3nd Lecture: Some Algorithms and Properties in Floating-Point Arithmetic

Jean-Michel Muller

CNRS - Laboratoire LIP

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

Summary of the previous episodes

- radix-β, precision-p FP number:

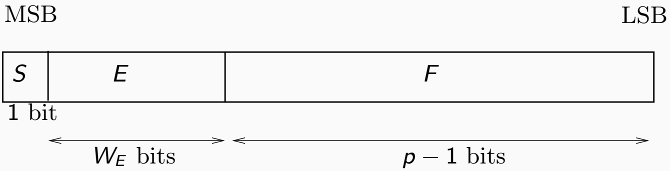
$$M \times \beta^{e-p+1},$$

  with $e_{\min} \leq e \leq e_{\max}$.
- round to nearest: $\mathrm{RN}(x)$ = FPN closest to x. If x halfway between two consecutive FPNs, a tie-breaking rule is needed (default: ties-to-even);
- $\mathrm{RN}(-x) = -\mathrm{RN}(x)$; $\mathrm{RN}(2^k x) = 2^k \mathrm{RN}(x)$ (unless subnormal or overflow); x multiple of $2^k \Rightarrow \mathrm{RN}(x)$ multiple of 2^k;
- if x is in the normal range (i.e., $\beta^{e_{\min}} \leq |x| \leq \Omega$), then

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

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

Internal binary representation of IEEE 754 formats (base 2)



- if $E = 2^{W_E} - 1$ (i.e., E is a string of ones) and $F \neq 0$, then a NaN is represented;
- if $E = 2^{W_E} - 1$ and $F = 0$, then $(-1)^S \times (+\infty)$ is represented;
- if $1 \leq E \leq 2^{W_E} - 2$, then the (normal) floating-point number being represented is
  $$(-1)^S \times 2^{E-b} \times \left(1 + F \cdot 2^{1-p}\right),$$
  where the *bias* b is defined as $b = e_{\max} = 2^{W_E - 1} - 1$;
- if $E = 0$ and $F \neq 0$, then the (subnormal) number being represented is
  $$(-1)^S \times 2^{e_{\min}} \times \left(0 + F \cdot 2^{1-p}\right);$$
- if $E = 0$ and $F = 0$, then the number being represented is the signed zero $(-1)^S \times (+0)$.

Internal binary representation of IEEE 754 formats (base 2)

format	binary16	binary32	binary64	binary128
former name	N/A	single precision	double precision	N/A
storage width	16	32	64	128
$p-1$, trailing significand width	10	23	52	112
W_E, exponent width	5	8	11	15
$b = e_{\text{max}}$	15	127	1023	16383
e_{min}	-14	-126	-1022	-16382

Example: Binary encoding of a normal number

Consider the binary32 number x whose binary encoding is

sign	*exponent*	*trailing significand*
0	01101011	01010101010101010101010

- the bit sign of x is a zero $\rightarrow x \geq 0$;
- biased exponent $01101011_2 = 107_{10} \notin \{00000000_2\ 11111111_2\} \rightarrow x$ is a normal number. Since the bias in binary32 is 127, the actual exponent of x is $107 - 127 = -20$;
- by placing the hidden bit (a 1, since x is not subnormal) at the left of the trailing significand, we get the significand of x:

$$1.01010101010101010101010_2 = \frac{5592405}{2^{22}};$$

- hence, x is equal to

$$\frac{5592405}{2^{22}} \times 2^{-20} = \frac{5592405}{2^{42}} \approx 1.2715657 \times 10^{-6}$$

Exercise

Consider, still in binary32 floating-point arithmetic, the 32-bit chain:

sign	*exponent*	*trailing significand*
1	00000000	01100000000000000000000

Which FP number does it represent ?

Exception handling: the show must go on...

- when an exception occurs: the computation must continue (default behaviour);
- two infinities and two zeros, with intuitive rules: $1/(+0) = +\infty$, $5 + (-\infty) = -\infty \ldots$;
- and yet, something a little odd: $\sqrt{-0} = -0$;
- Not a Number (NaN): result of $\sqrt{-5}$, $(\pm 0)/(\pm 0)$, $(\pm\infty)/(\pm\infty)$, $(\pm 0) \times (\pm\infty)$, NaN $+3$, etc.

$$f(x) = 3 + \frac{1}{x^5}$$

will give the very accurate answer 3 for huge x, even if x^5 overflows.

One should be cautious: behavior of

$$\frac{x^2}{\sqrt{x^3 + 1}}$$

for large x.

Just for the fun: quick and dirty square root

- game quake III, 1999;
- (very) low precision, very fast, software;
- use the fact that the exponent field of x encodes $\lfloor \log_2 |x| \rfloor$.
- Binary32 (a.k.a. single precision) representation of normal x:

S_x	E_x		F_x	
31	30	23	22	0

  - 1-bit sign S_x, 8-bit biased exponent E_x, 23-bit fraction F_x s.t.

$$x = (-1)^{S_x} \cdot 2^{E_x - 127} \cdot \left(1 + 2^{-23} \cdot F_x\right).$$

  - the same bit-chain, if interpreted as 2's complement integer, represents the number

$$I_x = (1 - 2S_x) \cdot 2^{31} + \left(2^{23} \cdot E_x + F_x\right).$$

Just for the fun: quick and dirty square root

In the following:

- I_x is the integer whose binary representation is the same as that of x, i.e.,

$$I_x = (1 - 2S_x) \cdot 2^{31} + \left(2^{23} \cdot E_x + F_x\right).$$

Beware, need to be cautious when we talk of equality: if y is the FP number equal to J, and $I_x = J$, x is not equal to y: we have

- mathematical equality of the integer J and the real y, and
- equality of the binary representations of J and x.

Just for the fun: quick and dirty square root

Remember:

$$x = (-1)^{S_x} \cdot 2^{E_x - 127} \cdot \left(1 + 2^{-23} \cdot F_x\right) = (-1)^{S_x} \cdot 2^{e_x} \cdot (1 + f_x).$$

- If $e_x = E_x - 127$ is even (i.e., E_x is odd), we use:

$$\sqrt{(1 + f_x) \cdot 2^{e_x}} \approx \left(1 + \frac{f_x}{2}\right) \cdot 2^{e_x/2}, \tag{1}$$

- if e_x is odd (i.e., E_x is even), we use:

$$\begin{aligned} \sqrt{(1 + f_x) \cdot 2^{e_x}} &= \sqrt{4 + \epsilon_x} \cdot 2^{\frac{e_x - 1}{2}} \\ &\approx \left(2 + \frac{\epsilon_x}{4}\right) \cdot 2^{\frac{e_x - 1}{2}} \\ &= \left(\frac{3}{2} + \frac{f_x}{2}\right) \cdot 2^{\frac{e_x - 1}{2}}, \end{aligned} \tag{2}$$

(Taylor series for $\sqrt{4 + \epsilon_x}$ at $\epsilon_x = 0$, with $\epsilon_x = 2f_x - 2$)

Just for the fun: quick and dirty square root

$$x = (-1)^{S_x} \cdot 2^{E_x - 127} \cdot \left(1 + 2^{-23} \cdot F_x\right) = (-1)^{S_x} \cdot 2^{e_x} \cdot (1 + f_x).$$

S_x	E_x	F_x
31	30 … 23	22 … 0

- E_x odd $\rightarrow \left(1 + \frac{f_x}{2}\right) \cdot 2^{\frac{e_x}{2}}$,

$$\left(1 + F_y \cdot 2^{-23}\right) \cdot 2^{E_y - 127} \approx \left(1 + F_x \cdot 2^{-24}\right) \cdot 2^{\frac{E_x - 127}{2}}$$
$$\Rightarrow E_y = \frac{E_x + 127}{2} \text{ and } F_y = \lfloor \frac{F_x}{2} \rfloor$$

- E_x even $\rightarrow \left(\frac{3}{2} + \frac{f_x}{2}\right) \cdot 2^{\frac{e_x - 1}{2}}$.

$$\left(1 + F_y \cdot 2^{-23}\right) \cdot 2^{E_y - 127} \approx \left(\frac{3}{2} + F_x \cdot 2^{-24}\right) \cdot 2^{\frac{E_x - 128}{2}}$$
$$\Rightarrow E_y = \frac{E_x + 127}{2} - \frac{1}{2} \text{ and } F_y = 2^{22} + \lfloor \frac{F_x}{2} \rfloor$$

In both cases:

$$I_y = \left\lfloor \frac{I_x}{2} \right\rfloor + 127 \cdot 2^{22}$$

Just for the fun: quick and dirty square root

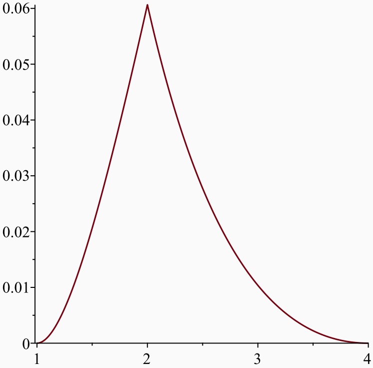


Figure 1: Plot of $(\text{approx} - \sqrt{x})/\sqrt{x}$.

- fast but rough approximation;
- always $\geq \sqrt{x}$ $\rightarrow$ replace $127 \cdot 2^{22}$ by a smaller value?

Just for the fun: quick and dirty square root

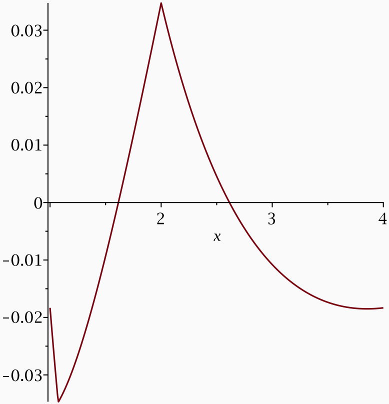
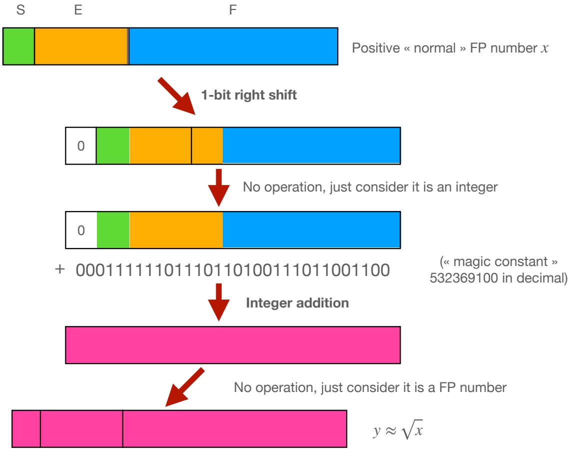


Figure 2: Plot of $(\text{approx} - \sqrt{x})/\sqrt{x}$ with $127 \cdot 2^{22}$ replaced by 532369100.

Just for the fun: quick and dirty square root



A similar trick first appears in
The game Quake III Arena

A useful property: Sterbenz Lemma

Lemma 1 (Sterbenz)

Let a and b be positive FP numbers. If

$$\frac{a}{2} \leq b \leq 2a$$

then $a - b$ is a FP number ($\rightarrow$ computed exactly, whatever the rounding function).

Beware: the "2"s in the formula are not the radix. In radix 10, 17 or 42, the same property holds, still with $\frac{a}{2} \leq b \leq 2a$.

A useful property: Sterbenz Lemma

We have $\frac{a}{2} \leq b \leq 2a$.

this implies $\frac{b}{2} \leq a \leq 2b$

→ a and b play a symmetrical role

→ without l.o.g., we can assume $a \geq b$

Consequence: a and b are multiple of

$$ulp(b) = \beta^{e_b - p + 1}$$

where e_b is the FP exponent of b.

Furthermore: $b = M_b \cdot ulp(b)$

with $M_b \leq \beta^p - 1$.

a - b is a multiple of ulp(b), ie

$$a - b = K \cdot ulp(b)$$

$$\begin{cases} a \geq b \\ a \leq 2b \end{cases} \Rightarrow 0 \leq a - b \leq b$$

Here $K \leq M_b \leq \beta^p - 1$

→ $a - b = K \cdot \beta^{e_b - p + 1}$ with $|K| \leq \beta^p - 1$

The error of (RN) FP addition is a FPN

Lemma 2

Let a and b be two FP numbers. Let

$$s = RN(a+b) \text{ and } r = (a+b) - s.$$

If no overflow when computing s, then r is a FP number.

Beware: does not always work with rounding functions $\neq$ RN .

Example: radix-2, precision-p, rounding function RD , $a = 1$, $b = -2^{-3p}$, give

$$s = \mathrm{RD}\,(a+b) = 0.\underbrace{111111\cdots 11}_{p} = 1 - 2^{-p},$$

and

$$(a+b) - s = \underbrace{1.1111111111\cdots 11}_{2p} \times 2^{-p-1},$$

which is not a precision-p FPN (would require precision $2p$).

The error of (RN) FP addition is a FPN

Proof. Without loss of generality, assume $|a| \geq |b|$

① s is "the" FPN nearest $a+b$ → it is closest to $a+b$ than a is

$$\rightarrow |s - (a+b)| \leq |a - (a+b)|$$

therefore $|r| \leq |b|$

② denote $a = M_a \cdot \beta^{e_a - p + 1}$; $b = M_b \cdot \beta^{e_b - p + 1}$

with $|M_a|, |M_b| \leq \beta^p - 1$ and $e_a \geq e_b$.

$a+b$ multiple of $\beta^{e_b - p + 1}$ $\Rightarrow$ s and r multiple of $\beta^{e_b - p + 1}$ too

$$\Rightarrow \exists\ R \in \mathbb{Z} \text{ s.t. } r = R \cdot \beta^{e_b - p + 1}$$

But $|r| \leq |b| \Rightarrow |R| \leq |M_b| \leq \beta^p - 1$

→ r is a FPN!

Get r: the fast2sum algorithm (Dekker)

Theorem 3 (Fast2Sum (Dekker))

(only radix 2). Let a and b be FP numbers, s.t. $|a| \geq |b|$.
Following algorithm: s and r such that

- $s + r = a + b$ *exactly;*
- s *is "the" FP number that is closest to* $a + b$*;*
- *incidentally (will serve later on)* $z = s - a$ *exactly.*

Algorithm 1 (FastTwoSum)

$s \leftarrow RN(a + b)$

$z \leftarrow RN(s - a)$

$r \leftarrow RN(b - z)$

C Program 1

```
s = a+b;
z = s-a;
r = b-z;
```

Important remark: Proving the behavior of such algorithms requires use of the correct rounding property.

Proof

$$s = \mathrm{RN}(a+b)$$
$$z = \mathrm{RN}(s-a)$$
$$t = \mathrm{RN}(b-z)$$

- if a and b have same sign, then $|a| \leq |a+b| \leq |2a|$ hence ($2a$ is a FP number, rounding is increasing) $|a| \leq |s| \leq |2a| \rightarrow$ (Sterbenz) $z = s - a$. Since $r = (a+b) - s$ is a FPN and $b - z = r$, we find $\mathrm{RN}(b-z) = r$.
- if a and b have opposite signs then
  1. either $|b| \geq \frac{1}{2}|a|$, which implies (Sterbenz) $a+b$ is a FPN, thus $s = a+b$, $z = b$ and $t = 0$;
  2. or $|b| < \frac{1}{2}|a|$, which implies $|a+b| > \frac{1}{2}|a|$, hence $s \geq \frac{1}{2}|a|$ ($\frac{1}{2}a$ is a FPN, rounding is increasing), thus (Sterbenz) $z = \mathrm{RN}(s-a) = s - a = b - r$. Since $r = (a+b) - s$ is a FPN and $b - z = r$,we get $\mathrm{RN}(b-z) = r$.

The TwoSum Algorithm (Moller-Knuth)

- no need to compare a and b;
- 6 operations instead of 3 yet, on many architectures, very cheap in front of wrong branch prediction penalty when comparing a and b;
- works in all bases.

Algorithm 2 (TwoSum)

$s \leftarrow RN(a+b)$
$a' \leftarrow RN(s-b)$
$b' \leftarrow RN(s-a')$
$\delta_a \leftarrow RN(a-a')$
$\delta_b \leftarrow RN(b-b')$
$r \leftarrow RN(\delta_a+\delta_b)$

Knuth: if no underflow nor overflow occurs then $a+b=s+r$, and s is nearest $a+b$.

Boldo et al: formal proof + underflow does not hinder the result (overflow does).

TwoSum is optimal, in a way we will explain.

The TwoSum Algorithm: proof in the case $\beta = 2$, $p \geq 3$.

1: $s \leftarrow \mathrm{RN}(a+b)$
2: $a' \leftarrow \mathrm{RN}(s-b)$
3: $b' \leftarrow \mathrm{RN}(s-a')$
4: $\delta_a \leftarrow \mathrm{RN}(a-a')$
5: $\delta_b \leftarrow \mathrm{RN}(b-b')$
6: $r \leftarrow \mathrm{RN}(\delta_a+\delta_b)$

Proof assuming base 2

(1) if $|b| \geq |a|$ then lines (1), (2) and (4) constitute Fast2Sum(b,a)

$\rightarrow a' = s - b$ (corresponds to the "z" of Fast2Sum)

$\delta_a = a+b-s$

Furthermore, $a' = s-b \Rightarrow s - a' = b \Rightarrow b' = b \Rightarrow \delta_b = 0$

Hence, $\delta_a + \delta_b = \delta_a = a+b-s$

and since $a+b-s$ is a FPN, $r = \mathrm{RN}(\delta_a+\delta_b) = \delta_a+\delta_b$

The TwoSum Algorithm (Moller-Knuth)

$s \leftarrow \mathrm{RN}\,(a+b)$
$a' \leftarrow \mathrm{RN}\,(s-b)$
$b' \leftarrow \mathrm{RN}\,(s-a')$
$\delta_a \leftarrow \mathrm{RN}\,(a-a')$
$\delta_b \leftarrow \mathrm{RN}\,(b-b')$
$r \leftarrow \mathrm{RN}\,(\delta_a+\delta_b)$

(2) If $|b| < |a|$ and $|s| < |b|$
then a and b have opposite signs
(otherwise we would have $|a+b| \geq |b|$ and therefore
$|s| \geq |RN(a+b)| \geq |RN(b)| = |b|$)

Also, $|b| \geq |\frac{a}{2}|$
(otherwise we would have $|a+b| > |\frac{a}{2}|$, so that
$|s| = |RN(a+b)| \geq |RN(\frac{a}{2})| = |\frac{a}{2}| > b$)

Therefore, Sterbenz lemma applies to line (1) of the algorithm

$\longrightarrow s = a+b$, so that $a' = a$, $b' = b$, $\delta_a = \delta_b = 0$, $r = 0$.

The TwoSum Algorithm (Moller-Knuth)

$s \leftarrow \mathrm{RN}\,(a+b)$
$a' \leftarrow \mathrm{RN}\,(s-b)$
$b' \leftarrow \mathrm{RN}\,(s-a')$
$\delta_a \leftarrow \mathrm{RN}\,(a-a')$
$\delta_b \leftarrow \mathrm{RN}\,(b-b')$
$r \leftarrow \mathrm{RN}\,(\delta_a+\delta_b)$

(3) If $|b| < |a|$ and $|s| \geq |b|$

We have $s = (a+b)(1+\varepsilon_1)$ with $|\varepsilon_1| \leq u$
$a' = (s-b)(1+\varepsilon_2)$ with $|\varepsilon_2| \leq u$
(with $u = 2^{-p}$)

Hence $a' = (a + a\varepsilon_1 + b\varepsilon_1)(1+\varepsilon_2)$

$|b|<|a| \Rightarrow a\varepsilon_1 + b\varepsilon_1$ can be written $2a\varepsilon_3$ with $|\varepsilon_3| \leq u$

$\longrightarrow a' = (a + 2a\varepsilon_3)(1+\varepsilon_2) = a(1+\varepsilon_4)$ with $|\varepsilon_4| \leq 3u+2u^2$

$p \geq 3 \Rightarrow u \leq \frac{1}{8} \Rightarrow |\varepsilon_4| < \frac{1}{2}$, hence $\left|\frac{a}{2}\right| \leq |a'| \leq |2a|$ and a and a' have the same sign

→ From Sterbenz lemma, $a - a'$ is a FPN

→ hence $\delta_a = a - a'$

The TwoSum Algorithm (Moller-Knuth)

$s \leftarrow \mathrm{RN}(a+b)$
$a' \leftarrow \mathrm{RN}(s-b)$
$b' \leftarrow \mathrm{RN}(s-a')$
$\delta_a \leftarrow \mathrm{RN}(a-a')$
$\delta_b \leftarrow \mathrm{RN}(b-b')$
$r \leftarrow \mathrm{RN}(\delta_a+\delta_b)$

follow up of case $|b|<|a|$ and $|s|\geq|b|$

- we have shown that $\delta_a = a - a'$
- lines (2), (3), (5) of the algorithm constitute Fast2Sum$(s, -b)$

$\longrightarrow b' = s - a'$ and $\delta_b = a' - (s-b)$

$\Longrightarrow \delta_a + \delta_b = (a+b) - s$

and since $(a+b)-s$ is a FPN:

$\mathrm{RN}(\delta_a+\delta_b) = \delta_a + \delta_b = a+b-s$

QED

TwoSum is “optimal”

Assume an algorithm satisfies:

- it is without tests or min/max instructions;
- it only uses rounded to nearest additions/subtractions: at step i we compute $\mathrm{RN}(u+v)$ or $\mathrm{RN}(u-v)$ where u and v are input variables or previously computed variables.

If that algorithm algorithm always computes the same results as 2Sum, then it uses at least 6 *additions/subtractions (i.e., as much as 2Sum).*

- proof: most inelegant proof award;
  - 480756 algorithms with 5 operations (after suppressing the most obvious symmetries);
  - each of them tried with 2 well-chosen pairs of input values.

Example of application: computing $x_1 + x_2 + x_3 + \cdots + x_n$

Naive algorithm:

$s \leftarrow x_1$
for $i = 2$ to n **do**
  $s \leftarrow \mathrm{RN}(s + x_i)$
end for
return s

- easy to show: $\left|s - \sum x_i\right| \leq \gamma_{n-1} \sum |x_i|$, with

$$\gamma_n = \frac{n\mathbf{u}}{1 - n\mathbf{u}}.$$

- much more tricky: replace γ_{n-1} by $(n-1) \cdot u$.

Example of application: computing $x_1 + x_2 + x_3 + \cdots + x_n$

Pichat, Ogita, Rump, and Oishi's algorithm:

Algorithm 3

$s \leftarrow x_1$

$e \leftarrow 0$

for $i = 2$ *to* n ***do***

  $(s, e_i) \leftarrow 2Sum(s, x_i)$

  $e \leftarrow RN(e + e_i)$

end for

return $RN(s + e)$

Example of application: computing $x_1 + x_2 + x_3 + \cdots + x_n$

Theorem 4 (Ogita, Rump and Oishi)
Applying the algorithm of P.,O., R., and O. to x_i, $1 \le i \le n$, and if $n\mathbf{u} < 1$, then, even in case of underflow (but without overflow), the final result σ satisfies

$$\left|\sigma - \sum_{i=1}^{n} x_i\right| \le \mathbf{u}\left|\sum_{i=1}^{n} x_i\right| + \gamma_{n-1}^2 \sum_{i=1}^{n} |x_i|.$$

What about products ?

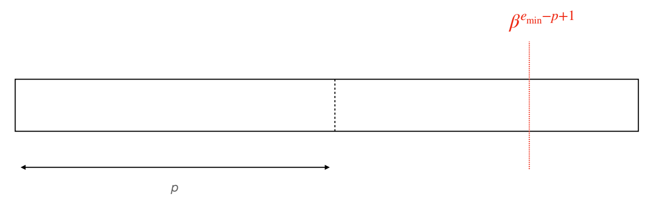
Theorem 5

Let a and b be FPNs:

$$a = M_a \cdot \beta^{e_a - p + 1}, \quad \text{and} \quad b = M_b \cdot \beta^{e_b - p + 1}, \quad \text{with}$$

$$|M_a|, |M_b| \leq \beta^p - 1 \quad \text{and} \quad e_{min} \leq e_a, e_b.$$

if $e_a + e_b \geq e_{min} + p - 1$ *then for any rounding function* $\circ \in \{RU, RD, RZ, RN\}$*, the number* $r = ab - \circ(ab)$ *is a FPN.*



What about products ?

Exercise: prove the theorem.

What about products ?

- We use the *fused multiply-add* (fma) instruction. It computes $\mathrm{RN}(ab+c)$. First ppeared in IBM RS6000, Intel/HP Itanium, PowerPC... Specified since 2008.
- We have seen: if a and b are FP numbers, then (under condition $e_a + e_p \geq e_{\min} + p - 1$), $r = ab - \mathrm{RN}(ab)$ is a FP number;
- obtained with algorithm TwoMultFMA $\begin{cases} p &= \mathrm{RN}(ab) \\ r &= \mathrm{RN}(ab-p) \end{cases}$

  $\rightarrow$ 2 operations only. $p + r = ab$.
- without fma, Dekker's algorithm: 17 operations (7 $\times$, 10 $\pm$). (only historical interest now)

Just an example: $ad - bc$ with fused multiply-add

Kahan's algorithm for $x = ad - bc$:

$\hat{w} \leftarrow \mathrm{RN}(bc)$
$e \leftarrow \mathrm{RN}(\hat{w} - bc)$
$\hat{f} \leftarrow \mathrm{RN}(ad - \hat{w})$
$\hat{x} \leftarrow \mathrm{RN}(\hat{f} + e)$
Return $\hat{x}$

We assume radix 2.

- using std model (2002):

$$|\hat{x} - x| \leq J|x|$$

  with $J = 2u + u^2 + (u + u^2)u\frac{|bc|}{|x|} \rightarrow$ high accuracy as long as $u|bc| \not\gg |x|$
- using properties of RN (2011):

$$|\hat{x} - x| \leq 2u|x|$$

  "asymptotically optimal" error bound.
- $\rightarrow$ complex $\times$, $\div$.

Newton-Raphson iteration for $1/b$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

with $f(x) = \frac{1}{x} - b$. Gives

$$x_{n+1} = 2x_n - bx_n^2 = x_n(2 - bx_n).$$

Gives

$$\begin{aligned} \left|x_{n+1} - \tfrac{1}{b}\right| &= \left|2x_n - bx_n^2 - \tfrac{1}{b}\right| \\ &= b \cdot \left|2bx_n - x_n^2 - \tfrac{1}{b^2}\right| \\ &= b \cdot \left(x_n - \tfrac{1}{b}\right)^2. \end{aligned}$$

But this is with exact arithmetic. What happens if we use FP arithmetic?

Reciprocation using Newton-Raphson iteration and an FMA

Division algorithm used on the Intel/HP Itanium. Precision p, radix 2. To simplify, we only compute $1/b$. We assume $1 \leq b < 2$ (significands of normal FP numbers).

- Newton-Raphson iteration to compute $1/b$:

$$y_{n+1} = y_n(2 - by_n)$$

- we lookup $y_0 \approx 1/b$ in a table addressed by the first (typically from 6 to 10) bits of b;
- the NR iteration is decomposed into 2 FMA instructions:

$$\begin{cases} e_n & = & \text{RN}\,(1 - by_n) \\ y_{n+1} & = & \text{RN}\,(y_n + e_n y_n) \end{cases}$$

Notice that $e_{n+1} \approx e_n^2$.

Property 1

If

$$\left|\frac{1}{b} - y_n\right| < \alpha 2^{-k},$$

where $1/2 < \alpha \leq 1$ *and* $k \geq 1$, *then*

$$\begin{aligned}\left|\frac{1}{b} - y_{n+1}\right| &< b\left(\frac{1}{b} - y_n\right)^2 + 2^{-k-p} + 2^{-p-1} \\ &< 2^{-2k+1}\alpha^2 + 2^{-k-p} + 2^{-p-1}\end{aligned}$$

$\Rightarrow$ it seems that we can get arbitrarily closer to error 2^{-p-1} (i.e., $1/2\,\mathrm{ulp}\,(1/b)$), without being able to show a bound below $1/2\,\mathrm{ulp}\,(1/b)$.

Example: binary64 format of the IEEE-754 standard

Assume $p = 53$ and $|y_0 - \frac{1}{b}| < 2^{-8}$ (small table), we find

- $|y_1 - 1/b| < 0.501 \times 2^{-14}$
- $|y_2 - 1/b| < 0.51 \times 2^{-28}$
- $|y_3 - 1/b| < 0.57 \times 2^{-53} = 0.57\,\mathrm{ulp}\,(1/b)$

Going further ?

Property 2

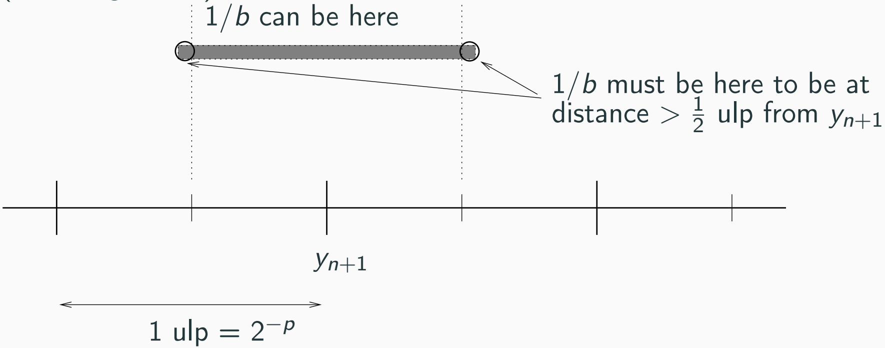
When y_n approximates $1/b$ within error $< 1\,ulp\,(1/b) = 2^{-p}$, then, since b is multiple of 2^{-p+1} and y_n is multiple of 2^{-p}, $1 - by_n$ is multiple of 2^{-2p+1}.

But $|1 - by_n| < 2^{-p+1} \rightarrow$ $1 - by_n$ is a FP number $\rightarrow$ exactly computed by one FMA.

$$\Rightarrow \left|\frac{1}{b} - y_{n+1}\right| < b\left(\frac{1}{b} - y_n\right)^2 + 2^{-p-1}.$$

$$\left| y_n - \frac{1}{b} \right| < \alpha 2^{-p} \Rightarrow \left| y_{n+1} - \frac{1}{b} \right| < b\alpha^2 2^{-2p} + 2^{-p-1}$$

(assuming $\alpha < 1$)



What can be deduced ?

- to be at distance $> 1/2$ ulp from y_{n+1}, $1/b$ must be within $b\alpha^2 2^{-2p} < b2^{-2p}$ from the midpoint of two consecutive FP numbers;
- implies that distance between y_n and $1/b$ has the form $2^{-p-1} + \epsilon$, with $|\epsilon| < b2^{-2p}$;
- implies $\alpha < \frac{1}{2} + b2^{-p}$ hence

$$\left| y_{n+1} - \frac{1}{b} \right| < \left(\frac{1}{2} + b2^{-p} \right)^2 b2^{-2p} + 2^{-p-1}$$

- so, to be at distance $> 1/2$ ulp from y_{n+1}, $1/b$ must be within $\left(\frac{1}{2} + b2^{-p}\right)^2 b2^{-2p}$ from the midpoint of two consecutive FP numbers.

- b is a FP number between 1 et 2 $\Rightarrow b = B/2^{p-1}$ where $B \in \mathbb{N}$, $2^{p-1} < B \leq 2^p - 1$;
- the midpoint of two consecutive FP numbers in the neighborhood of $1/b$ has the form $g = (2G+1)/2^{p+1}$ where $G \in \mathbb{N}$, $2^{p-1} \leq G < 2^p - 1$;
- we deduce

$$\left|g - \frac{1}{b}\right| = \left|\frac{2BG + B - 2^{2p}}{B.2^{p+1}}\right|$$

- the distance between $1/b$ and the midpoint of two consecutive FP numbers is a multiple of $1/(B.2^{p+1}) = 2^{-2p}/b$. It is $\neq 0$

Distance between $\frac{1}{b}$ and g, when $\left|\frac{1}{b} - y_{n+1}\right| > \frac{1}{2}\,\mathbf{ulp}\left(\frac{1}{b}\right)$

- has the form $k2^{-2p}/b$, $k \in \mathbb{Z}$, $k \neq 0$;
- we must have

$$\frac{|k| \cdot 2^{-2p}}{b} < \left(\frac{1}{2} + b2^{-p}\right)^2 b2^{-2p}$$

therefore

$$|k| < \left(\frac{1}{2} + b2^{-p}\right)^2 b^2$$

- since $b < 2$, as soon as $p \geq 4$, the only solution is $|k| = 1$;
- moreover, for $|k| = 1$, elementary manipulation shows that the only possible solution is

$$b = 2 - 2^{-p+1}.$$

How do we procede?

- we want
$$B = 2^p - 1,$$
$$2^{p-1} \le G \le 2^p - 1$$
$$B(2G+1) = 2^{2p} \pm 1$$
Only one solution: $B = 2^p - 1$ and $G = 2^{p-1}$: comes from $2^{2p} - 1 = (2^p - 1)(2^p + 1)$;
- except for that B (thus for the corresponding value $b = B/2^{p-1}$ of b), we are certain that $y_{n+1} = \text{RN}(1/b)$;
- for $B = 2^p - 1$: we try the algorithm with the two values of y_n within one ulp from $1/b$ (i.e. $1/2$ and $1/2 + 2^{-p}$). In practice, it works (otherwise: do dirty things).

Application: double precision ($p = 53$)

We start from y_0 such that $|y_0 - \frac{1}{b}| < 2^{-8}$. We compute:

$$
\begin{array}{lcl}
e_0 & = & \mathrm{RN}\left(1 - b y_0\right) \\
y_1 & = & \mathrm{RN}\left(y_0 + e_0 y_0\right) \\
e_1 & = & \mathrm{RN}\left(1 - b y_1\right) \\
y_2 & = & \mathrm{RN}\left(y_1 + e_1 y_1\right) \\
e_2 & = & \mathrm{RN}\left(1 - b y_1\right) \\
y_3 & = & \mathrm{RN}\left(y_2 + e_2 y_2\right) \quad \text{error} \le 0.57 \text{ ulps} \\
e_3 & = & \mathrm{RN}\left(1 - b y_2\right) \\
y_4 & = & \mathrm{RN}\left(y_3 + e_3 y_3\right) \quad 1/b \text{ rounded to nearest}
\end{array}
$$

In practice: two iterations

Markstein iterations

$$\begin{cases} e_n &= \mathrm{RN}\left(1 - by_n\right) \\ y_{n+1} &= \mathrm{RN}\left(y_n + e_n y_n\right) \end{cases}$$

More accurate ("self correcting"), sequential

Goldschmidt iterations

$$\begin{cases} e_{n+1} &= \mathrm{RN}\left(e_n^2\right) \\ y_{n+1} &= \mathrm{RN}\left(y_n + e_n y_n\right) \end{cases}$$

Less accurate, faster (parallel)

In practice: we start with Goldschmidt iterations, and switch to Markstein iterations for the final steps.