Decimal Arithmetic Encoding
Strawman 1

5th July 2002

Mike Cowlishaw

IBM Fellow
IBM UK Laboratories
mfc@uk.ibm.com

Draft – Version 0.72

Table of Contents

Introduction

This document describes proposed encodings suitable for supporting the general purpose decimal arithmetic defined in the **General Decimal Arithmetic Specification**.[1]

This proposal has been submitted to the committee which is currently revising the IEEE 754-1985 standard.[2] This committee intends to add decimal encodings to the next version of the standard; as a result of this process it is expected that the standardized encodings will be similar in principle but different in detail from those proposed here.

The primary audience for this document is implementers, so examples and explanatory material are included. Explanatory material is identified as Notes, Examples, or footnotes, and is not part of the formal specification.

Additional explanatory material can be found in the paper *A Decimal Floating-Point Specification*.[3] For further background details, including other specifications and related decimal arithmetic links, please see the material at the associated web site: `http://www2.hursley.ibm.com/decimal`

Appendix A (see page 7) details the rationale behind the various choices made to arrive at the current specification.

Appendix B (see page 18) summarizes the changes to this specification since the first public proposals.

Comments on this draft are welcome. Please send any comments, suggestions, and corrections to the author, Mike Cowlishaw (`mfc@uk.ibm.com`).

Acknowledgements

The author is indebted to Glenn Colon-Bonet, Dave Raggett, Fred Ris, Eric Schwarz, Ronald Smith, and Charles Webb, who have all directly contributed to this document.

Also, of course, thanks are due to the contributors to earlier work in the area – especially the members of the Radix-Independent Floating-Point Arithmetic Working Group of the Microprocessor Standards Subcommittee of the IEEE, and the members of the X3 Secretariat/CBEMA (now NCITS) Subcommittee J18.

---

1 See `http://www2.hursley.ibm.com/decimal/decarith.html`

2 ANSI/IEEE 754-1985 – *IEEE Standard for Binary Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., New York, 1985.

3 *A Decimal Floating-Point Specification*, Schwarz, Cowlishaw, Smith, and Webb, in the *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (Arith15)*, `http://arith15.polito.it` , IEEE, June 2001

Scope

Objectives

This document describes proposed encodings (concrete representations) which are suitable for supporting the general purpose decimal arithmetic defined in the **General Decimal Arithmetic Specification**.[4]

Inclusions

This specification defines the following:

- The format and layout of single and double precision decimal numbers (64 and 128 bits respectively)
- The range of numerical values which can be represented by the encodings
- The range of precisions which can be represented by the encodings
- Recommendations for other fixed sizes of decimal encodings.

Exclusions

This specification does not define the following:

- The semantics of arithmetic and other operations on encoded numbers
- Exceptions and other consequences of operations on encoded numbers
- Encodings of context information
- Encodings for arbitrary precision decimal arithmetic beyond the sizes for which recommendations are made.

[4] See `http://www2.hursley.ibm.com/decimal/decarith.html`

Specification

This section defines the proposed encodings for both single and double precision decimal numbers. Both precisions allow for values of ±0, ±Infinity, and two kinds of Not-a-Number (NaN) values (which may be signed).

A *single precision* number is encoded in 64 consecutive bits.

A *double precision* number is encoded in 128 consecutive bits.

Recommendations are also included for representations of some other useful lengths.

Fields in the encodings

Each encoding comprises four fields, whose length and permitted range of values may vary depending on the encoding.

The fields are:

1. *sign* – a single bit indicating the polarity of the number.

   In numbers for which the *sign* has meaning (for ordinary numbers, Infinity, and possibly NaN) a 1 indicates the number is negative (or negative zero) and a 0 indicates it is positive or non-negative zero. Where the *sign* does not have meaning it must be 0.

2. *exponent* – an unsigned (non-negative) binary integer in the range 0 through 2^n-1, where *n* is the number of bits in the field. Some encodings in this range are not used.

   The *exponent* has the following characteristics:

   - For numbers (encodings which are not *special values*, that is, neither infinite nor a NaN), the value of the exponent is given by subtracting a *bias* from the binary integer in the field. In all cases, the bias is given by $2^{n-1}-1$, where *n* is the number of bits in the field, as before.

     The range of values used for the *exponent* is limited by the value E_{max}, which is typically related to an integral number of decimal digits and depends on the length of the encoding. The maximum value used is E_{max}, and the minimum value used is given by negating the value of $E_{max}+d-1$, where *d* is the maximum length of the *coefficient* in decimal digits (see below).

     For example, if E_{max} is +999 and *d* is 15, then the range of exponent values used for numbers is –1013 through +999.

- The *special values* are encoded in the binary integer of the *exponent* field thus:

  Infinity is encoded as the binary integer 2

  Quiet NaN is encoded as the binary integer 1

  Signaling NaN is encoded as the binary integer 0.[5]

- All other possible values of *exponent* are not valid and must not be used. Note that in the future some of these values could be used to extend the encoding and would then become valid.

3. *padding* – zero or more bits, depending on the length of the encoding, which are used to ensure right-alignment of the final field. The *padding* bits must all be 0.

4. *coefficient* – an unsigned (non-negative) integer in the range 0 through 10^d-1, where *d* is the maximum length of the integer in decimal digits.

   The *coefficient* fills the remainder of the encoding. It is encoded as zero or more groups of 10 bits, possibly preceded by a group of either 4 or 7 bits, depending on the length of the encoding.

   Each 10-bit group represents three decimal digits, using Densely Packed Decimal encoding.[6] A 4-bit or 7-bit group represents one or two decimal digits respectively, using the same encoding.

   Within the *coefficient*, the most significant group is on the left (is placed first). For example, if the *coefficient* were five decimal digits long, it would be encoded with the two most significant digits first (in a group of 7 bits) followed by the hundreds, tens, and units digits encoded in a group of 10 bits.

   The number of decimal digits required to represent the value of the *coefficient* is called the *length* of the value of the coefficient, which is a positive number (a value of 0 has length 1). This length will often be smaller than the space available in the encoding; in this case leading 0 digits are added before encoding (that is, the *coefficient* is right-aligned).

   The *coefficient* is undefined when the *exponent* indicates that the number is a *special value*. In this case, an implementation may use the bits in the field for its own purposes (for example, to indicate the origin of a NaN value).

The *length* of the value of the *coefficient* may further restrict the possible values of the *exponent*. In the arithmetic defined in the X3.274 subset of the *specification*, the value of the exponent must be in the range $-E_{max}-(d-1)$ through $E_{max}-(d-1)$, where *d* here is the *length* of the value of the *coefficient*. The lower bound is extended down to the smallest *exponent* value that can be used (independent of the value of the *coefficient*) when the full specification is followed, because this allows *subnormal* numbers.[7]

---

[5] An encoded number whose bits are all zero therefore has the value *signaling NaN*, not 0.

[6] See `http://www2.hursley.ibm.com/decimal/DPDecimal.html` for a summary of Densely Packed Decimal encoding.

[7] Subnormal numbers are those below the balanced range of IEEE 854 exponents (see page 16).

The value of an encoded number

The value of an encoding is either a *special value* (see page 4), or it is a finite number whose numerical value is given exactly by: $(-1)^{sign} \times coefficient \times 10^{exponent}$, where each name represents the value of the named field.

For example, if the *sign* had the value 1, the *exponent* had the value –2, and the *coefficient* had the value 250, then the numerical value of the number is exactly –2.5.

Note that more than one encoding may have the same numerical value; if the *sign* again had the value 1, but the *exponent* had the value –1 and the *coefficient* had the value 25, then the numerical value of the number would also be exactly –2.5.

These redundant encodings allow the representation of numbers with trailing fractional zeros, which often convey additional information about a decimal number beyond its simple numerical value.[8]

Single precision encoding

A *single precision* decimal number is encoded in 64 bits, with the following parameters defining the fields:

sign	1 bit
exponent	11 bits; E_{max} is +999
padding	2 bits
coefficient	50 bits (encoding up to 15 decimal digits)

Double precision encoding

A *double precision* decimal number is encoded in 128 bits, with the following parameters defining the fields:

sign	1 bit
exponent	15 bits; E_{max} is +9999
padding	2 bits
coefficient	110 bits (encoding up to 33 decimal digits)

[8] See `http://www2.hursley.ibm.com/decimal/decifaq.html#tzeros`

Ordering of fields and bits in the encodings

The fields of encodings are laid out in the order they are described in the section *Fields in the encodings* (see page 3), with the first (*sign*) field being considered the most significant and the coefficient being the least significant field.

Within each field, the most significant bit (for the *exponent*) or the most significant code group (for the *coefficient*) is placed first. The bits of any *padding* field are always 0 and are unordered.

The *network byte order* (the order in which the bytes of an encoding are transmitted in a network protocol such as TCP/IP) of an encoding is such that the byte which includes the *sign* is transmitted first.

Example:

In single precision, the fields are laid out as follows:

Bits	1	11	2	50
Content	Sign	Exponent	Pad	Coefficient

Hence, the number –7.50E+11 would be encoded (in hexadecimal, shown in network byte order) as:

```
C0 80 00 00 00 00 03 D0
```

The first bit is 1, indicating that the number is negative. The exponent will be 9 plus the bias (1023) giving 1032 (hexadecimal 408). The coefficient will be 750, which is encoded as the ten bits 1111010000 and right-aligned in the coefficient field.

Other recommended encodings

The following parameters are recommended for decimal numbers of other lengths. The parameters for the single and double precision encodings described above are also included in this table, for comparison.

Length	**Sign**	**Exponent**		**Padding**	**Coefficient**	
		Length	E_{max}		Length	Digits
32	1 bit	7 bits	+49	0	24 bits	7
64	1 bit	11 bits	+999	2 bits	50 bits	15
80	1 bit	15 bits	+9999	0	64 bits	19
128	1 bit	15 bits	+9999	2 bits	110 bits	33
192	1 bit	17 bits	+49999	0	174 bits	52
256	1 bit	18 bits	+99999	0	237 bits	71

Appendix A – Rationale

This section, which is not part of the encoding specification, describes the design rationale that led to the encoding specified in the body of this document (see page 3).

The primary design goals are that the representation should be suitable for:

(Goal1) efficient hardware or software implementation of decimal floating-point (that is, where the exponent indicates a power of ten)

(Goal2) representing numbers to be used for the floating-point and integer decimal arithmetic defined in ANSI X3.274-1996,[9] including the subset defined in ECMA-334[10] and ECMA-335[11]

(Goal3) representing numbers and values to be used for the floating-point arithmetics defined in ANSI/IEEE 854-1987.[12]

(Goal4) efficient use of existing integer or fixed-point decimal data (in binary coded decimal)

(Goal5) future enhancements.

These goals can be satisfied by a representation which comprises two integers (both of which may be positive, zero, or negative). One is a *coefficient* of a power of ten, and the other is an an appropriately bounded *exponent* (the power of ten). In addition, it is necessary to allow for a negative zero, and at least four *special values* from IEEE 854 (including ±infinity, Quiet NaN, and Signaling NaN), and some means to allow for future expansion.

Further, existing hardware architectures suggest that the representations be 32, 64, 80, or 128 bits in length. This rationale concentrates on 64-bit and 128-bit representations, forming the *single* and *double* precisions described by IEEE 854. (IEEE 854 requires that there be a single precision and recommends that there should be an extended precision; double precision satisfies the requirements for single-extended precision.)

The remainder of this rationale takes this form of representation, and the two precisions, as given. This still allows considerable flexibility in the representation. To facilitate

---

9 *American National Standard for Information Technology – Programming Language REXX, X3.274-1996*, American National Standards Institute, New York, 1996.

10 Standard ECMA 334 – *C# Language Specification*, ECMA, Geneva, December 2001.

11 Standard ECMA 335 – *Common Language Infrastructure (CLI)*, ECMA, Geneva, December 2001.

12 IEEE 854-1987 – *IEEE Standard for Radix-Independent Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, Inc., New York, 1987.

discussion, each design choice below is identified by a letter, and each concrete proposal is numbered. The proposals contain only factual material; subjective comments are contained in a discussion section. Notes contain other, non-controversial, background information.

The specific choices below are:

A. Representation of the *coefficient* (see page 9)

B. Representation of the *exponent* (see page 11)

C. Representation of the *special values* (see page 13)

D. Ordering of the fields (see page 15)

E. Length of the *exponent* (see page 16).

Representation of the coefficient

A.1	Binary (the coefficient is a plain binary number)	This allows the greatest number of decimal digits in a given amount of storage, the fastest multiplication or division, and the fastest addition when two numbers have the same exponent. Extra processing is required for shifting numbers during addition or subtraction when the exponents differ, for conversions to and from character form, for conversions to and from BCD form (Goal4), and for rounding to a given number of decimal digits (for Goal2).
A.2	Binary Coded Decimal (BCD: 4 bits per digit, weighted 8.4.2.1)	BCD allows efficient loading and storage of existing data decimal formats (notably Packed Decimal data), efficient rounding to a specified number of decimal digits (required for Goal2 and Goal3), and simple conversions to and from character form. A secondary advantage is that the value is easily readable when the representation is displayed as a hexadecimal "dump". However, BCD arithmetic is more complex than a pure binary arithmetic, and fewer digits can be encoded in a given number of bits.
A.3	Chen-Ho encoding (10 bits per 3 digits)	Chen-Ho encoding compresses three decimal digits into 10 bits with a 0.34% wastage, giving a 20% more efficient encoding than simple BCD. BCD encoding can be derived from Chen-Ho encoding using boolean combination, without multiplications or divisions. Extra processing is required to convert between BCD and Chen-Ho, or to work directly on Chen-Ho encoded numbers.
A.4	Densely Packed Decimal encoding (10 bits per 3 digits)	Densely Packed Decimal encoding compresses three decimal digits into 10 bits using a modified Chen-Ho encoding. Unlike Chen-Ho encoding, it is not limited to multiples of three digits, numbers can be padded without re-encoding, and BCD encoding is preserved for the numbers 0–79.

Discussion

The primary advantage of A.1 is that it would allow 33 decimal digits to be encoded in a 128-bit floating-point representation (including an exponent), along with very fast operations when no rounding or alignment is necessary.

A.2 seems an attractive choice, due to the simplicity of using existing decimal data (which are often encoded in BCD format). However, even with no exponent only 31 digits (with a sign) can be encoded in 128 bits. This is insufficient, given that numbers can often approach this length and (for example) the forthcoming COBOL standard[13] requires that intermediate results be computed to 32 digits. A.2 is therefore not a viable candidate.

[13] ISO/IEC standard 1989:2002 – *Information technology – Programming languages, their environments and system software interfaces – Programming language COBOL*, ISO/IEC, publication date *tbd*

A.3 and A.4, like A.1, would allow up to 33 digits in a 128-bit representation (with exponent), with better performance for rounding and conversions than A.1. However, a hardware implementation would require either additional encoding and decoding (leading to a wider-than-128 bits internal BCD representation) or additional (coded 10-bit base) processing units.

On balance, the extra processing costs associated with A.1 (especially the costs and complexity of conversion from BCD and character formats and of rounding to a decimal precision after operations) make this choice inferior to A.3 and A.4, despite the increased complexity of these.

Of the latter, **A.4** is the better choice, because of its ability to encode arbitrary-length decimal data (and hence make best use of the space available in all formats) and because it allows conversions between lengths to be carried out simply by adding or stripping leading zero bits.

Notes

1. Chen-Ho encoding is described in *Storage-Efficient Representation of Decimal Data*, Tien Chi Chen & Irving T. Ho, CACM (18)1, pp.49-52, January 1975. A summary is available.[14]
2. Densely Packed Decimal encoding is described in a paper by the author, recently accepted for publication by the IEE in the UK. A summary is available.[15]
3. Other 3 in 10 bit encodings are possible. For example, simple base 1000 binary encoding would allow byte-by-byte comparison of numbers provided their exponents and sign were the same. Base 1000, however, is more costly to convert to or from BCD.
4. With BCD encoding, the sign would held separately, either as a “packed decimal” sign digit, or as a single sign bit. Either scheme allows for –0 as a value.
5. With Binary encoding, the coefficient could either be twos-complement, biased, or unsigned. In the first two cases a special value would be required to encode –0; in the third case a separate sign bit would be used.
6. Other representations investigated but not under consideration include Bi-quinary, Gray, Excess-3, 2-of-5, 1-of-10, 6.3.1.1, 2.4.2.1, negadecimal (base –10), 3-digit/10-bits (base 1000), and the Base/1 Number Class encoding.[16]

[14] See `http://www2.hursley.ibm.com/decimal/chen-ho.html` for the summary.

[15] See `http://www2.hursley.ibm.com/decimal/DPDecimal.html` for the summary.

[16] See `http://www.boic.com` for a description of the Base/1 Number Class.

Representation of the exponent

B.1	Binary twos-complement	This allows the greatest range of exponent in a given amount of storage, and fast processing. If all the bits in an encoding were set to zero then the exponent would be zero. Extra processing is needed for conversions to and from character form.
B.2	Binary with bias (all exponents appear positive, and unsigned)	This allows the same range of exponent as twos-complement, with faster processing. In hardware, comparisons are simpler when testing for a restricted range of exponents or when comparing exponents. Binary floating-point implementations usually use a biased exponent. Extra processing is needed for conversions to and from character form.
B.3	Binary Coded Decimal (BCD)	BCD allows simple conversions to and from character form, and implicitly restricts the exponent to an exact number of digits (that is, no explicit hardware or software check would be needed). A secondary advantage is that the value is easily readable when the representation is displayed as a hexadecimal "dump". A smaller range of exponents would be available for a given space, with a significant reduction for single precision using the likely layout.

Discussion

If a binary representation is chosen, then B.2 has the advantage over B.1 and would be the preference.

B.3 seems to be an attractive choice in that it appears to force the "natural" decimal limit to the exponent normally required by software, and would have a certain subjective elegance if both numbers in the representation were encoded in the same manner. However, there are two disadvantages:

1. A significantly reduced exponent range would be available, especially for single precision or smaller encodings.

   For example, if choice A.2 were made (BCD integer) then the space available for sign, flags, and exponent would be a multiple of 4 bits, with 12 bits (the same number as IEEE 754) being an obvious choice. With 12 bits available, choice B.3 would force a 2 digit exponent (–99 through +99) whereas with binary encoding a 3 digit exponent would be possible (–999 through +999).

2. As discussed below (see page 16), the exponent in the representation will have a smaller upper bound than lower bound, and hence the range of values of the exponent is larger than twice the power of ten which defines the limit. A BCD representation therefore would require one more digit than might be expected, which exacerbates the first disadvantage. An additional test to determine whether a limit had been exceeded would still be required.

For these reasons, a binary representation (**B.2**) appears to be the best choice here.

Notes

1. There is a choice of bias. For single precision, for example, it could be either 1023 (matching IEEE 754) or 999+*d*–1, where *d* is the maximum number of digits in the coefficient (so the minimum exponent appears as 0). IEEE 754 values have precedent (they would already be available in hardware) and are currently preferred.

2. It is assumed that the exponent limit will be strictly checked on a boundary related to decimal digits, because user definitions of decimal arithmetic naturally express exponent limits and layouts in terms of the number of decimal digits supported in a scientific notation. If implementations did not enforce an appropriate limit then significant extra checking would need to be carried out in software (possibly after every operation).

3. Chen-Ho or other 3-in-10 encodings are not proposed because the lengths of exponents are small and the extra complexity is not justified; these encodings also suffer the same second disadvantage as BCD encodings.

4. With BCD encoding, the sign would held separately, probably as a single sign bit. In this case, an exponent of –0 should probably be in error.

Representation of the special values

C.1	Reserved values of exponent, following IEEE 754	This scheme follows the precedent of existing binary floating-point representations. It does not require extra dedicated bits in the number. If the exponent used a binary representation and was limited to a decimal range there would be ample special values available.
C.2	Other reserved values of exponent	This scheme also uses special values of exponent, but uses values different to those of IEEE 754. If the exponent used a binary representation and was limited to a decimal range the special values can all be encoded without requiring the coefficient to be inspected.
C.3	Separate bits	This scheme is the simplest to decode. It would remove either one or two bits from those available for the integer and exponent (depending on whether sign bits were available for indicating special values).

Discussion

This choice is needed for Goal3, and is partly determined by the answer to B:

- If a binary representation for the exponent is chosen, then the maximum exponent range can be achieved by either choice C.1 or C.2 (that is, no extra bits are used for special values). It is likely that the exponent range will be related to a decimal digit boundary, which would leave many encodings available for the special values (and for future expansion, Goal5).
- If a BCD representation for the exponent is chosen, then it is likely that there will be a multiple of four bits left over after the exponent and integer are laid out. These give sufficent space for the special values (and signs), as needed for C.3.

With B.2 (binary with bias) chosen for the exponent format, C.1 or C.2 are preferred over C.3.

At first sight, C.1 would seem to have the advantage over C.2, as it offers the possibility of preserving detection hardware for the special values. However, there are several arguments in favor of C.2:

1. The coefficient does not have an “implied 1-bit”, so zero is a possible and legal value. Therefore, there is no need to have a special value of exponent to represent zero (indeed, it would be a disadvantage to have two unrelated mechanisms for representing zero).
2. An “all-bits-zero” value is quite likely to occur in practice, yet it would not be a valid bit pattern. It therefore makes sense for this encoding to represent Signaling NaN, in effect making it a signal indicating an *uninitialized number*.
3. If there is a special value at 0, and there are unused encodings adjacent to that value, then it is useful to have the other two special values (NaN and Infinity) at that “end” of the range, too. This makes it possible to check for the special cases in software with a single test.

4. If all three special values are explicitly encoded by values of exponent then there is no need to inspect the coefficient to distinguish between them.

5. Having the special values all at the low end of the exponent range simplifies both documentation and software, as the special values then have the same numerical value (0, 1, or 2) regardless of the size of the exponent.

For these reasons, a scheme meeting choice **C.2** is preferred. The actual values suggested are 0 for Signaling NaN (as already discussed) with Quiet NaN as the adjacent value (1) and Infinity having the value 2.

Notes

1. With choice C.2 and encodings as described, the coefficient would have no effect on the value when the exponent indicated a special value. This would allow implementations to convey diagnostic information in the integer part if desired, as recommended by IEEE 854.

2. If the representation of the exponent were twos-complement (B.1 rather than B.2) then C.2 would still be preferred, but an “all-bits-zero” value would represent zero. In this case, the three special values could be at either the most-positive or the most-negative end of the range of exponent encodings.

3. If the choice were C.3 then one possible design would be as follows. Three bits from the four available could be assigned: Integer sign, Exponent Sign, and Special. With this scheme, if “Special” were 1 then the second bit could select Infinity/NaN, and the first could then indicate sign (for Infinity) or signaling (for NaN).

   The fourth bit could then be reserved for future expansion (Goal5). One use for this (in conjunction with other bits) might be to indicate explicitly whether a number was in single or double format (if the special bit were 0).

Ordering of the fields

D.1	Exponent first: – sign/flags – exponent – coefficient	This scheme follows the precedent of many existing binary floating-point representations, including IEEE 754 and IBM S/390.
D.2	Coefficient first: – coefficient – exponent – sign/flags	This scheme follows the precedent of the character representation of a floating-point number (*e.g.*, 12.3E+3), and some software representations (*e.g.*, Borland real48).

Discussion

For software, there is no strong preference for either choice.

For hardware, however, existing floating-point dataflows will already have many wires from Floating-Point Registers that depend on the structure defined for IEEE 754 (with the sign and exponent first). Having a different ordering and hence a conflicting wiring pattern would strongly discourage hardware designers from implementing decimal floating-point arithmetic.

The correct choice here would therefore seem to be **D.1**.

Notes

1. It is assumed that if there is a sign bit it would precede any other flags, independent of the choice of order.
2. If a decimal encoding is used for the exponent or coefficient then it is possible that some bits may be unused. These “padding” bits would be placed after the exponent so that the coefficient is right-aligned in the remainder of the encoding (this simplifies conversions between different encodings).
3. Another possible position for the padding bits might be before or after the sign, which would allow the possibility of self-defining lengths for encodings. However, this loses the correspondence with IEEE 754 formats, and restricts the range of representable numbers in the cases where no padding would have been needed.

Length of the exponent

Only one proposal was considered.

E.1	Exponent lengths should match those of IEEE 754 and IEEE 754R quad	This scheme simplifies hardware design of a combined binary/decimal floating-point unit. It meets the constraints of IEEE 854. With a binary exponent representation (as in B.1 and B.2) a greater exponent range than IEEE 754 is achieved as exponents are powers of 10.

Discussion

IEEE 854 puts several constraints and recommendations on the exponent of any representation (see the Notes below), and in particular requires that the exponent be longer for a double precision number than for a single precision number. These constraints are all met by using the IEEE 754 allocation of bits for sign and exponent (or the quad format proposed for the IEEE 754R revision).

For a single precision number, reducing the exponent to two decimal digits would require 8 bits for the exponent, so (with the added sign bit) no additional integer digits would be available if a BCD encoding were used for the integer.

For a double precision number, the exponent could be reduced to three digits to gain a digit in the integer, but this would leave one bit unused and would also force the single precision exponent down to two digits unnecessarily.

For these reasons, **E.1** is the only proposal.

Notes

1. It is assumed that given the result of this choice and the previous choices the length of the coefficient is determined. That is, it will be the remainder of the number after any exponent and necessary sign and flag bits are allocated.

2. IEEE 854 requires that the exponent range ($E_{max} - E_{min}$) be greater than 5 times the maximum precision in digits, and recommends that it be greater than 10 times the precision.

   This gives the minimum values for E_{max} shown in the second and third rows of the following table. Plausible single precisions are shown to the left of the table, and plausible double precisions are on the right.

Precision (digits)	**10**	**11**	**12**	**13**	**14**		**24**	**25**	**26**	**27**	**28**	**29**
required E_{max}	26	28	31	33	36		61	63	66	68	71	73
preferable E_{max}	51	56	61	66	71		121	126	131	136	141	146
double E_{max}	415	455	495	535	575							

   The bottom row in the table above shows, for each of the plausible single precisions, the recommended minimum E_{max} for double precision. This must be greater than or equal to 8 times the E_{max} for single precision, plus 7.

It is apparent from the table that if the latter constraint is satisfied then the preferred E_{max} for double precision will also be satisfied.

3. IEEE 854 recommends that, for base 10 representations, the minimum adjusted exponent E_{min} should have the same absolute value as the maximum adjusted exponent E_{max}.

   That is, $E_{min} = -E_{max}$ when the exponent is adjusted to represent the value of the number in scientific notation. Balancing the range in this way minimizes overflows and underflows when the inverse of a number is calculated.

   Since the representation comprises an integer and exponent (instead of a fraction and an exponent), the maximum exponent in the representation must be reduced so that the effective exponent range is balanced. For example, if the coefficient were 13 digits and the exponent 3 digits (–999 through +999) then the range of positive numbers would be from 1E–999 through 9.999999999999E+1011, which is unbalanced.

   Instead, the exponent limits in the representation must be adjusted down by d–1 (where d is the number of digits in the coefficient).

   For example, if the coefficient were 13 digits long and all nines in this case, the allowed range of the exponent would be –1011 through +987, leading to a balanced range of numbers with a guaranteed maximum exponent length when converted to character form. That is, positive numbers with this exponent would range from 9.999999999999E–999 through 9.999999999999E+999.

4. Common floating-point formats use the following layouts:

Binary bits			**Decimal** (approx.)	
Total	**Fraction**	**Exponent**	**precision**	E_{max}
32	23	8	7	10^{+38}
64	52	11	16	10^{+308}
80	64	15	19	10^{+4931}
128	112	15	34	10^{+4931}

The 32-bit and 64-bit layouts are the IEEE 754 single and double formats. The 128-bit layout is the quad format being proposed for the current revision of IEEE 754 (IEEE 754R).

Appendix B – Changes

This appendix documents changes since the first public draft of the proposals which led to this specification (July 2000). It is not part of the specification.

Changes in Draft 0.60 (23 May 2001)

This specification is a formalization of the "strawman" proposal for decimal concrete representations which was previously published as `decconc.html`.

No technical changes have been made to the proposal detailed in the most recent version of that document (version 0.57, 22 April 2001); however, there have been numerous editorial and formatting changes.

Changes in Draft 0.71 (19 March 2002)

- The rationale has been updated, especially for choice C (the representation of special values).
- Minor editorial changes and clarifications have been made.

Changes in Draft 0.72 (5 July 2002)

Minor editorial changes have been made to reflect the change from a two-layered arithmetic specification to a single document.

Index

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

