Posits:
the good, the bad and the ugly

Florent de Dinechin
Luc Forget
Yohann Uguen

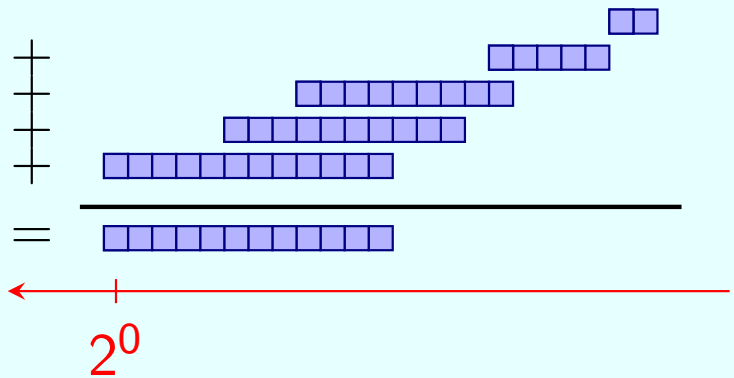
Jean-Michel Muller

Posits: the good that we all know

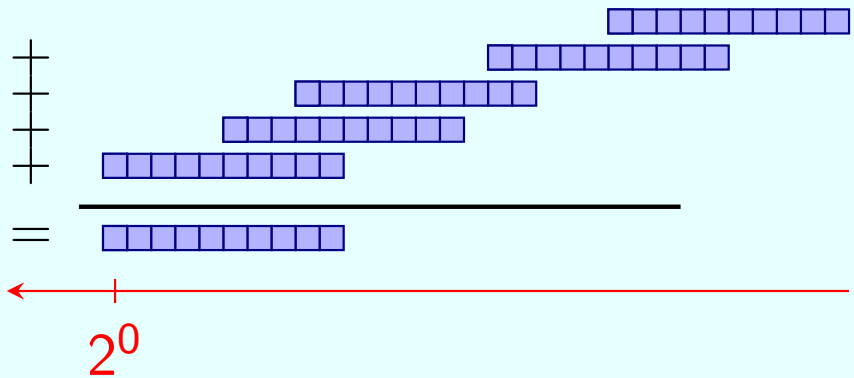
Accuracy on large/small numbers is traded for accuracy "around 1".

Sum of small terms with result "around 1"

Alignment of significands:



posit16



float16

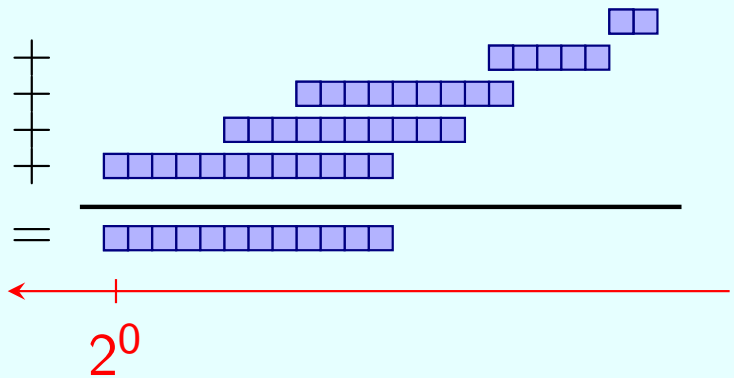
- In this case, floats are accurate in vain.
- The zone around 1 ("golden zone") is quite large.

Posits: the good that we all know

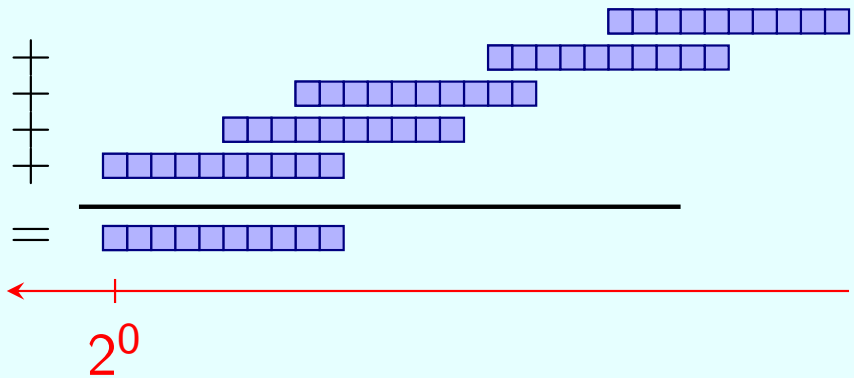
Accuracy on large/small numbers is traded for accuracy "around 1".

Sum of small terms with result "around 1"

Alignment of significands:



posit16



float16

- In this case, floats are accurate in vain.
- The zone around 1 ("golden zone") is quite large.

And then you have the quire...

But really, no need of a quire for posits to beat floats.

More good will follow

We come in peace.

Motivation by intimidation (1)

In my other life I implement elementary functions.

Simplified sketch of a **floating-point exponential**

(code in red, the rest is explanations)

1. Compute the integer $E \approx \lfloor X/\log(2) \rceil$
   (the tentative result exponent)
2. Compute the float $Y \approx X - E \times \log(2)$
   (a reduced argument in the interval $I \approx [\frac{-\log(2)}{2}, \frac{\log(2)}{2}]$)
   Remark that
   $Y \approx X - E \times \log(2)$ can be rewritten $e^X \approx 2^E e^Y$.
3. Evaluate a polynomial $Z \approx P(Y)$
   where the polynomial P is a good approximation of e^Y on I
4. Construct a float of value $S = 2^E$
5. Return $R = S \times Z \quad \approx e^X$.

How does all this translate to posits?

Motivation (2)

- the computation $S = 2^E \times Z$ was exact in FP.
  No longer in posits. Workaround?
- $Y \approx X - E \times \log(2)$ is a cancellation by construction.
  An evil?

Motivation (2)

- the computation $S = 2^E \times Z$ was exact in FP.
  No longer in posits. Workaround?
- $Y \approx X - E \times \log(2)$ is a cancellation by construction.
  An evil? No, a blessing.

A subtraction that cancels is exact.

$3.14159 - 3.14123 = 0.00036 = 3.60000 \cdot 10^{-4}$

No rounding error here.

Motivation (2)

- the computation $S = 2^E \times Z$ was exact in FP.
  No longer in posits. Workaround?
- $Y \approx X - E \times \log(2)$ is a cancellation by construction.
  An evil? No, a blessing.

A subtraction that cancels is exact.

$3.14159 - 3.14123 = 0.00036 = 3.60000 \cdot 10^{-4}$
No rounding error here.

Cody and Waite argument reduction trick

A **very accurate** computation of $Y \approx X - E \times \log(2)$,
based on exact-by-construction floating-point subtractions and multiplications.

Motivation (3)

So, can we use such tricks with posits?

Motivation (3)

So, can we use such tricks with posits?

Extended precision needed

... for a correctly-rounded elementary function

- always: a few bits more than the target precision
- rarely: 2-3 times more than the target precision

Motivation (3)

So, can we use such tricks with posits?

Extended precision needed

... for a correctly-rounded elementary function

- always: a few bits more than the target precision
- rarely: 2-3 times more than the target precision

Extended precision? That's the quire, right?

The quire: A naive architect's point of view

Operation on the quire == operation on w_q bits

	range $(\min, \max)$	quire parameters		
		w_q	w_l	w_r
Posit8, es=0	$[2^{-6}, 2^{6}]$	32	16	12
Posit16, es=1	$[2^{-28}, 2^{28}]$	128	64	72
Posit32, es=2	$[2^{-120}, 2^{120}]$	512	256	332
Posit64, es=3	$[2^{-496}, 2^{496}]$	2048	1024	1429

Latency of quire-to-posit conversion (amortized for large sums):

- leading zero count on w_l bits
- wide OR on w_r bits for rounding

The quire: A naive architect's point of view

Operation on the quire == operation on w_q bits

	range (min, max)	quire parameters		
		w_q	w_l	w_r
Posit8, es=0	$[2^{-6}, 2^{6}]$	32	16	12
Posit16, es=1	$[2^{-28}, 2^{28}]$	128	64	72
Posit32, es=2	$[2^{-120}, 2^{120}]$	512	256	332
Posit64, es=3	$[2^{-496}, 2^{496}]$	2048	1024	1429

Latency of quire-to-posit conversion (amortized for large sums):

- leading zero count on w_l bits
- wide OR on w_r bits for rounding

For our limited extended precision: cheaper alternatives?

- this paper: what works and what doesn't
- future work: a quantitative assessment (i.e. which is cheaper)

Posits: the good that you learn in this paper (1)

Boring theorems or conjectures like:

Conjecture

The rounding error in the addition of two posits of same format is a posit of the same format, except in the "twilight zone".

(verified by exhaustive test on Posit8 and Posit16)

Posits: the good that you learn in this paper (1)

Boring theorems or conjectures like:

Conjecture

The rounding error in the addition of two posits of same format is a posit of the same format, except in the "twilight zone".

(verified by exhaustive test on Posit8 and Posit16)

Why is this useful?

Posits: the good that you learn in this paper (2)

Why is this useful?
Because then we can (hopefully) compute it as a posit:

FastTwoSum

def FastTwoSum(a, b):

$$\begin{aligned} s &= a + b \\ \beta &= s - a \\ t &= b - \beta \end{aligned}$$

return s, t

Posits: the good that you learn in this paper (2)

Why is this useful?
Because then we can (hopefully) compute it as a posit:

FastTwoSum

def FastTwoSum(a, b):

$$\begin{aligned} s &= a + b \\ \beta &= s - a \\ t &= b - \beta \end{aligned}$$

return s, t

Yet another lemma

If neither a nor b belongs to the twilight zone, and $|a| > |b|$, then the posits s and t computed by the FastTwoSum sequence of posit operations verify s+t = a+b exactly.

Cheap (?) doubled precision?

Posits: the good that you learn in this paper (2)

Another very useful property:

Don't call me Sterbenz Lemma

For any two PositN of the same format a and b, where a and b are different from NaR,

$$\frac{a}{2} \leq b \leq 2a \implies a \ominus b = a - b \quad .$$

So what again?
It will help compute $Y \approx X - E \times \log(2)$

The bad (1)

Multiplication-related lemmas and tricks that work with floats don't work with posits.

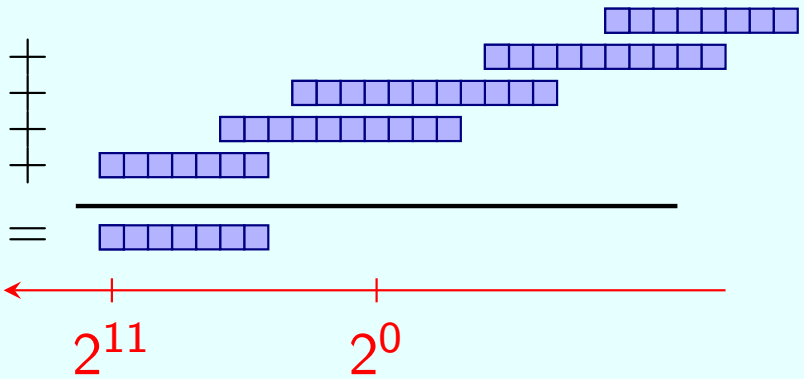
- Multiplications by powers of two are not exact.
- The error of the product of two posits is not always a posit.
- ...

It will make elementary function implementation more challenging...

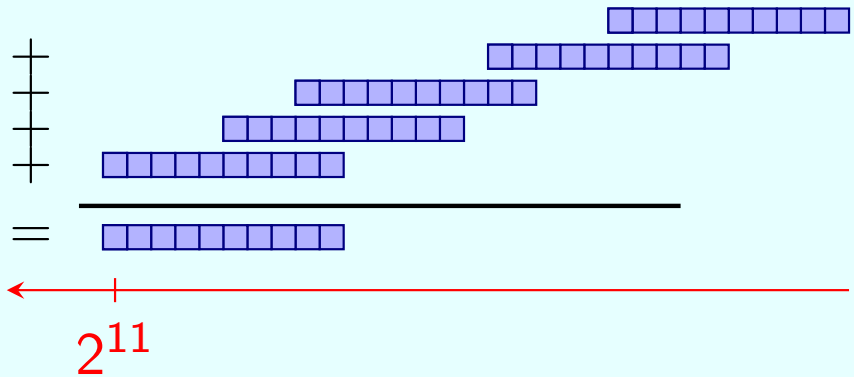
The bad (2): Beware of scale

Sum of small terms with a large result

Alignment of significands:



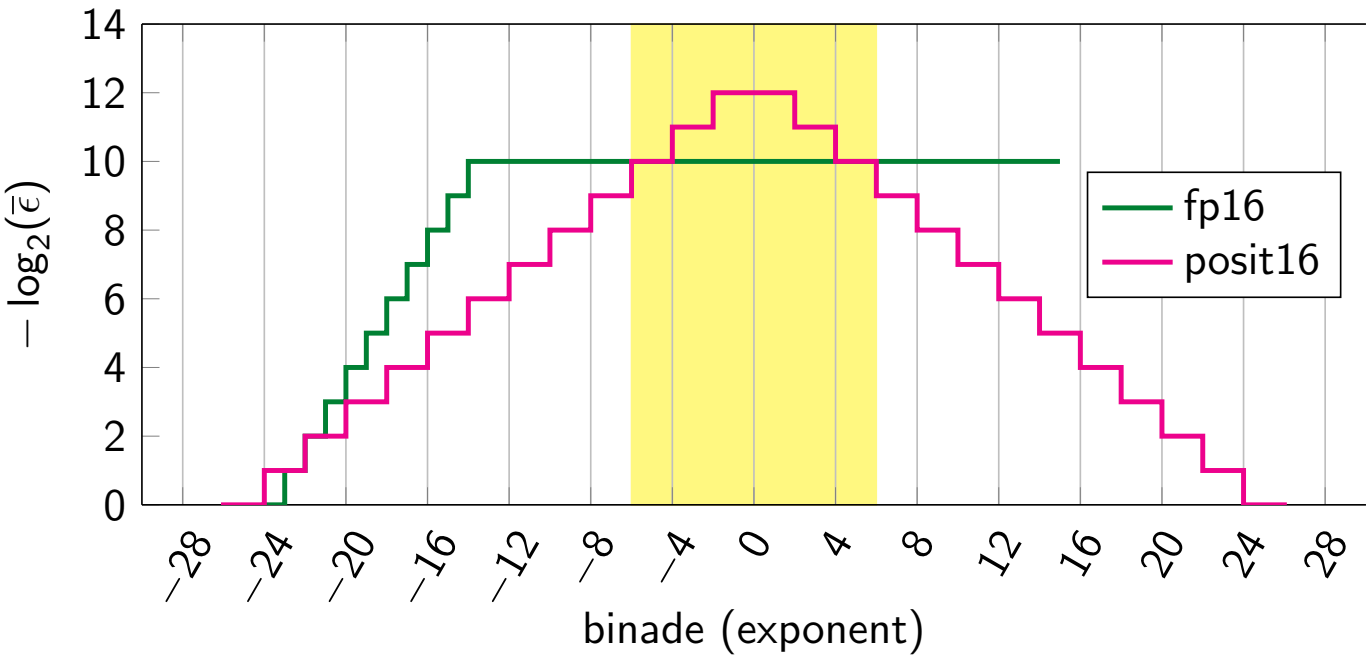
posit16



float16

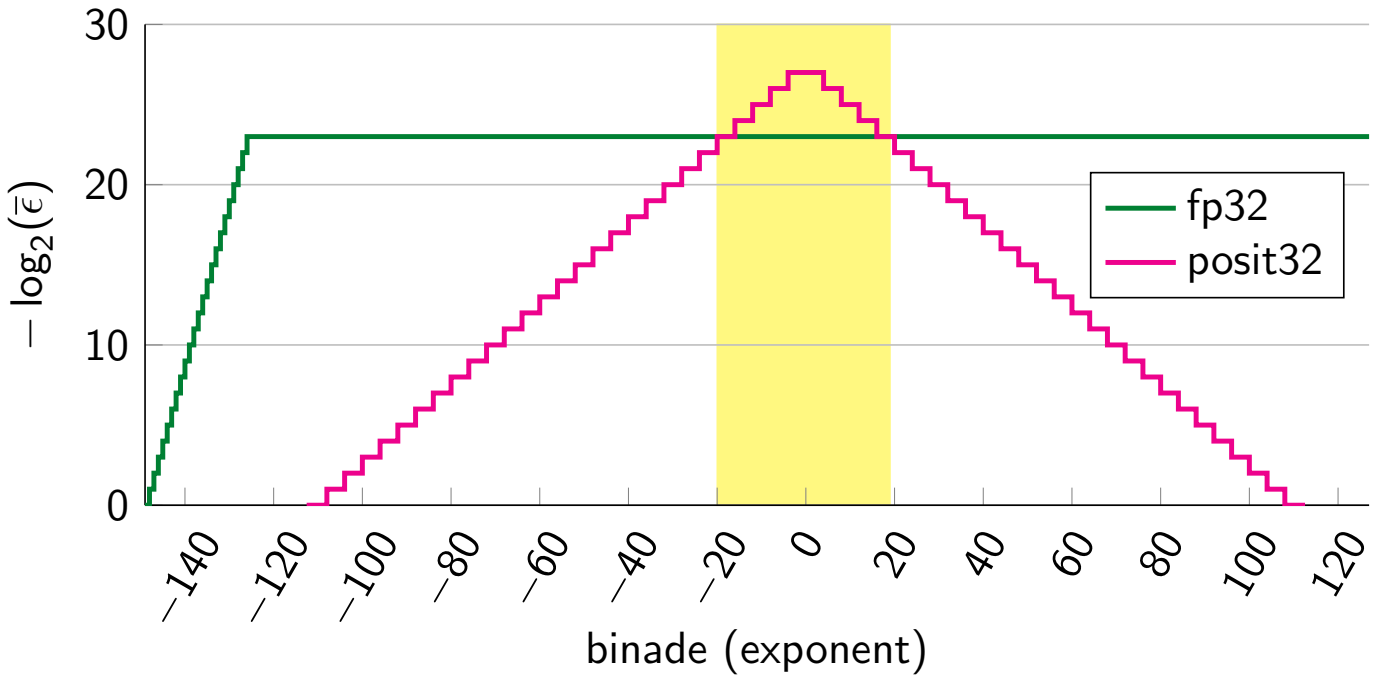
- Floats still accurate in vain, but posits even more so...
- Posits cannot represent the result accurately
  even if the summation is performed in the quire.
- float+quire $>$ posit+quire?

Still, Posit16 arguably better than Float16



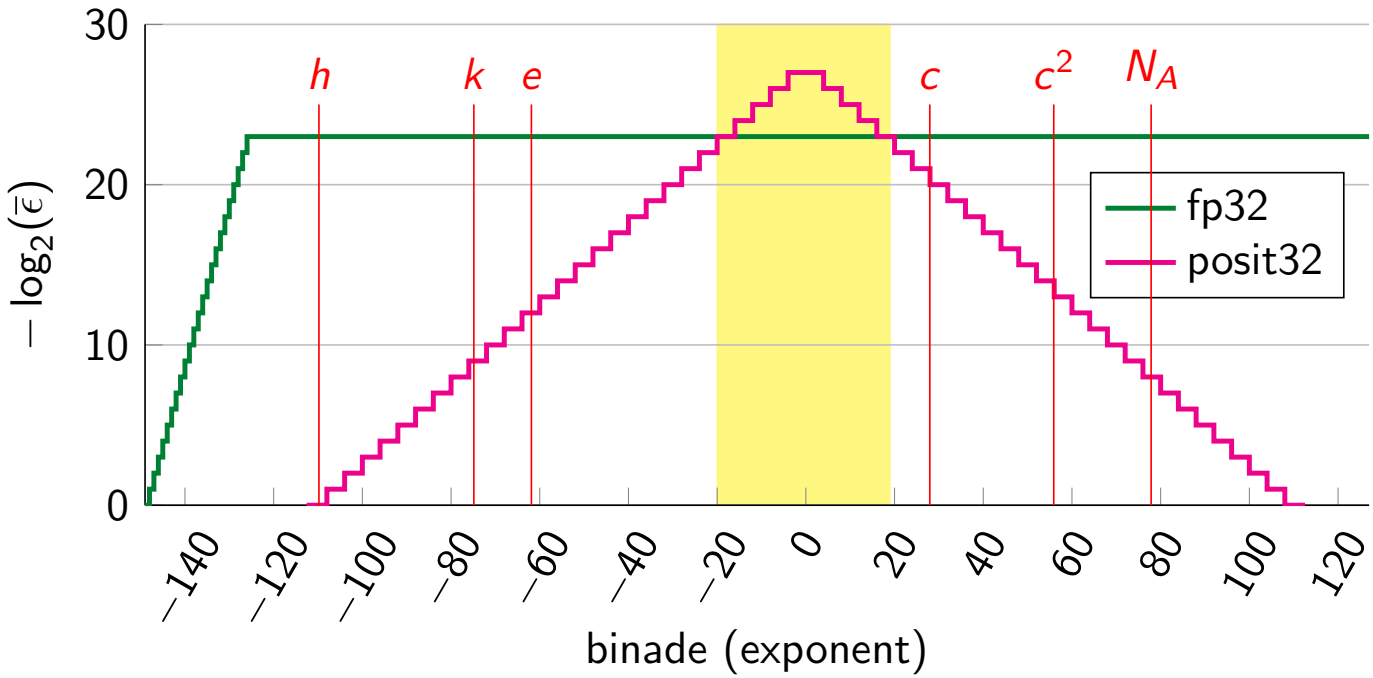
- Larger range for large numbers
  - (for small numbers, range is similar and accuracy lower)
- Golden zone (better than floats) between 1/64 and 64
- Subnormal-like behaviour outside the golden zone
  - but so do the floats

Posit32: don't call me subnormal



- Between exponents -20 and 20, the golden zone
  - very large : 12 orders of magnitude, about 10^{-6} to 10^{6}
  - where posits are always more accurate than floats
  - (and where most of the posits are concentrated)
- Outside of the golden zone, ~~subnormal~~ tapered accuracy

Posit32: don't call me subnormal



- Between exponents -20 and 20, the golden zone
  - very large : 12 orders of magnitude, about 10^{-6} to 10^{6}
  - where posits are always more accurate than floats
  - (and where most of the posits are concentrated)
- Outside of the golden zone, ~~subnormal~~ tapered accuracy

Unfortunately, physics likes to live out of the golden zone

`https://en.wikipedia.org/wiki/International_System_of_Units`

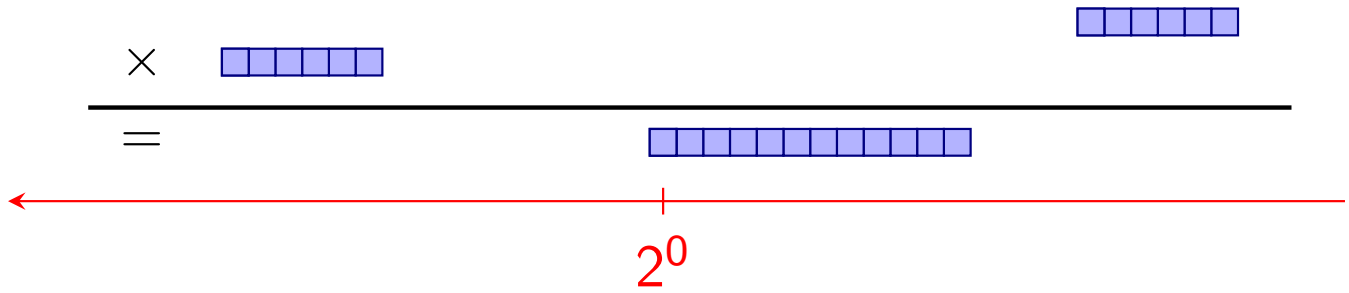
The ugly (1): very large and very small

Constants that define the International System of Units:

Planck constant h		$6.626070150 \cdot 10^{-34}$
Posit32 value	$\approx$	$7.7 \cdot 10^{-34}$
FP32 value	$\approx$	$6.626070179 \cdot 10^{-34}$
Avogadro number N_A		$6.02214076 \cdot 10^{23}$
Posit32 value	$\approx$	$6.021 \cdot 10^{23}$
FP32 value	$\approx$	$6.0221406 \cdot 10^{23}$
Speed of light c		299792458
Posit32 value		299792384
FP32 value		299792448
charge of e^-		$1.602176634 \cdot 10^{-19}$
Posit32 value	$\approx$	$1.6022 \cdot 10^{-19}$
FP32 value	$\approx$	$1.60217659 \cdot 10^{-19}$
Boltzmann constant k		$1.380649 \cdot 10^{-23}$
Posit32 value	$\approx$	$1.3803 \cdot 10^{-23}$
FP32 value	$\approx$	$1.1.38064905 \cdot 10^{-23}$

`https://en.wikipedia.org/wiki/International_System_of_Units`

The ugly (2): multiplicative cancellation



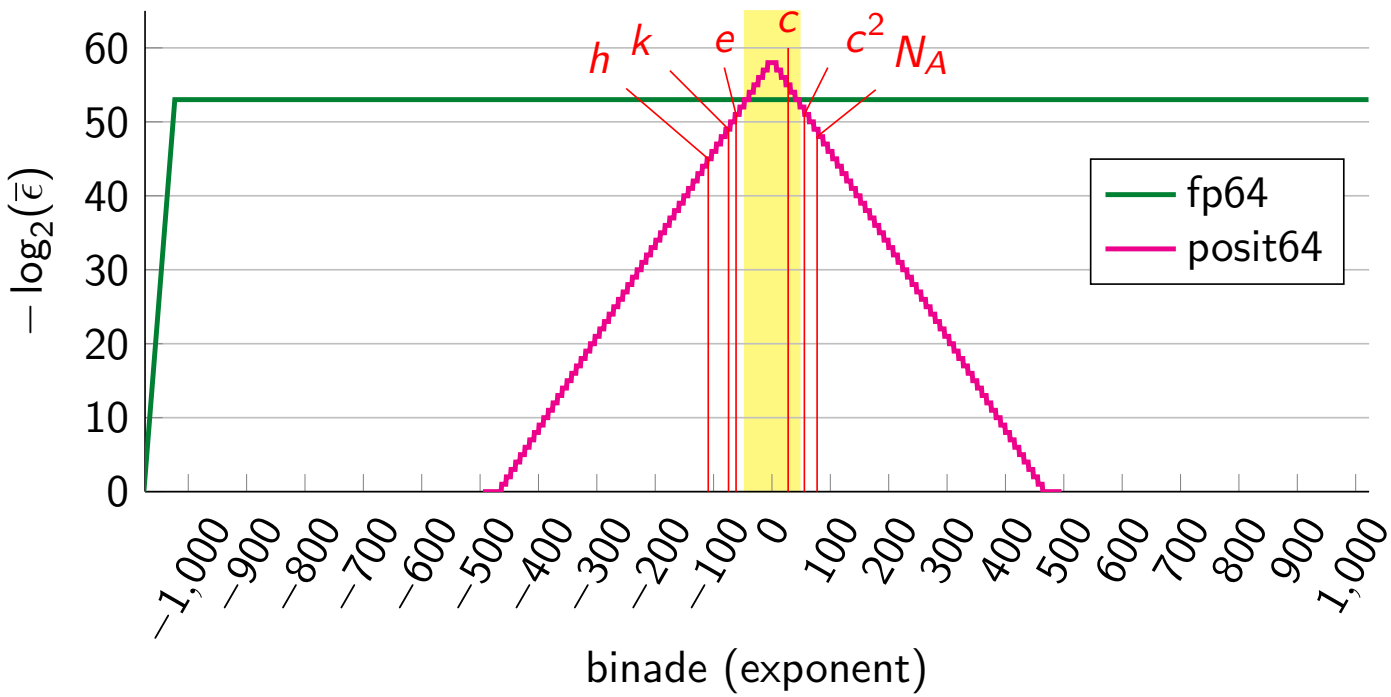
- ⊖ The result looks accurate (it is in the golden zone), but it is no more accurate than the inputs
- ⊕ (on the figure) the multiplication is exact.

Two equations you may have heard of:

$$e = mc^2$$

$$e = h\nu$$

Is there a place for Posit64?



- Sure, the golden zone is even larger
  - **28 orders of magnitude** should be enough for anybody
  - even OK for physics (all constants defined with < 10 decimal digits)
- range less than half of the float64 range
  - Will somebody complain? Probably not the physicists.

Gain in the iceberg tip: 17 decimal digits instead of 15

Do we need them? Physics units are defined with < 10 decimal digits...

A transitory proposal for general-purpose processors: embrace and extend floating-point

- keep FP32 unit and registers, use posit16 as a memory format
- keep FP64 unit and registers, use posit32 as a memory format
- in the paper: latency and resource consumption are small

Summary

- 8 bits: are floats really the competition ?
  - Logarithm Number System (Jeff Johnson)
  - **Tabulation** allows for completely bizarre and ad-hoc formats

Summary

- 8 bits: are floats really the competition ?
  - Logarithm Number System (Jeff Johnson)
  - **Tabulation** allows for completely bizarre and ad-hoc formats

- 16 bits: Posit16 better than float16
  - with or without a hardware quire?
  - state of the art in float16 is to use an exact accumulator (N. Brunie)

Summary

- 8 bits: are floats really the competition ?
  - Logarithm Number System (Jeff Johnson)
  - **Tabulation** allows for completely bizarre and ad-hoc formats
- 16 bits: Posit16 better than float16
  - with or without a hardware quire?
  - state of the art in float16 is to use an exact accumulator (N. Brunie)
- 32 and 64 bits: beware of extrapolation from smaller formats
  - Posits have smaller range
  - Posits have variable accuracy
  - The cost of a hardware quire begins to show (latency counts, too)

Conclusion: challenges ahead

- A rebuild of error analysis is needed
  - ... and I'm not saying I know how to do it
  - but I feel the very notion of a *numerical library* depends on it
- Posits force programmers to think about the scale of their numbers
  - This may actually be a Good Thing
  - ("educate programmers" versus "protect programmers")

- Proofs of low-level properties: beyond exhaustive test
  - need to embark these formal proof people
  - for proofs that scale, but more importantly, for insight
- Some properties are lost:
  - are they important?
  - how to get around them?

Conclusion: challenges ahead (2)

- Write some complete elementary function code
- Write a complete hardware posit unit
  - including good quire hardware
  - in progress

This will allow quantitative assessment : is this quire-less trick efficient?

But then, another big-picture question

posit+quire versus float+quire ?

Thank you for your attention

Questions?
(otherwise I have controversial backup slides)