Decimal Arithmetic Encodings

20th March 2009

Mike Cowlishaw

IBM Fellow
IBM UK Laboratories

mfc@uk.ibm.com

Version 1.01

Copyright © IBM Corporation 2000, 2009. All rights reserved.

Table of Contents

Introduction

This document describes decimal 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 revised the IEEE 754-1985[2] and IEE 854-1987[3] standards. These encodings are now included in the new IEEE-SA 754 standard approved in June 2008.

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 rationale and 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://speleotrove.com/decimal`

Comments on this document 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://speleotrove.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 decimal 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 possible exponent values – however, because the base is 10, the exponent range is always greater than that of the IEEE 754 binary format of the same size.

In the encoding of these dual-integer numbers, the *sign* is a single bit, as for IEEE 754 binary numbers. 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 15).

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 decimal encodings (concrete representations) which are suitable for supporting the general purpose 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 with 96 bits or wider than 128 bits; however, any multiple of 32 bits can be used for an IEEE 754 interchange format that follows the same encodings and rules described here (each 32 bits after the first adds 2 bits to the exponent and nine decimal digits to the coefficient).

5 See `http://speleotrove.com/decimal/decarith.html`

Specification

This section defines decimal 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:

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.

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.

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**
`1 1 1 1 0`	`- -`	Infinity
`1 1 1 1 1`	`0 -`	quiet NaN
`1 1 1 1 1`	`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).

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 8 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 $10^n - 1$.

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

6 See *Densely Packed Decimal Encoding*, Mike Cowlishaw, in *IEE Proceedings – Computers and Digital Techniques*, ISSN 1350-2387, Vol. 149, No. 3, pp102-104, IEE, May 2002.
Abstract: Chen-Ho encoding is a lossless compression of three Binary Coded Decimal digits into 10 bits using an algorithm which can be applied or reversed using only simple Boolean operations. An improvement to the encoding which has the same advantages but is not limited to multiples of three digits is described. The new encoding allows arbitrary-length decimal numbers to be coded efficiently while keeping decimal digit boundaries accessible. This in turn permits efficient decimal arithmetic and makes the best use of available resources such as storage or hardware registers.
A summary is available at: `http://speleotrove.com/decimal/DPDecimal.html`

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 15). 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: $(-1)^{sign} \times coefficient \times 10^{exponent}$.

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
```

Simlarly, the value **+Infinity** is encoded as:

```
78 xx xx xx xx xx xx xx
```

(Where the bytes `xx` are undefined and could be repetitions of the `78`.)

Note that only the first byte has to be inspected to determine whether the number is finite or is a special value. Also, if the number is a special value, its specific value is fully defined in that first byte.

Appendix A – Calculation of Emax and bias

This section, which is not part of the encoding specification, describes how E_{max} and E_{min} (the maximum and minimum exponents available in a format) and bias (the amount to be subtracted from the encoded exponent to form the exponent's value) are calculated.

Except for the calculation of E_{limit}, these calculations are general for any format where the coefficient and exponent are both integers.

1. Let p be the precision (total length of the coefficient) of a format, in digits.
2. Let E_{max} be the maximum positive exponent for a value in a format, as defined in IEEE 854. That is, the largest possible number is

   $(10^{p}-1) / (10^{(p-1)}) \times 10^{E_{max}}$

   For example, if p=7 this is $9.999999 \times 10^{E_{max}}$.
3. The *exponent* needed for the largest possible number is $E_{max}-(p-1)$ (because, for example, the largest coefficient when p=7 is 9999999, and this only needs to be multiplied by $10^{E_{max}} / 10^{(p-1)}$ to give the largest possible number).
4. $E_{min}=-E_{max}$ (as defined by IEEE 854 for base 10 numbers). That is, the smallest normal number is $10^{E_{min}}$. The *exponent* needed for this number is E_{min} (its coefficient will be 1).
5. The number of exponents, $E_{normals}$, used for the normal numbers is therefore $2 \times E_{max} - p + 2$. (The values $-E_{max}$ through -1, 0, and 1 through $E_{max}-(p-1)$.)
6. Let E_{tiny} be the exponent of the smallest possible (tiniest, non-zero) subnormal number when expressed as a power of ten. This is $E_{min} - (p-1)$.

   For example. if p=7 again, the smallest subnormal is $0.000001 \times 10^{E_{min}}$, which is $10^{E_{tiny}}$.

   The number of exponents needed for the subnormal numbers, $E_{subnormals}$, is therefore $E_{min} - E_{tiny}$, which is $p - 1$.
7. Let E_{range} be the number of exponents needed for both the normal and the subnormal numbers; that is, $E_{normals} + E_{subnormals}$. This is $(2 \times E_{max} + 1)$.
8. Place E_{tiny} so its encoded exponent (the exponent with bias added) is 0 (the encoded exponent cannot be less than 0, and we want an all-zeros number to be valid – hence an encoded exponent of 0 must be valid).
9. Let E_{limit} be the maximum encoded exponent value available. For the formats in the specification, this is $3 \times 2^{ecbits} - 1$, where *ecbits* is the length of the exponent continuation in bits (for example, E_{limit} is 191 for the 32-bit format).
10. Then, the number of exponent values available is $E_{limit} + 1$, which is 3×2^{ecbits}.

11. Now, to maximize E_{max}, $E_{range} = E_{limit} + 1$

    That is, $2 \times E_{max} + 1 = 3 \times 2^{ecbits}$.

12. Hence: $E_{max} = (3 \times 2^{ecbits} - 1)/2 = E_{limit}/2$

    Note that the divisions by 2 must be truncating integer division.

13. If E_{limit} is odd (always the case in these encodings), one value of exponent would be unused. To make full use of the values available, E_{min} remains as the value just calculated, negated, and E_{max} is increased by one.[7]

    Hence: $\mathbf{E_{min}} = -E_{limit}/2$

    and: $\mathbf{E_{max}} = E_{limit}/2 + 1$

    (where the divisions by 2 are truncating integer division).

14. And: **bias** $= -E_{tiny} = -E_{min} + p - 1$

For example, let $E_{limit} = 191$ and $p = 7$ (the 32-bit format). Then:

$E_{max} = 191/2 + 1 = 96$
$E_{min} = -95$
$E_{tiny} = -101$
bias = 101

The parameters and derived values for all three formats are as follows:

Format	***ecbits***	$\mathbf{E_{limit}}$	***p***	$\mathbf{E_{max}}$	$\mathbf{E_{min}}$	**bias**
32-bit	6	191	7	96	-95	101
64-bit	8	767	16	384	-383	398
128-bit	12	12287	34	6144	-6143	6176

Note that it is also possible to consider the coefficients in these formats to have a decimal point after the first digit (instead of after the last digit). With this view, the bit patterns for the layouts are identical, but the bias would be decreased by $p-1$, resulting in the same value for a given number.

7 As decided by the IEEE 754r committee at its January 2003 meeting.

Appendix B – Changes

This appendix documents changes since the first circulation of the “Strawman 3” proposal (7 November 2002). It is not part of the specification.

Changes in Draft 0.88 (19 Nov 2002)

- It is pointed out that, although infinities and NaNs should be recognized as such based on only the first byte of the number, arithmetic may well be required to fully define all bits of an infinite or NaN result (for example, by clearing the remaining bytes to 0).
- A note has been added to Appendix A, pointing out that the coefficient can be considered to have an internal decimal point if an appropriate change to the bias is made.
- The term U_{max} has been renamed E_{limit} for consistency with other documents.

Changes in Draft 0.93 (17 Dec 2002)

- The exponent range of all three formats has been expanded by making the largest normal number be the largest representable number. This also simplifies the calculations of E_{max} and bias. There are now no “supernormal” values, and one value of encoded exponent is always unused (and is therefore available for future expansion of the encoding scheme).
- The names *compact*, *single precision*, and *double precision* have been replaced by the names *decimal4*, *decimal8*, and *decimal16* respectively, to avoid confusion with the binary floating-point formats.

Changes in Draft 0.94 (12 Jan 2003)

- The distinction between signaling NaN and quiet NaN, using the top bit of the exponent continuation field, has been made part of the specification.
- For an Infinity, the top bits of the exponent continuation field have been defined to be 0 (instead of 1). This places Infinity logically adjacent to the finite numbers, and allows Infinities to have all bits zero after the combination field.

Changes in Draft 0.95 (3 Feb 2003)

During the IEEE 754r committee meeting in January 2003, the following changes were agreed.

- Signaling NaN has been removed.
- Redundant Densely Packed Decimal encodings are permitted in operands.

- The value of E_{max} is raised by one for all encodings (leaving E_{min} unchanged), hence using all exponent values available.

These changes mean that all possible bit patterns in the formats are valid.

Changes in Draft 0.96 (21 Feb 2003)

During the IEEE 754r committee meeting in February 2003, the following changes were agreed.

- Signaling NaN has been restored.
- The names *decimal4*, *decimal8*, and *decimal16* have been replaced by the names *decimal32*, *decimal64*, and *decimal128* respectively.

Changes in Version 1.00 (29 Jul 2008)

The decimal encodings are now included in the IEEE 754 standard approved in June 2008, so this document is no longer a proposal and has been updated to reflect that. There are no technical changes from the previous version (Strawman 4d).

Also, all references to the General Decimal Arithmetic website have been updated to `http://speleotrove.com/decimal` (its new location).

Changes in Version 1.01 (20 Mar 2009)

The document is now formatted using OpenOffice (generated from GML), for improved PDF files with bookmarks, hot links, *etc.* There are no technical changes.

Index