Intel and Floating-Point

Updating One of the Industry's Most Successful Standards

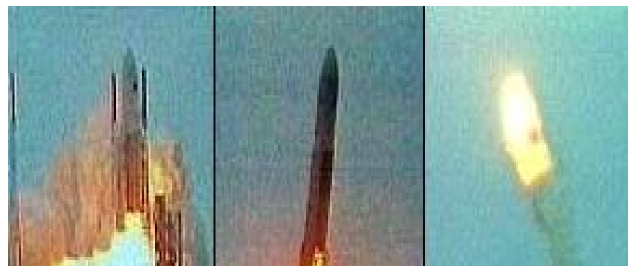
The Technology Vision for the Floating-Point Standard

Most people today would never expect different answers to the same mathematical calculation performed on different microprocessors. But before 1985, it happened all the time. That's because there was no standard for floating-point numbers — the way computers handle real numbers.

Real numbers with non-recurring decimal representations, such as pi, are problematic for computers because they only have a finite number of bits with which to represent each number.

IEEE Standard 754 for Binary Floating-point Arithmetic changed all that by providing a set of specifications for computers to follow. Approved in 1985, this standard became vitally important nearly immediately — especially when one considers the type of things that happen when it is disregarded. According to Professor William Kahan, University of California at Berkeley, a classic case occurred in June 1996. A satellite-lifting rocket named *Ariane 5* turned cartwheels shortly after launch and scattered itself and a payload worth over half a billion dollars over a marsh in French Guiana. Kahan found the disaster could be blamed upon a programming language that disregarded the default exception-handling specifications in IEEE 754. Upon launch, sensors reported acceleration so strong that it caused a conversion-to-integer overflow in software intended for recalibration of the rocket's inertial guidance while on the launching pad. This software could have been disabled upon rocket ignition, but leaving it enabled had mistakenly been deemed harmless. The software ended up triggering a system diagnostic that dumped its debugging data into an area of memory being used by the programs guiding the rocket's motors. At the same time, control was switched to a backup computer that unfortunately had the same data. This was misinterpreted as necessitating strong corrective action and the rocket's motors swiveled to the limits of their mountings. Disaster ensued. Had overflow merely obeyed the IEEE 754 default policy, the recalibration

Computers approximate real numbers using floating-point arithmetic. This operation involves some approximation or rounding because a number may be too long to represent. Floating-point arithmetic employs scientific notation and a "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 (10^{12} x 1.0) to 0.000000000001 (10^{-12} x 1.0) with ease.



The Ariane 5 exploded seconds after launching.

software would have raised a flag, delivered an invalid result to be ignored by the motor guidance programs, and the Ariane 5 would have pursued its intended trajectory.

Preventing mishaps like these was an important part of the technology vision for IEEE 754. Another important consideration was the ever-increasing performance capabilities of personal computers. Every generation of computers is much more powerful than the computers sold in 1976 when Intel first began to design a floating-point co-processor for its i8086/8 and i432 microprocessors. In the late 70s, the typical microprocessor was an 8-bit CPU with 40,000 transistors and an 8-bit bus. These microprocessors ran a few million instructions per second on programs and data that fit into a computer's main memory of a megabyte or less. The floating-point processor was often an attached unit running 1 million floating-point operations per second (FLOPS).

In *binary* floating-point arithmetic, a floating-point number represents an integer or fixed-point number multiplied by the base 2 to some integer power. Floating-point numbers are the binary analog of scientific notation in base 10.

Fast forward to today. The average laptop computer is many times faster than the fastest supercomputer that existed in 1975. For that matter, the computer on your desk could nearly handle the processing done by all the world's computers back then and still have enough capacity left over to play a computer game. An Intel® Core™ i7 processor introduced in 2010 with six 64-bit CPUs (cores) has 1.17 billion transistors and at 3.3 GHz can reach up to 158.4 GFLOPs in single precision ($158 \cdot 10^9$ floating-point operations per second), and half that in double precision.

With change as large as that, the technology vision for floating-point calculations merits change as well. Where once a floating-point program might have run into a problem every billion or trillion operations (say, every few hours or a few times a year), today that problem comes up anywhere from several times a second to many times an hour.

The number of programs requiring floating-point operations has increased dramatically as well. It's not just scientific, CAD and other math-intensive applications. It's also communications, security, graphics, and games. Consider a game character throwing

Professor Kahan cites another example of the dangers of not abiding by IEEE 754. In 1997, the *Aegis* missile-cruiser *Yorktown* spent almost three hours adrift off Cape Charles, Virginia. Its software-controlled propulsion and steering was disabled as it waited for its operating system to be rebooted after a division-by-zero error from a database program that had interpreted an accidentally blank field as zero. If the software had followed the IEEE standard, the mistake would have generated an answer of infinity that would have resulted in an unknown and an error message. Instead, the software tried to compute it, crashed the operating system, and left the ship traveling in a broad circle until its crew finally succeeded in rebooting the operating system. Meanwhile, the ship had no control over its engines, steering or weaponry.

an axe. Everything from the force with which it is thrown to its flight path and where it lands requires determining the physics of motion and how that object looks at each instant in real time as it moves across the screen. Such realistic rendering requires an immense amount of calculations. Back in 2000 when IEEE 754 came up for renewal, it was time to look for ways to update the standard for the present day and the upcoming demands of tomorrow's computing.

Before There Ever Was a Floating-Point Standard

Programmers of floating-point computations in the 1960s and 1970s had to cope with each computer brand (and often models within that brand) supporting its own range and precision for floating-point numbers. Each manufacturer rounded off arithmetic operations in their own, sometimes peculiar, way. Not only that, but some computers were binary and some were decimal. (A Russian computer even used trinary arithmetic.)

Equally challenging were the peculiar ways each handled various situations that a particular arithmetic problem could create. For instance, with one computer, numbers behaving as non-zeros during comparison and addition might behave as zeros in multiplication and division. This was a problem because you can't divide by zero (think back to your math classes.) Before a number could safely be used as a divisor with this computer, a programmer had to have inserted code for multiplying the number by 1 and then comparing to zero. If the number was equal to zero, it couldn't be used. You couldn't necessarily try this with another computer though. The same trick of multiplying by 1 on a different computer might lop off the last four bits and give a different answer.

Having so many different ways of handling floating-point arithmetic created an anarchy that forced people to deal with individual anomalies in their own way. This made making reliable numerical software that was "portable" from one type of computer to another extremely expensive. A few forward thinkers began to realize that as microprocessors continued to proliferate, no one would soon be able to afford adapting a single math-intensive application to them all. The computers, the I/O devices, the languages, compliers and arithmetic were so different from one machine to the next, that rewriting a program for another computer was becoming a monumental task that involved a great deal of debugging to make it work.

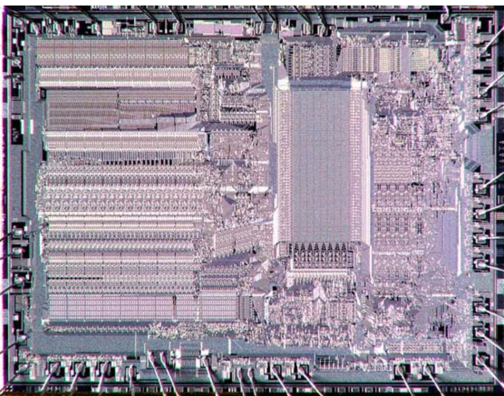
Professor William Kahan (Circa 1975, Photo: Peg Skorpinski)

Setting the Stage for the First IEEE Floating-Point Standard

In 1976, in the midst of all these different ways of handling floating-point numbers, Intel began designing a floating-point co-processor for its i8086/8 and i432 processors. Dr. John Palmer, manager of Intel's floating-point effort, persuaded the company it needed an internal arithmetic standard to prevent different Intel microprocessors from giving different mathematical results. Palmer remembered a professor, William Kahan, he had heard at Stanford some 10 years earlier. Kahan had been involved with computers since 1953, back in the days when vacuum tube computers had a mean time between failures of about five minutes. In the second half of the 60s, Kahan had worked with IBM on improving their handling of floating-point numbers. In the 70s, Kahan had helped enhance a successful line of Hewlett-Packard calculators. Palmer recruited Kahan as a consultant to Intel's floating-point efforts.

Intel began seeing excellence in floating-point arithmetic as a potential competitive advantage — a way to differentiate their processors from other chip manufacturers at the time. Intel also realized it was on the cusp of something big — the market for microprocessors was soon to explode and so was the opportunity to sell massive numbers of floating-point co-processors. Intel gave Kahan license to pursue the best floating-point solution possible.

Kahan assembled and integrated all the best floating-point arithmetic features he knew of to achieve a mathematical regularity and integrity that previous solutions had lacked. Kahan then worked with Intel engineers on solutions for fitting all the necessary algebraic operations and library of functions into the i8087's read-only memory (ROM). At first, because of the size, this seemed an impossible task. But Intel engineers in Israel developed a solution enabling storage of two bits per transistor instead of one —and thus solved a potentially limiting space issue.



Intel 8087 Floating-Point Co-Processor Die

Rumors about the i8087 started circulating and other companies began to look at a standards effort as a way of keeping a level playing field. Professor Kahan attended one of the meetings and then requested permission from Intel to participate. Palmer gave Kahan the go-ahead to disclose most of the specifications for the i8087, but not its architecture or transcendental functions. (Transcendental functions are functions which "transcend," i.e., cannot be expressed in terms of, algebra. Examples of transcendental functions include the exponential function, the trigonometric functions, and the inverse functions of both.)

What Kahan could share included precisions, exponent ranges, special values, and storage formats, and the reasoning behind the decisions that had been made. For competitive reasons, Intel didn't want to give away its upcoming surprise — a chip with only 40,000 transistors that had most of the essentials of a math library on it.

The Proof is in the Performance

Kahan collaborated with a student, Jerome Coonen, and a visiting professor, Harold Stone, at U.C. Berkeley on a draft specification that they submitted to the IEEE p754 working group. It became known as the K-C-S draft. It was one of several proposals. Initial reaction to the K-C-S draft was that it was complicated, but had a good rationale for everything. Kahan, knowing that a lot of code involving floating-point would be written in the future by people who knew little about numerical analysis, wanted to make sure their programs would get the right results. His other goal was to ensure that the standard would enable people who really were expert in floating-point to write truly portable software that would work as well on one microprocessor as another.

The double precision format requires two 4-byte storage locations in computer memory, at *address* and *address+1* in order to represent a floating-point number. This is how computers with 32-bit stores (single precision) provide the 64-bit double precision format.

The selection came down to two proposals: Kahan's work for Intel and an existing DEC VAX format that had a large installed base. Initially DEC's format was seen as inferior because its exponent range was too narrow for some double precision computations. But then DEC introduced a proven double precision format with the same exponent range as the K-C-S draft.

Attention at this point turned to another big difference: How the two floating-point proposals handled underflow. Underflow occurs when the result of a floating-point operation is smaller in magnitude (closer to zero, either positive or negative) than the smallest quantity representable. The DEC solution flushed underflow to zero, a strategy favoring performance but that had the troubling side effect of occasionally causing software to malfunction. While these malfunctions were rare in the 1970s, some of the working group was concerned about what would happen when computers became a thousand times more numerous and arithmetic became a thousand times faster.

Underflow refers to the condition that occurs when a computer attempts to represent a number that is too small for it (that is, a number too close to zero). For example, if your computer supports a precision of 6 digits and the exponent range allows a minimum of -99, then the smallest non-zero number it can support is 10^{-99} x 1.00000. If a calculation produces a smaller number such as 10^{-99} x 0.3, an underflow condition occurs. Programs respond to underflow conditions in different ways. Some report an error, while others approximate as best they can (a process called gradual underflow) and continue processing.

The K-C-S draft employed gradual underflow to reduce the risk of software malfunctions. This meant that subnormal numbers (non-zero numbers smaller than

the smallest normal number allowed by the precision in the floating-point implementation) were produced that would allow a calculation to lose precision slowly when the result was small, rather than all at once.

The argument against gradual underflow was that it would degrade performance of the fastest arithmetic because of the extra steps it required even if no underflows occurred. Intel and Kahan had already come up with a solution for implementing gradual underflow in hardware without delaying all floating-point operations, but didn't want to reveal it. The dispute was resolved when a U.C. Berkeley graduate student built the K-C-S floating-point solution onto two accelerator boards for a VAX. Substituting these boards in a VAX and running the VAX instruction set proved that there were no performance sacrifices due to the K-C-S draft's use of gradual underflow.

The IEEE Standard 754-1985 for Binary Floating-Point Arithmetic was a nearly decade-long effort by a 92-person working group of university mathematicians, computer scientists and engineers, computer manufacturers, and microprocessor companies.

By now, support was growing for the K-C-S draft. The main argument against it finally centered on the value of its gradual underflow solution for numerical software. This solution clearly helped prevent some software malfunctions, but did it really do anything for the arithmetic?

To determine this, a highly respected error-analyst, Professor G.W. Stewart III from the University of Maryland, was commissioned in 1981 to assess the value of gradual underflow. He concluded it was the right course to take.

The Standard's Adoption and Success

Despite the strong support, the movement of the K-C-S draft towards ratification was slow as wording changes and various small compromises delayed it. Nevertheless, by 1984 the draft was already being implemented in products by Intel, AMD, Apple, AT&T, IBM, Motorola, National Semiconductor, and others. In 1985, IEEE 754 officially became an industry standard.

In 1989, Professor Kahan received the ACM Turing Award (the unofficial Nobel Prize for the computing industry) for his work on floating-point and IEEE 754.

Looking back, the standard has been an enormous success. All computers now conform either fully or to a large extent to the standard — including specialized chips such as DSP or graphics chips. According to one IEEE 754 revision committee member, it has been the "pivotal flagship example of IEEE standards and one of the most implemented and far-reaching of any IEEE standard." Since 1984, more than 1.2 *billion* Intel processors alone have conformed to the standard (source: IDC database). The standard's influence has extended up to mainframes. Many general purpose computers (computers designed to perform functions required by both

sophisticated scientific and business applications) claim conformance to at least a large subset of the standard.

Software developers have benefited tremendously from the standard as well. They can more or less take it for granted that when they write a program that works with real numbers, it will behave in a specific way when run on various microprocessors. Instead of many different floating-point formats, there's just one. Instead of having to troubleshoot and come up with ingenious little bits of code to make answers come out right on each different microprocessor, everyone simply counts on IEEE 754 to ensure consistency. Having the standard has allowed people to move beyond trying to make floating-point work properly on a computer to basing new work on it.

"The standard doesn't provide a guarantee that the answers are the "right" answer. That may require careful error analysis. However the standard provides many mechanisms that make it easier for a program to get the "right" answer on conforming computers from different vendors." — John Crawford, Intel Fellow

"The fact that Intel presented this gift [Intel's specifications for floating-point arithmetic] was a phenomenal act of altruism on Intel's part." — Professor William Kahan, U.C. Berkeley

That said, it's important to note that IEEE 754 doesn't guarantee a "right" answer, only a consistent one. Determining the correctness of an answer can in special cases require careful error analysis. What IEEE 754 does do is provide a high level of mathematical regularity and integrity for software that requires it.

Revising the IEEE 754 Standard

IEEE standards have about a 15-year lifespan. That means IEEE 754 was up in 2000. Through yearly extensions, the standard and the revision process were extended for a number of years.

But more than simple expiration was driving the revision process. As mentioned early in this article, each generation of computers and their microprocessors drastically advance technology. Each generation is incredibly more powerful and runs much more sophisticated and math-intensive software — programs we don't even think of math-intensive, such as games. Whereas the original Intel floating-point co-processor required 40,000 transistors, today's many-core processors have parallel single instruction multiple data (SIMD) instruction sets for floating-point, and dedicate around 1 million transistors per core to handling floating-point operations. This means a quad-core processor could have around 4 million transistors for floating-point operations. Despite this large number, each transistor in a floating-point unit is carefully thought out for its contribution to performance before the design is replicated in each core of millions of processors.

The nature of computing also has changed dramatically since 1985. Back then, the best one could hope for in graphics rendering for a game would be moving one large

or complicated object around on the screen and putting up with frequent waits for the image to refresh. But today, with much more powerful computers, we're asking for far more. Going back to the physics of motion involved in a game character throwing an axe, if we do this in 32-bit calculations, small round-off errors will cause tiny shape changes or details in the image to come and go as the axe moves across the screen. Some of these will be big enough to be seen by the eye. Perform these calculations in 64-bit arithmetic (double precision) and the round-off errors become small enough that any imperfections will be too small to be seen by the human eye. In a growing number of cases, double precision may not be enough. For example supernova simulations, climate modeling, planetary orbit calculations, quantum field theory, and experimental mathematics require a higher level of numeric precision.[1] Such precision is generally only required in scientific computing applications where a very long train of computations may lead to a single result. To meet this need, quad precision was added to the new IEEE 754r revised floating-point standard (which became IEEE 754-2008 upon its adoption).

To revise IEEE 754, a committee of as many of the original working group as could be assembled was formed. This included Professor Kahan and some of his former students. A number of new people in the field were included as well. The group started with the question "what needed to be changed," and came up with some key things to consider:

1. Fixing various minor ambiguities in the 1985 standard that were known only to a few experts.
2. Extending the standard to cover "half precision" (also known as "Float 16," a 16-bit storage format used for graphics and in DirectX), and "quad precision" (128-bit format), together with generalized formulae for some wider formats.
3. Adding fused multiply-add.
4. Incorporating decimal floating-point into the standard.
5. Improving exception handling and providing recommendations to language standard implementers.
6. Introducing optional data types for supporting other fixed width floating-point formats, as well as arbitrary precision formats (cases where the precision of representation and rounding are determined at execution time) and the ability to specify the size of the significand.
7. Adding transcendental functions.
8. Making minor changes to the operations that convert floating-point numbers to integers and back again.
9. Not invalidating any existing computer through a proposed change for the revised standard.

In computing, a *fused multiply-add* (FMA) computes a multiply-accumulate — FMA(A, B, C) = AB + C — with a single rounding of floating-point numbers. When implemented in a microprocessor, this is typically faster than a multiply operation followed by an add. FMA is already implemented on Intel® Itanium® and Itanium 2 processors.

This last point was particularly important. When the original standard was introduced, there wasn't any standardized way to do floating-point operations. Now several billion computers from manufacturers all over the world use the standard. They can't be changed to conform to a revised standard. For the committee, that meant new capabilities could be added, but not changes that would invalidate existing computers.

Intel was prominent in the revision process. Former Intel mathematician Jeff Kidder was vice chair. Intel members of the committee included former Intel senior principal engineer Peter Tang, Intel Fellow John Crawford, senior principal engineer Roger Golliver, senior software engineer John Harrison, and principal engineer Marius Cornea.

Implementation Strategies for Decimal Floating-Point Arithmetic

IEEE 854, commonly known as the radix-independent floating-point standard, came out in 1987, two years after IEEE 754. While not truly radix-independent, it defined both binary and decimal floating-point arithmetic. Decimal arithmetic makes numerical calculations more human-friendly. Results will be as people expect them, identical to what would be obtained using pencil and paper. Decimal arithmetic also provides a robust, reliable framework for financial applications that are often subject to legal requirements concerning rounding and precision of the results in the areas of banking, telephone billing, tax calculation, currency conversion, insurance, or accounting in general.

Decimal (base 10) floating-point arithmetic provides an exact representation of displayed numbers and provides a precise round at the decimal radix point (i.e., 10 in base 10). This type of arithmetic is used in financial calculations.

Unlike IEEE 754, IEEE 854 was never implemented or used widely. Decimal floating-point arithmetic calculations were done just as easily (and often faster) with software solutions or in binary floating-point. Having a variety of software solutions for how people do decimal floating-point was never much of a problem because, through the present day, there haven't been enough decimal arithmetic-intensive applications, or speed and accuracy issues, to create interest in IEEE 854.

Nevertheless, the revision of IEEE 754 brought an opportunity to combine in one standard both binary and decimal floating-point standards. It's expected combining the standards will do for decimal floating-point what IEEE 754 did for binary floating-point — namely, get everyone on the same page in how they implement decimal floating-point arithmetic.

IEEE 754-2008 defines three decimal floating-point formats with sizes of 32, 64, and 128 bits. Two encodings for each of these formats have been adopted: the

decimal, or Densely Packed Decimal (DPD) encoding, and the binary, or Binary Integer Decimal (BID) encoding (the terms *decimal encoding* and *binary encoding* are the used in IEEE Std 754-2008). BID-based encoding is the best suited for software implementations of the decimal floating-point arithmetic and is seeing wide adoption from companies that make financial software and database applications.

BID-based encoding may also turn out to be the best solution for hardware implementation as well. A 2007 paper presented at the IEEE International Conference on Computer Design reported that BID is particularly well suited for hardware implementations, since it can share hardware with binary arithmetic units.[2] The paper discussed a proposed hardware design of a BID-based floating-point adder and showed that it could be achieved with a modest area increase on a processor core. According to the paper, "over 70 percent of the BID adder's area is due the 64-bit fixed-point multiplier, which can be shared with a binary floating-point multiplier and hardware for other DFP [decimal floating-point] operations."[3]

From a cost-effectiveness and performance standpoint, a software implementation offers the best solution for now. To help ensure compliance with IEEE 754-2008 when implementing the functions defined for decimal floating-point arithmetic operations, Intel offers the Intel® Decimal Floating-Point Math Library. This software package is used as a computation engine by compilers or other tools requiring decimal floating-point support and provides a fast and efficient solution for financial calculations that cannot be carried out correctly in binary floating-point arithmetic. It's designed to run on any platform in Linux, Windows, HP-UX, Solaris, and OSX.

What Lies Ahead

IEEE 754-2008 was published in August 2008. Top programming languages such as C, C++ and Fortran will be addressing the new standard in the course of working on revisions of their standards in coming years.

Just as the computing landscape changed dramatically from the ratification of the first floating-point standard, equally big changes are in store for the next 15 to 20 years. The increasing number of cores in processors and Intel's continual innovation in microprocessor architecture are driving an ongoing revolution in software languages as software developers continue to adopt new ways, such as parallelism, in their applications to make use of the advancing technology. This will enable users to better tap the ever-greater leaps in performance (and performance per watt) being made in personal computers. In this new landscape, the revised IEEE floating-point standard is a welcome update and should continue the standard's legacy as one of the most implemented and successful IEEE standards for years to come.

The revised standard provides an opportunity for the industry to benefit from its improvements, new features and performance enhancements, as well as to improve

implementation of the standard in products. This will lead to greater realism in real-time graphics, more accurate simulations, and greater precision and more consistent implementation of decimal floating-point arithmetic in financial calculations.

Learn More

Visit the IEEE 754: Standard for Binary Floating-Point Arithmetic Web site (http://grouper.ieee.org/groups/754/)

Read:

- "An Interview with the Old Man of Floating-Point — Reminiscences from William Kahan" (http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html) by Charles Severance.
- "Why Do We Need a Floating-Point Arithmetic Standard?" (http://www.cs.berkeley.edu/~wkahan/ieee754status/why-ieee.pdf) by William Kahan
- "BID — Binary-Integer Decimal Encoding for Decimal Floating-point: A Format Friendly to Software Emulation and Compiler Native Support" (http://754r.ucbtest.org/issues/decimal/bid_rationale.pdf) by Ping Tak Peter Tang, Software and Solutions Group, Intel Corporation
- "A Software Implementation of the IEEE 754r Decimal Floating-Point Arithmetic Using the Binary Encoding Format" (http://www.lirmm.fr/arith18/papers/CorneaM_Decimal_ARITH18.pdf) by Marius Cornea, Cristina Anderson, John Harrison, Ping Tak Peter Tang, Eric Schneider, and Charles Tsen

---

[1] To learn more about scientific computing applications requiring high precision computation, see: http://crd.lbl.gov/~dhbailey/dhbpapers/dhb-jmb-acat08.pdf

[2] C. Tsen, M. J. Schulte, and S. Gonzalez-Navarro, "A Binary Integer Decimal-based Multiplier for Decimal Floating-point Adder," October 2007.

[3] Ibid.