Rexx Arithmetic
– inspiration for a Standard

Rexx Symposium – 31 August 2016

Mike Cowlishaw
http://speleotrove.com
mfc@speleotrove.com
Overview

- Rexx arithmetic

- Status in 2001
  - Software
  - Hardware
  - Standards

- The IEEE 754 standard 2001–2008
  - Current status

- Questions?
Why is decimal arithmetic important?

- Decimal arithmetic represents numbers in base ten, so uses the same number system that people have used for thousands of years.

- Pervasive for financial and other commercial applications; often a legal requirement.

- 55% of numeric data in commercial databases are decimal (and a further 43% are integers).
The trouble with binary

Decimal:

\[ 0.1 = \frac{1}{10} = 1 \times 10^{-1} = 1\text{E}-1 \]

Binary:

\[ = 0.0001100110011\ldots \]

\[ = \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{512} + \frac{1}{4096} + \frac{1}{8192} + \ldots \]
Repeated division by 10; what users expect:

<table>
<thead>
<tr>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
</tr>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>0.09</td>
</tr>
<tr>
<td>0.009</td>
</tr>
<tr>
<td>0.0009</td>
</tr>
<tr>
<td>0.00009</td>
</tr>
<tr>
<td>0.000009</td>
</tr>
<tr>
<td>0.000009</td>
</tr>
<tr>
<td>9E - 7</td>
</tr>
<tr>
<td>9E - 8</td>
</tr>
</tbody>
</table>
Repeated division by 10; what they get:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>float (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>0.09</td>
<td>0.0899999996</td>
</tr>
<tr>
<td>0.009</td>
<td>0.0090</td>
</tr>
<tr>
<td>0.0009</td>
<td>9.0E-4</td>
</tr>
<tr>
<td>0.00009</td>
<td>9.0E-5</td>
</tr>
<tr>
<td>0.000009</td>
<td>9.0E-6</td>
</tr>
<tr>
<td>9E-7</td>
<td>9.00000003E-7</td>
</tr>
<tr>
<td>9E-8</td>
<td>9.0E-8</td>
</tr>
</tbody>
</table>
Where it costs real money …

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

- $1.05 \times 0.70$ using binary double is exactly

  \[
  0.7349999999999998667732370449812151491641998291015625
  \]

  (should have been 0.735)

- rounds to $0.73$, instead of $0.74$
'Minimal' arithmetic – minimum necessary to be useful (loop counters, etc.)

Integers only

This was a 'holding' implementation, while other parts of the interpreter were designed and implemented
Plain decimal arithmetic (no exponents)

Up to 9 digits after the decimal point

Precision of result is determined by the more precise of the two terms involved in an operation

\[ \frac{8}{3} \rightarrow 2 \quad \frac{8}{3.00} \rightarrow 2.67 \]

Worked well, but results could often surprise
Rexx 2.50 – the 'new' arithmetic [May – July 1981]

- Developed primarily by e-mail

- Initially controversial (because it changed the behaviour of existing programs)

- Widely discussed and researched (REX [sic] was in use in 43 countries by then)

- Essentially the same as the arithmetic in "The Rexx Language" book (1985 & 1990)
The choice of arithmetic

- The principle:

"REX arithmetic attempts to carry out the usual operations in as 'natural' way as possible. What this really means is the rules which are followed are those which are conventionally taught in schools and colleges."

(7 Oct. 1981)
Rexx arithmetic

- Full-function finite decimal floating point arithmetic

- Preserves significand length, etc. For example, $1.23 + 1.17$ gives 2.40 (not 2.4)

- Integers are a seamless subset of all numbers

- Precision is user-selectable (numeric digits)

- Exponents from E-999999999 through E+999999999
ANSI (X3-J18) refinements

- Trigger to exponential notation after 0.000001
  (not dependent on DIGITS setting)

- LostDigits condition (raised if input data too precise)

- Input data rounded to DIGITS (not DIGITS+1)

- Published as ANSI X3.274-1996 (see www.rexxla.org), refined through 1999
Decimal arithmetic in 2001
Decimal fixed scale numbers

- Example: 1234.50
- Integer and scale

123450 2
Status in 2001 (software)

- C/C++, PL/I, etc: 15-31 digit fixed-scale decimal (e.g., 2 digits after the decimal point)
- COBOL: 31 digit fixed-scale decimal
- Databases: 31 or 38 digits, various arithmetics
- Java: unlimited finite floating point decimal (by Sun and IBM)
- C#, VB, etc. (Microsoft .Net platform): 28 digit partly-floatiing-point decimal
- Rexx family: unlimited finite floating point decimal
Status in 2001 (hardware)

- z-Series (IBM S/390): decimal integer instructions (Store-to-Store) used for fixed-scale arithmetic

- Most non-RISC processors (Intel x86, Motorola 68xxx, etc.) had decimal adjust instructions to aid decimal integer arithmetic

- In general, decimal arithmetic had to be carried out in software; 100x to 1000x slower than hardware (or worse)
Standards in 2001

- ANSI X3.274-1996 (Programming Language REXX)
  - floating-point arbitrary precision decimals

- IEEE 854-1987 (Radix-Independent Floating-Point Arithmetic)
  - generalization of IEEE 754, to allow for base-10
  - fixed precision
Decimal representations

- Example: 1234.50

- Traditional fixed-scale numbers: integer and scale

- Floating point: coefficient and exponent

  - many advantages in the coefficient being an integer
Rationalizing decimal arithmetic

- General Decimal Arithmetic specification:
  - Arithmetic and encodings, suitable for hardware or software implementation
  - Core floating point and integer operations based on ANSI X3-274 (Rexx)
  - Specification extended to comply with IEEE 854

- Open specification; (still) available on the web
IEEE 754 revision, 2001–2008

- Proposal based on Rexx arithmetic
- UC Berkeley key supporters (Prof. Kahan, etc.)
- Refined to allow efficient hardware (registers etc.)
- By 2005, agreed decimal formats and arithmetic
- Ballot process took another three years …
64-bit Decimal encoding

- Sign, exponent, and coefficient encoded:

  1 5 bits  8 bits  50 bits

- First bit is sign
- Combination field indicates NaN or Infinity, or holds first digit of coefficient and ~two bits of exponent
- Third field has further 8 bits of exponent
- Fourth field has 3x10 bits (3x5 digits) of coefficient (densely packed decimal; DPD)
## IEEE 754 decimal formats

<table>
<thead>
<tr>
<th>size (bits)</th>
<th>digits</th>
<th>exponent range</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>7</td>
<td>-95 to +96</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>-383 to +384</td>
</tr>
<tr>
<td>128</td>
<td>34</td>
<td>-6143 to +6144</td>
</tr>
</tbody>
</table>

(range is always larger than same-size binary format)
Benefits of the new types

- Hardware performance
  - Individual operations up to 150x faster than software packages
  - Applications up to 2x faster overall
  - Almost all commercial and financial applications get a performance boost
Power6 processor core layout

DFPU (Decimal Floating-Point Unit)
Benefits of the new types \[2\]

- Standard data types for decimal bring all the benefits that binary applications enjoy:
  - known, standard, formats, so no conversions (less checking needed at interfaces)
  - faster processing (especially with hardware)
  - well-defined arithmetic, rounding rules, \textit{etc.}
  - safer and easier-to-write applications
IEEE 754 revision, 2015 – 2018?

- Start point IEEE 754-2008 (ISO/IEC 60559:2011)
- Clarifications and minor additions only
- No changes to decimal arithmetic (so far…)
- New Chairman (David Hough); the Editor is still Mike Cowlishaw
Summary

- Decimal data and arithmetic predominate in commercial calculation

- IEEE 754-2008 includes decimal floating-point arithmetic, based on Rexx arithmetic

- Now well-established, in hardware and software

- New revision of IEEE 754 is in progress
Questions?

http://speleotrove.com/decimal