2nd Lecture: Basic properties of Floating-Point Arithmetic

Jean-Michel Muller

CNRS - Laboratoire LIP

http://perso.ens-lyon.fr/jean-michel.muller/

Base β Floating-Point System

Parameters:

$$\begin{cases} \text{radix (or base)} & \beta \geq 2 \quad \text{(in practice } \beta = 2 \text{ or } 10\text{)} \\ \text{precision} & p \geq 1 \\ \text{extremal exponents} & e_{\min}, e_{\max} \quad \text{(in practice } e_{\min} = 1 - e_{\max}\text{)} \end{cases}$$

A Floating-Point number (FPN) x is represented by 2 integers:

- integral significand: M, $|M| \leq \beta^p - 1$;
- exponent e, $e_{\min} \leq e \leq e_{\max}$.

such that

$$x = M \cdot \beta^{e+1-p}$$

with e smallest under these constraints (necessary for unicity: $1200 \times 10^{-2} = 12 \times 10^0$).

$$\Rightarrow |M| \geq \beta^{p-1}, \text{ unless } e = e_{\min}.$$

Base β Floating-Point System

- Fractional significand of x (sometimes called mantissa): the number $m = M \cdot \beta^{1-p}$, so that $x = m \cdot \beta^e$.
- normal number: of absolute value $\geq \beta^{e_{\min}}$. The absolute value of its integral significand is $\geq \beta^{p-1}$. Example with $\beta = 10$, $e_{\min} = -99$ and $p = 4$:

$$4235 \times 10^{0-3} = 4.235 \times 10^0.$$

- subnormal number: of absolute value $< \beta^{e_{\min}}$. The absolute value of its integral significand is $< \beta^{p-1}$. Example with $\beta = 10$, $e_{\min} = -99$ and $p = 4$:

$$3 \times 10^{-99-3} = 0.003 \times 10^{-99}.$$

Subnormal numbers represented with worse accuracy: graceful underflow.

Base β Floating-Point System

In practice: normality/subnormality is encoded in the exponent field.

Radix 2: the leftmost bit of the significand is always:

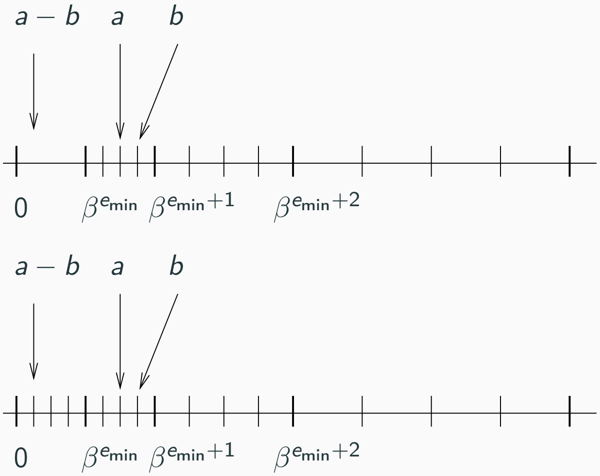
- a "1" for a normal number,
- a "0" for a subnormal number.

$\rightarrow$ no need to store it (implicit 1 convention).

Exercise:

- what is the largest representable number Ω?
- let $x \in [\beta^k, \beta^{k+1})$, with $k \geq e_{\min}$. Assume x is a FPN. The number x^+ (FP successor of x) is the smallest FPN $> x$. What is the value of $x^+ - x$?
- in the binary32 format, $\beta = 2$, $e_{\max} = 127$, and $p = 24$. What is the representation of 1? What is the smallest positive FP number?

Subnormal numbers difficult to implement efficiently, but...



If a and b are FPN, $a \neq b$ equivalent to "computed $a-b \neq 0$".

Theorem 1 (Hauser)

If the absolute value of the sum/difference of two floating-point numbers is $\leq \beta^{e_{min}+1}$ then it is a floating-point number (i.e., it is exactly representable in FP arithmetic).

$$\left.\begin{aligned} a &= M_a \cdot \beta^{e_a - p + 1} \\ b &= M_b \cdot \beta^{e_b - p + 1} \\ e_a, e_b &\geq e_{min} \end{aligned}\right\} \Rightarrow$$

a and b are multiple of $\beta^{e_{min} - p + 1}$ $\Rightarrow$ $a + b$ is multiple of $\beta^{e_{min} - p + 1}$

$$|a + b| = K \cdot \beta^{e_{min} - p + 1}$$

It is $\leq \beta^{e_{min}+1} \Rightarrow K \leq \beta^{p}$

① $K \leq \beta^{p} - 1 \Rightarrow$ Integral significant K & exponent e_{min}

② $K = \beta^{p} \Rightarrow |a + b| = \beta^{e_{min}+1}$ which is a FPN

Before 1985: a total mess. . .

Machine	Underflow λ	Overflow Λ
DEC PDP-11, VAX, F and D formats	$2^{-128} \approx 2.9 \times 10^{-39}$	$2^{1}27 \approx 1.7 \times 10^{3}8$
DEC PDP-10; Honeywell 600, 6000; Univac 110x single; IBM 709X, 704X	$2^{-129} \approx 1.5 \times 10^{-39}$	$2^{1}27 \approx 1.7 \times 10^{3}8$
Burroughs 6X00 single	$8^{-51} \approx 8.8 \times 10^{-47}$	$8^{76} \approx 4.3 \times 10^{68}$
H-P 3000	$2^{-256} \approx 8.6 \times 10^{-78}$	$2^{256} \approx 1.2 \times 10^{77}$
IBM 360, 370; Amdahl1; DG Eclipse M/600; ...	$16^{-65} \approx 5.4 \times 10^{-79}$	$16^{63} \approx 7.2 \times 10^{75}$
Most handheld calculators	10^{-99}	10^{100}
CDC 6X00, 7X00, Cyber	$2^{-976} \approx 1.5 \times 10^{-294}$	$2^{1070} \approx 1.3 \times 10^{322}$
DEC VAX G format; UNIVAC, 110X double	$2^{-1024} \approx 5.6 \times 10^{-309}$	$2^{1023} \approx 9 \times 10^{307}$

Source: Kahan, *Why do we need a Floating-Point Standard*, 1981.

Before 1985: a total mess...

- on some Cray supercomputers, overflow in multiplication was computed just from the exponent of the entries, in parallel with the actual computation of the product;

→ `1 * x` could overflow;

- still on the Crays, only 12 bits of x were examined to detect a division by 0 when computing y/x

→ `if (x = 0) then z := 17.0 else z := y/x`
could lead to a “zero divide” error message...
but since the multipler too examined only 12 bits to decide if an operand is zero,

```
if (1.0 * x = 0) then z := 17.0 else z := y/x
```

was just fine.

Writing reliable and portable numerical software was a challenge!

IEEE-754 Standard for FP Arithmetic (1985, 2008, 2019)

- put an end to a mess (no portability, variable quality);
- leader: W. Kahan (father of the arithmetic of the HP35 and the Intel 8087);
- formats (in radices 2 and 10);
- specification of operations and conversions;
- exception handling (max+1, 1/0, $\sqrt{-2}$, 0/0, etc.);
- successive versions of the standard: 2008, 2019, more to come.

IEEE-754 Standard for FP Arithmetic (1985, 2008, 2019)

Name	binary16	binary32 (basic)	binary64 (basic)	binary128 (basic)
Former name	N/A	single precision	double precision	N/A
p	11	24	53	113
$e_{\max}$	$+15$	$+127$	$+1023$	$+16383$
$e_{\min}$	-14	-126	-1022	-16382

Table 1: Main parameters of the binary interchange formats of size up to 128 bits specified by the 754-2019 Std. In some articles and software libraries, 128-bit formats were sometimes called "quad precision".

IEEE-754 Standard for FP Arithmetic (1985, 2008, 2019)

Name	decimal32	decimal64 (basic)	decimal128 (basic)
p	7	16	34
$e_{\max}$	$+96$	$+384$	$+6144$
$e_{\min}$	-95	-383	-6143

Table 2: Main parameters of the decimal interchange formats of size up to 128 bits specified by the 754-2019 Std.

Standardizing has some interest. . .

- The Mars Climate Orbiter probe crashed on Mars in 1999;
- one of the software teams assumed the unit of length was the meter;
- the other team assumed it was the foot.

Roundings

- in general, the sum, product, quotient. . . of two FP numbers is not a FP number $\rightarrow$ it must be rounded;
- first systems: the only information was that the computed result was “close to” the exact result;
- now: a rounding function is defined, and the computed result is obtained by applying that function to the exact result

$\rightarrow$ notion of correct rounding:

- easy to guarantee for $+$, $-$, $\div$, $\times$, $\sqrt{}$;
- very difficult for sin, exp, Γ, etc.

Rounding function

Denote $\mathbb{F}$ the set of FP numbers in a given system $(\beta, p, e_{\min}, e_{\max})$.
A function $r : \mathbb{R} \to \mathbb{F}$ is a rounding function if:

- $\forall x \in \mathbb{F}, r(x) = x$;
- $x_1 \leq x_2 \Rightarrow r(x_1) \leq r(x_2)$.

Here are some usual rounding functions:

- round toward $-\infty$: $\mathrm{RD}(x)$ is the largest FP number $\leq x$;
- round toward $+\infty$: $\mathrm{RU}(x)$ is the smallest FP number $\geq x$;
- round toward zero: $\mathrm{RZ}(x) = \begin{cases} \mathrm{RD}(x) & \text{if} \quad x \geq 0, \\ \mathrm{RU}(x) & \text{if} \quad x < 0; \end{cases}$
- round to nearest: $\mathrm{RN}(x) =$ FPN closest to x. If x halfway between two consecutive FPNs, a tie-breaking rule is needed.

Classical tie-breaking rules for RN (round to nearest)

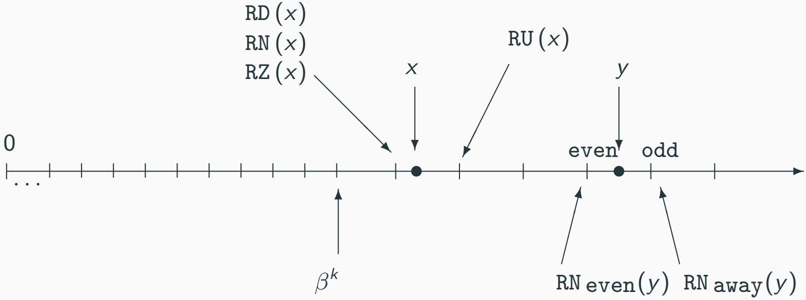
When x is exactly halfway between 2 consecutive FP numbers:

- ties-to-even: choose the one whose integral significand is even (default);
- ties-to-away: choose the one with largest magnitude;
- ties-to-zero: choose the one with smallest magnitude.

These 3 rules are easy to implement in hardware, and with them we have

- $\mathrm{RN}(-x) = -\mathrm{RN}(x)$,
- $\mathrm{RN}(2^k x) = 2^k\,\mathrm{RN}(x)$ (x and $2^k x$ in normal range), and
- x multiple of $2^k \Rightarrow \mathrm{RN}(x)$ is multiple of 2^k.

The standard rounding functions



Here we assume that the real numbers x and y are positive.

Correct rounding

- IEEE 754 specifies the rounding functions RD, RU, RZ, and RN with ties-to-even and ties-to-away (ties-to-zero allowed for some special, not-yet-implemented operations);
- the user can choose the rounding function (not always simple);
- the default function is RN ties to even.

Correctly rounded operation: returns what we would get by exact operation followed by rounding.

- correctly rounded $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$ are required;
- correctly rounded sin, cos, exp, ln, etc. are only recommended (not mandatory).

$\rightarrow$ in practice, when the operation `c = a + b` appears in a program, we obtain $c = \mathrm{RN}(a + b)$.

Correct rounding

IEEE-754 (since 1985): Correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{}$ and some conversions. Advantages:

- if the result of an operation is exactly representable, we get it;
- if we just use the 4 arith. operations and $\sqrt{}$, deterministic arithmetic: one can elaborate algorithms and proofs that use the specifications;
- accuracy and portability are improved;
- playing with rounding towards $+\infty$ and $-\infty \rightarrow$ guaranteed lower and upper bounds: interval arithmetic.

FP arithmetic becomes a structure in itself, that can be studied.

Just for the fun: what does this program?

$\circ$ is any of RD, RU, RZ, or RN.

$A \leftarrow 1.0$
$B \leftarrow 1.0$
while $\circ(\circ(A + 1.0) - A) = 1.0$ **do**
    $A \leftarrow \circ(2 \times A)$
end while
while $\circ(\circ(A + B) - A) \neq B$ **do**
    $B \leftarrow \circ(B + 1.0)$
end while
return B

It computes β

$A \leftarrow 1.0$
$B \leftarrow 1.0$
while $\circ(\circ(A + 1.0) - A) = 1.0$ **do**
  $A \leftarrow \circ(2 \times A)$
end while
while $\circ(\circ(A + B) - A) \neq B$ **do**
  $B \leftarrow \circ(B + 1.0)$
end while
return B

Assume radix β, precision p. Define A_i = value of A after i^{th} execution of the first while loop.

Consider the first while loop:

- Induction $\rightarrow$ if $2^i \leq \beta^p - 1$, then $A_i = 2^i$ exactly. Gives $A_i + 1 \leq \beta^p \rightarrow \circ(A_i + 1.0) = A_i + 1$.
- Hence, $\circ(\circ(A_i + 1.0) - A_i) = \circ((A_i + 1) - A_i) = 1$. Therefore while $2^i \leq \beta^p - 1$, we stay in the 1st loop.
- Consider the smallest j s.t. $2^j \geq \beta^p$. We have $A_j = \circ(2A_{j-1}) = \circ(2 \times 2^{j-1}) = \circ(2^j)$. Since $\beta \geq 2$, we conclude
$$\beta^p \leq A_j < \beta^{p+1}.$$
- Hence, the FP successor of A_j is $A_j + \beta \rightarrow \circ(A_j + 1.0)$ is either A_j or $A_j + \beta$ therefore $\circ(\circ(A_j + 1.0) - A_j)$ is 0 or β: in any case it is $\neq 1.0 \rightarrow$ we exit the loop.

At the end of the 1st while loop, A satisfies $\beta^p \leq A < \beta^{p+1}$.

It computes β

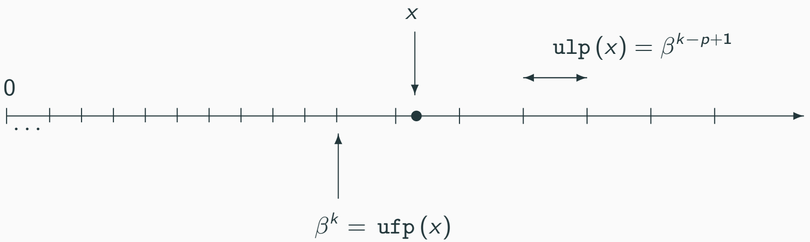
$A \leftarrow 1.0$
$B \leftarrow 1.0$
while $\circ(\circ(A + 1.0) - A) = 1.0$ **do**
    $A \leftarrow \circ(2 \times A)$
end while
while $\circ(\circ(A + B) - A) \neq B$ **do**
    $B \leftarrow \circ(B + 1.0)$
end while
return B

Consider the 2nd while loop.

- We have seen that the FP successor of A is $A + \beta$.
- Hence, while $B < \beta$, $\circ(A + B)$ is either A or $A + \beta \rightarrow \circ(\circ(A + B) - A)$ is 0 or β: in any case, we do not exit the loop.
- As soon as $B = \beta$, $\circ(A + B)$ is $A + B$ exactly, so $\circ(\circ(A + B) - A) = B$.

We exit the 2nd loop as soon as $B = \beta$

Some useful notions



- this figure: We assume that x is in the normal range: $\beta^{e_{\min}} \leq x \leq \Omega$;
- distance between consecutive FPNs of absolute value $< \beta^{e_{\min}}$: $\beta^{e_{\min}-p+1}$.

ulp ("unit in the last place") and ufp ("unit in the first place")

Definition 2 (ulp function)

If $|x| \in [\beta^e, \beta^{e+1})$, then $\text{ulp}(x) = \beta^{\max\{e, e_{\min}\}-p+1}$.

It is the distance between consecutive FP numbers in the neighborhood of x.

Properties:

- $|x - \text{RN}(x)| \leq \frac{1}{2}\text{ulp}(x)$;
- $|x - \text{RU}(x)|$, $|x - \text{RD}(x)|$, and $|x - \text{RZ}(x)|$ are $< 1\,\text{ulp}(x)$;
- if x is a FP number then it is an integer multiple of $\text{ulp}(x)$.

Function ulp is frequently used for expressing errors.

Definition 3 (ufp function)

If $|x| \in [\beta^e, \beta^{e+1})$, then $\text{ufp}(x) = \beta^e$.

Relative error due to rounding

The most frequent measure of error in numerical computing is relative error. If $\hat{x}$ approximates an exact value x, it is defined as:

- $\left|\frac{\hat{x}-x}{x}\right|$ if $x \neq 0$;
- 0 if $\hat{x} = x = 0$;
- $+\infty$ if $x = 0$ and $\hat{x} \neq 0$.

In practice, when the relative error is ≥ 1 this means we have lost all information on x (we even do not know its sign).

Relative error due to rounding

- if x is in the normal range (i.e., $\beta^{e_{\min}} \leq |x| \leq \Omega$), then

$$|x - \mathrm{RN}(x)| \leq \frac{1}{2}\,\mathrm{ulp}(x) = \frac{1}{2}\beta^{\lfloor \log_\beta |x| \rfloor - p + 1},$$

  therefore,

$$|x - \mathrm{RN}(x)| \leq u \cdot |x|, \tag{1}$$

  with $u = \frac{1}{2}\beta^{-p+1}$ (base 2, $u = 2^{-p}$). Hence the relative error

$$\frac{|x - \mathrm{RN}(x)|}{|x|}$$

  (for $x \neq 0$) is $\leq u$.

- u, called rounding unit is frequently used for expressing errors.
- Exercise (for next time): show that in (1), u can be replaced by $\frac{u}{1+u}$.

Relative error due to rounding

- similarly, for x in the normal range, $|x - \circ(x)|$, where $\circ \in \{\,\mathrm{RU}\,, \mathrm{RD}\,, \mathrm{RZ}\,\}$, is $\leq \beta^{-p+1}$;
- if $|x|$ is below $\beta^{e_{\min}}$ (subnormal range), we only have

$$|x - \mathrm{RN}\,(x)| \leq \frac{1}{2}\beta^{e_{\min}-p+1},$$

(i.e., the relative error can be much larger than u).

Consequence: the "standard model"

- correctly rounded operation $\top \in \{+, -, \times, \div\}$;
- a, b FP numbers;
- if $a \top b$ is in the normal range, then the computed result $\hat{r} = \text{RN}(a \top b)$ and the exact result $r = a \top b$ satisfy

$$\hat{r} = r \cdot (1 + \epsilon_1),$$

  with $|\epsilon_1| \leq u$;
- very similarly, we have

$$\hat{r} = \frac{r}{1 + \epsilon_2},$$

  with $|\epsilon_2| \leq u$.

Hauser's theorem $\Rightarrow$ if $\top \in \{+, -\}$ these relations hold even in the subnormal range.

The FMA (*Fused Multiply-Add*) instruction

- added in 2008 to the standard, now implemented in all commercially-significant processors;
- computes $\circ(ab + c)$, where $\circ$ is the chosen rounding function;
- allows faster (and frequently more accurate) complex multiplication, division, evaluation of polynomials or dot-products;
- facilitates the implementation of correctly-rounded division and square-root using slightly modified Newton-Raphson iterations.