Smith College
Computer Science

Week 14
Fixed & Floating Point Formats

CSC231—Fall 2017
Week #14

Dominique Thiébaut
dthiebaut@smith.edu

Reminder!

- In C, strings are terminated by a byte containing 0 decimal, or 0000 0000 binary. In C, we express this quantity as '\0'.
- In assembly, 0 as a byte is expressed as 0
- '\0' in C = 0000 0000 = 0
- '0' in assembly = 0011 0000 = 0x30

```
Cmsg    db      "hello", 0
        cmp     al, 0
```

Reference

http://cs.smith.edu/dftwiki/index.php/CSC231_An_Introduction_to_Fixed-_and_Floating-Point_Numbers

```java
public static void main(String[] args) {

   int n = 10;
   int k = -20;

   float x = 1.50;
   double y = 6.02e23;

}
```

```java
public static void main(String[] args) {

   int n = 10;
   int k = -20;

   float x = 1.50;
   double y = 6.02e23;

}
```

```
public static void main(String[] args) {

   int n = 10;
   int k = -20;

   float x = 1.50;
   double y = 6.02e23;

}
```

Nasm knows what 1.5 is!

```
        section    .data

x       dd       1.5
```

in memory, x is represented by

00111111 11000000 00000000 00000000

or

0x3FC00000

Outline

- **Fixed-Point Format**
- Floating-Point Format

Fixed-Point Format

- Used in very few applications, but **programmers know about it**.

- Some micro controllers (e.g. Arduino Uno) do not have Floating Point Units (**FPU**), and must rely on libraries to perform Floating Point operations (VERY SLOW)

- Fixed-Point can be used when storage is at a premium (can use small quantity of bits to represent a real number)

Review Decimal Real Numbers

Decimal Point

$123.45 = 1x10^2 + 2x10^1 + 3x10^0 + 4x10^{-1} + 5x10^{-2}$

Can we do the same in binary?

- Let's do it with **unsigned numbers** first:

$$1101.11 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 1x2^{-2}$$

Binary Point

Can we do the same in binary?

- Let's do it with **unsigned numbers** first:

$$1101.11 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 1x2^{-2}$$

$$= 8 \quad + 4 \quad\quad\quad + 1 \quad + 0.5 \quad + 0.25$$

$$= 13.75$$

Observations

- If we know where the binary point is, we do not need to "store" it anywhere. (Remember we used a bit to represent the +/- sign in 2's complement.)
- A format where the binary/decimal point is fixed between 2 groups of bits is called a **fixed-point format**.

Definition

- A number format where the numbers are **unsigned** and where we have *a* integer bits (on the left of the decimal point) and *b* fractional bits (on the right of the decimal point) is referred to as a ***U(a,b)*** *fixed-point format.*

- Value of an *N*-bit binary number in U(a,b):

$$x = (1/2^b) \sum_{n=0}^{N-1} 2^n x_n$$

Exercise 1

typical final exam question!

x = 1011 1111 = 0xBF

- What is the value represented by x in ***U(4,4)***
- What is the value represented by x in ***U(7,3)***

Exercise 2

typical final exam question!

- z = 00000001 00000000
- y = 00000010 00000000
- v = 00000010 10000000
- What values do z, y, and v represent in a ***U(8,8)*** format?

Exercise 3

typical final exam question!

- What is 12.25 in ***U(4,4)***? In ***U(8,8)***?

What about *Signed* Fixed-Point Numbers?

Observation #1

- In an N-bit, **unsigned** integer format, the weight of the MSB is 2^{N-1}

nybble	Unsigned
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	+8
1001	+9
1010	+10
1011	+11
1100	+12
1101	+13
1110	+14
1111	+15

$N = 4$

$2^{N-1} = 2^3 = 8$

Observation #2

- In an N-bit **signed** 2's complement integer format, the weight of the MSB is $\mathbf{-2^{N-1}}$

nybble	2's complement
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

$N=4$

$-2^{N-1} = -2^3 = -8$

Fixed-Point Signed Format

- **Fixed-Point signed** format = sign bit + *a* integer bits + *b* fractional bits = *N* bits = **A(*a*, *b*)**
- *N* = number of bits = 1 + a + b
- Format of an *N*-bit A(a, b) number:

$$x = (1/2^b)\left[-2^{N-1}x_{N-1} + \sum_{0}^{N-2} 2^n x_n\right],$$

Examples in A(7,8)

- 000000001 00000000 = 00000001 . 00000000 = ?
- 100000001 00000000 = 10000001 . 00000000 = ?
- 00000010 00000000 = 0000010 . 00000000 = ?
- 10000010 00000000 = 1000010 . 00000000 = ?
- 00000010 10000000 = 00000010 . 10000000 = ?
- 10000010 10000000 = 10000010 . 10000000 = ?

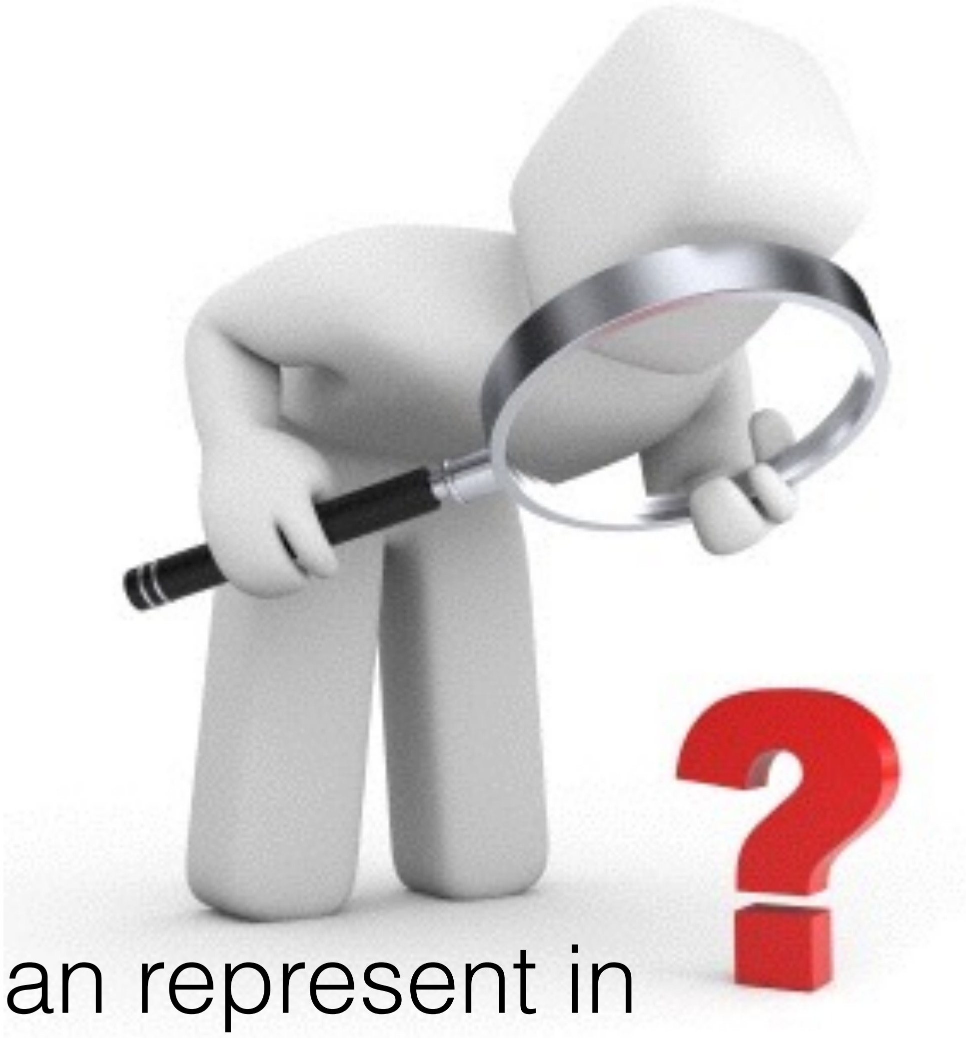
Examples in A(7,8)

- 000000001 00000000 = 00000001 . 00000000 = 1d
- 100000001 00000000 = 10000001 . 00000000 = -128 + 1 = -127d
- 00000010 00000000 = 0000010 . 00000000 = 2d
- 10000010 00000000 = 1000010 . 00000000 = -128 + 2 = -126d
- 00000010 10000000 = 00000010 . 10000000 = 2.5d
- 10000010 10000000 = 10000010 . 10000000 = -128 + 2.5 = -125.5d

Exercises

- What is -1 in ***A(7,8)***?
- What is -1 in ***A(3,4)***?
- What is 0 in ***A(7,8)***?
- What is the smallest number one can represent in ***A(7,8)***?
- The largest in ***A(7,8)***?

Exercises



- What is -1 in ***A(7,8)***?

  11111111 00000000

- What is -1 in ***A(3,4)***?

  1111 0000

- What is 0 in ***A(7,8)***?

  00000000 00000000

- What is the smallest number one can represent in ***A(7,8)***?

  10000000 00000000

- The largest in ***A(7,8)***?

  01111111 11111111

Exercises

- What is the largest number representable in ***U(a, b)***?

- What is the smallest number representable in ***U(a, b)***?

- What is the largest positive number representable in ***A(a, b)***?

- What is the smallest negative number representable in ***A(a, b)***?

Exercises

- What is the largest number representable in ***U(a, b)***?
  1111…1 111…1 = 2^{a}-2^{-b}

- What is the smallest number representable in ***U(a, b)***?
  0000…0 000… 01 = 2^{-b}

- What is the largest positive number representable in ***A(a, b)***?
  0111…11   111.-.11 = 2^{a-1} -2^{b}

- What is the smallest negative number representable in ***A(a, b)***?
  1000…00   000…000 = 2^{a-1}

- Fixed-Point Format
  - Definitions
    - Range
    - Precision
    - Accuracy
    - Resolution
- Floating-Point Format

Range

- Range = difference between most positive and most negative numbers.

- **Unsigned Range**:
  The range of **U(*a*, *b*)** is $0 \leq x \leq 2^a - 2^{-b}$

- **Signed Range**:
  The range of **A(*a*, *b*)** is $-2^a \leq x \leq 2^a - 2^{-b}$

Precision

2 different definitions

- **Precision** = *b*, the number of fractional bits

https://en.wikibooks.org/wiki/Floating_Point/Fixed-Point_Numbers

- **Precision** = *N*, the total number of bits

Randy Yates, Fixed Point Arithmetic: An Introduction, Digital Signal Labs, July 2009.
http://www.digitalsignallabs.com/fp.pdf

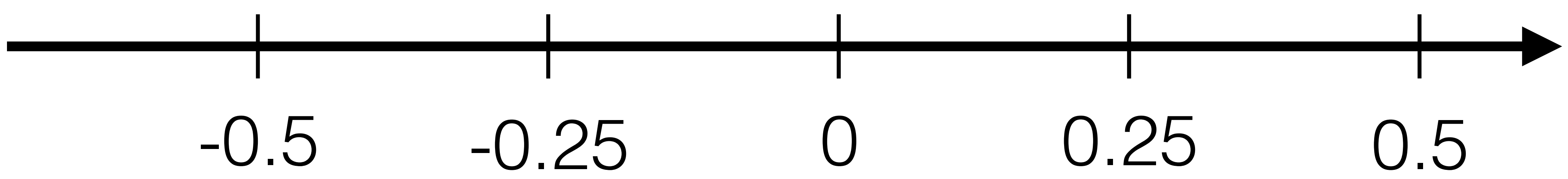
Resolution

- The **resolution** is the smallest non-zero magnitude representable.
- The **resolution** is the size of the intervals between numbers represented by the format
- Example: ***A(13, 2)*** has a resolution of 0.25.

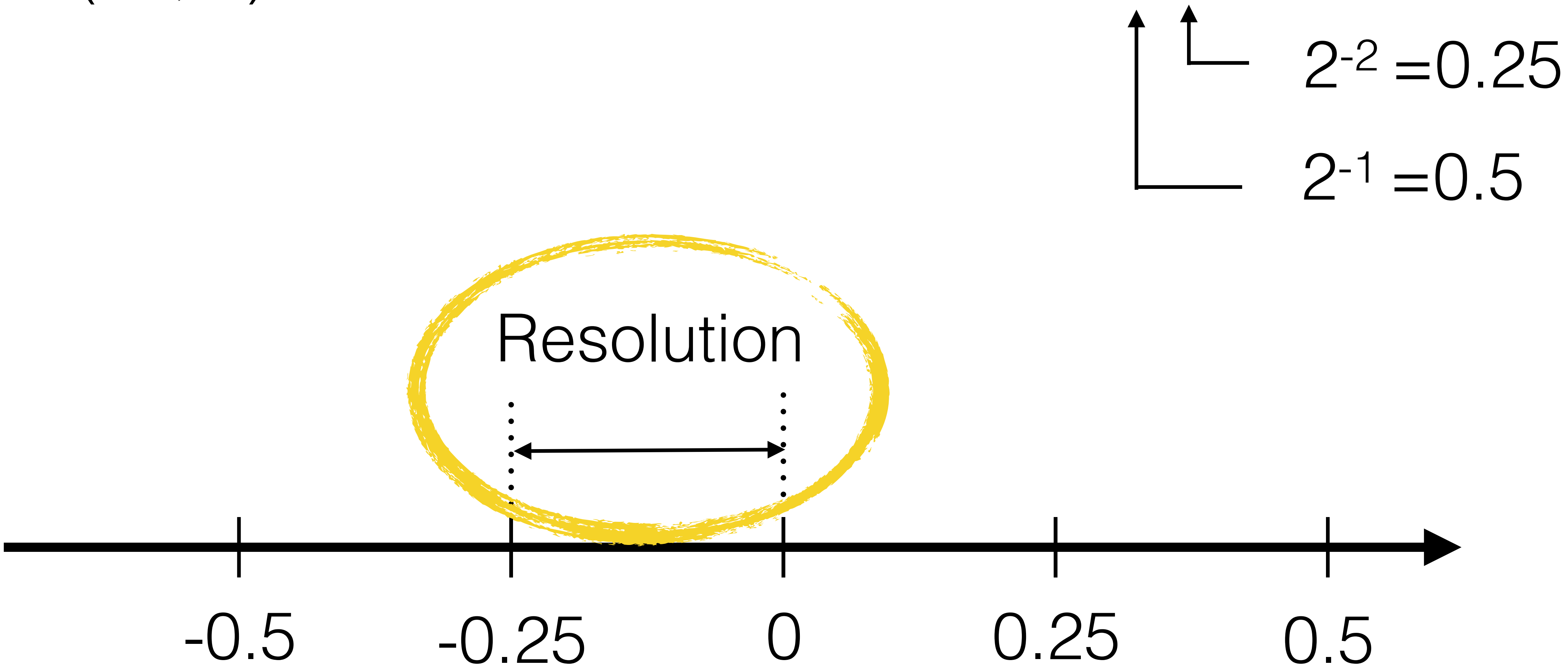
A(13, 2) —> sbbb bbbb bbbb bb . bb

$2^{-2} = 0.25$

$2^{-1} = 0.5$



A(13, 2) —> sbbb bbbb bbbb bb . bb



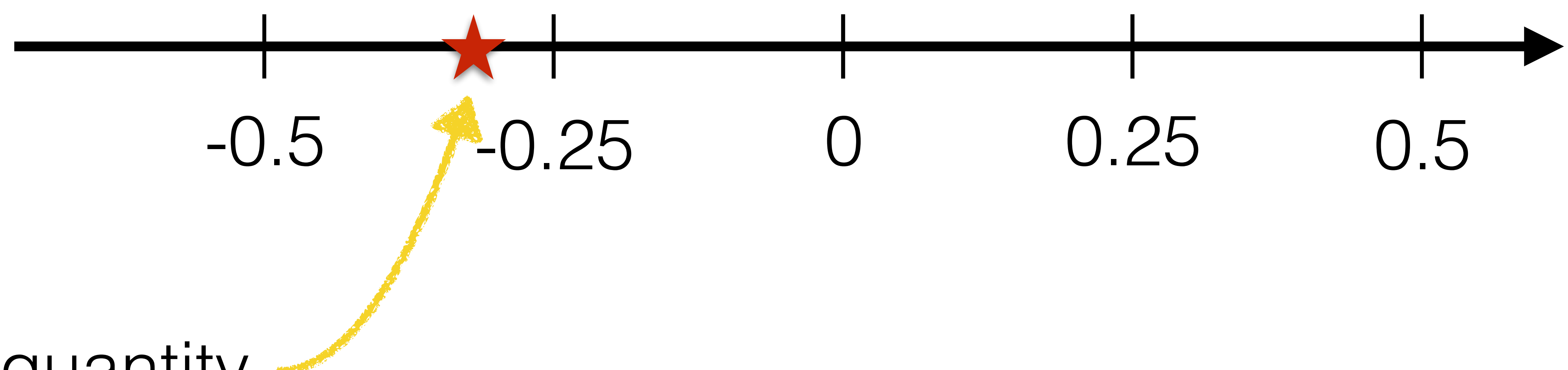
Accuracy

- The **accuracy** is the largest magnitude of the difference between a number and its representation.
- **Accuracy** = 1/2 **Resolution**

A(13, 2) —> sbbb bbbb bbbb bb . bb

$2^{-2} = 0.25$

$2^{-1} = 0.5$

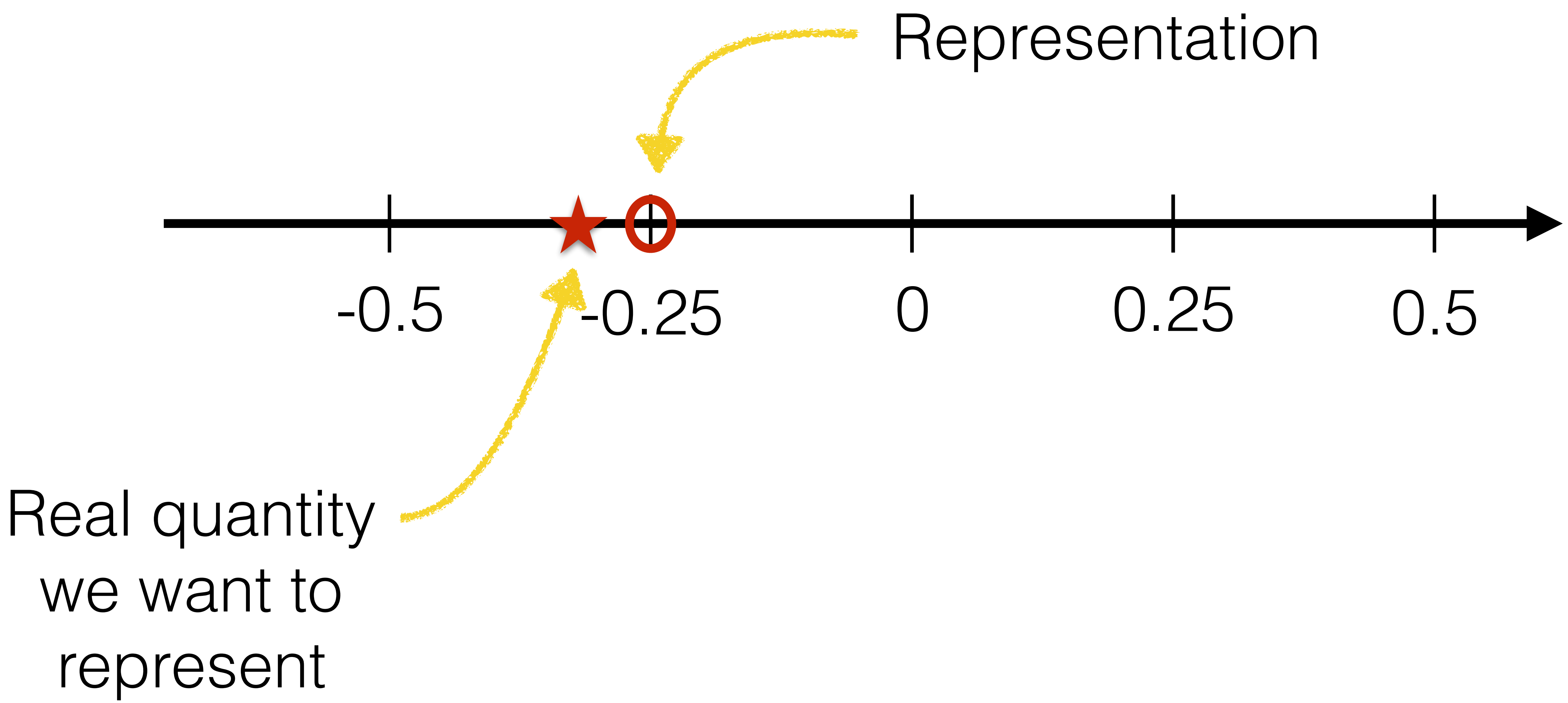


Real quantity we want to represent

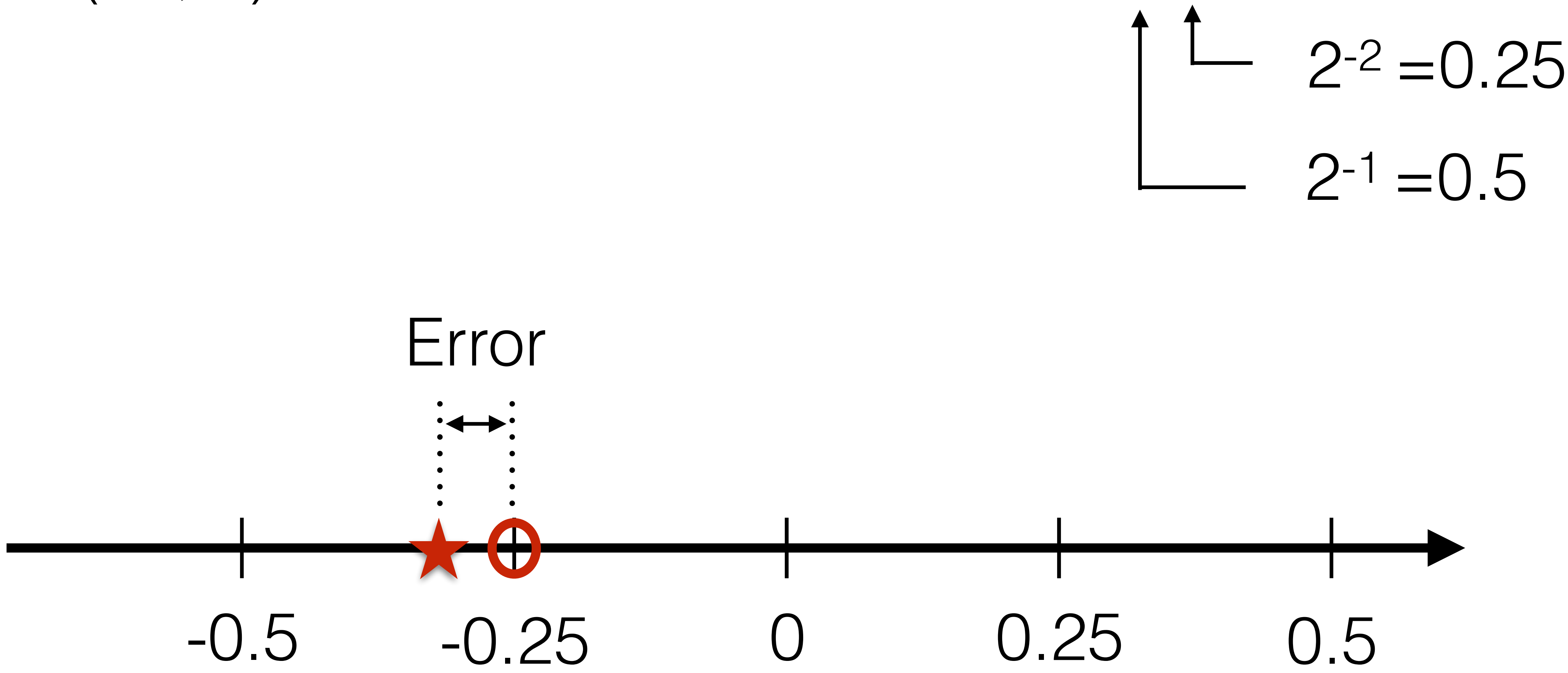
A(13, 2) —> sbbb bbbb bbbb bb . bb

2^{-2} =0.25

2^{-1} =0.5



A(13, 2) —> sbbb bbbb bbbb bb . bb

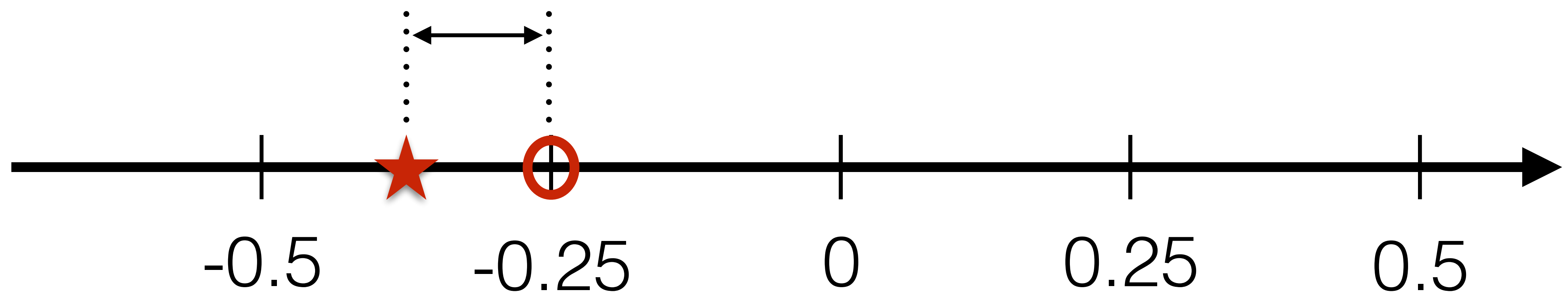


A(13, 2) —> sbbb bbbb bbbb bb . bb

$2^{-2} = 0.25$

$2^{-1} = 0.5$

Largest Error = **Accuracy**



We stopped here last time…

A look at Homework 8

f1

```
File Edit Options Buffers Tools Asm Help
;;; hw8 f1 function
;;; D. Thiebaut

        section .data
s1              db      "Hello world! 123 <>$#", 10, 0
s1Len           equ     $-s1

        section .text
        global  _start
        extern  _printString, _println

_start:
        mov     ebx, s1                 ;ebx points to start of s1
for:    cmp     byte[ebx], 0            ;char at ebx == 0?
        je      done                    ;if so, we're done
        cmp     byte[ebx], 'a'          ;char at ebx lower than 'a'
        jb      next                    ;if so, not interested
        cmp     byte[ebx], 'z'          ;char at ebx higher than 'z'
        ja      next                    ;if so, not interested
        add     byte[ebx], -'a'+'A'     ;else, it's a lowercase, make it
next:                                   ;uppercase
        inc     ebx                     ;point to next char
        jmp     for                     ;loop back

done:
        mov     ecx, s1                 ;we're done, print the string
        mov     edx, s1Len-1
        call    _printString
        call    _println

-UU-:----F1  hw8_f1.asm      Top L1      (Assembler) --------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

getcopy hw8_f1.asm

f1
v2

```
File Edit Options Buffers Tools Asm Help
;;; -------------------------------------------------
;;; f1: takes string address passed in stack at ebp+8
;;;     and transforms in into its uppercase equivalent.
;;;     f1 assumes string is terminated by 0
;;; -------------------------------------------------
f1:     push    ebp
        mov     ebp, esp

        push    ebx                     ;save ebx

;;      mov     ebx, s1
        mov     ebx, dword[ebp+8]

.for:   cmp     byte[ebx], 0 ;char at ebx == 0?
        je      .done                   ;if so, we're done
        cmp     byte[ebx], 'a'          ;char at ebx lower than 'a'
        jb      .next                   ;if so, not interested
        cmp     byte[ebx], 'z'          ;char at ebx higher than 'z'
        ja      .next                   ;if so, not interested
        add     byte[ebx], -'a'+'A'     ;else, it's a lowercase, make it
.next:                                  ;uppercase
        inc     ebx                     ;point to next char
        jmp     .for                    ;loop back

.done:  pop     ebx
        pop     ebp
        ret     4

-UU-:----F1  hw8_f1b.asm     32% L40    (Assembler) -----
```

```
File Edit Options Buffers Tools Asm Help
        section .data
s1      db      "Hello world! 123 <>$#", 10, 0
s1Len   equ     $-s1

        section .text
        global  _start
        extern  _printString, _println

_start: mov     eax, f1
        push    eax
        call    f1

-UU-:----F1  hw8_f1b.asm     4% L7     (Assembler) -----
```

getcopy hw8_f1b.asm

f3

```
File Edit Options Buffers Tools Asm Help
;;; hw8 f3 solution
;;; D. Thiebaut

        section         .data
array    dd       3, 5, 0, 1, 2, 10, 100, 4, 1
arrayLen equ      ($-array)/4 ; figure out the /4 part.

        section .text
        global  _start
        extern  _printInt
        extern  _println
_start:
        mov     ebx, array
        mov     ecx, arrayLen
        mov     eax, 0          ;set counter of even numbers
                                ; to 0

for:    mov     edx,dword[ebx]  ;get int at ebx in edx
        and     edx, 1          ;test last bit of edx for parity
        jnz     next            ;if 1, then odd, skip increment
        inc     eax             ;if 0, then even, increment counter
next:   add     ebx, 4          ;ebx points to next int
        loop    for             ;keep looping...

        call    _printInt       ;print # of even ints found
        call    _println

-UU-:**--F1  hw8_f3.asm     Top L14    (Assembler)
```

getcopy hw8_f3.asm

f3 v2

```
File Edit Options Buffers Tools Asm Help
;;; ------------------------------------------------------------
;;; f3: gets array and array length in stack.
;;;     counts the number of even ints in array and returns
;;;     it in eax.
;;; ------------------------------------------------------------
f3:     push    ebp
        mov     ebp, esp            ;set up stack frame

        push    ebx                 ;save regs used (but not eax)
        push    ecx
        push    edx
                                    ;prepare to loop
        mov     ebx, dword[ebp+12]
        mov     ecx, dword[ebp+8]
        mov     eax, 0              ;set counter of even numbers
                                    ; to 0

.for:   mov     edx,dword[ebx]      ;get int at ebx in edx
        and     edx, 1              ;test last bit of edx for parity
        jnz     .next               ;if 1, then odd, skip increment
        inc     eax                 ;if 0, then even, increment counter
.next:  add     ebx, 4              ;ebx points to next int
        loop    .for                ;keep looping...

        pop     edx                 ;restore regs saved
        pop     ecx
        pop     edx
        pop     ebp                 ;restore old stack frame
        ret     2*4                 ;return and pop 2 dwords

-UU-:**--F1  hw8_f3b.asm     Bot L40     (Assembler) ------
```

getcopy hw8_f3b.asm

Documentation is IMPORTANT!

A word about Hw7a

```
@                @                  @
 @              @ @                @
  @            @   @              @
 @ @          @ @ @ @            @ @
    @        @       @          @
   @ @      @ @     @ @        @ @
  @   @    @   @   @   @      @   @
 @ @ @ @  @ @ @ @ @ @ @ @    @ @ @ @
        @@               @  @
       @@@@             @ @@ @
```

1 more slide!

- **Fixed-Point Format**
- **Floating-Point Format**

Exercise

- What is the accuracy of an U(7,8) number format?

- How good is U(7,8) at representing small numbers versus representing larger numbers? In other words, **is the format treating small numbers *better* than large numbers, or the opposite?**

- **Fixed-Point Format**
- **Floating-Point Format**

IEEE

Advancing Technology for Humanity

The world's largest professional association for the advancement of technology

IEEE Floating-Point Number Format

A bit of history…

http://datacenterpost.com/wp-content/uploads/2014/09/Data-Center-History.png

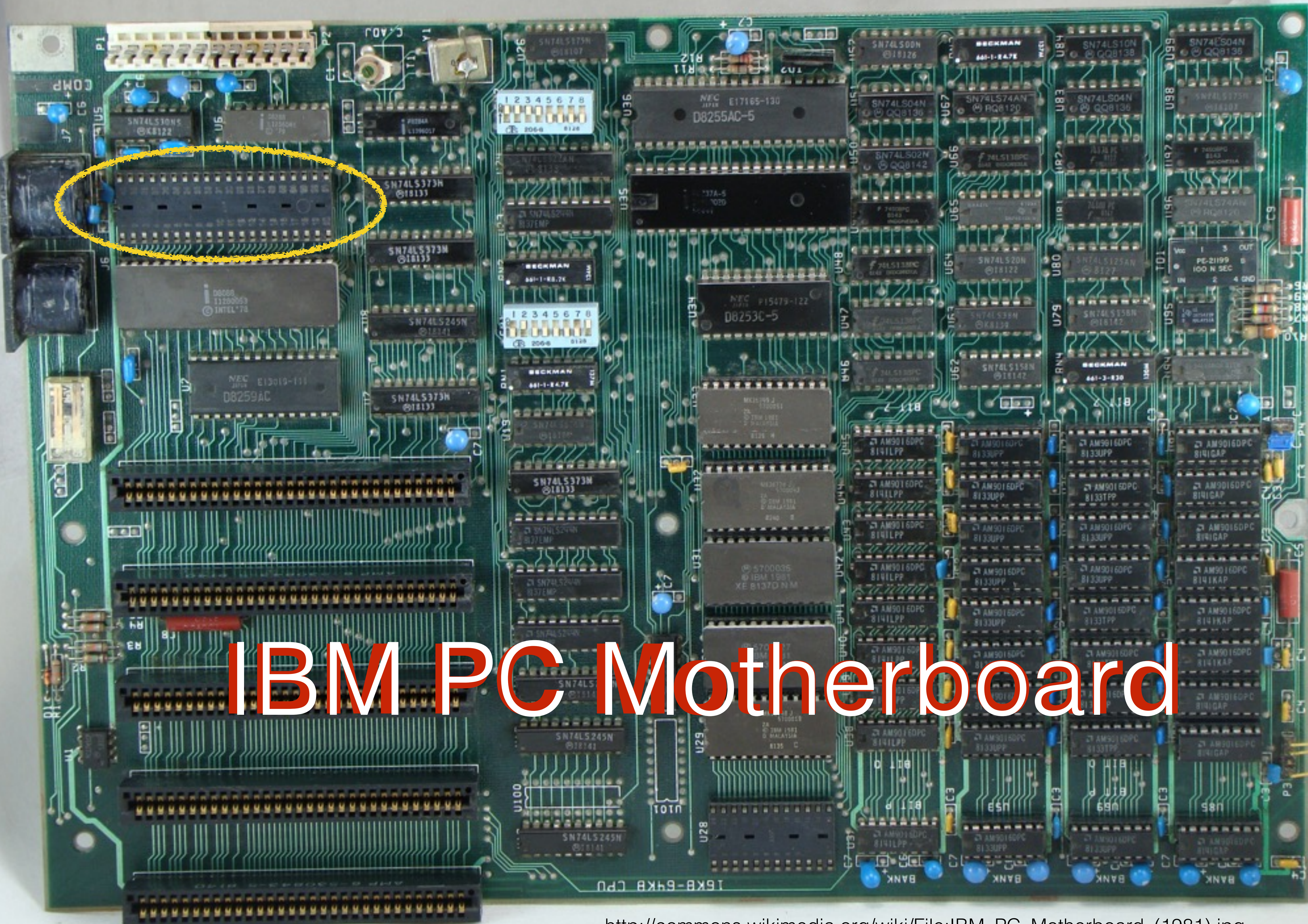
- 1960s, 1970s: many different ways for computers to **represent** and **process** real numbers. Large variation in way real numbers were operated on

- 1976: **Intel** starts design of first hardware floating-point **co-processo**r for 8086. Wants to define a **standard**

- 1977: Second meeting under umbrella of **Institute for Electrical and Electronics Engineers** (IEEE). Mostly microprocessor makers (IBM is observer)

- Intel first to put whole **math library** in a processor

IBM PC Motherboard



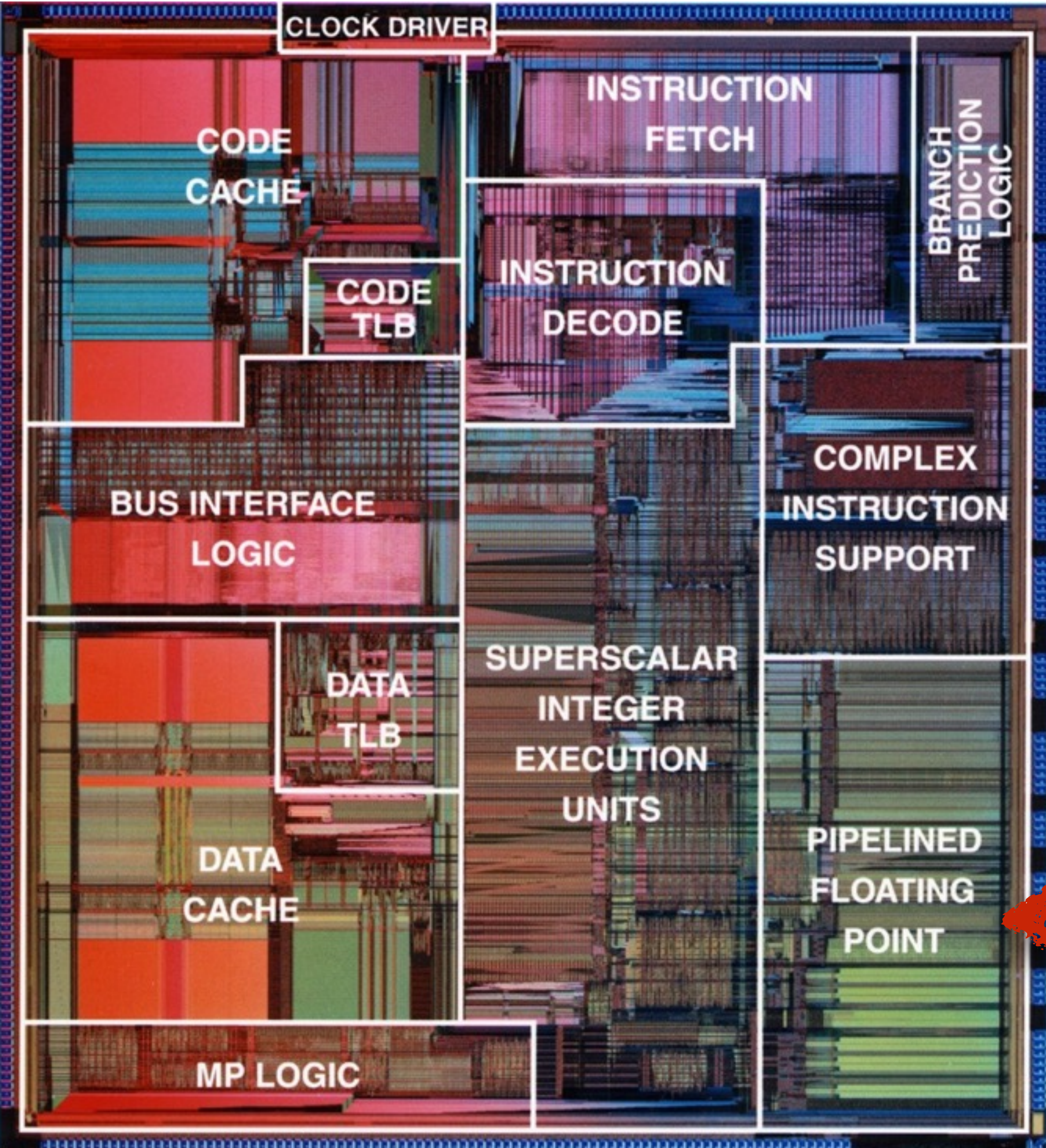
http://commons.wikimedia.org/wiki/File:IBM_PC_Motherboard_(1981).jpg

Intel Coprocessors

Intel		
Processor	Year	Description
8087	1980	Numeric coprocessor for 8086 and 8088 processors.
80C187	19??	Math coprocessor for 80C186 embedded processors.
80287	1982	Math coprocessor for 80286 processors.
80387	1987	Math co-processor for 80386 processors.
80487	1991	Math co-processor for SX versions of 80486 processors.
Xeon Phi	2012	Multi-core co-processor for Xeon CPUs.

Pentium
March 1993

(Ear
Intel Pe



http://semiaccurate.com/assets/uploads/2012/06/1993_intel_pentium_large.jpg

Integrated Coprocessor

Some Processors that do not contain FPUs

- Some ARM processors
- Arduino Uno
- Others

Some Processors that do not contain FPUs

Few people have heard of ARM Holdings, even though sales of devices containing its flavor of chips are projected to be 25 times that of Intel. The chips found in ***99 percent*** *of the world's smartphones and tablets are ARM designs.* ***About 4.3 billion people, 60 percent of the world's population, touch a device carrying an ARM chip each day****.*

Ashlee Vance, Bloomberg, Feb 2014

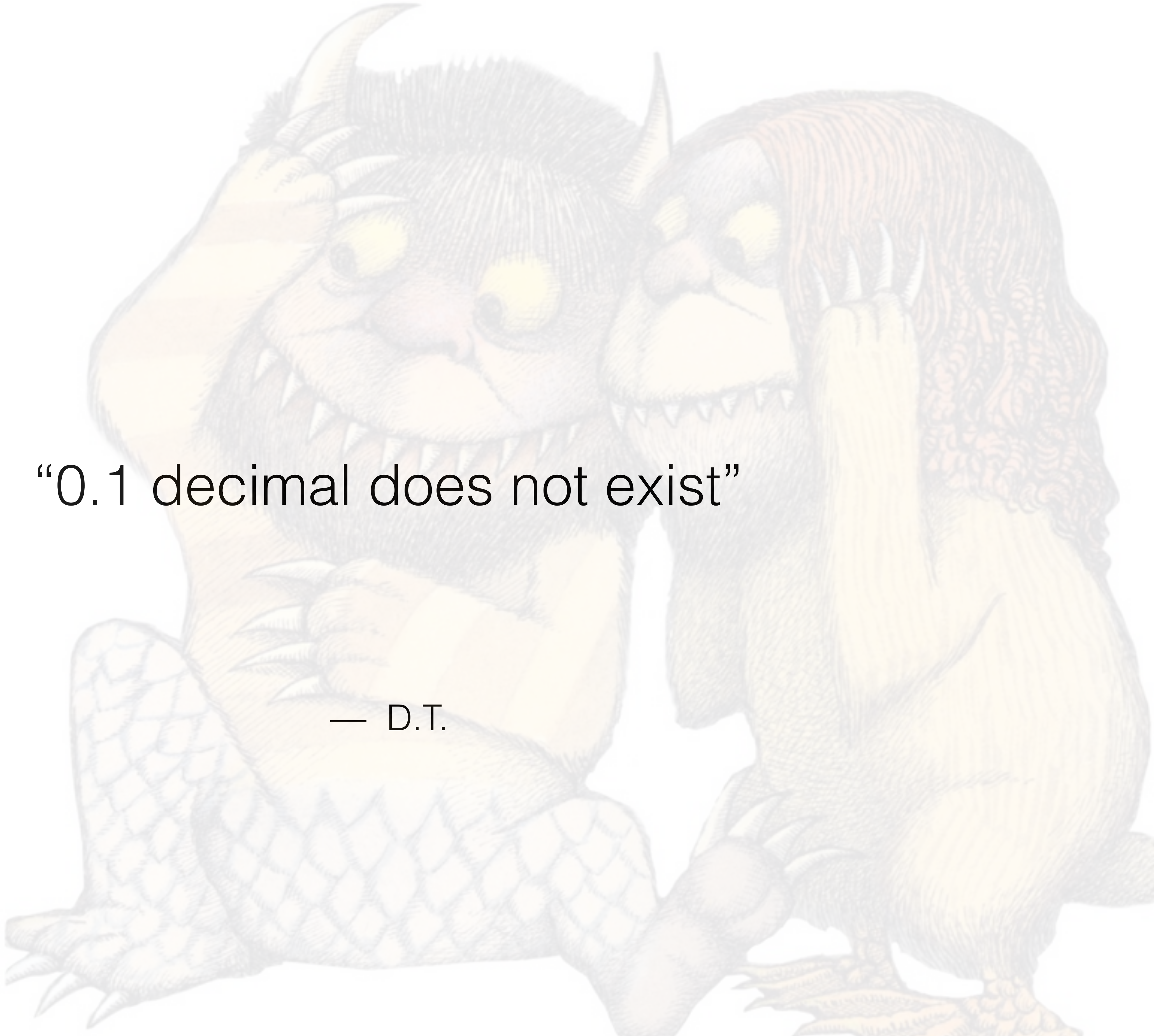
- Some ARM processors
- Arduino Uno
- Others

How Much Slower is Library vs FPU operations?

- Cristina Iordache and Ping Tak Peter Tang, "An Overview of Floating-Point Support and Math Library on the Intel XScale Architecture", In *Proceedings IEEE Symposium on Computer Arithmetic*, pages 122-128, **2003**
- http://stackoverflow.com/questions/15174105/performance-comparison-of-fpu-with-software-emulation

Library-emulated FP operations = **10 to 100 times slower** than hardware FP operations executed by FPU

Floating Point Numbers Are Weird…



"0.1 decimal does not exist"

— D.T.

```java
import java.util.*;

public class SomeFloats {
        public static void main(String args[]) {
            float x = 6.02E23f,
                y = -0.000001f,
                z = 1.23456789E-19f,
                t = -1.0f,
                u = 8000000000f;

            System.out.println(   "\nx = " + x
                                + "\ny = " + y
                                + "\nz = " + z
                                + "\nt = " + t
                                + "\nu = " + u );
        }
}
```

```java
import java.util.*;

public class SomeFloats {
        public static void main(String args[]) {
            float x = 6.02E23f,
                y = -0.000001f,
                z = 1.23456789E-19f,
                t = -1.0f,
                u = 8000000000f;

            System.out.println(   "\nx = " + x
                                + "\ny = " + y
                                + "\nz = " + z
                                + "\nt = " + t
                                + "\nu = " + u );

        }
}
```

```
231b@aurora ~/handout $ java SomeFloats

x = 6.02E23
y = -1.0E-6
z = 1.2345678E-19
t = -1.0
u = 8.0E9
```

$$1.230$$

$$= 12.30\ 10^{-1}$$

$$= 123.0\ 10^{-2}$$

$$= 0.123\ 10^{1}$$

IEEE Format

- 32 bits, single precision (floats in Java)
- 64 bits, double precision (doubles in Java)
- 80 bits*, extended precision (C, C++)

$$x = +/- 1.bbbbbb....bbb \times 2^{bbb...bb}$$

---

* 80 bits in assembly = 1 Tenbyte

10110.01

10110.01

1.011001×2^4

10110.01

1.011001×2^4

1.011001×2^{100}

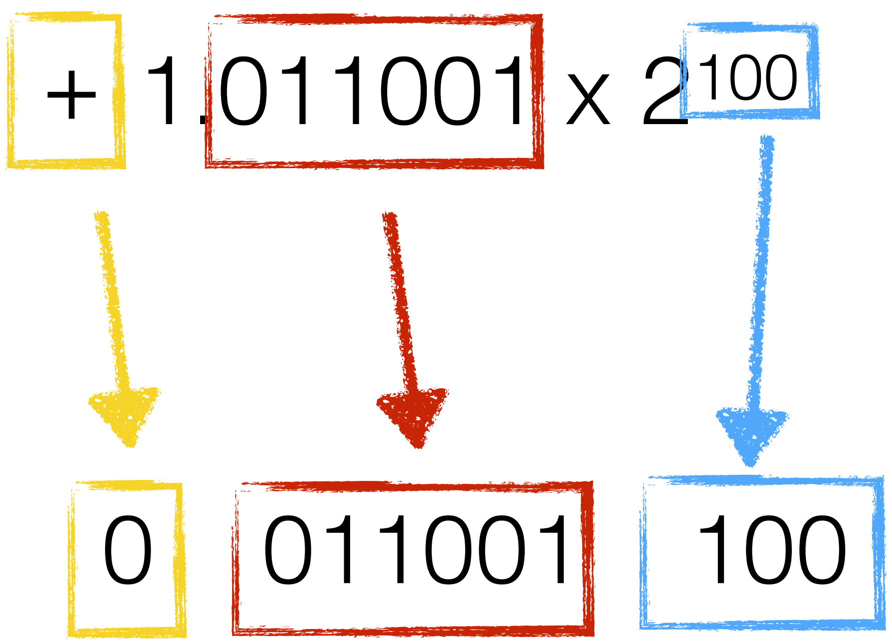
10110.01

1.011001×2^{4}

$+ \ 1.011001 \times 2^{100}$

10110.01

1.011001×2^4



0 011001 100

10110.01

0 | 011001 | 100

Multiplying/Dividing by the Base

In Decimal

1234.56

1234.56 x 10 = 12345.6

12345.6 x 10 = 123456.0

1234.56

1234.56 / 10 = 123.456

123.456 / 10 = 12.3456

Multiplying/Dividing by the Base

In Decimal	In Binary
1234.56	**101.11**
1234.56 x 10 = 12345.6	101.11 x 2 = 1011.1
12345.6 x 10 = 123456.0	1011.1 x 2 = 10111.0
1234.56	**101.11**
1234.56 / 10 = 123.456	101.11 / 2 = 10.111
123.456 / 10 = 12.3456	10.111 / 2 = 1.0111

Multiplying/Dividing by the Base

In Decimal	In Binary	
1234.56	**101.11**	=5.75*d*
1234.56 x 10 = 12345.6	101.11 x 2 = 1011.1	=11.50*d*
12345.6 x 10 = 123456.0	1011.1 x 2 = 10111.0	=23.00*d*
1234.56	**101.11**	=5.75*d*
1234.56 / 10 = 123.456	101.11 / 2 = 10.111	=2.875*d*
123.456 / 10 = 12.3456	10.111 / 2 = 1.0111	=1.4375*d*

Multiplying/Dividing by the Base

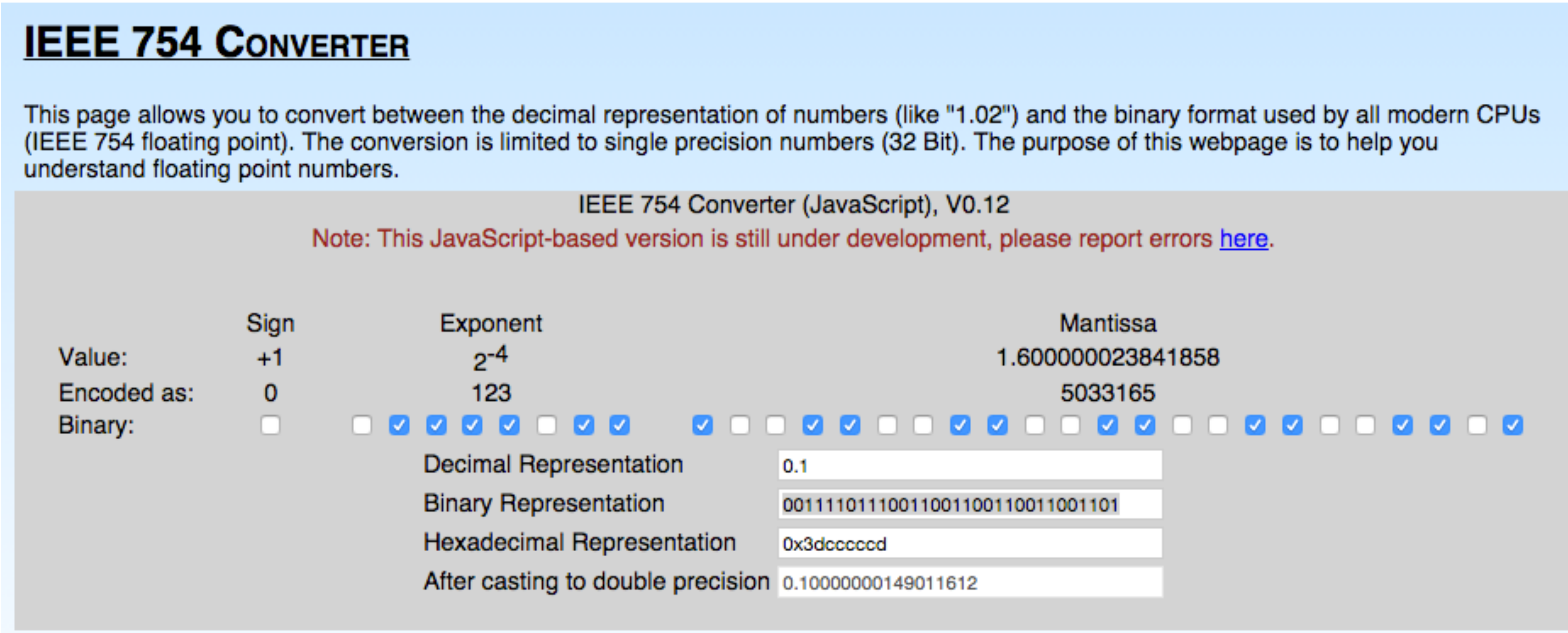
101.11 x 10 = 1011.1

In Decimal	In Binary	
1234.56	**101.11**	=5.75*d*
1234.56 x 10 = 12345.6	101.11 x 2 = 1011.1	=11.50*d*
12345.6 x 10 = 123456.0	1011.1 x 2 = 10111.0	=23.00*d*
1234.56	**101.11**	=5.75*d*
1234.56 / 10 = 123.456	101.11 / 2 = 10.111	=2.875*d*
123.456 / 10 = 12.3456	10.111 / 2 = 1.0111	=1.4375*d*

Observations

$$x = +/- 1.bbbbbb....bbb \times 2^{bbb...bb}$$

- +/- is the sign. It is represented by a bit, equal to **0** if the number is **positive**, **1** if **negative**.
- the part 1.bbbbbb....bbb is called the **mantissa**
- the part bbb...bb is called the **exponent**
- **2** is the **base** for the exponent (could be different!)
- the number is **normalized** so that its binary point is moved to the right of the leading 1
- because the leading bit will always be 1, we don't need to store it. This bit will be an **implied bit**



IEEE 754 Converter

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point). The conversion is limited to single precision numbers (32 Bit). The purpose of this webpage is to help you understand floating point numbers.

IEEE 754 Converter (JavaScript), V0.12

Note: This JavaScript-based version is still under development, please report errors here.

	Sign	Exponent	Mantissa
Value:	+1	2^{-4}	1.600000023841858
Encoded as:	0	123	5033165
Binary:	0	01111011	10011001100110011001101

Decimal Representation	0.1
Binary Representation	00111101110011001100110011001101
Hexadecimal Representation	0x3dcccccd
After casting to double precision	0.10000000149011612

http://www.h-schmidt.net/FloatConverter/IEEE754.html