IEEE Standard for Floating-Point Arithmetic revision due in 2019

revised 06/18/19 04:25:25 PM

IEEE Std 754-2008 for Floating-Point Arithmetic has expired, and so a bug-fix-and-minor-enhancements revision activity began in 2015. A draft has now been approved by the IEEE Standards Board as IEEE Std 754-2019. The simplified Scope of the new draft:

> This standard specifies formats and operations for floating-point arithmetic in computer systems. Exception conditions are defined and handling of these conditions is specified.

The 754 working group considered 50 drafts over a period of three and a half years. That's a lot of work for bug fixes and minor enhancements like augmented arithmetic and payload operations! To put that effort into perspective, let's review the history of 754:

Diversity: The 1960's were the flowering of the mainframe era, epitomized by IBM 360 and CDC 6600 and DEC 10, but with many other contenders as well. The 1970's saw a corresponding boom in minicomputers, epitomized by the DEC PDP-11 and, again, many others. These systems all had floating-point arithmetic, usually programmed with Fortran compilers. But that was about all one could generalize. Each arithmetic was different in greater or lesser ways, sometimes even within the same instruction-set architecture. Quite an engineering art developed in how to write programs that would run equally correctly when compiled with these differing compilers targeted at different hardware. The "portable" numerical programs that resulted were often much more complicated than corresponding programs targeted at just one system.

Standards: Meanwhile microprocessor development was taking over the steepest part of the development curve. Although the single-chip processors available around 1976 had no floating-point hardware and often not much integer arithmetic hardware beyond addition and subtraction, technologists and executives could foresee that powerful arithmetic engines would be feasible in a few years. So Intel hired John Palmer to gather customer requirements and coordinate a specification for floating-point software for Intel processors, with the expectation that hardware would be the next step. Intel also consulted William Kahan from the University of California, who in the course of mathematical error analysis for scientific computing, had unearthed and publicized the consequences of many bad choices made in mainframe and minicomputer floating-point hardware, system software, and compilers. At the same time, Bob Stewart and others in the IEEE decided it would be more efficient and timely to standardize aspects of microprocessors in advance, by achieving consensus among individual technical experts, rather than trying to standardize the traditional way, after the fact, by achieving consensus among commercial organizations. Thus the IEEE Microprocessor Standards Committee was commissioned to promote "timely development of great technical standards." In 1977 these diverse threads converged in the IEEE 754 working group for binary floating-point arithmetic. Under Dick Delp and Dave Stevenson, the working group labored to produce IEEE Std 754-1985 for Binary Floating-Point Arithmetic.

1985: The motivation for 754-1985 was to make it easier to provide portable robust mathematical software. It's far more efficient to put a little more effort into the hardware and system software specification than much more effort into ~~the~~ keeping all the world's portable mathematical software working. 754-1985 specifies formats, operations, rounding, and exceptions, which affect all levels of hardware and software, but in several places it reads mostly like a hardware specification, referring to

condition codes and traps and global mode and status bits. Mathematical software providers would have preferred to standardize high-level language facilities, but there was no chance of getting language standards interested until there were some hardware implementations and some compilers and libraries proving the concepts. So the intent was to get some hardware implementations, then subroutine libraries to provide access, then compilers to generate the obvious mappings of language constructs to hardware, and finally compilers to add language features that would allow access to the novel aspects of the standard. Typical contentious issues of this effort included – why not standardize DEC VAX arithmetic which was closest to the proposed standard; how should underflow be handled ; why not do something really novel like symbols for overflow and underflow intervals, or variable-length exponent fields? Over time these issues were resolved and the final consensus was not very different from the early "CKPPS": proposal drafted by Jerome Coonen, Kahan, Palmer, Tom Pittman, and Harold Stone. Dave Goldberg wrote an excellent summary of 754-1985 for ACM Computing Surveys [1]; see also [5].

1987: In the midst of 754's work, a parallel effort under Jim Cody tackled decimal floating-point arithmetic and word lengths other than 32 and 64 bits. Although potentially of greater import to ordinary computer users who think in decimal rather than binary, this process was not contentious as the main issues had been settled in 754-1985. IEEE Std 854-1987 for Radix-Independent Floating-Point Arithmetic was the result, but it had little industrial impact compared to 754-1985.

2008: 754-1985 was adopted in whole or part by most microprocessor architectures as they began to incorporate hardware floating-point instructions. Because 754-1985 couldn't be implemented entirely in hardware, however, manufacturers developed subroutine libraries to provide access to the novel aspects of the standard – in particular, modes and exception flags. The libraries tended to be different, so that once again portable mathematical software development was inhibited. And the latitude allowed to implementors in 754-1985 had more negative consequences than anticipated, as described by Doug Priest ["Differences Among IEEE 754 Implementations" in https://docs.oracle.com/cd/E19060-01/stud8.compiler/817-0932/ncg_goldberg.html]

By now, C had become an important system and application programming language, so Rex Jaeschke convened a subgroup of the C Language standards committee to start developing proposals for numerical C extensions, including full support of 754-1985. Much of their work was incorporated in the 1999 standard for C. Meanwhile, 754-1985 had expired and had been quietly renewed several times, So in 2001, a new 754 working group convened under Bob Davis and Dan Zuras to produce a comprehensive revision to supersede 754-1985 and 854-1987. The goal was ambitious – encompassing incorporating insights from 854-1987, decimal arithmetic proposed by Mike Cowlishaw of IBM, the fused multiply-add operation, a new 16-bit binary format, specifications for elementary transcendental functions, and higher-level language facilities for dealing with modes and exceptions and expression evaluation and optimization. Such an ambitious project became quite contentious on a number of technical points. But the biggest challenge was a dispute that led to standardizing two encodings for decimal floating-point formats. No end user asked for that, but the compromise was accepted to prevent stalling out on the rest of the standard. The most significant change from 754-1985 is turning from specification close to hardware to specification close to higher-level language constructs. This was the ultimate intent all along, but seemed premature in 754-1985. But now, ISO/IEC Technical Specification 18661 specifies extensions to C to support nearly all of 754-2008 [8]. These specifications are candidates for inclusion in the next version of the C standard, C2X. Read more about 754-2008 in [7].

2019: 754-2008 expired in 2018. 754-2008 veterans, eager to avoid another 8 year ordeal, intended 754-2019 to be an upwardly compatible bug-fix-and-minor-enhancements revision. The bug fixes are many, and mostly refinements of language and elimination of accidental inconsistencies; the description of comparison operations was substantially rewritten to better clarify the original intent. The minor enhancements are new recommended operations

- quantum for decimal formats
- tanPi, aSinPi, and aCosPi
- augmented{Addition,Subtraction,Multiplication}
- {min,max}imum{,Number,Magnitude,MagnitudeNumber}; NaN and signed zero handling are changed from 754-2008 5.3.1.
- {getPayload,setPayload,setPayloadSignaling}

All new operations are "recommended" even when we really meant "required," to retain upward compatibility from 754-2008. Several additions were inspired by existing features of C Language support for 754-2008 arithmetic. For various reasons, these had been omitted from 754-2008. The new Pi operations complete the set defined in 754-2008. The payload operations allow applications to read and write payloads of NaNs in an implementation-independent way.

Somehow 754-2008 incorporated a defective definition of the {min,max}{Num,NumMag} operations, which weren't associative in the presence of NaNs. So that definition was removed from 754-2019. Instead new operations {min,max}imum{,Number,Magnitude,MagnitudeNumber} are defined that are associative. Implementations can conform to both 754-2008 and 754-2019 by providing all of these operations, but the defective ones are deprecated.

The most interesting new feature of 754-2019 is the augmented arithmetic operations. They provide the exact result of an addition, subtraction, or multiplication in two parts that add up to the exact result. These operations were added because hardware implementations of similar functionality appeared imminent and we hoped to have them work identically and to provide the most useful functionality. Toward that end, a new rounding method – round to nearest, ties toward zero – was defined for these operations. So defined, they are useful for two target applications – bit-wise reproducible vector summation independent of the order of summation, for example in the presence of varying numbers of threads[2, 3], and "double-double" software that extends the highest hardware precision [4], used in high-precision mathematical calculations and some physics simulations such as climate prediction and solar system stability.

More extensive background discussions of the rationales for the new operations and explanations for some confusing parts retained from 754-2008 may be found in http://754r.ucbtest.org/background/.

2029: 754-2019 was self-constrained to be upwardly compatible with only minor extensions from 754-2008. But new kinds of computational demands might eventually encompass new kinds of standards, particularly for fields like artificial intelligence, machine vision and speech recognition, and machine learning. Some of these fields obtain greater accuracy by processing more data faster rather than by computing with more precision – rather different constraints from those for traditional scientific computing. Old ideas like block floating point have become new again [6]. There might be arithmetic standards dedicated to very specific application areas, rather than compromises intended to be suitable for a wide range of diverse applications. So the next generation of application programmers

and error analysts will face new challenges and have new requirements for standardization. Good luck to them!

References

1. David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.* 23, 1 (March 1991), 5-48. DOI: 10.1145/103162.103163

2. James Demmel, Jason Riedy, and Peter Ahrens. Reproducible BLAS: Make Addition Associative Again! *SIAM News*, 51 (8): 8, October 2018. https://sinews.siam.org/Details-Page/reproducible-blas-make-addition-associative-again

3. Jason Riedy and James Demmel. Augmented Arithmetic Operations Proposed for IEEE-754 2018. In *25th IEEE Symposium on Computer Arithmetic (ARITH 25)*, June 2018. DOI 10.1109/ARITH.2018.8464813.

4. D. H. Bailey, "High-precision floating-point arithmetic in scientific computation," in *Computing in Science & Engineering*, vol. 7, no. 3, pp. 54-61, May-June 2005.
DOI: 10.1109/MCSE.2005.52

5. Michael L. Overton. *Numerical computing with IEEE floating point arithmetic*. Vol. 76. SIAM, 2001.

6. J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London; Prentice-Hall, Englewood Cliffs, N.J., 1963.

7. J..-M. Muller et al, *Handbook of Floating-Point Arithmetic*, Birkhäuser, Basel, 2018.

8. ISO/IEC TS, Information technology - Programming languages, their environments, and system software interfaces - Floating-point extensions for C:

    18661-1:2014, Part 1: Binary floating-point arithmetic.

    18661-2:2015, Part 2: Decimal floating-point arithmetic.

    18661-3:2015, Part 3: Interchange and extended types.

    18661-4:2015, Part 4: Supplementary functions.

    18661-5:2016, Part 5: Supplementary attributes.