Decimal Arithmetic Encoding Strawman 4d

21st February 2003

Mike Cowlishaw

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

Draft – Version 0.96

Table of Contents

Introduction

This document describes proposed encodings suitable for supporting the general purpose floating-point decimal arithmetic defined in the **Decimal Arithmetic Specification**,[1] which allows fixed-point and integer decimal arithmetic as subsets.

The encodings are the product of discussions by a subcommittee of the IEEE committee (known as 754R) which is currently revising the IEEE 754-1985[2] and IEE 854-1987[3] standards.

These encodings are now included in an unapproved draft of the proposed IEEE-SA 754 standard. However, they are still subject to change; use at your own risk.

The primary audiences for this document are implementers and standards-makers, so examples and explanatory material are included. This informative 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*.[4] For further background details, please see the material at the associated web site: `http://www2.hursley.ibm.com/decimal`

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 David Bindel, Glenn Colon-Bonet, James Demmel, William Kahan, Dave Raggett, Andy Rawson, Jason Riedy, Fred Ris, Eric Schwarz, Ronald Smith, Charles Webb, and Dan Zuras, who have all directly contributed to this document.

Also, of course, thanks are due to all the contributors to standards work in the area – especially the members of the Radix-Independent Floating-Point Arithmetic Working Group of the Microprocessor Standards Subcommittee of the IEEE, the members of the X3 Secretariat/CBEMA (now NCITS) Subcommittee J18, and the members of the IEEE 754r committee.

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

[4] *A Decimal Floating-Point Specification*, Schwarz, Cowlishaw, Smith, and Webb, in the *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (Arith15)*, IEEE, June 2001.

Overview

(This overview is not part of the specification.)

This document describes the proposed encodings for decimal numbers. These encodings allow for a range of positive and negative values (including both normal and subnormal numbers), together with values of ±0, ±Infinity, and Not-a-Number (NaN).

Three formats of decimal numbers are described:

- A *decimal32* number, which is encoded in four consecutive bytes (32 bits)
- A *decimal64* number, which is encoded in eight consecutive bytes (64 bits)
- A *decimal128* number, which is encoded in 16 consecutive bytes (128 bits).

The finite numbers are defined by a *sign*, an *exponent* (which is a power of ten), and a decimal integer *coefficient*. The value of a finite number is given by $(-1)^{sign} \times coefficient \times 10^{exponent}$. For example, if the *sign* had the value 1, the *exponent* had the value –1, and the *coefficient* had the value 25, then the value of the number is –2.5.

This *dual integer* description of numbers permits redundant encodings of some values. 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 this number is also –2.5.

The advantage of this representation is that it exactly matches the definition of decimal numbers used in almost all databases, programming languages, and applications. This in turn allows a decimal arithmetic unit to support not only the floating-point arithmetic used in languages such as Java and C# but also the strongly-typed fixed-point and integer arithmetic required in databases and other languages.

The cost of the redundant encodings is approximately a 17% reduction in available exponent range – however, because the base is 10, the exponent range is always greater than that of the binary format of the same size.

In the encoding of these dual-integer numbers, the *sign* is a single bit, as in IEEE 754. The *exponent* is encoded as an unsigned binary integer from which a *bias* is subtracted to allow both negative and positive exponents. The *coefficient* is an unsigned decimal integer, with each complete group of three digits encoded in 10 bits (this increases the precision available by about 15%, compared to simple binary coded decimal).

Given the length of the coefficient and possible values for the encoded exponent, the maximum positive exponent (E_{max}) and bias can be derived, as described in Appendix A (see page 10).

Calculating the values leads to the following results for the three formats:

Format	**decimal32**	**decimal64**	**decimal128**
Coefficient length in digits	7	16	34
Maximum Exponent (E_{max})	96	384	6144
Minimum Exponent (E_{min})	–95	–383	–6143
Bias	101	398	6176

For the decimal32 format, the largest normal number is $9.999999 \times 10^{E_{max}}$ The coefficient and exponent for some sample positive numbers in this format are:

Number	**Dual-integer representation**		**Encoded exponent**
	coefficient	*exponent*	
$9.999999 \times 10^{E_{max}}$	9999999	+90	191
$1.234567 \times 10^{E_{max}}$	1234567	+90	191
$1.23 \times 10^{E_{max}}$	1230000	+90	191
$1 \times 10^{E_{max}}$	1000000	+90	191
12345	12345	0	101
1	1	0	101
1.23	123	–2	99
123.45	12345	–2	99
$1 \times 10^{E_{min}}$	1	–95	6
$1.000000 \times 10^{E_{min}}$	1000000	–101	0
$1.000001 \times 10^{E_{min}}$	1000001	–101	0
$0.000001 \times 10^{E_{min}}$	1	–101	0

Of these positive numbers, the number $9.999999 \times 10^{E_{max}}$ (which has no redundant encoding) is the largest normal number, the number $1 \times 10^{E_{min}}$ (and its redundant encodings) is the smallest normal number, and $0.000001 \times 10^{E_{min}}$ (which has no redundant encoding) is the smallest subnormal number.

Note that numbers with the same “scale” (such as 1.23 and 123.45) have the same encoded exponent.

As shown in the first table, each format has a coefficient whose length is a multiple of three, plus one. One digit of the coefficient (the most significant) cannot be included in a 10-bit group and instead is combined with the two most significant digits of the exponent into a 5-bit *combination field*. This scheme is more efficient than keeping the exponent and coefficient separated, and increases the exponent range available by about 50%.

The combination field requires 30 states out of a possible 32 for the finite numbers; the other two states are used to identify the special values. This localization of the special values means that only the first few bits of a number have to be inspected in order to determine whether it is finite or is a special value. Further, bulk initialization of storage to values of ±0, NaN, or ±Infinity can be achieved by simple byte replication.

Scope

Objectives

This document describes proposed encodings (concrete representations) which are suitable for supporting the general purpose IEEE-854-conforming decimal arithmetic defined in the **Decimal Arithmetic Specification**.[5]

Inclusions

This specification defines the following:

- The formats and layouts of 32-bit, 64-bit, and 128-bit decimal numbers (*decimal32*, *decimal64*, and *decimal128* respectively)
- The range of numerical values which can be represented by the formats
- The range of exponents which can be represented by the formats.

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 specified sizes.

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

Specification

This section defines the proposed encodings for decimal numbers. These encodings allow for a range of positive and negative values (including both normal and subnormal numbers), together with values of ±0, ±Infinity, and Not-a-Number (NaN).

Fields in the encodings

Each encoding comprises four fields, as follows:

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

   In numbers for which the *sign* has meaning (for finite numbers and Infinity) a 1 indicates the number is negative (or is negative zero) and a 0 indicates it is positive or is non-negative zero.

2. *combination field* – a 5-bit field which which encodes the two most significant bits (MSBs) of the exponent (which may take only the values 0 through 2) and the most significant digit (MSD) of the coefficient (4 bits, which may take only the values 0 through 9).

   When any of the first four bits of the field is 0, the whole encoding describes a *finite number*. When all of the first four bits of the field are 1, the whole encoding describes a *special value* (an Infinity or NaN).

   The following table defines the encoding of the combination field. The leftmost of the bits in the combination field is placed first.

Combination field (5 bits)	**Type**	**Exponent MSBs** (2 bits)	**Coefficient MSD** (4 bits)
`a b c d e`	Finite	`a b`	`0 c d e`
`1 1 c d e`	Finite	`c d`	`1 0 0 e`
`1 1 1 1 0`	Infinity	`- -`	`- - - -`
`1 1 1 1 1`	NaN	`- -`	`- - - -`

   Note that either one or both of the exponent MSBs will always be 0, so in the first line of the table, either `a` or `b` (or both) will be 0, and in the second line of the table, either `c` or `d` (or both) will be 0.

3. *exponent continuation* – the remaining, less significant, bits of the exponent. The most significant of these bits is on the left (is placed first).

   The *encoded exponent* is formed by appending these continuation bits as a suffix to the two exponent bits derived from the combination field. The whole encoded exponent forms a unsigned binary integer whose largest unsigned value, E_{limit}, is given by $3 \times 2^{ecbits} - 1$, where *ecbits* is the number of bits in the exponent continuation. *ecbits* varies with the format, as detailed below.

   The value of the *exponent* is calculated by subtracting a *bias* from the value of the encoded exponent, in order to allow both negative and positive exponents. The value of the bias varies with the format, and is also detailed below. In each format, all values of encoded exponent (0 through E_{limit}) can be used.

   When the number is a NaN or an Infinity, the first two bits of the exponent continuation field are used as follows:

Combination field	**Exponent continuation field most significant bits**	**Value**
11110	--	Infinity
11111	0–	quiet NaN
11111	1–	signaling NaN

   where "–" means undefined (an implementation may use these undefined bits for its own purposes, such as to indicate the origin of a NaN value); however, a future standard might require that the results of arithmetic set these bits to 1 for a NaN or 0 for an Infinity.

   These assignments allow the bulk initialization of consecutive numbers in storage through byte replication (for initial values of NaNs, ±Infinity, or ±0).

4. *coefficient continuation* – the remaining, less significant, digits of the coefficient. The coefficient continuation is a multiple of 10 bits (the multiple depending on the format), and the most significant group is on the left (is placed first).

   Each 10-bit group represents three decimal digits, using Densely Packed Decimal encoding.[6] Note that certain 10-bit groups encode the same value (all 6 possibilities where all three digits in the value are either 8 or 9 have four possible encodings). For these numbers, all four encodings are accepted as operands, but only the encoding with the first two bits being 0 will be generated on output.

   The *coefficient* is formed by appending the decoded continuation digits as a suffix to the digit derived from the combination field. The value of the *coefficient* is an unsigned integer which is the sum of the values of its digits, each multiplied by the appropriate power of ten. That is, if there are *n* digits in the coefficient which are labeled $d_n\ d_{n-1} \ldots d_1\ d_0$, where d_n is the most significant, the value is $\mathrm{SUM}(d_i \times 10^i)$, where *i* takes the values 0 through *n*.

   The maximum value of the coefficient, C_{max}, is therefore $10^n - 1$.

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

The *coefficient continuation* field is undefined when the *combination field* indicates that the number is an Infinity or NaN. 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); however, a future standard might require that the results of arithmetic set these bits to 1 for a NaN or 0 for an Infinity.

The fields of encodings are laid out in the order they are described above. Within each field, the bits are laid out as described for each field (that is, the combination field has its bits in the order `abcde`, the exponent continuation field has its most significant bit first, and the coefficient continuation field has its most significant 10-bit group first).

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.

Lengths of the fields

This specification defines three formats for decimal numbers:

- a four-byte *decimal32* format (32 bits)
- an eight-byte *decimal64* format (64 bits)
- a sixteen-byte *decimal128* format (128 bits).

Of these, the decimal32 format is required if the decimal64 format is provided, and the decimal64 format is required if the decimal128 format is provided.

In all three formats, the sign is always one bit and the combination field is always 5 bits. The lengths of the other two fields vary with the format, and from these lengths the maximum exponent (E_{max}) and bias can be derived, as described in Appendix A (see page 10). The following table defines the field lengths (in bits, unless specified) and details the corresponding derived values.

Format	**decimal32**	**decimal64**	**decimal128**
Format length	32	64	128
Exponent continuation length (*ecbits*)	6	8	12
Coefficient continuation length	20	50	110
Total Exponent length	8	10	14
Total Coefficient length in digits	7	16	34
E_{limit}	191	767	12287
E_{max}	96	384	6144
E_{min}	–95	–383	–6143
bias	101	398	6176

The value of an encoded number

The value of an encoding is either a special value (a NaN or an Infinity) or it is a finite number whose numerical value is given exactly by: $(-1)^{sign} \times \mathit{coefficient} \times 10^{exponent}$.

For example, if the *sign* had the value 1, the *exponent* had the value –1, and the *coefficient* had the value 25, then the numerical value of the number is exactly –2.5.

Notes

1. More than one encoding may have the same numerical value; if the *sign* again had the value 1, but the *exponent* had the value –2 and the *coefficient* had the value 250, then the numerical value of the number would also be exactly –2.5.
2. The largest value of the *exponent* in a format is less than E_{max} because the coefficient is an integer. For a format with precision *p* digits and maximum exponent E_{max}, IEEE 854 requires that the maximum absolute value of a number be exactly $(10^p-1) \times 10^{-(p-1)} \times 10^{E_{max}}$. (For example, if *p*=7 and E_{max}=96 then the largest value allowed is 9.999999E+96.)

   The maximum value of the *exponent* for a given format is therefore $E_{max}-(p-1)$. For example, if *p*=7 and E_{max}=96 then the number whose coefficient=9999999 and exponent=+90 has the value 9.999999E+96, which is the maximum normal number.
3. The method for deriving the values of E_{max} and bias ensures that all combinations of exponent (0 through E_{limit}) and coefficient (0 through 10^p-1) are allowed. For example, the *exponent* encoded as zero is always allowed.

Examples

In the decimal64 format, the length and content of the fields are:

Length (bits)	1	5	8	50
Content	Sign	Combination field	Exponent continuation	Coefficient continuation

In this format, the finite number –**7.50** would be encoded as follows:

- The *sign* is 1 indicating that the number is negative.
- The *coefficient* will be 750, with 13 leading zeros. This is encoded with the first digit (0) in the combination field, and the remaining 15 digits in the coefficient continuation field (four 10-bit groups of all zero bits and the final group being the encoding of 750, which is the ten bits 11 1101 0000).
- The *exponent* will be –2, so the encoded exponent is this plus the bias, or 396. This is 01 1000 1100 in binary, with the first two bits being embedded in the combination field and the remainder being placed in the exponent continuation field.

The bits of the combination field are therefore 01000 (the last three bits are 0 because the most significant digit of the coefficient is 0). The full encoding is therefore (in hexadecimal, shown in network byte order):

```
A2 30 00 00 00 00 03 D0
```