2006 RexxLA International Rexx Language Symposium Proceedings

René Vincent Jansen (ed.)

THE REXX LANGUAGE ASSOCIATION
RexxLA Symposium Proceedings Series
ISSN 1534-8954

Publication Data

A publication of **RexxLA Press**

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

The RexxLA Symposium Series is registered under ISSN 1534-8954
The 2006 edition is registered under ISBN 978–9-4-0-36-0-000-0

2024-03-12 First Edition
2024-05-05 Second Edition

Introduction

History of the International Rexx Language Symposium

In 1990, Cathie Dager of SLAC[1] convened the organizing committee for the first independent Rexx[2] Symposium for Developers and Users. SLAC continued to organize this annual event until the middle of the 1990's when the RexxLA took over that responsibility. Symposia have been held annually since 1990.

About RexxLA

During the 1993 Symposium in La Jolla, California, plans for a Rexx User Group materialized. The Rexx Language Association (RexxLA), as it was called, is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language. RexxLA manages several open source implementations of Rexx.

The selection procedure

Presentation proposals are solicited yearly using a CFP[3] procedure, after which the RexxLA symposium comittee reviews them and votes which presentations are selected for the symposium. The presentations are peer reviewed before being presented. Presenters are not compensated for their presentations.

Location

The 2006 symposium was held in Austin, Texas, USA from 9 Apr 2006 to 13 Apr 2006.

Organizing Committee

- Chip Davis
- David Ashley
- Gil Barmwater
- Lee Peedin
- Mark Hessling
- Rony G. Flatscher

[1] Stanford Linear Accelerator Center, since 2008 SLAC National Accelerator Laboratory
[2] Cowlishaw, M. F., **The REXX Language** (second edition), ISBN 0-13-780651-5, Prentice-Hall, 1990.
[3] Call For Papers.

Contents

1

Enhanced Arithmetic for Rexx – Mike Cowlishaw

Date and Time

10 Apr 2006, 14:15:00 CET

Presenter

Mike Cowlishaw

Presenter Details

Mike Cowlishaw is the creator of Rexx and has worked in both hardware and software design and is currently the Editor of the IEEE 754 Standard for Floating-Point Arithmetic. He has long been interested in the human aspects of computing, including the Rexx and Java programming languages, colour perception, neural networks, text editing, mapping, panorama viewers, and decimal arithmetic. He is an IBM Fellow (retired), a Fellow of the Royal Academy of Engineering, and a Visiting Professor in the Department of Computer Science at the University of Warwick, UK.

Enhanced Rexx Arithmetic

RexxLA, Austin — 10 April 2006

Mike Cowlishaw
IBM Fellow

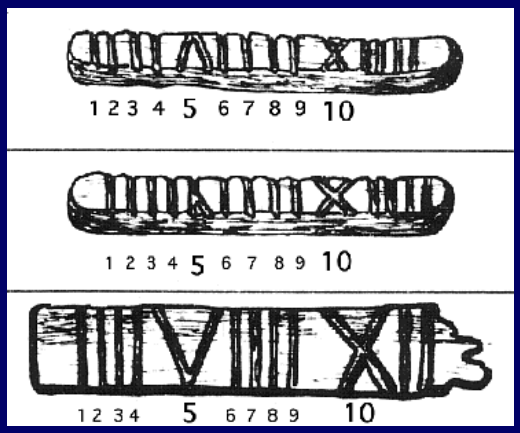
Overview

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

2

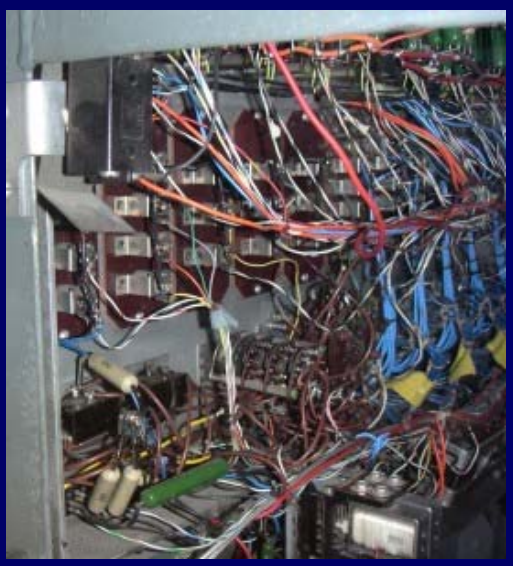
Origins of decimal arithmetic

- 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: 0.0001100110011001…)

5

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

  0.7349999999999999866773237044981215149164199829101562 5

  (should have been 0.735)
- rounds to $ 0.73, instead of $ 0.74

6

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

9

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)

10

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

11

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.*

12

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)

17

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

18

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

19

IEEE 754r support in Rexx

- Support could be very simple:

numeric form scientific
engineering
ieee

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

20

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

- 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

31

2

But I don't use objects... or do I? – Rick McGuire

Date and Time

10 Apr 2006, 16:00:00 CET

Presenter

Rick McGuire

Presenter Details

As at 2008: Rick was the developer charged with integrating Mike Cowlishaw's original Rexx interpreter into VM/CMS back in 1982. From 1982 until 1995, Rick was IBM's lead architect for Rexx issues and principal developer of the Classic Rexx and Object Rexx interpreters for OS/2. Since 1995, Rick has been heavily involved in IBM projects involving programming languages, including 3 years as a member of IBM's Java Virtual Machine development team.