IEEE 754r arithmetic for Rexx

RexxLA, Raleigh — April 2008

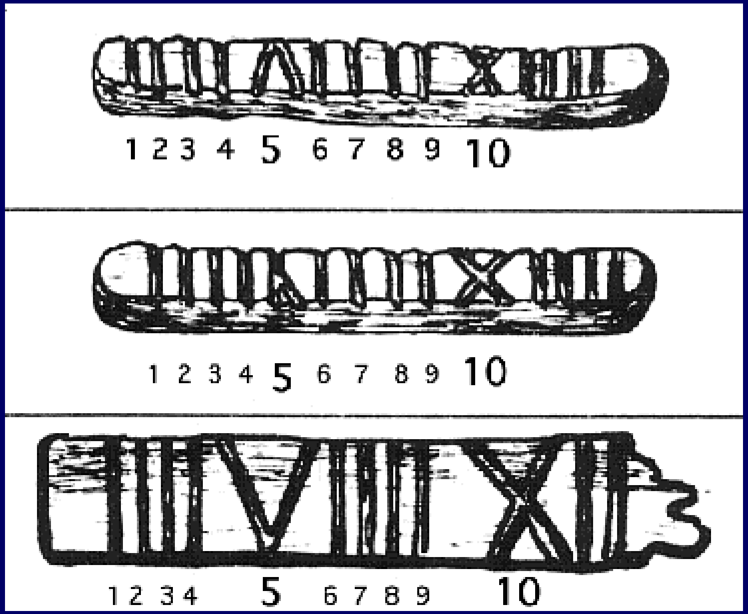
Mike Cowlishaw
IBM Fellow

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 has been 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 Products

- PowerPC (POWER6) and mainframe (z10) processors now have decimal floating-point units in hardware, compliant with current 754r draft
- Appropriate software support:
  - operating system (z/VM, z/OS, AIX, *etc*.)
  - C compilers (GCC, IBM AIX, z/Os, i/OS, Linux) and PL/I, *etc.*
  - DB2 database (z/OS, UNIX, Windows, Linux)

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
  - basic support released in GCC 4.2

Other standards, *etc.*

- COBOL already has floating-point decimal, adding new type for 2008 standard
- ECMAScript (JavaScript/JScript) editions 3.1 and 4 converging on a decimal type
- XML Schema 1.1 draft now has *pDecimal*
- New SPEC benchmarks (SPECjbb, *etc.*)

Other standards, *etc.* [2]

- Other languages have added decimal arithmetic (Python, Eiffel, Ruby, *etc.*)
- ANSI/ISO SQL … new types accepted in principle (waiting on IEEE 754)
- Strong support expressed by Microsoft, SHARE, academia, and many others

Differences from Rexx arithmetic

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

Format	precision	normal range
64-bit	16 digits	-383 to +384
128-bit	34 digits	-6143 to +6144

… there are some subtle edge effects at the exponent extremes because all hardware encodings are valid data

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
  - (optional, so Rexx does not have to change)

Other differences [3]

- 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 … better to provide a BIF

Other differences [4]

- IEEE 754r has five rounding modes; Java and hardware have more (eight)
  - HALF_UP, HALF_EVEN, TRUNCATE are the most important
  - Rexx has only the one rounding mode

IEEE 754r support in Rexx

- The differences are very minor, but are sufficiently obscure that they could be surprising if applied to current programs
- 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

Proposed IEEE 754r support

- Turned on by: numeric form ieee16
  or: numeric form ieee34
- Sets digits=16 or 34
  - numeric digits can then be used to switch between these, but not any other value
  - numeric fuzz an error; current setting ignored
- Arithmetic then follows IEEE rules

Rounding modes

- New: numeric rounding <mode>
- Sets rounding mode
  - only allowed or has effect if form is ieeeNN?
  - ‘numeric rounding value <expr>’ too?
  - 5, 7, or 8 modes defined?
  - strings ‘HALF_UP’, *etc.,* more or less de facto standard

Infinities and NaNs

- With ieee16 or ieee34: "Infinity", "NaN", and "sNaN" accepted for arithmetic
  - 'sNaN' is signaling NaN (with error message, perhaps 35.2 "Signalling NaN encountered")
  - payloads accepted on NaNs (*e.g.*, 'NaN99')
- Environment symbols .!, .?, and .?? preset constants with those values (no payload)

Essential BIFs/Methods

- Quantize [similar to format(x,,n)]
  - quantize(x, 0.01) is format(x, , 2)
  - explicit rounding mode very useful: quantize(x, 0.01, ‘HALF_EVEN’)
- Round [to precisions other than 16 or 34]
  - again, explicit rounding mode very useful
- Rounding() [returns current numeric rounding]
- Num2ieeebits [convert actual bits & vice versa]

Useful BIFs/Methods

- IsNaN, IsInfinite
- Fused multiply-add [FMA]
- SquareRoot
- CompareTotal [with total ordering]
- Normalize [strip trailing zeros]
- logb [return exponent] and scaleb [x 10^N]
- log10, exp10, generalized power

BIF changes

- DataType(x, *option*)
  - do not change existing behavior for option ‘N’
  - add a new option (‘E’?) for extended numbers

- Form() can return ‘IEEE16’ or ‘IEEE34’

- Other BIFs need no changes
  - *e.g.*, D2X is still an error if passed ‘Infinity’

Better class support

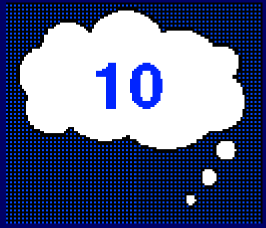
- ::OPTIONS directive
  - *e.g.,* OPTIONS FORM IEEE16
  - applies to entire package/source file
  - Rick suggest might have other uses

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