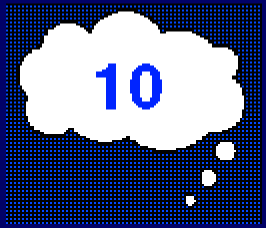
Enhanced Rexx Arithmetic

RexxLA, Austin — 10 April 2006

Mike Cowlishaw

IBM Fellow

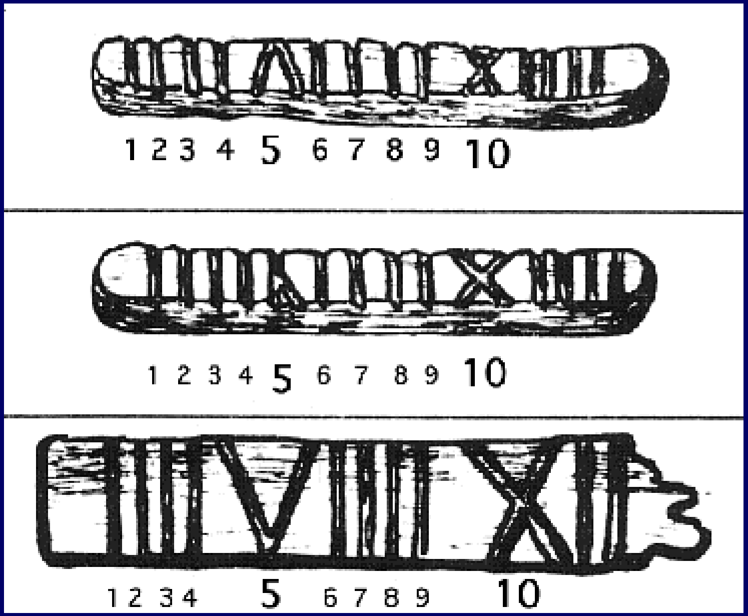


Overview

- Why is Rexx arithmetic decimal?
- Adoption by other standards and languages
- Enhancements and differences
- Adding the new type(s) to Rexx?

Origins of decimal arithmetic

- Decimal (base 10) arithmetic has been used for thousands of years



- Algorism (Indo-Arabic place value system) in use since 800 AD

- Calculators and many computers were decimal …

IBM 650 (in Böblingen)

Bi-quinary digit

Binary computers

- In the 1950s binary floating-point was shown to be more efficient
  - minimal storage space
  - more reliable (20% fewer components)

- But binary fractions *cannot* exactly represent most decimal fractions (*e.g.*, 0.1 requires an infinitely long binary fraction: 0.000110011001100110011…)

Where it costs real money…

- Add 5% sales tax to a $ 0.70 telephone call, rounded to the nearest cent

- 1.05 x 0.70 using binary double is exactly

  0.73499999999999998667732370449812151491641998291015625

  (should have been 0.735)

- rounds to $ 0.73, instead of $ 0.74

Hence…

- Binary floating-point cannot be used for commercial or human-centric applications
  - cannot meet legal and financial requirements
- Decimal data and arithmetic are pervasive
- 55% of numeric data in databases are decimal (and a further 43% are integers, often held as decimal integers)

Why decimal hardware?

Software is slow: typical Java BigDecimal add is 1,708 cycles, hardware might take 8 cycles

	software penalty
add	210x – 560x
quantize	90x – 200x
multiply	40x – 190x
divide	260x – 290x

penalty = Java BigDecimal cycles ÷ DFPU clock cycles

Effect on real applications

- The ‘telco’ billing application
  1,000,000 calls (two minutes)
  read from file, priced, taxed,
  and printed

	Java BigDecimal	C, C# packages	Itanium hand-tuned
% execution time in decimal operations	93.2%	72 – 78%	45% *

* Intel™ figure

The path to hardware…

- A 2x (maybe more) performance improvement in applications makes hardware support *very* attractive

- Standard formats are essential for language and hardware interfaces
  - IEEE 754 is being revised (since 2001)
  - incorporates IEEE 854 (radix-independent)

IEEE 754 agreed draft ('754r')

- Now has decimal floating-point formats with decimal significands and arithmetic
  - suitable for mathematical applications, too
- Fixed-point and integer decimal arithmetic are subsets (no normalization)
- Compression maximizes precision and exponent range of formats

IBM Product Plans

- Future processors will have decimal floating-point units in hardware, compliant with current 754r draft

- Appropriate software support:
  - operating system
  - compiler (GCC, IBM)
  - database
  - *etc*.

Other standards, *etc.*

- Java 5 BigDecimal (compatible arithmetic)

- C# and .Net ECMA and ISO standards
  - arithmetic changed to match, and now allow use of 745r decimal128

- ISO C and C++ are jointly adding decimal floating-point as first-class primitive types
  - work on adding to GCC almost complete

Other standards, *etc.*

- COBOL already has floating-point decimal, adding new type for 2008 standard
- ECMAScript (JavaScript/JScript) edition 4 will add decimal type
- XML Schema 1.1 draft now has *pDecimal*
- New SPEC benchmarks (SPECjbb, *etc.*)

Other standards, *etc.* [2]

- Other languages are adding decimal arithmetic (Python, Eiffel, *etc.*)
- ANSI/ISO SQL … new types accepted in principle (draft about to be submitted)
- Strong support expressed by Microsoft, SHARE, academia, and many others

Differences from Rexx arithmetic

- The IEEE types are fixed size, encoded to get maximum range and precision

Format	precision	normal range
32-bit	7	-95 to +96
64-bit	16	-383 to +384
128-bit	34	-6143 to +6144

… edge effects at the exponent extremes

Other differences [1]

- Full floating-point value set, including –0, ±infinity, and NaNs (Not-a-Number).
- Positive exponents are not forced to integers (2E+3 + 0 is 2E+3, not 2000)
- Zeros have exponents (just like other numbers) so can affect the exponent of results (1 + 0.000 is 1.000, not 1)

Other differences [2]

- Trailing zeros are preserved for divide and power operators (2.40/2 is 1.20, not 1.2)

- Subtraction rounds to length of result, not lengths of operands (with numeric digits 5, 12222 – 10000.5 is 2221.5, not 2222)

- 0 ** 0 is an error (not 1), but n ** 0.5 is OK

IEEE 754r support in Rexx

- The differences are very minor, but are sufficiently obscure that they could be surprising

- Support would allow exact emulation of other languages using the IEEE 754r types (and potentially exploit hardware)

- Built-in much easier to use than a library

IEEE 754r support in Rexx

- Support could be very simple:

	scientific
numeric form	engineering
	ieee

- Sets digits=16 (?), only digits 7, 16, 34 then allowed (or digits must already be one of these three values)

Infinities and NaNs

- String: “Infinity” (*etc.*) could be a valid number – but this could ‘surprise’ some algorithms (a+b not an error)
  - this really mostly affects the datatype BIF
- Could use original idea: ‘!’ = Infinity, ‘?’ = NaN – and these are valid symbols now
  - perhaps ‘??’ = sNaN (signaling NaN)
  - ‘payloads’ on NaNs?

Ordering

- IEEE 754r has a *total order* for numbers
  - –0 is ‘lower’ than +0
  - 1.000 is ‘lower’ than 1.0
  - +Infinity is ‘lower’ than ‘NaN’
  - etc.

- Could define the strict comparison operators to work this way on numbers
  - risky … probably better to provide a BIF

Useful BIFs

- IsNaN, IsInfinite
- Quantize (shorthand for format(x,,n))
- Normalize (strip trailing zeros)
- Num2ieeebits (convert actual bits)
  - and vice versa

BIF changes

- DataType(x, ‘N’)
  - could accept Infinities/NaNs
  - or a new option (‘E’?) for extended numbers
- Format() would probably need some work
  - (reduced exponent range)
- Sign(x) … need to be careful about –0

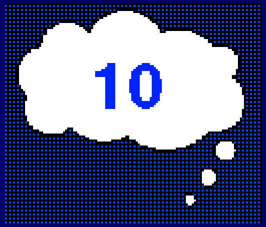
Implementation

- The decNumber C package supports both IEEE 754r arithmetic and formats and the ANSI X3.274 (Rexx) arithmetic
  - and it's open source (in GCC tree)…

- Includes enhanced power function, exp, log10, ln ($\log_e$), square-root, quantize

Questions?

Google: decimal arithmetic



Format details

IEEE 754r: common ‘shape’

Sign	Comb. field	Exponent	Coefficient

- Sign and combination field fit in first byte
  - combination field (5 bits) combines 2 bits of the exponent (0−2), first digit of the coefficient (0−9), and the two special values
  - allows ‘bulk initialization’ to zero, NaNs, and ± Infinity by byte replication

Exponent continuation

Sign	Comb. field	Exponent	Coefficient

Simple concatenation

Format	exponent bits	bias	normal range
32-bit	2+6	101	-95 to +96
64-bit	2+8	398	-383 to +384
128-bit	2+12	6176	-6143 to +6144

(All ranges larger than binary in same format.)

Coefficient continuation

Sign	Comb. field	Exponent	Coefficient

- Densely Packed Decimal – 3 digits in each group of 10 bits (6, 15, or 33 in all)
- Derived from Chen-Ho encoding, which uses a Huffman code to allow expansion or compression in 2–3 gate delays