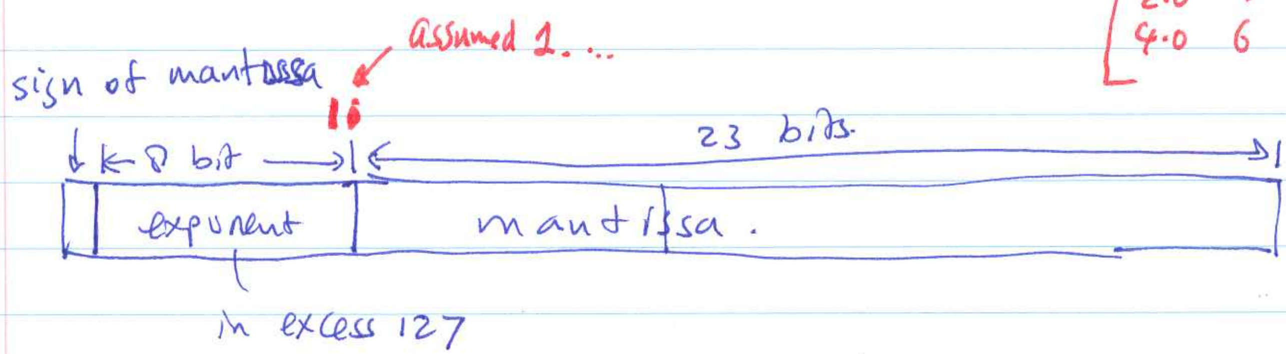
Example:

$101 = 1.01000\ldots \times 2^2$

printfloat.c

use

1.0	1.5	1.75
2.0	3	3.5
4.0	6	7



- mantissa is stored in sign-magnitude representation

  magnitude: unsigned value or absolute value.

Where the assumed decimal point is immediately after the leading digit.

(mantissa is always 1. . . . . . .)

Also: the leading 1 in mantissa is NOT stored, but it is assumed (since it's always there!)

- Exponent is stored in 8 bits with excess <u>127</u> encoding (ie.: value is stored as absolute (value+127))

- Example:

$$(1.0)_{10} = (1.0)_2 \times 2^0$$

mantissa $= 1.\underbrace{00000 \dots 0}_{23 \text{ bits.}}$ sign $= \emptyset$.

exponent $= 0$

representation in excess 127

value x is repr. by absol. value $x+127$
0 is repr. by ~~abs~~: unsigned value
$0+127 = 127 = 0111.1111$
(8 bits).

Put it in the format specified:

Ø	0111.1111	0000 - - - - - - - - 0

Example: $(-1.0)_{10}$

$(-1.0)_{10} = (-1.0)_2 \times 2^0$.

mantissa = sign = 1
magnitude = $1.\underbrace{000 \ldots 0}_{23 \text{ bits}}$

exponent = 0 → 127 = 0111.111

Repr:

$\boxed{1}$ $\boxed{0111.1111}$ $\boxed{000 \cdots\cdots 0}$

Example: $(1.25)_{10}$.

$(1.25)_{10} = (1.01)_2$

$= (1.01)_2 \times 2^0$.

mantissa : sign = ∅.
magnitude = $1.\underbrace{010000 \ldots}_{23 \text{ bits}}$

exponent = ∅.

Repr:

$\boxed{\emptyset}$ $\boxed{0111.1111}$ $\boxed{01000000 \ldots 0}$

Example:

$(16.25)_{10} = (10000.01)_2$

$= (1.000001)_2 \times 2^4$

mantissa: sign $= 0$

magnitude $= 1.\underbrace{0000010000000}_{23 \text{ bits}}$.

exponent $= 4$

in excess 127 repr. by unsigned $4 + 127$
$= 131$

$131 = 128 + 3$
$= 1000.0000 + 11$
$= 1000.0011$.

Representation is:

$\boxed{0}\ \boxed{1000.0011}\ \boxed{00001000 \ldots 0}$

Given : repr : $\boxed{1}\ \boxed{0111.1110}\ \boxed{110000\ldots0}$

what is the ~~number~~ value ?

mantissa: sign = 1 → neg.
magnitude = 1.110000

= ~~1 + 1/2 + 1/4 = 1.75~~.

so: ~~-1.75~~

exponent : 0111.1110 = 126

repr. value $x + 127 = 126$
$x = 126 - 127$
$= -1.$

so:

value in binary $= -1.11 \times 2^{-1}$

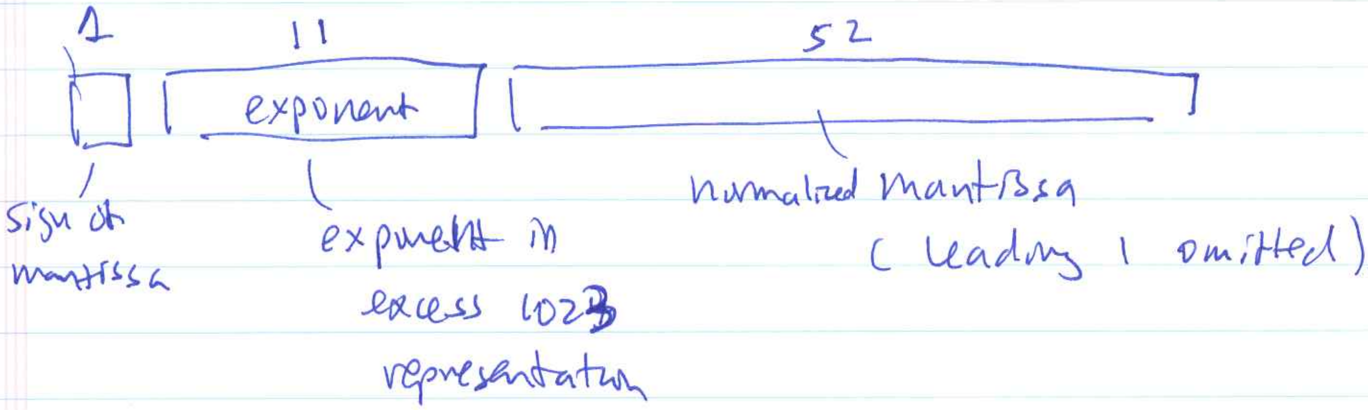
$= -0.111$

$= -\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8}\right)$

$= -\frac{7}{8}$

$= -0.875$

- <u>Double precision IEEE format</u> :



The IEEE Floating point standard

- Upto about 1980, each computer manufacturer has its own way to represent floating point numbers.

- To ease exchange of data (information) between computers, a universal standard for floating point numbers was established by IEEE (Institute of Electrical & Electronical Engineers). in 1985 (The IEEE standard 754)

- Most modern CPUs (Motorola, SPARC, MIPS) include a floating point processor and all of them conform to the IEEE standard.

- The standard defines 3 types of floating point numbers:

  (1) single precision (float in C)
  (2) double precision (double in C)
  (3) quad precision (long double in C)

They are all similar.