RETROSPECTIVE:

How to Read Floating Point Numbers Accurately

William D Clinger

College of Computer and Information Science
Northeastern University
Boston, MA 02115, USA
will@ccs.neu.edu

ABSTRACT

Converting decimal scientific notation into binary floating point is nontrivial, but this conversion can be performed with the best possible accuracy without sacrificing efficiency.

1. INTRODUCTION

Having learned to count on their fingers, humans like to express real numbers in decimal scientific notation. Most computers are designed to calculate using numbers that are expressed in IEEE-standard binary floating-point notation.

Although every binary floating-point number can be expressed in decimal scientific notation by using enough digits, most numbers that are expressible in decimal scientific notation cannot be expressed in binary floating-point. For example, 0.1 is not expressible in binary floating-point. The value of the closest IEEE double precision floating-point approximation to 0.1 is

.1000000000000000055511151231257827021181583404541015625

A decimal-to-binary conversion routine that always delivers the closest floating-point approximation to its input, and breaks ties according to the current rounding mode (typically round-to-even), is said to perform *correct rounding*.

My PLDI paper gave the first description of an efficient algorithm for correctly rounded decimal-to-binary conversions [3]. Section 10 of that paper describes its motivation and its relationship to the paper by Steele and White in that PLDI and this collection [16].

IEEE-conforming decimal-to-binary conversions have been allowed to lose almost twice as much accuracy as would be lost by correct rounding [9]. When my PLDI paper was published, almost all implementations did lose this much accuracy at times. This source of inaccuracy could conceivably affect the result of a numerically unstable computation. More importantly, this loss of accuracy has made decimal scientific notation less attractive for exchanging numerical data between different systems [15]:

> Unfortunately, the IEEE standard does not guarantee that the same program will deliver identical results on all conforming systems. Most programs will actually produce different results on different systems for a variety of reasons. For one, most programs involve the conversion of numbers between decimal and binary formats, and the IEEE standard doesn't completely specify the accuracy with which such conversions must be performed.

20 Years of the ACM/SIGPLAN Conference on Programming Language Design and Implementation (1979-1999): A Selection, 2003.

In consequence of the research described below, most computer systems now perform correct rounding, and several language standards now require correct rounding of decimal-to-binary conversions. The committee that is revising the IEEE standard for binary floating point arithmetic is considering a proposal to require correct rounding.

2. IMPLEMENTATIONS

Within months of PLDI '90, David Gay improved upon my Algorithm Bellerophon by replacing the extended precision floating point calculation by a standard-precision floating point calculation combined with a high-precision integer calculation [5]. Gay also noted an easy case that I had missed, simplified the algorithm by using a uniform error bound for all hard cases, and reduced the table sizes.

Gay also improved upon Steele and White's Dragon algorithm. He implemented both improved algorithms in C, and made his code available for anyone to use [6]. Gay's code was slower than previous conversion routines on hard cases (for which the previous routines were often less accurate), but was so much faster on typical cases that Gay's code was faster overall.

Gay's code was also more robust and more accurate. These advantages were demonstrated by David Hough's *Testbase* program, and were documented in a manuscript written by Vern Paxson under the direction of William Kahan [8, 14]. Most major workstation vendors soon incorporated Gay's code into their standard libraries.

The implementation of correctly rounded decimal-to-binary conversion for the IBM S/390 apparently began with my algorithm instead of Gay's code [1].

Meanwhile I had implemented correctly rounded conversions in Scheme for MacScheme, which ran on the Apple Macintosh, and made my code available to other implementors of Scheme and Common Lisp [13]. Robert Burger and Kent Dybvig implemented correct rounding for Chez Scheme, and made their code available to other implementors [2]. Most major implementations of Scheme now provide correctly rounded conversions.

3. STANDARDS

The Scheme standards cite the PLDI '90 papers, and require numeric conversion routines to preserve the value of a number across a *round-trip* of output followed by input, but do not actually require correct rounding [10, 11]. Scheme also requires the output routine, for each individual number, to generate the minimum number of digits that allows this round-trip requirement to be satisfied. This makes Scheme's round-trip requirement more stringent than the round-trip requirement of the IEEE floating-point standard, where the number of digits that are needed to avoid loss of accuracy during a round-trip is independent of the number. Hence implementations of Scheme cannot just rely on IEEE-conforming conversion routines. The best way to satisfy Scheme's i/o requirements is to provide correctly rounded conversions.

It appears that Java was the first programming language to require correctly rounded decimal-to-binary conversions [17]. The specification of `java.lang.Double.valueOf(String)`, for example, says that a syntactically correct input string

> is regarded as representing an exact decimal value in the usual "computerized scientific notation"; this exact decimal value is then conceptually converted to an "infinitely precise" binary value that is then rounded to type double by the usual round-to-nearest rule of IEEE 754 floating-point arithmetic.

XML Schema, which was approved as a W3C Recommendation on 2 May 2001, requires correct rounding of decimal-to-binary conversions, citing both my paper and Gay's [3, 5, 19].

The committee that is revising the IEEE 754 standard for binary floating-point arithmetic has already voted to encourage correct rounding of all binary-decimal conversions. A proposal that would actually require correct rounding has been written and will soon be considered by the committee [18].

4. CORRECTION

In section 9 of my PLDI paper, I reported that "some compilers do not implement IEEE arithmetic correctly." This was an overstatement, as the IEEE 754 standard concerns itself primarily with the low-level operations as they would be implemented in hardware or in library routines, and does not specify many of the language-level and compiler-level details that determine the behavior of floating-point arithmetic as seen by most programmers and users. In particular, "the IEEE standard requires that each result be rounded correctly to the precision of the destination into which it will be placed, but the standard does not require that the precision of that destination be determined by a user's program" [15].

5. ACKNOWLEDGEMENTS

I have relied on BibTeX entries that were collected or written by Guy Steele and Nelson Beebe for the committee that is revising the IEEE 754 standard [18].

REFERENCES

[1] P. H. Abbott, D. G. Brush, C. W. Clark III, C. J. Crone, J. R. Ehrman, G. W. Ewart, C. A. Goodrich, M. Hack, J. S. Kapernick, B. J. Minchau, W. C. Shepard, R. M. Smith, Sr., R. Tallman, S. Walkowiak, A. Watanabe, and W. R. White. Architecture and software support in IBM S/390 Parallel Enterprise Servers for IEEE floating-point arithmetic. *IBM Journal of Research and Development*, 43(5/6):723–760, 1999. At `www.research.ibm.com/journal/rd/`, see `435/abbott.html`.

[2] Robert G. Burger and R. Kent Dybvig. Printing floating-point numbers quickly and accurately. *ACM SIGPLAN Notices*, 31(5):108–116, May 1996. (Proceedings of PLDI '96).

[3] William D. Clinger. How to read floating point numbers accurately. *ACM SIGPLAN Notices*, 25(6):92–101, June 1990. (Proceedings of PLDI '90).

[4] W. H. J. Feijen, A. J. M. van Gasteren, D. Gries, and J. Misra, editors. *Beauty is our business: a birthday salute to Edsger W. Dijkstra*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1990.

[5] David Gay. Correctly rounded binary-decimal and decimal-binary conversions. Technical Report 90-10, AT&T Bell Laboratories, November 1990. At `http://www.ampl.com/REFS/`.

[6] David Gay. `dtoa.c`. At `http://www.netlib.org/fp/`, November 1990.

[7] David Gries. Binary to decimal, one more time. In Feijen et al. [4], chapter 16, pages 141–148. This paper presents an alternate proof of Knuth's algorithm [12] for conversion between decimal and fixed-point binary numbers.

[8] David Hough and Vern Paxson. Testbase. At `http://www.netlib.org/fp/`, 1991.

[9] IEEE Computer Society, New York. *IEEE Standard for Binary Floating-Point Arithmetic*, 1985. IEEE Standard 754-1985.

[10] IEEE Computer Society, New York. *IEEE Standard for the Scheme Programming Language*, 1991. IEEE Standard 1178-1990.

[11] Richard Kelsey, William D. Clinger, and Jonathan Rees. Revised^5 report on the algorithmic language Scheme. *Journal of Higher-Order and Symbolic Computation*, 11(1):7–105, 1998. Also appears in *ACM SIGPLAN Notices* 33(9), September 1998.

[12] Donald E. Knuth. A simple program whose proof isn't. In Feijen et al. [4], chapter 27, pages 233–242. This paper discusses the algorithm used in TEX for converting between decimal and scaled fixed-point binary values, and for guaranteeing a minimum number of digits in the decimal representation. See also [3] for decimal to binary conversion, [16] for binary to decimal conversion, and [7] for an alternate proof of Knuth's algorithm.

[13] Lightship Software. *MacScheme manual and software*, 1990.

[14] Vern Paxson and William Kahan. A program for testing IEEE decimal-binary conversion. At `ftp.ee.lbl.gov`, see `testbase*`, May 1991.

[15] Doug Priest. Differences among IEEE 754 implementations. At `http://www.validlab.com/`. This was written as Appendix D for David Goldberg's *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, *ACM Computing Surveys*, March 1991.

[16] Guy L. Steele Jr. and Jon L. White. How to print floating-point numbers accurately. *ACM SIGPLAN Notices*, 25(6):112–126, June 1990. (Proceedings of PLDI '90).

[17] Sun Microsystems. *Java 2 Platform, Standard Edition, v1.2.2, API Specification*, 1999.

[18] 754R working group. Some proposals for revising ANSI/IEEE Std 754-1985. At `http://754r.ucbtest.org/`.

[19] World-Wide Web Consortium. *XML Schema Part 2: Datatypes*, May 2001. At `http://www.w3.org/TR/xmlschema-2/`.