Floating Point Arithmetic

(The IEEE Standard)

Floating Point Arithmetic (and The IEEE Standard)

* **Floating Point Arithmetic**
  - Representations
  - Issues
  - Normalized, Unnormalized, Subnormal
  - Precision
  - Wobble

* **The IEEE Standard**
  - Why
  - What it contains, what it doesn't contain
  - Formats
  - Rounding
  - Operations
  - Infinities, NANs
  - Exceptions
  - Traps

Several Issues Come Up:

* How many bits for range, how many bits for precision?
* What to do with numbers too small to represent with this scheme?
* What to do with numbers that do not correspond to exact representations?
* What to do with numbers too large to be represented?
* Shall we distinguish numbers too large with true infinities?
* What about nonsense numbers?

  (Examples:

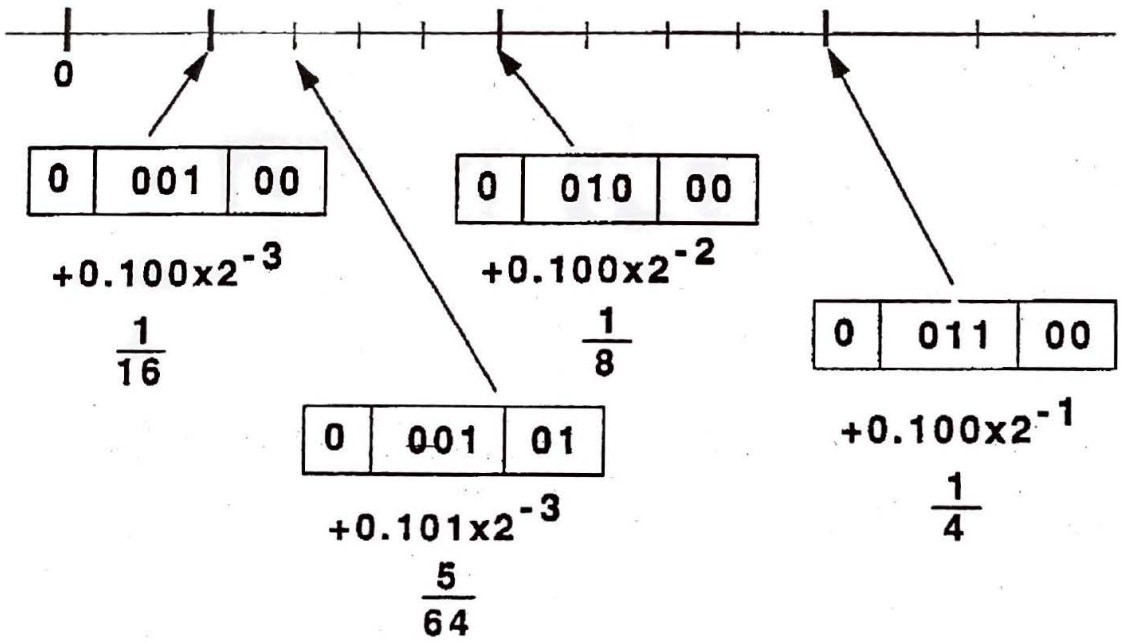
  $$\text{Arcsin } 2, \frac{0}{0}, \infty - \infty)$$

First, An Example

We Simplify:

S	EXP	FRA

In DEC format: $(-1)^S * 0.1\,\underline{fra} * 2^{EXP-4}$

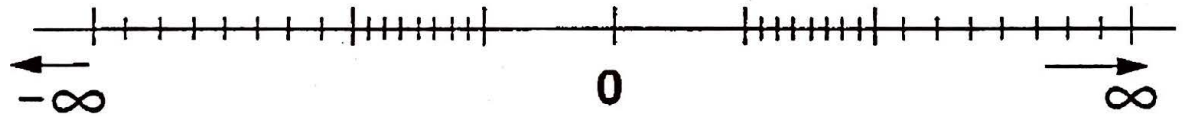


First, Some General Stuff:

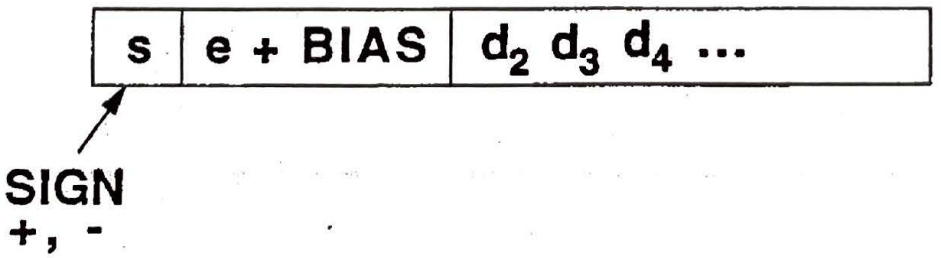
A number can be represented as

$$\pm\, d_0 . d_1 d_2 \ldots \beta^e$$

These numbers correspond to points on the real line. If we insist that all representations be normalized, then the points are shown (normalized can mean: $d_0 = \not{0}$, $d_1 = 1$)



(We can, incidentally, store the number in signed-magnitude format:)



Normalized, Unnormalized, Subnormal

Again, we are looking at $\pm\ d_0.d_1 d_2 \ldots * \beta^e$

1. If it is <u>normalized</u>, it is:

$$\pm\ 0.1\ d_2\ d_3\ \cdots * \beta^e$$

2. <u>Unnormalized</u> (after a subtract of like signs, for example)

$$\pm\ 0.0001\ d_2\ d_3\ \cdots * \beta^{e+3}$$

3. <u>Subnormal</u> means it can't be represented in the machine in normalized format

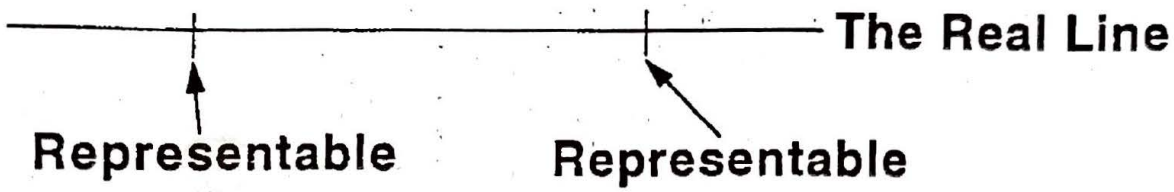
   - Recall the format

±	e+BIAS	$d_2\ d_3$ ...

   Corresponds to $\pm\ 0.1\ d_2 d_3 \ldots * \beta^e$

   - Suppose we successively divide by β. We can do this <u>until</u> e+BIAS = 1. Below that we can't represent numbers (except 0). Why? Suppose we let e+BIAS = 0. How do we now represent 0?

Precision

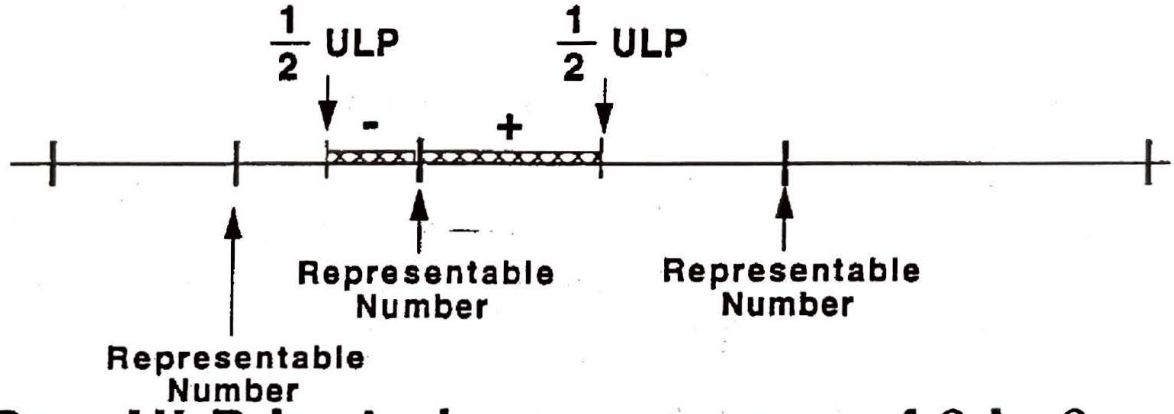


* Uncertainty is at Most: $\frac{1}{2}$ ULP

* Precision deals with worst unavoidable error

* Precision is a function of representation
  Accuracy is a function of your algorithm

* Relative uncertainty (the issue of wobble)



One ULP just above a power of β is β times as large as one ULP just below.

The IEEE Standard

Reasons:

1. **Direct Support for:**
   - **Execution-time diagnosis of anomalies**
   - **Smoother handling of exceptions**
   - **Interval arithmetic at reasonable cost**

2. **Provide for development of:**
   - **Standard elementary functions**
   - **Very high precision arithmetic**
   - **Coupling of numeric & symbolic computation**

The IEEE Standard (Continued)

What does it contain:

- Formats: single, double, extended
- Operations: +,-, *, ÷, $\sqrt{\ }$, REM,CMP
- Rounding modes
- Conversion: Int/Fl., Dec/Fl., Fl/Fl
- Exceptions: Underflow, Overflow, Div Ø, Inexact, Invalid

What it does not contain:

- Requirements for implementation in HDWR or SFWR
- Interpretation of NaNs
- Formats for Integers, BCD
- Conversions other than above

The Formats

There are four; we start with one as an example.

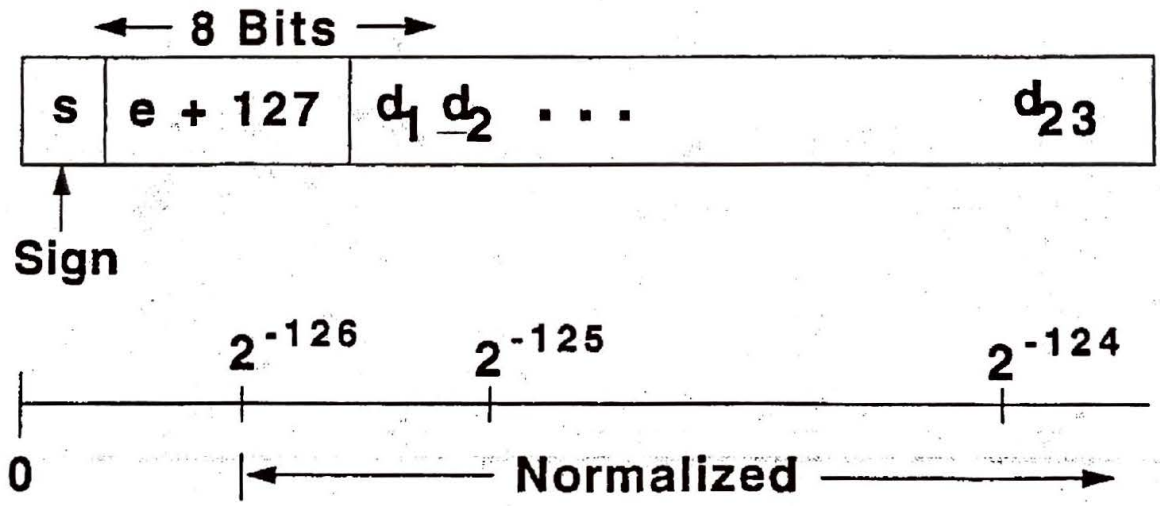
Single

Representable Numbers:

* Normalized

$$1.d_1 d_2 d_3 \ldots d_{23} * 2^e$$

where $-126 \leq e \leq +127$



Note: The range of exponents

$$-126 \leq e \leq +127$$

Coupled with the BIAS (127) which is added to the exponent yields an 8 bit string from 00000001

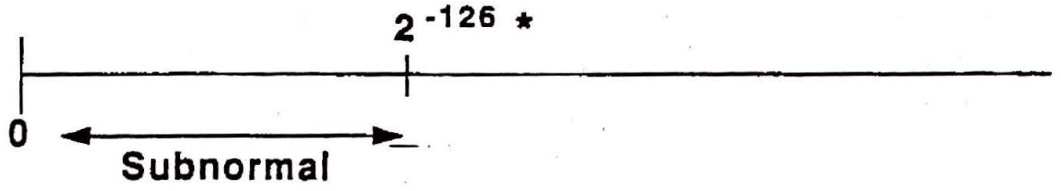
⋮

11111110

Two strings remain: 00000000, 11111111

* Subnormal numbers (Exp field = 00000000)

$$0.d_1 d_2 \ldots d_{23} \quad 2^{-126}$$



* Infinities (Exp field = 11111111)

s	11111111	000 ... 0

Formats (Continued)

That still leaves those strings characterized:

s	11111111	Not Zero

These are defined as NaNs.

They result from invalid operations

(Like,: $\frac{0}{0}, \frac{\infty}{\infty}, \infty - \infty$**)**

Generalizing to the other formats

	Single	Single-X	Double	Double-X
Precision	24 bits	≥32	53	≥64
Exponent	8 bits	≥11	11	≥15
Word Length	32 bits	≥43	64	≥79
Exp BIAS	+127	--	+1023	--
e_{max}	+127	≥1023	+1023	≥16382
e_{min}	-126	≤-1022	-1022	≤-16382

Rounding

1st We perform the operation & produce the infinitely precise result

2nd We round to fit it into the destination format

Four Rounding Modes

1. Default: To nearest. If equally near, then to the one having A Ø in LSB

2. Directed roundings

   - Toward + ∞
   - Toward - ∞
   - Toward Ø (Chop)

Operations

* Arithmetic: +, - , * , ÷, REM

When $y \neq \emptyset$, r = x REM y, is defined:

r = x-y * n, where n is the integer nearest $\frac{x}{y}$

whenever $\left| n - \frac{x}{y} \right| = \frac{1}{2}$, then n is EVEN

∴ Remainder is always exact

* Square root: Result defined if ARG $\geq \emptyset$.

* Conversion from one format to another

  - To fewer bits: rounded
  - To more bits: exact

Operations (Continued)

- ★ Conversion Fl. Pt. <----> Integers
  Binary <----> Decimal

- ★ Comparison
  - Always exact
  - Never underflow, overflow
  - Four relations are possible
    {>, =, <, unordered}

Note: Invalid is signaled if unordered operands are compared and unordered is not the basis but > or < is the basis.

Examples:

Predicate	>	<	=	?	Invalid if unordered
=	F	F	T	F	No
? ≠	T	T	F	T	No
>	T	F	F	F	Yes
?<≠	F	T	T	T	No

Infinities, NaNs, ±Ø

∞:

* - ∞ < (finite) < + ∞
* Arithmetic on ∞ is <u>exact</u>
* ∞ is created by
  - Overflow
  - "Divide by zero"

NaN:

* Signaling & Quiet

  <u>Signaling</u> - Reserved operand that signals the invalid Op. Exception for all operations in the standard. If no trap occurs, a quiet NaN is delivered

  <u>Quiet</u> - Operations on quiet NaNs produce quiet NaNs. They provide hooks to retrospective diagnostic information.

Exceptions

When detected: Take Trap, or
Set Flag, or
Both

Flag can be reset only under program control

* **Invalid**
  - **Operation on a signaling NaN.**
  - **$\infty - \infty$ O/O**
  - **$0 * \infty$ ∞ / ∞**
  - **x REM y, where y=0 or x= ∞**
  - **$\sqrt{NEG}$**
  - **Conversion from Fl. to int. or decimal, when overflow, infinity, or NaN prevents the conversion**
  - **Comparison via predicates involving > or <, and Not?, when the operands are unordered**

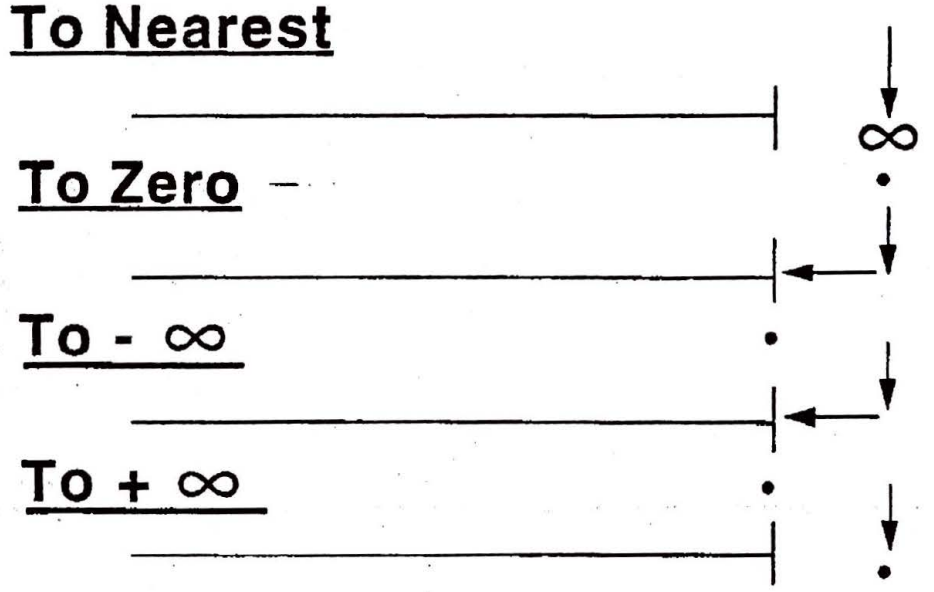
Exceptions (Continued)

* **Divide by zero**

  When f(finite) --> Infinite and exact

* **Overflow**

  When the destinations largest finite number is exceeded by what would have been the rounded floating point result if the exponent range were unbounded



Exceptions(Continued)

* **Overflow (Continued)**

  Trapped overflows! [Except for conversions]

  1st, Divide infinitely precise Result by 2^a

$$a = \begin{array}{lll} \text{Single} & \text{Double} & \text{Extended} \\ \hline 192 & 1536 & 3 * 2^{n-2} \end{array}$$

n = | exponent bits |

Why?

* **Underflow**

  - **Tiny value (which could cause subsequent overflow)**

  - **Loss of precision**

Delivered result may be zero, subnormal No., or $\pm 2^{\text{min-exp}}$

Exceptions (Continued)

* **Underflow (continued)**

  Trapped underflows!

  [All operations except conversions]

  1st, Multiply infinitely precise Result by 2^a

* **Inexact**

  When the result of an operation is not exact, or on non-trapped overflow.

Traps

For any of the five exceptions, a user should be able to:

- **Specify a handler**
- **Request that an existing handler be disabled, saved, restored.**

When a system traps, the trap handler should be able to determine:

- **Which exception occurred on this operation**
- **The kind of operation being performed**
- **The destination format**
- **In overflow, underflow, & inexact, the correctly rounded result**
- **In invalid & divide by zero, the operand values**