

# Support for Decimal Floating-Point in C (Document N1016)

ISO/JTC1/SC22/WG14 — 21 October 2003

Mike Cowlshaw  
IBM Fellow

<http://www2.hursley.ibm.com/decimal>

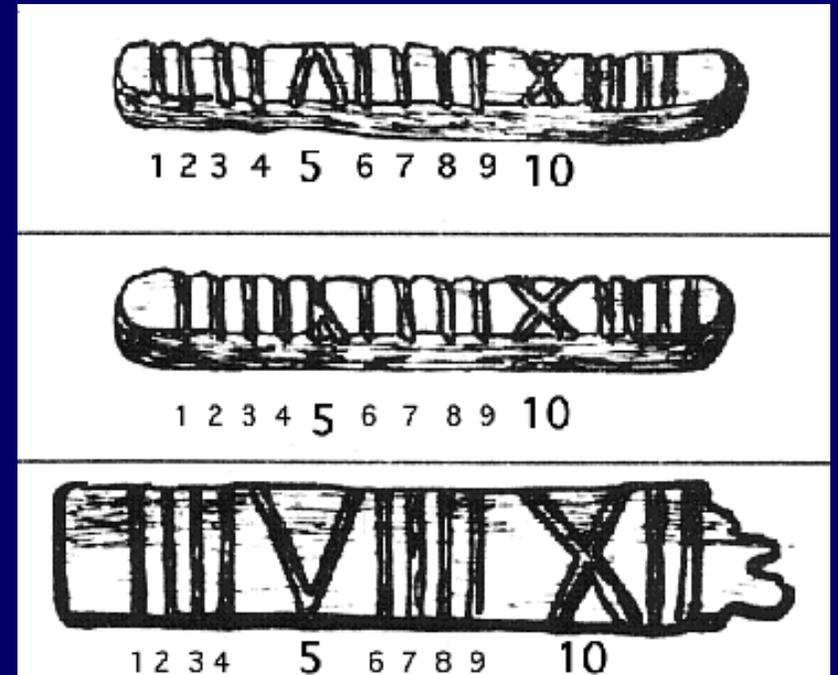


# Overview

- Why decimal arithmetic is increasingly important
- Why hardware support is being built
- The agreed IEEE 754R decimal types
- Summary of proposed C support

# Origins of decimal arithmetic

- Decimal (base 10) arithmetic has been used for thousands of years
- Algorithm (Indo-Arabic place value system) in use since 800 AD
- Many early computers were decimal



# Decimal computation today

- Pervasive in financial, commercial, and human-centric applications
  - often a legal requirement
- Benchmarks do not reflect actual use
- 55% of numeric data in databases are decimal (and a further 43% integers)

# Why floating-point?

- Traditional integer arithmetic with 'manual' scaling is awkward and error-prone
- Floating-point is increasingly necessary
  - division and exponentiation
  - interest calculated daily
  - telephone calls priced by the second
  - taxes more finely specified
  - financial analysis, statistics, *etc.*

# Why not use binary FP?

- binary fractions *cannot* exactly represent all decimal fractions
- $1.2 \times 1.2 \rightarrow 1.44 ?$ 
  - 1.2 in a 32-bit binary float is actually:  
1.2000000476837158203125
  - and this squared is:  
1.440000057220458984375

# A financial example...

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

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

0.734999999999999999998667732370449812151491641998291015625

(should have been 0.735)

- rounds to \$ 0.73, instead of \$ 0.74

# Hence...

- Binary floating-point cannot be used for commercial applications
  - cannot match values computed by hand
  - cannot meet legal and financial requirements, which are based on 3,700+ years of decimal arithmetic
- So applications use decimal software floating-point packages...

# ...but decimal software is slow...

some measurements ...

times in $\mu s$	$x=y+z$	$x=y*z$	$x=y/z$
Java BigDecimal	1.250	1.100	8.440
Binary hardware	0.006	0.006	0.078
Decimal penalty	208x	183x	108x

(These are 9-digit timings. 27-digit calculations are 9x worse for multiply and divide.)

# Effect on real applications

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



	Java BigDecimal	fastest C package
time in decimal operations	93.2 %	72.2 %

# Effect on real applications [2]

- A 'Web Warehouse' benchmark used float binary for currency and tax calculations
- We added realistic decimal processing...

	transactions per second
float binary	3,862
decimal	1,193

69% of workload is decimal arithmetic



# The path to hardware...

- A 2x to 10x performance improvement in applications makes hardware support very attractive; IBM is developing it now
- IEEE 854 tells us how to compute the *value* of floating-point results
- We can use redundant encodings to allow fixed-point and integer arithmetic, too

# Two-integer decimals

- value = coefficient  $\times 10^{\text{exponent}}$
- integer, fixed-point, and floating-point numbers in one representation
  - integers always have exponent = 0
  - in general: *numbers with the same number of decimal places have the same exponent, and need no alignment for addition*

e.g.,      1.23 and 123.45 both have exponent -2  
          [123, -2] and [12345, -2]

# Example: multiplication

- The significands are multiplied (an integer operation), and the exponent is the sum of the operand exponents

$$123E-2 \times 45E-1 \text{ gives } 5535E-3$$

$$122E-2 \times 45E-1 \text{ gives } 5490E-3$$

- Independent calculations for the two parts
- No further processing is necessary unless rounding (*etc.*) is needed

# Integer-based floating-point

- Compatible with:
  - IEEE 754/854
  - manual processes (algorithm)
  - legal requirements
  - programming language data types  
(COBOL, PL/I, Java, C#, Rexx, Visual Basic, *etc.*)
  - databases (DB2, SQL Server, Oracle, *etc.*)
  - application testcase data formats
  - mixed-type arithmetic: 12 x \$ 9.99



# IEEE 754R decimals

- These are new primitive data types, in hardware, coexisting with binary floating-point, but maybe ...

“... in the relatively distant future, the continuing decline in the cost of processors and of memory will result (in applications intended for human interaction) in the displacement of substantially all binary floating-point arithmetic by decimal”

# C support

- C currently allows radix 10 floating-point (in principle) — but this would redefine `float` and `double`, preventing the use of binary libraries and data in decimal programs (and vice versa)
- Hence the need for new primitive types
- Support for these is mostly obvious, but...

# Some discussion areas

- Naming (N1016 uses `_Decimal32`, *etc*)
- Conversions to/from float & double (automatic or require cast?)
- Constants and constant expressions  
For example, in C#:

```
decimal d=1.23; // a compile-time error
```

# C support

- Naming, conversions, and other details are sufficiently tricky that standardization is both necessary and valuable
  - a Technical Report is probably the best route
  - N1016 (Raymond Mak's document) is intended to show what it might look like

# Questions?

<http://www2.hursley.ibm.com/decimal>

(Google: decimal arithmetic)