Brakefield Research = *Optimal Architecture* 10/17/25
13423 Blanco Rd. #144, San Antonio, Texas 78216

An Engineer's rework of IEEE-754 Floating-Point

by
James C. Brakefield

The work herein, "Alt-754", is presented within the framework of a numerical model. The model provides a general setting lending perspective and insight into floating-point design issues. The model came after the invention of Alt-754.

Most Alt-754 implementation issues can be examined in the context of this model. Issues of significant impact on hardware implementation were resolved with a unique combination of economy and generality.

Given adoption of Alt-754 several larger possibilities emerge: Universal calibration standards for instrumentation, e.g. given a signal of a specific kind, software scaling and offsetting are no longer needed. 16-bit floating-point can be used to save memory when accuracy is limited. Low precision floating-point is made more affordable by low cost controllers (microprocessors & DSPs).

What Follows:

13423 Blanco Rd. #144, San Antonio, Texas 78216

Outline:

Justifications:

Justification for floating-point datum (i.e. Engineered Floating-Point):
- Software robustness (via type checking & OOP)
- Algorithm simplicity
  - Fewer arithmetic operations
  - Quicker development
- Calibration uniformity
- Wide dynamic range (via optimizing floating-point format)

Justification for underflow rework
- Eliminate floating-point exception handling in low cost implementation
- Eliminate memory load delay uncertainty in pipeline implementations
- Lower circuit cost

Justification for rounding rework
- Eliminate delay uncertainty of IEEE-754 rounding
- Lower circuit cost

Examples of engineered “floating-point” datum
- Logarithmic “db”
- u-law & a-law codecs
- Variety of floating-point formats (past & present)

Brakefield Research = *Optimal Architecture* 10/17/25
13423 Blanco Rd. #144, San Antonio, Texas 78216

The Alt-754 Model:

Sign
Unbounded bit string
    Embedded decimal point
Von-Neumann or “jam” rounding
    Unbiased rounding
Encoding issues
    Exponent range
    Mantissa size
    Corner treatment
        Symmetric corners
        Corner taper
Special values
    Zero
    Infinity
    NaN (not a number)

Alt-754 Implementation:

Memory unit design
- Engineer memory formats

Load/store unit design
- Load unit
- Store unit

ALU design
- Choose register format
- Exception handling
- Status bits and state bits
- Add/subtract unit
- Multiply unit
- Divide unit

Effect of unbiased von-Neumann “jam” rounding
- Equivalent bias
  - Both IEEE-754 & Alt-754 use unbiased rounding
- Equivalent ARRE
  - By treating “jam” bit as hidden
- Equivalent error growth
  - By treating “jam” bit as hidden

Abstract Register Model

This model is most intuitive if a floating point "value" is considered as a range of numbers rather than a single number. Whereas IEEE 754 considers a floating point number as a specific rational number, the floating point format disclosed herein considers a floating point "value" (i.e. the specific contents of a memory or register interpreted as floating point) as representing a range of numbers.

It is represented in binary. For the abstract model, a value has only its sign and a binary encoding of its numeric content.

Consider a binary register of indefinite extent to the left and to the right. One particular bit of the register is marked. This is the location of the decimal point.

```
s, ---------------------------------.----------------
```

The most significant bit (MSB) of the number is to the left of the remaining bits. The displacement of this bit with respect to the decimal point is the logical exponent. I.e. any of the following (where the "x" denotes arbitrary bits):

```
s, ------------------------1xxxxxx--.---------------- exponent=8

s, ----------------------------1xxx.xx-------------- exponent=3

s, ---------------------------------.--1xxxxxx------- exponent=-3
```

The accuracy of the number is marked by a right most one bit (RMOB). The displacement of the RMOB with respect to the MSB gives the number of bits of accuracy. E.g.:

```
s, ----------------------------1xxx.xx1-------------- exp=3, accuracy=6
                                ^MSB   ^RMOB
```

The sign and the bits from the MSB to the RMOB constitute the floating point value. It represents a range of numbers. That range of numbers consists of all numbers with bits to the right of the RMOB where the RMOB is clear or set.

```
s, ----------------------------1ddd.dd0xxxxxxxxx... and

s, ----------------------------1ddd.dd1xxxxxxxxx...
```

13423 Blanco Rd. #144, San Antonio, Texas 78216

Memory Encoding of Special Values
(32-bit floats shown)

There is only one zero and it's code is all zero bits (IEEE-754 has both a positive and negative zero, standard Alt-754 has a total of only four codes in all available for zero, NaN, and the two infinities).

```
Zero:        0 00000000 00000000000000000000000
```

There is only one code for "Not-a-Number" (NaN) and it is the code that IEEE-754 uses for negative zero (IEEE-754 has a "huge" number of NaNs: $2^{mantissa\_length}$ less the two codes for infinity)

```
NaN:         1 00000000 00000000000000000000000
```

The codes for positive and negative infinity correspond to the largest possible magnitude (see overflow coding below).

```
+infinity:   0 11111111 00000000000000000000000

-infinity:   1 11111111 00000000000000000000000
```

The codes for underflow use the smallest possible exponent. The exponent adjustment is the number of trailing zeros in the mantissa. The adjustment is subtracted from the exponent.

$$underflow:\ s\ 00000000\ xxxxxxxxxxx100000000000 = (1-2s)*1.x*2^{(0-128-11)}$$

The codes for overflow use the largest possible exponent. The exponent adjustment is the number of trailing zeros in the mantissa. The adjustment is added to the exponent.

$$overflow:\ s\ 11111111\ yyyyyyyyyyyyyyyy1000000 = (1-2s)*1.y*2^{(255-128+6)}$$

Standard Alt-754 uses one exponent value for underflow codes and one exponent value for overflow codes. Using more than one exponent value for underflow or overflow results in increased exponent range via more rounded "corners". I.e., each additional exponent value used for the "corners" adds as many counts of exponent as there are bits in the mantissa. For the 32-bit format, this is a 23 to 1 exchange ratio. Below a total of four exponent codes are used for underflow and overflow each.

$$undfw:\ s\ 000000zz\ xxxxxxxxxxx100000000000 = (1-2s)*1.x*2^{(0-128-11*4+z)}$$
$$ovfw:\ s\ 111111zz\ yyyyyyyyyyyyyyyy1000000 = (1-2s)*1.y*2^{(252-128+6*4+z)}$$

If explicit accuracy is implemented, the accuracy is the number of bits not in the trailing zero bit count. (The hidden leading one bit is considered accurate, the trailing one bit is not considered accurate)

$$Explic\text{-}acc:\ s\ uuuuuuuu\ zzzzzzzzzzzzz100000000 = (1-2s)*1.z*2^{(u-128)}$$

Text:

1. Alt-754 Justification

IEEE-754 exists and is universally implemented in non-embedded micro-processors. However, for embedded (i.e. low cost) and hard real-time applications either IEEE-754 is not fully implemented or fixed-point is used. It is time to consider alternatives to IEEE-754.

1.1 Justification for floating-point datum

Floating-point is preferred to fixed-point in most cases. Keeping track of the decimal point is time consuming. The additional numeric range is also welcome.

1.1.1 Software robustness (via type checking & OOP)

Various industries and processes could adopt specific floating-point scales. The type checking and object oriented features of modern programming languages would ensure that calculations would be properly scaled and offset (i.e. avoid bugs due to incorrect units).

1.1.2 Algorithm simplicity

Software productivity scales with inversely with code size.

1.1.2.1 Fewer arithmetic operations

For voice compression, a floating-point implementation takes about half as many operations as the equivalent fixed-point version.

1.1.2.2 Quicker development

Accuracy is easier to control with floating-point

1.1.3 Calibration uniformity

With process specific datum, calibration is standardized. Data from different plants can be immediately compared. As more automation enters the consumer sector, the importance of consistent and uniform calibration will increase. The 16-bit float is ideal for most automation as there is about 12-bits of accuracy (better than three digits accuracy) and a dynamic range of 36 binary octaves (~11 decades) available using Alt-754.

1.1.4 Wide dynamic range (via optimizing floating-point format)

Using 16-bit datum and a four-bit exponent, by maximizing the taper, an binary exponent range of 174 is possible (equivalent to a decimal exponent range of +/- 26). With minimal taper, an exponent range of 36 is available.

1.2 Justification for underflow rework

IEEE-754 was not designed with circuit area or delay costs in mind. IEEE-754 was introduced in a time when floating-point was implemented via micro-code. IEEE-854, a derivative of IEEE-754, does not specify the encoding for underflow numbers. US Patent number 6,131,106 uses the Alt-754 approach to implement underflow.

1.2.1 Eliminate floating-point exception handling in low cost implementation

The standard way to reduce chip cost is to implement infrequent operations, such as IEEE-754 denorm processing, via exception handlers.

1.2.2 Eliminate memory load delay uncertainty in pipeline implementations

If the processor is fully pipelined, denorm processing will cause time variability in the load/store path.

1.2.3 Lower circuit cost

Alt-754 denorm processing circuitry has a minimal effect on size or speed. Getting rid of the exception handlers and their software complexity will justify the use of Alt-754. Alt-754 has a small increase in the load/store path (assuming IEEE-754 denorm processing is done in the ALU) and an overall reduction in the ALU path.

1.3 Justification for rounding rework

IEEE-754 rounding occasionally causes the carry to lengthen the mantissa requiring a right shift and exponent adjustment. Even more rarely, the exponent adjust causes the value to become and infinity.

1.3.1 Eliminate delay uncertainty of IEEE-754 rounding

Alt-754 never generates a carry. Therefore no exponent adjust. Therefore: no additional processing.

1.3.2 Lower circuit cost

Techniques are available to mitigate the IEEE-754 rounding problem. Typically, two results are computed, one of which assumes the mantissa carry and exponent adjust. The correct result is chosen at the last moment. This approach requires duplicate adders.

1.4 Examples of engineered “floating-point” datum

1.4.1 Logarithmic “db”

1.4.2 u-law & a-law codecs

1.4.3 Variety of floating-point formats (past & present)

13423 Blanco Rd. #144, San Antonio, Texas 78216

Two Alt-754 16-bit Floating-Point formats

"Standard format Alt-754 (i.e. 16-bit memory floats a.k.a. IEEE-754)"

s 0000 yyyyyyyyyy	11 octaves, 1 to 11 bits accuracy (denorms)
s xxxx yyyyyyyyyy	14 octaves, 12 bits accuracy
s 1111 yyyyyyyyyy	11 octaves, 1 to 11 bits accuracy (gradual overflow)

"All underflow/overflow (i.e. with all exponents treated as under/overflow)"

s xxxx yyyyyyyyy1	16 octaves, 11 bits accuracy
s xxxx yyyyyyyy10	16 octaves, 10 bits accuracy
s xxxx yyyyyyy100	16 octaves, 9 bits accuracy
s xxxx yyyyyy1000	16 octaves, 8 bits accuracy
s xxxx yyyyy10000	16 octaves, 7 bits accuracy
s xxxx yyyy100000	16 octaves, 6 bits accuracy
s xxxx yyy1000000	16 octaves, 5 bits accuracy
s xxxx yy10000000	16 octaves, 4 bits accuracy
s xxxx y100000000	16 octaves, 3 bits accuracy
s xxxx 1000000000	16 octaves, 2 bits accuracy
s xxxx 10000000000	16 octaves, 1 bit accuracy
s xxxx 00000000000	16 octaves, 0 bits accuracy?
s 1111 00000000000	infinity
0 0000 00000000000	zero
1 0000 00000000000	NaN

2. **The Alt-754 Model:**
2.1 Sign
2.2 Unbounded bit string
2.2.1 Embedded decimal point
2.3 Von-Neumann or "jam" rounding
2.3.1 Unbiased rounding
2.4 Encoding issues
2.4.1 Exponent range
2.4.2 Mantissa size
2.4.3 Corner treatment
2.4.3.1 Symmetric corners
2.4.3.2 Corner taper
2.5 Special values
2.5.1 Zero
2.5.2 Infinity
2.5.3 NaN (not a number)

3. Alt-754 Implementation:

3.4 Effect of unbiased von-Neumann "jam" rounding

Von-Neumann or "jam" rounding is generally considered inferior to IEEE-754 rounding. Alt-754 takes several steps to improve "jam" rounding.

3.4.1 Equivalent bias

3.4.1.1 Both IEEE-754 & Alt-754 use unbiased rounding

Standard jam rounding is biased. Using the same rule as IEEE-754 (round to nearest/even) results in unbiased results. Prior to IEEE-754 bias in rounding was often overlooked.

3.4.2 Equivalent ARRE

3.4.2.1 By treating "jam" bit as hidden

ARRE (Absolute Relative Roundoff Error) is the same for IEEE-754 and Alt-754 if one does not include the "jam" bit as part of the mantissa (i.e. consider it as a hidden bit).

3.4.3 Equivalent error growth

3.4.3.1 By treating "jam" bit as hidden

Error growth rate is approximately the same for IEEE-754 and Alt-754 if one does not include the "jam" bit as part of the mantissa (i.e. consider it as a hidden bit).

References:

Patents:
Brakefield, J.C. (1999) US Patent 5892697: Method and Apparatus for Handling Overflow and Underflow in Processing Floating-Point Numbers.

Eckard, Clinton B. (1999) US Patent 5995992: Conditional truncation indicator control for a decimal numeric processor employing result truncation (references Alt-754)

Smith, Lane Allen (2000) US Patent 6112220: Programmable overflow protection in digital processing (references Alt-754)

Steele Jr., Guy L. (2000) US Patent 6131106: System and method for floating-point computation for numbers in delimited floating point representation (uses Alt-754's method of handling denormalized numbers in the IEEE-754 context, references Alt-754)

Floating-Point Problems:
Berkeley Design Technology, Inc. (1997). DSP on General-Purpose Processors: http://www.bdti.com/products/gpp604.htm. (floating-point exception processing on IBM PowerPC 604)

Coonen, Jerome (1997) A Proposal for RealJava (uses merged multiply-add of IBM PowerPC) http://www.validgh.com/java/realjava.html

Huckle, Thomas (2000): Collection of Software Bugs (http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html) (several rounding error bugs, several incorrect units' bugs)

Kahan, W. & Joseph D. Darcy (1998): How Java's Floating-Point Hurts Everyone Everywhere http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf (an ardent supporter of **ALL** IEEE-754 features, better content and writing than previous articles)

IEEE-754 & IEEE-854:
Cody, W.J. (1984) A proposed radix- and word-length-independent standard for floating-point arithmetic, pg. 86-100 of IEEE Micro 4:4 (IEEE-854).

Coonen, J.T. (1980) An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic, pg. 68-79 of IEEE Computer, Jan 1980.

Goldberg, David (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. ACM Computing Surveys, 23:1:5-48.
or
Goldberg, David (1990) Appendix A: Computer Arithmetic, pg. A1-A62 of Computer Architecture A Quantitative Approach by Hennessy & Patterson, Morgan-Kaufmann.)

Von-Neumann Rounding:

Cavanagh, J.J.F. (1984) Von Neumann Rounding, pg. 429-430 of Digital Computer Arithmetic: Design and Implementation, McGraw-Hill.

Von Neumann, John (1945?) First Draft of a Report on the EDVAC, pg. 50-51 of IEEE Annals of the History of Computing, 15:4, 1993. (Von Neumann rounding)

Kuck, D. J., D. S. Parker Jr., A. H. Sameh (1977) Analysis of Rounding Methods in Floating-Point Arithmetic (IEEE Trans. Comput. C-26:643-650) (in Swartzlander I)

Other:

Cody, W.J. (1984) A proposed radix- and word-length-independent standard for floating-point arithmetic, pg 86-100 of IEEE Micro 4:4 (IEEE-854).

Cody, W. J. (1987) Analysis of Proposals for the Floating-Point Standard (Computer Mar. pp63-68) (in Swartzlander II)

Hamming, R. W. (1970) On the Distribution of Numbers (Bell Syst. Tech. J. 49:1609-1625) (in Swartzlander I) (basis for ARRE analysis)

Muller, J-M., A. Scherbyna, A. Tisserand (1998) Semi-Logarithmic Number Systems (IEEE Trans. Comput. C-47:145-151) (most advanced of logarithmic coding approach)

Parhami, Behrooz (2000): Computer Arithmetic, Algorithms and Hardware Designs (through and current coverage of computer arithmetic)

Swartzlander Jr., Earl E. ed. (1990): Computer Arithmetic vols. I & II (classic articles on computer arithmetic covering mathematics, logic and silicon implementations)

Synopsys Inc, CoWare Inc & Frontier Design Inc (2000): SystemC User's Guide http://www.systemc.org (Chapter 8 covers rounding, overflow and truncation modes for integer arithmetic)

13423 Blanco Rd. #144, San Antonio, Texas 78216

Appendix:
The Case for Alt-754 Floating-Point

By James C. Brakefield

The current floating-point standard, IEEE-754, is the basis for most computer floating-point arithmetic. It offers a variety of numerical properties (some advocates want a simpler standard, others want better software availability of the less used features). It offers 32-bit and 64-bit memory values. Cost or performance sensitive applications typically have variances or incomplete implementations. Some implementations use exception handlers in place of non-cost-effective hardware.

The floating-point advocated in this paper, herein called **Alt-754**, was motivated by certain hardware and software complexities of IEEE-754. The process of finding solutions to IEEE-754 complexity illuminates the choices faced in floating-point architecture.

Choice #1:

IEEE-754 uses exact numbers to stand for both approximate values and for exact values. The impact is that the rounding mechanism is forced to be speed or complexity sub-optimal. Alt-754 has separate and distinct sets of approximate and exact values.

Choice #2:

Values near zero, termed herein as the "low corner", must deal with decreased resolution. The representatives chosen and their encoding affect hardware and software complexity. Alt-754 has a "high corner" symmetric to the "low corner". The Alt-754 encoding is chosen for implementation ease.

Choice #3:

Wrapping the choices in a floating-point architecture into a model provides the clearest presentation of the floating-point design. Alt-754 offers a relatively clean model with some useful mathematical properties:

3.1 The model illuminates an implementation as the process of encoding a subset of the model.
3.2 Symmetry between lower accuracy values near zero and similar accuracy at the high end of the magnitude scale.
3.3 Rounding is defined so as to clarify the issue of rounding bias.
3.4 A powerful model provides generality in the form of defined variations.
3.5 Alt-754 offers the option to track accuracy.

Begin by having exact and inexact values. The exact values are integers scaled by a power of two. The inexact values are a range of values. A specific range is encoded by an Alt-754 floating-point bit pattern. As the two kinds of values are treated distinctly, the center point of the inexact value ranges can be chosen to facilitate hardware implementation.

The model of an inexact range is:

A floating-point number is a sign bit and a bit string unbounded on both ends. Between two specific bits of the string is a decimal point. The number of positions between the decimal point and the leading one bit is the exponent. The number of positions between the leading one bit and the trailing one bit is the accuracy. The bits less significant than and including the trailing one bit are indeterminate; e.g. a specific sign bit and bit string represent a range of values.

If there is no leading one bit, the value is infinite (positive or negative depending on the sign bit). If there is no trailing one bit, the value is exact. If all bits are zeros, the value is mathematical zero. There is also a single value called NaN ("not a number").

The model provides a mathematical set of floating point values. For a given implementation some subset is chosen: The exponent and accuracy limits are specified. The exponent size is chosen as so many bits. Accuracy is chosen to match the arithmetic unit size.

The model of rounding is:

The sequence of number ranges (e.g. for a specific exponent and accuracy) consists of alternating open and closed intervals. Values at the boundary between open and closed intervals are "moved" to the closed interval. (this rule is the same as IEEE-754).

Since the interval center for approximate numbers is a design parameter, and exact and inexact numbers are not co-mingled, the choice for Alt-754 is that of "von Neumann" rounding: chop the excess least significant bits and replace by a single one bit. For unbiased rounding, chose the open and closed intervals so as to avoid carry propagation.

Corner treatment:

IEEE-754 uses denorms to represent values close to zero. The format chosen requires a normalization to occur in the memory read path and a denormalization to occur in the memory write path. IEEE-854 does not force a particular representation of denorms.

For Alt-754 the number range is symmetric: loss of accuracy is supported at both ends of the number scale. One set of exponents is allocated to the low end and another set to the high end. The low end is also used to represent zero and NaN, and the high end is also used to represent the two infinities.

The Alt-754 encoding scheme for denorms is that the mantissas remain normalized (with the usual leading hidden most significant one bit), but that the least significant part of the number contain an exponent adjustment: A one bit followed by zero or more zero bits. The number of zero bits is the exponent adjustment: At the low end (near zero), the adjustment is subtracted, and at the high end, the adjustment is added to the exponent field to get the true exponent.

With symmetric corners the zero value becomes a number range encompassing both positive and negative values. The infinities also become extended ranges encompassing everything larger in magnitude than the highest high-end corner values.

Variations:

The number of exponents allocated to the corners is a design parameter. The choice of no corners corresponds to not implementing denorms in IEEE-754. The choice of all corners (all values are denorms) corresponds to accuracy gradually decreasing as values occur further from the center (the center is +/- 1.0). The "all corners" choice provides a greatly expanded exponent range at the cost of the accuracy taper.

The number of bits allocated to the exponent and mantissa does not affect the model. Instead, shortening the mantissa or exponent or varying the corners only affects the subset of numerical values (e.g. set of specific ranges) implemented. E.g. a less accurate implementation lacks the finer resolution or the exponent range of the more accurate implementation. Formal statement: The model contains a superset of all possible implementations.

When more than one exponent code is allocated to each corner there are unused codes which can be used to provide several NaNs and two zeros.

Accuracy tracking:

Two modes of operation of the arithmetic unit are possible: round to arithmetic unit accuracy and round to arithmetic result accuracy. Either way the arithmetic unit will use additional register bits to encode result accuracy. Then when the result is stored to memory, a trailing one bit code can be inserted to indicate remaining accuracy. The trailing one bit code matches the representation used for denorms and thus does not require additional hardware (unless both denorm and accuracy codes are used simultaneously).

Hardware issues:

The difficult parts of IEEE-754 are:
denorm processing,
rounding causing carries and possible overflow,
the intricacies of negative zero, exception handling, and four rounding modes.

Limitations of IEEE-754 are:
no accuracy tracking,
no gradual overflow,
asymmetric exponent range,
no provision for varying exponent range,
no provision for varying mantissa accuracy,

no provision for varying denorm “corners”

Denorm processing: IEEE-754 requires a normalization step for memory reads and a denormalization step for memory writes. Alt-754 requires examining the trailing one bit code and adjusting the exponent for memory reads and the insertion of the trailing one bit code for memory writes.

Rounding for IEEE-754 requires (50% of the time) adding one to the mantissa. Occasionally the add one causes exponent overflow. Alt-754 chops the mantissa. The exponent never changes.

In retrospect IEEE-754 was developed in the era of micro-coded processors and non-pipelined arithmetic units. Rare cases do not have much effect on overall runtime or micro-code size. The current situation is that PCs use pipelining for add/subtract/multiply. Rare cases cause increased logic and increased delay in pipelined arithmetic units. Thus IEEE-754 is technologically dated.

Some features of IEEE-754 reflect historical biases:

Most early floating-point formats had a bit ordering which allowed an integer compare to be used on a pair of floats. This saved an op-code and allowed fast floating-point compares.

Mathematicians wanted features to make numerical analysis easier.

Embedded applications with their emphasis on low cost were not considered.

Trade-offs between hardware and software (what was put into the instruction set versus what was put into the programming language) was slanted towards hardware implementation.

Summary:

A more through examination of the issues in floating-point design shows that some features of IEEE-754 have superior alternatives. One set of design decisions is shown by Alt-754. Alt-754 has a simple mathematical model to motivate the design and guide implementation decisions.