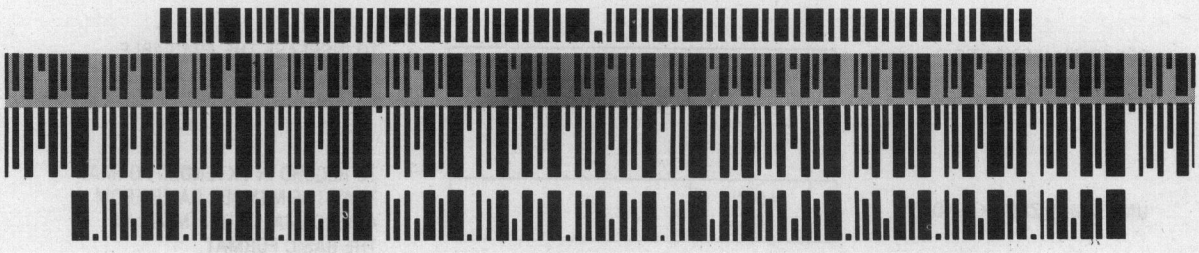
The committee considered three major proposals. Although the one adopted is nontraditional and seems radically different from the others, it shares many details with them.

Analysis of Proposals for the Floating-Point Standard



W. J. Cody
Argonne National Laboratory

During its deliberations, the IEEE Computer Society's Floating-Point Committee, Task 754, considered three main proposals. The first, the so-called KCS proposal, was originally drafted by William Kahan, Jerome Coonen, and Harold Stone. It was based on early work by Konrad Zuse and others, Kahan's 15 years of experience writing mathematical software, and a paper by John Palmer.[1] The other two were counterproposals to certain features of the KCS proposal. The first, the FW proposal, was drafted by Robert Fraley and J. Stephen Walther, and the second, the PS proposal, was drafted by Mary Payne and William Strecker, based on their long experience in designing and implementing minicomputers. After extended, serious discussion within the committee, the KCS proposal received the two-thirds majority support necessary for formal adoption.

The major proposals considered by the committee are publicly available. In addition to the draft standard, which is the final version of the KCS proposal, this presentation borrows heavily from the references, which we highly recommend to the reader. The *ACM SIGNUM Newsletter*[2] contains the formal KCS and PS proposals as they stood in October 1979, a discussion by Fraley and Walther of their objections to the KCS proposal, a discussion by Kahan and Palmer of the advantages of the KCS proposal, and a number of other articles. Coonen's article[3] is an explanation of the KCS proposal without the formalism necessary in a draft standard (no such explanation exists for the other proposals). Finally, Payne has carefully outlined the differences between the proposals.[4,5] The tabular presentation of these differences is especially valuable.[5] There are some minor discrepancies in the discussions because the proposals were "moving targets." Coonen's article, for example, includes a quad-precision data type not contained in the KCS proposal but advocated in the PS proposal.

We outline the KCS proposal first and then indicate how the FW and PS proposals differ. Many of the details as to formats, methods of rounding, and operations are common to all proposals, so some of the discussion of the KCS proposal carries over to the others as well.

The KCS proposal

The KCS proposal specifies

- floating-point number formats;
- results for add, subtract, multiply, divide, square root, remainder, and compare;
- conversions between integers and floating-point numbers;
- conversions between different floating-point formats;
- conversions between basic-format floating-point numbers and decimal strings; and
- floating-point exceptions and their handling, including non-numbers, called NaNs for not-a-number.

The KCS proposal specifies two basic floating-point formats, single and double, that are intended to be fully supported when they are implemented. Any standard-conforming system must support single precision and may also optionally support double precision. In addition, the system may optionally provide an extended format, with limited support, for the widest basic format that is provided. Coonen[3] included a quad-precision which essentially replaced double-extended. We include quad in this discussion, with the understanding that it must be filtered out when considering the pure KCS proposal.

The number of bits in each basic format and the allocation of those bits to the various fields of the format are all

0018-9162/81/0300-0063$00.75

Table 1.
Bit patterns of data formats.

TYPE	BIT PATTERN (SIGN | EXPONENT | SIGNIFICAND)	PURPOSE
NORMALIZED BASIC	≠MIN, MAX |	TO PROVIDE THE GREATEST ACCURACY FOR A FIXED NUMBER OF BITS IN THE SIGNIFICAND
NORMALIZED EXTENDED	≠MIN, MAX | 1	
DENORMALIZED BASIC DENORMALIZED EXTENDED	MIN | NOT ZERO	TO INCREASE THE AVAILABLE FLOATING-POINT NUMBERS TO MITIGATE THE EFFECT OF UNDERFLOW
UNNORMALIZED EXTENDED	≠MIN, MAX | 0	TO RECORD IN EXTENDED FORMAT THAT THE NUMBER CAME FROM A DENORMALIZED NUMBER IN THE BASIC FORMAT.

specified (see Table 1). The basic formats are sign-magnitude representations containing an implicit bit in the significand. Nonzero normalized numbers contain a nonzero exponent field and a significand between 1 and 2. A normalized zero is represented by zero exponent and significand fields. In addition, there are certain "denormalized"numbers with zero exponent fields, no implicit bits, and nonzero (fractional) significands which may contain one or more leading zero bits. They are the default results for underflow. Specific bit patterns are also reserved to represent infinity and NaNs, the default results for invalid operations.

All of the possible single-precision entities are well ordered in the natural lexicographic ordering of their machine representations interpreted as sign-magnitude binary integers (see Table 2). This ordering facilitates compare operations and gives easy access to the successor of a floating-point number.

Coonen's quad format is more conventional in that all significand bits are explicit. The representations of NaNs, infinities, zeros, and unnormalized numbers in quad are what one might expect. Positive infinity, for example, is represented by the largest possible exponent and either a zero significand or a significand with only the first bit nonzero.

Table 2.
Floating-point quantities.

SIGN	EXPONENT	SIGNIFICAND	QUANTITY
0	MAX	>0	+NaN
0	MAX	0	+INFINITY
0	0<E<MAX	>0	+REAL
0	0	>0	+DENORMALIZED
0	0	0	+0
1	0	0	−0
1	0	>0	−DENORMALIZED
1	0<E<MAX	>0	−REAL
1	MAX	0	−INFINITY
1	MAX	>0	−NaN

The easiest way to implement single- and double-precision arithmetic is probably to unpack the bit strings representing numbers into their constituent parts and to manipulate them separately. Extended formats are intended to allow natural exploitation of this unpacked format in certain intermediate computations. Expansion of both the exponent and significand fields of the extended formats simplifies the accurate computation of the elementary functions, and otherwise protects intermediate computations in the corresponding basic format from unnecessary underflow, overflow, and loss of significance. Such an extended format greatly simplifies the computation of the Euclidean norm of a vector, for example. Extended formats would probably only be accessible to assembly language at first, but we suspect that they would rapidly become available to algebraic languages through local enhancements. We can only hope that the language people get into the act early enough to standardize these enhancements in some way—but that is another issue. The double format meets or exceeds the requirements for the single-extended format, and quad similarly passes for a double-extended. Therefore, it is not required to have both single-extended and double, for example.

The proposed standard, which will be known as IEEE Standard 754, carefully specifies several different rounding modes. In each case the rounded result of an operation will be one of the representable neighbors of the infinitely precise true result. This is a source of minor disagreement between KCS and FW on the one hand and PS on the other in the case of the remainder operation. In the RN—round to the nearest even—mode of rounding, the rounded result is the nearest representable neighbor of the true result, with ties being broken in favor of the neighbor with the least significant bit zero. The other modes are RZ, forced round toward zero; RP, round toward plus in-

finity; and RM, round toward minus infinity. RP and RM are provided to facilitate implementation of interval arithmetic. A standard-conforming system must provide all four rounding modes.

The KCS proposal also provides for control over the handling of infinities and zeros. The projective mode, in which there is only one infinity, is the default mode of operation. In this mode the algebraic signs on zero and infinity are ignored and infinity obeys no order relationships. This mode is ordinarily used for rational arithmetic, such as in the evaluation of continued fractions, because it allows for division by zero without risk of undetected anomalies.

The second mode is the affine mode, in which the algebraic signs of zero and infinity are preserved. With appropriate care to be certain that the algebraic signs are not determined by rounding error, the affine mode preserves order relations while fixing up overflow. Thus, for example, the reciprocal of a negative number which underflows is still negative.

The KCS proposal provides for a number of different arithmetic exceptions: invalid-operation, underflow, overflow, division-by-zero, and inexact-result. In every case the default response to an exception is to provide a specified result and proceed after either trapping or setting a "sticky" flag. The use of traps with transfer of control to the user is an implementation option. If traps are used, the system should provide sufficient information for correction of the fault and subsequent continuation of processing. It is intended that the user will have full control over enabling and disabling traps and over clearing and interrogating flags.

Default results for invalid-operations such as 0*infinity are NaNs which are intended to either progagate through subsequent arithmetic operations or be trapped. Their precise informational content is left to the implementation; for example:

- An operating system might initialize storage to NaNs which specify the storage location. Propagation of these NaNs would then aid in debugging programs which reference uninitialized data.
- Trapping NaNs might be used as pointers to nonstandard data formats for unusual arithmetics.

The handling of underflow is one of the most controversial issues in the KCS proposal. Basically, underflow occurs when the exponent of a result becomes too small. This is always signalled by setting a sticky flag. If the underflow trap is enabled, the exponent of the result is wrapped around into the accepted range and the resulting value is delivered to the trap handler. If the trap is disabled, the result is denormalized by right-shifting its significand, incrementing the exponent for each shift, until the exponent is acceptable. This process could result in what appears to be a normal zero, but the underflow sticky bit distinguishes this case from the true normal zero.

Denormalized numbers do not propagate in the usual manner. It is intended that descendants of these numbers will be forever marked, unless the loss in significance traceable to the original denormalization becomes of the same order of magnitude as normal rounding errors. Ordinarily, a denormalized number will disappear quietly only when it is added to a normalized number and the result is also normalized. In most other cases an invalid operation is signalled. This is the default mode of operation, called the warning mode. There is provision for an optional normalizing mode, in which all results are computed as though the operands were normalized. The results for all manipulations of denormalized numbers, such as conversion from one format to another, are carefully specified in the proposal.

An arithmetic exception for an inexact result is signalled whenever a rounding error occurs. When the floating-point system is used for integer arithmetic, floating-point error may be introduced into the least-significant bits of a result. The usual integer arithmetic expects those bits to be correct. If the inexact-result flag is cleared before doing floating-point integer arithmetic and is still clear afterwards, then the floating-point results agree with what would have been obtained using ordinary interger arithmetic.

That is a basic outline of the KCS proposal. We have not discussed some aspects of the proposal, such as the arithmetic tables and the conversions between various formats, nor have we given much detail on those things that were discussed. We fill in some of the missing information below.

Counterproposals

The FW proposal. Fraley and Walther made specific counterproposals to the KCS scheme for the handling of underflow, overflow, zeros, and infinities. The latest version of their draft, which is not reproduced in the references, was distributed to committee members in November 1979. In it they propose arithmetic on a set of operands consisting of zero, infinity, normal numbers, special overflow symbols +OV and −OV, special underflow symbols +UN and −UN, an operand ERR representing an error condition, and a set of reserved operands. These quantities are not well-ordered in the sense that comparisons involving INF and ERR return "unordered" as a result, and comparisons involving a reserved operand usually result in an exception for an invalid operation. The remaining quantities are ordered +OV, +reals, +UN, 0, −UN, −reals, and −OV.

FW provides for essentially the same exceptions as KCS, but the exception for an inexact result is omitted and an exception for a "risky" operation is introduced. As in KCS, the default mode of operation is to set a flag, provide a default result, and proceed unless a trap is enabled. Default results for underflow and overflow are the special symbols UN and OV, respectively, with appropriate algebraic signs. Each of the special operands enters a computation and propagates in a sensible but somewhat pessimistic manner. Once introduced, a UN symbol tends to propagate as either another UN or an OV, or to degenerate into an ERR or a reserved operand. It will disappear quietly only when it is added to a real of sufficiently large magnitude. OV and UN are reciprocals, as are zero and infinity. An ERR operand is generated and propagates in much the same way as a nontrapping NaN,

and a reserved operand behaves much like a trapping NaN.

There are two fundamental modes of operation called "risky" and "careful." The term "risky" is also applied to an operation whenever the result of the operation cannot be guaranteed because of underflow and overflow. Default results are provided for risky operations in both modes. If the result destination of a risky operation has a binary floating-point form, ERR is returned in the careful mode. Otherwise, a propagating risky flag is set. For example, when X is very close to the underflow threshold, UN + X is X with a risky flag in risky mode, and is ERR in careful mode. Otherwise, exception and trap-handling facilities are very similar to those in the KCS proposal, with the obvious mappings between NaNs and ERR, NaNs and reserved operands, etc.

This proposal retains the rounding rules of the KCS proposal while introducing the additional concept of bounding, which is used to avoid − OV and + UN as right endpoints, and + OV and − UN as left endpoints in interval arithmetic. For example, bounding in the RP rounding mode changes − OV and + UN to the largest negative and smallest positive normalized operands, respectively. There are analogous changes for + OV and − UN when bounding in the RM rounding mode.

In most other matters, such as data formats, conversions, and the like, the FW draft is similar to the KCS draft. There are some minor differences but nothing of great importance. For example, there are differences in the minimum and maximum exponents in the basic formats as a result of eliminating denormalized numbers. FW does not provide either quad precision or the affine mode. It also leaves the specific bit patterns for the various entities that it treats to the implementation. Thus it requires two reserved exponent fields for INF, ERR, and reserved operands but does not specify what they shall be.

The PS proposal. Payne and Strecker object to essentially the same features of the KCS proposal as Fraley and Walther do, but they repair things in a more conventional manner. Briefly, quad is required, extended formats are not allowed, there is only one reserved exponent field available for special operands, trapping is mandatory rather than optional when an exception occurs, underflow is flushed to zero when the underflow trap is disabled, and remaindering and conversion between floating point and decimal are not specified. The trapping mechanism of PS and FW may be used to incorporate many of the features of KCS.

To be more specific, the representation of a floating-point number is specified much as in the KSC proposal except that the significand is interpreted as a fraction and the minumum and maximum exponents are different (they also differ from those in the FW proposal). A zero exponent field with a "+" sign field indicates true zero regardless of significand, while a zero exponent with a "−" sign indicates a reserved operand. The normal mode of operation is to trap on an exception and to pass control to a trap handler with enough information for recovery. Trapping is provided for underflow, overflow, division by zero, attempting the square root of a negative number, and encountering a reserved operand. When the user does not provide a trap handler, a default trap handler replaces underflow with a zero result and provides a reserved operand as the result in all other cases. These reserved operands then propagate in the expected way.

There are some other minor differences between PS and KCS. For example, PS has no affine mode. Again, these remaining differences are small.

The critical difference between PS and KCS is that PS has only one reserved exponent for representing special operands. Leaving aside questions of which options are defaults, the major features of KCS could be imbedded in PS were it not for that missing reserved field. In fact, with this one exception, the main features of each of the proposals can be imbedded in each of the other proposals.

These then are the proposals that were debated within the committee. The KCS proposal is radically different from anything most of us have seen before, the PS proposal is almost traditional in its details, and the FW proposal lies somewhere between the others. We urge that each of the proposals be read for technical content. Do not be misled by rough edges in language or sometimes contradictory statements regarding minor points, because the published proposals were primarily working drafts and some had undergone more corrective iterations than others.

The underflow controversy

Now let us return to the basic controversy regarding underflow. The KCS scheme is to implement underflow "gradually" through denormalized numbers. The justification is that the effect of underflow then becomes comparable to that of roundoff. Proponents argued that gradual underflow works better in this sense than flush to zero when both work, that it occasionally works when flush to zero does not, and that it usually provides a warning when it does not work. Opponents argued that it is too complicated to be understood and is therefore likely to be misused, and that it is too expensive to implement. They proposed instead either the familiar flush-to-zero approach or special underflow symbols with appropriate properties.

Fraley and Walther include an example in their SIGNUM paper[2] which is worth examining in detail because it does point up some of the conceptual problems with gradual underflow. As indicated earlier, it is intended that the arithmetic operations on basic format numbers be carried out by unpacking the numbers into wider registers in which the implicit bit in the significand becomes explicit. Denormalized numbers become unnormalized numbers in this unpacked representation. That is, they have the exponent corresponding to the minimum exponent in the basic format, and their significand has the appropriate number of leading zero bits to properly represent the number. Multiplication in general is carried out by generating the sign of the result according to convention, adding the unbiased exponents of the operands, and then multiplying the significands (see Table 3). Recall that the significands of normalized numbers lie between 1 and

2, so the significand of the product of two normalized numbers lies between 1 and 4. If the resulting significand is greater than 2, it is shifted right one place and the exponent is increased by one. Finally, the result is converted back to the basic format and an arithmetic exception is signalled if the result in the basic format is nonstandard in some way. If one of the original operands is denormalized, the result will usually be denormalized. However, normalized results may occur when the original operand is only slightly denormalized. For example, suppose we multiply the slightly denormalized number $X=2^{**}(-126)*(3/4)$ by the normalized number $5=2^{**}(2)*(5/4)$. Then the significand in the unpacked format is $15/16<1$, and an arithmetic exception is raised when the result is converted to single precision. If X is multiplied by $6=2^{**}(2)*(3/2)$, the significand of the unpacked result is properly normalized and no exception is raised. But if X is multiplied by $8=2^{**}(3)*(1)$, the significand of the unpacked result is again unnormalized, and once again an arithmetic exception is signalled.

One side of this controvesy focused attention on the raggedness of the boundary between normalized and denormalized results in this case, claiming that the behavior was difficult to justify or to explain to ordinary users. The other side felt that this seemingly erratic behavior was actually quite normal and no more complicated than rounding error. Their explanation was that the denormalization error for some of the products had simply become comparable to normal rounding error, hence the products need no longer be branded as "different," while for others it had not. They pointed out that some denormalized numbers had contributed to acceptable results in this case, and that this was not possible with the competing proposals for handling underflow.

Each side of this controversy supported its position with similar examples purporting to show that its particular view was superior to other views. In my opinion an infinite number of such examples existed for each side, and the only way to reach a reasonable decision in this matter was to look at the overall picture. We had to choose that alternative which works best in the most commonly occurring stituations and not worry about the rest. I personally support gradual underflow, because I believe it enlarges the set of problems that can be safely solved in a natural way without penalizing previously successful methods. Like any new tool, it is possible to misuse this facility and to have a malfunction, but I do not believe that the facility introduces malfunctions into processes that previously worked. Consider the simple computation

$$(Y-X)+X$$

where $Y-X$ underflows. Then gradual underflow always returns Y exactly, flush to zero returns X, and the FW scheme returns X with a risky flag in the risky mode and ERR in the careful mode. We could look at this as another isolated example, but I prefer to look at it as the preservation of the associative law of addition to within rounding error. That is, under gradual underflow we always have

$$(Y-X)+X=Y+(-X+X)$$

Table 3.
KCS multiplication table, $X*Y$.

		Y			
		0	W	INF	NaN
X	0	a	a	b	Y
	W	a	c	d	Y
	INF	b	d	d	Y
	NaN	X	X	X	M

W is any finite number except normal zero.

M indicates the system's precedence rule is to be applied to nontrapping NaNs.

a: Result = 0 with sign.

b: Signal invalid-operation. If result must be delivered, use a NaN.

c: Compute as follows:

(1) Generate sign and exponent per convention. Multiply significands.
(2) If significand overflows, shift right one and adjust exponent.
(3) Check underflow, round, and check invalid and overflow.

d: If either operand is an unnormal zero, proceed as in b; otherwise result = INF with sign.

to within rounding error. This is compelling, in my opinion.

The controversy between the various proposals was fueled from time to time by manufacturers who announced intentions to produce floating-point chips or systems con-

forming to various versions of the KCS and PS proposals. DEC announced its intention to implement a substantial subset of the PS proposal,[5] and two chips are currently available, the AMD 9512 and the Intel 8232, which almost conform to a subset of the KCS proposal. At this writing, several manufacturers—including Advanced Micro Devices, Intel,[6] Motorola, National Semiconductor, and Zilog—have either produced full implementations of the KCS proposal or are rumored to be working on them. ■

Acknowledgment

This work was supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the US Department of Energy under Contract W-31-109-Eng-38.

References

1. J. Palmer, "The INTEL Standard for Floating-Point Arithmetic," *Proc. COMPSAC 77,* pp. 107-112.
2. *ACM SIGNUM Newsletter,* special issue on the Proposed IEEE Floating-Point Standard, October 1979.
3. J. T. Coonen, "An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic," *Computer,* January 1980, Vol. 13, No. 1, pp. 68-79.
4. M. H. Payne, "Floating Point Standardization," *Proc. COMPCON Fall 79,* pp. 166-169.
5. M. H. Payne and D. Bhandarkar, "VAX Floating Point: A Solid Foundation for Numerical Computation," *Computer Architecture News,* Vol. 8, No. 4, June 1980, pp. 22-33.
6. R. Nave, "A Numeric Data Processor," *Proc. IEEE ISSCC 1980,* pp. 108-109.

W. J. Cody is a senior mathematician at Argonne National Laboratory where he has worked since 1959. His current interests include the approximation and evaluation of elementary and special functions, the design and evaluation of numerical software, and the interaction between computer arithmetic design and numerical algorithms. He received the BS degree in mathematics from Elmhurst College, Elmhurst, Ill., in 1951, the MA degree in mathematics from the University of Oklahoma, Norman, in 1956, and an honorary ScD degree from Elmhurst College in 1977.