

Novel Aspects of IEEE SA P3109

Arithmetic Formats for Machine Learning

Andrew Fitzgibbon
Graphcore

Christoph M. Wintersteiger
Imandra

Jeffrey Sarnoff
IEEE

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

P3109™/D1
Draft Standard for
Arithmetic Formats for Machine Learning

version of June 29, 2026

Developed by the
Microprocessor Standards Committee
of the
IEEE Computer Society

Approved (Date Approved)

Copyright © 2026 by The Institute of Electrical and Electronics Engineers, Incorporated
Three Park Avenue
New York, New York 10016-5997, USA
All rights reserved.

This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to change. **USE AT YOUR OWN RISK!** IEEE copyright statements SHALL NOT BE REMOVED from draft or approved IEEE standards, or modified in any way. Because this is an unapproved draft, this document must not be utilized for IEEE performance/compliance purposes. Permission is hereby granted for officers from each IEEE Standards Working Group or Committee to reproduce the draft document developed by that Working Group for purposes of international standardization consideration. IEEE Standards Department must be informed of the submission for consideration prior to any reproduction for international standardization consideration (stds-ipr@ieee.org). Prior to submission of this document, in whole or in part by another standardization organization, IEEE Standards Department will require a copy of the standard development organization's document highlighting the use of IEEE content. Other entities seeking permission to reproduce this document, in whole or in part, must also obtain permission from the IEEE Standards Department.

IEEE Standards Department
445 Hoos Lane
Piscataway, NJ 08854, USA

09 e Learning

I. Wintersteiger

Jeffrey Sarnoff
IEEE

Participants

At the time this draft was completed, the P3109 Working Group had the following membership:

Kiran Gunam, *Chair*
Leonard Tsai, *Vice Chair*
Jeffrey Sarnoff, *Secretary*

Malia Zaman, *IEEE Standards Board Liaison*
Tom Thompson, *past IEEE Standards Liaison*

Editors

Jeffrey Sarnoff (Editor in Chief), Andrew Fitzgibbon, Christoph M. Wintersteiger
Contributing Editors
Amos Omond, Guy Lemieux

Contributors

Nima Badzadegan	Michel Hack	Michael Overton
Paul Bainea	Dandan Huang	Katherine Perry
David Chen	Lanso Hunhold	Nathalie Revol
Marco Cococcioni	Seokbum Ko	Jason Riedy
George Constantinides	Carlo Lucchi	Ali Skozogari
Marius Cornea	David Lutz	Eric Schwarcz
Mike Cowlishaw	The Ly	Olivier Serracys
James Davenport	Albert Martin	Michael Siu
James Demmel	Mantas Mikaitis	Gil Tibek
Kenneth Docker	Aufab Mushi	Julio Villalba
Massimiliano Fusi	Santosh Nagarkatte	Kristopher Wong
Silvio-Joan Filip	Aisha Naseer	Thomas Yeh
John Gustafson	Badreddine Noutte	Aleksandr Zakharchenko
	Siuari Oberman	

The following members of the individual Standards Association balloting group voted on this draft. Balloters may have voted for approval, disapproval, or abstention.
[To be supplied by IEEE.]
When the IEEE SA Standards Board approved this Standard on <Date Approved>, it had the following membership:
[To be supplied by IEEE.]

Trademarks and service marks: IEEE Std 754-2019™ is a trademark owned by The Institute of Electrical and Electronics Engineers, Incorporated.

Learning

Wintersteiger

Jeffrey Sarnoff
IEEE

Contents

1 Overview	11
1.1 Scope	11
1.2 Word usage	11
2 Definitions and abbreviations	12
2.1 Definitions	12
2.2 Abbreviations	13
2.3 Mathematical notations	13
3 Floating-point formats	14
3.1 Formats	14
3.2 Naming	15
4 Operations	16
4.1 General	16
4.2 Projection specifications	16
4.3 Operation definitions	17
4.3.1 Operation definition schema	17
4.3.2 Pattern-matching declarations	19
4.4 Approximate implementations	20
4.5 Conforming implementations	22
4.6 Conformance declarations	23
4.7 Interact functions	23
4.7.1 General	23
4.7.2 Decoding	24
4.7.3 Projection	25
4.7.4 Rounding to precision	27
4.7.5 Saturation	28
4.7.6 Encoding	28
4.8 Decoding and encoding for external formats	29
4.8.1 Decoding	29
4.8.2 Encoding	29
4.9 Converting between formats	30
4.9.1 Conversion	30
4.10 Arithmetic operations	31
4.10.1 Absolute value, Negation	31
4.10.2 Copying the sign	31
4.10.3 Addition, Subtraction	32
4.10.4 Multiplication	33
4.10.5 Division	33
4.10.6 Fused Multiply-Add	34
4.10.7 Fused Add-Add	35
4.10.8 Square root, Reciprocal, Reciprocal square root	36
4.10.9 Logarithm, Exponentiation	37
4.10.10 Trigonometric functions	38
4.10.11 Hyperbolic functions	39
4.10.12 Trigonometric π -functions	40
4.10.13 Sigmoid	41
4.10.14 Hypotenuse	42
4.10.15 Inverse tangent of two variables	43

ng

eiger

Jeffrey Sarnoff
IEEE

2 Definitions and abbreviations

2.1 Definitions

For the purposes of this document, the following terms and definitions apply:

- bitwidth:** the minimum number of bits required to encode a floating-point value in a given format, denoted K .
- canonical form:** of a nonzero finite floating-point datum X in format f , its representation as $\text{sign}(X) \times S \times 2^{1-P} \times 2^E$ for minimal integer $E > -B$, and nonnegative integer S , where $P = \text{PrecisionOf}(f)$ and $B = \text{ExponentBiasOf}(f)$. For zero, $S = 0$ and $E = 0$.
- closed extended reals:** set denoted \mathbb{R}^{ce} , consisting of the real numbers augmented with positive infinity, negative infinity, and non-a-number; $\mathbb{R}^{\text{ce}} := \mathbb{R} \cup \{-\infty, +\infty, \text{NaN}\}$.
- code point:** integer in the range 0 to $2^K - 1$ that encodes a floating-point datum occurring in a format of bitwidth K .
- datum set:** see floating-point datum.
- datum set:** the subset of the closed extended reals representable in a format of bitwidth K .
- defined operation:** operation whose signature and behavior are defined by this standard.
- defined result:** result of a defined operation for a given set of operands.
- domain:** format-defining parameter in $\{\text{Finite}, \text{Extended}\}$ that specifies whether the format's datum set includes infinities.
- encoding:** a format's unique mapping from its datum set to code points.
- exponent:** of a floating-point datum X expressed in canonical form as $\text{sign}(X) \times S \times 2^{1-P} \times 2^E$, the integer E .
- exponent bias:** format-specific constant, denoted B , used in encoding and decoding the exponent field.
- exponent field:** the integer value of the biased exponent of a code point.
- floating-point datum:** a closed extended real value representable in a given format, an element of the datum set of the format.
- floating-point value:** a code point representing a floating-point datum in a given format, an element of the datum set of the format.
- format:** binary floating-point representation parameterized by bitwidth, precision, signadness, and domain.
- format-defining parameters:** bitwidth (K), precision (P), signadness (Σ), and domain (Δ).
- NaN (Not a Number):** special non-numeric value used to represent undefined or unacceptable results.
- normal:** finite floating-point datum whose significand S satisfies $2^{P-1} \leq S < 2^P$. A floating-point value is normal iff its datum is normal.
- operand:** an argument to an operation.
- operation:** a set of operative specializations parameterized over operation parameters, e.g., format, projection specification, and constants such as block size.
- operation specialization:** mapping from a tuple of operands to one or more results.

Jeffrey Sarnoff
IEEE

3 Floating-point formats

3.1 Formats

Define the set of *closed extended reals* to be the reals augmented with positive and negative infinity and NaN:

$$\mathbb{R}^{\infty} := \mathbb{R} \cup \{-\infty, +\infty, \text{NaN}\}$$

A floating-point format f comprises a *datum set* D_f and an *encoding*. The datum set is a subset of the closed extended reals. The encoding is a unique bijective mapping from D_f to integer code points $0 \dots 2^B - 1$, where K is the bitwidth of the format.

A floating-point format has four *format-defining parameters*:

- Bitwidth K , an integer greater than two;¹
- Precision P , an integer greater than zero. P shall be strictly less than K ($0 < P < K$) for signed formats, and less than or equal to K ($0 < P \leq K$) for unsigned formats.
- Signfulness Σ , in {Signed, Unsigned};
- Domain Δ , in {Finite, Extended};²

The *exponent bits* is derived from the format-defining parameters.³ For signed formats, the exponent bias shall be $B = 2^{P-1}$. For unsigned formats, the exponent bias shall be $B = 2^{P-1}$.

A floating-point datum in a format f is an element of D_f , i.e., a closed extended real that encodes to a code point in f . A floating-point value is a code point representing a floating-point datum in a given format. The code points for infinities are denoted *Inf* and *-Inf*.

NOTE 2—While a floating-point value is synonymous with a code point, the term is used in contexts where a format is associated with the value. For example, in an operation definition, a value might be declared “ x : floating-point value, format f ”. In such contexts, the interpretation of a real constant, e.g., 2.0 , as a floating-point value is to be read as $\omega\text{Encode}_f(2.0)$. In cases where the format may be ambiguous, a real constant may be subscripted with its associated format, e.g., $X_j = \omega\text{Encode}_f(X)$. For example, $2\text{bits}_{\text{significance}} = \omega\text{Encode}_{\text{Significance}}(2.0)$ represents the code point $0x7F$.

The word *finite* applied to a datum signifies that the datum is not $\pm\infty$ or NaN. Applied to a value, it signifies that the value encodes a finite datum.

A finite floating-point datum is a real number whose absolute value has the form

$$S \times 2^{1-P} \times 2^E$$

where the *exponent* E is an integer such that $E > -B$, and S , the *significand*, is an integer $0 \leq S < 2^P$. This decomposition is made canonical for nonzero values by choosing the smallest $E > -B$. For $S = 0$, E is chosen to be 0. The datum is *normal* if $2^{P-1} \leq S < 2^P$. The datum is *subnormal*⁴ if $0 < S < 2^{P-1}$. The datum zero is neither normal nor subnormal.

The *trailing significand* is the integer S used 2^{P-1} .

NOTE 3—The encoding of formats is described in §4.7.6; properties of encodings are described in Annex B.

¹See Annex C for constraints for $K > 2$.
²See Annex A.4 for rationale for parameterization of domain.
³See Annex A.5 for rationale for the choice of exponent bias.
⁴See Annex A.6 for rationale for the choice of subnormal.

effrey Sarnoff
EEE

4.10.4 Multiplication

Multiplication of two floating-point values, returning a floating-point value.
Signature
 $\text{Multiply}_{f_x, f_y, f_r, \rho}(x, y) \rightarrow r$

Parameters
 f_x : format of operand x
 f_y : format of operand y
 f_r : format of result r
 ρ : projection specification

Operands
 x : floating-point value, format f_x
 y : floating-point value, format f_y

Result
 r : floating-point value, format f_r .

Behavior

- $\omega \text{Multiply}(\text{NaN}, *) \rightarrow \text{NaN}$
- $\omega \text{Multiply}(*, \text{NaN}) \rightarrow \text{NaN}$
- $\omega \text{Multiply}(+, +) \rightarrow +\infty$
- $\omega \text{Multiply}(-, -) \rightarrow +\infty$
- $\omega \text{Multiply}(-, +) \rightarrow -\infty$
- $\omega \text{Multiply}(+, -) \rightarrow -\infty$
- $\omega \text{Multiply}(+, Y) \text{ if } Y > 0 \rightarrow +\infty$
- $\omega \text{Multiply}(+, Y) \text{ if } Y = 0 \rightarrow \text{NaN}$
- $\omega \text{Multiply}(+, Y) \text{ if } Y < 0 \rightarrow -\infty$
- $\omega \text{Multiply}(X, +) \text{ if } X > 0 \rightarrow +\infty$
- $\omega \text{Multiply}(X, +) \text{ if } X = 0 \rightarrow \text{NaN}$
- $\omega \text{Multiply}(X, +) \text{ if } X < 0 \rightarrow -\infty$
- $\omega \text{Multiply}(-, Y) \text{ if } Y > 0 \rightarrow -\infty$
- $\omega \text{Multiply}(-, Y) \text{ if } Y = 0 \rightarrow \text{NaN}$
- $\omega \text{Multiply}(-, Y) \text{ if } Y < 0 \rightarrow +\infty$
- $\omega \text{Multiply}(X, -) \text{ if } X > 0 \rightarrow -\infty$
- $\omega \text{Multiply}(X, -) \text{ if } X = 0 \rightarrow \text{NaN}$
- $\omega \text{Multiply}(X, -) \text{ if } X < 0 \rightarrow +\infty$
- $\omega \text{Multiply}(X, Y) \rightarrow X \times Y$
- $\text{Multiply}(x, y) \rightarrow \omega \text{Project}_{f_r, \rho}(\omega \text{Multiply}(\omega \text{Decode}_{f_x}(x), \omega \text{Decode}_{f_y}(y)))$

/ Sarnoff

Appendices

Annex A (Informative)

Rationales and discussion

A.1 General

The principle underpinning the design of floating-point formats is suitability for use in machine learning systems, where narrow bitwidths are important for reducing energy usage, conserving memory space, and delivering timely results. Therefore, rationales consider the cost of a feature in terms of number of code points required and its utility in machine learning systems.

A.2 Not a number (NaN)

Datum sets include exactly one NaN.

NaN is obtained from operation results that are outside the set of numerical values, e.g., division of zero by zero, or addition of positive and negative infinities. Other floating-point systems define multiple NaN values. These encodings are used to allow different exceptional conditions to be distinguished.

In the context of machine learning systems, uses of NaN include:

- Debugging of code running on accelerator hardware. In machine learning accelerators, exceptions may be difficult or expensive to convey back to user code, so it is common practice to allow NaN values to propagate through calculations to indicate that an error has occurred.
- Use as a sentinel value. In some datasets, for example, where individual element values may be missing or out of range, a sentinel may be used to record the position of these values. In many cases, this will require less memory than storing such information out-of-band, such as in a coordinate-list (COO) format array. In some cases, `NaN` can be used as a missing value, but given the restricted range of the formats, it is likely that infinity will be used as a separate indicator of rounding from values outside of the finite range.
- The use of multiple NaN payloads is known in statistical code (e.g., NaN and “not available” or N/A) but it is not widely used in large-scale machine learning systems.

For other purposes, supporting multiple NaNs would reduce the already limited encoding space (e.g., occupying code points where the exponent field is all ones, reducing dynamic range) and potentially increase hardware complexity.

A.3 Zero

Datum sets include exactly one zero.

The inclusion of negative zero (-0) would incur the cost of an additional code point. Given the decision to encode only a single NaN, placing that NaN at the code point where negative zero would be encoded enables the strictly positive and strictly negative number ranges to be symmetric for signed formats.

A key rationale for including -0 in IEEE-754 was the consistent implementation of branch cuts in the ArcTan2 function and the complete trigonometric functions [8, 9]. The ArcTan2 function is rarely used in deep learning applications. The related ArcTan function is common in deep learning, however it is generally used as an activation function, rather than a trigonometric operation.

loff

The PAR (Project Authorization Request)

“This standard defines a binary arithmetic and data format for machine learning optimized domains. It also specifies the default handling of exceptions occurring in this arithmetic. This standard provides a consistent and flexible arithmetic framework optimized for Machine Learning Systems (MLSs) in hardware and/or software implementations to minimize the work required to make MLSs interoperable with each other as well as other dependent systems. This standard is aligned with the IEEE Std 754-2019 Standard for Floating-Point Arithmetic.”

IEEE SA P3109 Active Project Authorization Request, Approved 15 Feb 2023
<https://standards.ieee.org/ieee/3109/11165/>

The PAR (Project Authorization Request)

“This standard defines a binary arithmetic and data format for machine learning optimized domains. It also specifies the default handling of exceptions occurring in this arithmetic. This standard provides a **consistent and flexible** arithmetic framework optimized for Machine Learning Systems (MLSs) in hardware and/or software implementations to minimize the work required to make MLSs interoperable with each other as well as other dependent systems. This standard is aligned with the IEEE Std 754-2019 Standard for Floating-Point Arithmetic.”

IEEE SA P3109 Active Project Authorization Request, Approved 15 Feb 2023
<https://standards.ieee.org/ieee/3109/11165/>

Why P3109?

Context:

- Machine learning workloads pushed common arithmetic from 32-bit to 16, 8, and even 4-bit formats, saving power, silicon area, and memory bandwidth.
- Existing low-precision hardware formats disagree on infinities, NaNs, scaling, and supplied operations.
- Approximate operations (e.g. flushing subnormals, unusual accumulator widths) are common in practice.
- It's not clear which formats or operations will endure.

Why P3109?

Context:

- Machine learning workloads pushed common arithmetic from 32-bit to 16, 8, and even 4-bit formats, saving power, silicon area, and memory bandwidth.
- Existing low-precision hardware formats disagree on infinities, NaNs, scaling, and supplied operations.
- Approximate operations (e.g. flushing subnormals, unusual accumulator widths) are common in practice.
- It's not clear which formats or operations will endure.

Response:

- A coherent family of signed and unsigned floating-point **formats** focused on narrow-bitwidth computations used in machine learning systems.
- A family of arithmetic **operations** parameterized over operand formats and projection modes.
- Operation definitions are automatically generated from **formal specifications**, ensuring well-defined outputs for all formats and over all input values.
- Multiple rounding and saturation modes.
- Operations on scaled blocks.
- A new scale invariant measure of operation accuracy, named κ -approximation, that is computed entirely in representation space.

Principles

Priority order

1. correct
2. precise
3. easy to read

What that means

Some terminology or notation may be hard to grasp at first reading. We chose to settle correctness and precision first; clearer teaching material can follow. This paper and talk aim to be more approachable.

Describe, don't prescribe

The key role of the standard is to **define** behavior, not to insist that every vendor implement every variant.

Specializations

Each defined operation induces many specializations over formats and projection modes. No vendor is expected to supply all of them in hardware, or even in software; the point is to define them cleanly for present and future use.

Design Choices Driven by ML

Single zero, no negative zero

Frees a scarce code point, keeps signed ranges symmetric.

Single NaN

Keeps a useful debugging and sentinel value without spending multiple encodings on payload variants.

Signed or Unsigned

With small bitwidths, naturally unsigned quantities (e.g. block scale factors) need not pay for a sign bit.

Optional infinities

The standard supports both extended and finite domains so narrow formats can trade range signaling against extra finite values.

Stable bias choice

Finite values keep the same encoding whether or not infinities are present; precisions greater than 1 include subnormals.

Consistent and Flexible

Format Subformat	OCP		AGQ		TSL		P3109	
	E4M3	E5M2	E4M3	E5M2	E4M3	E5M2	k8p4se	k8p3se
Special values shared	✗		✓		✗		✓	
Number of NaNs	2	6	1	1	1	1	1	1
Include negative zero	Y	Y	N	N	N	N	N	N
Has infinity	N	Y	N	N	N	N	Y	Y
Max exponent e_{max}	8	15	7	15	N/A	N/A	7	15

OCP = Open Compute Platform, AGQ = AMD/Graphcore/Qualcomm, TSL = Tesla Dojo.
“Special values shared” means that related formats reuse the same encodings for special values.

Flexible Formats: Binary $\{K, P, \Sigma, \Delta\}$

A floating-point format has four *format-defining* parameters:

- Bitwidth $K > 2$
- Precision $P > 0$, with $P < K$ for signed formats, $P \leq K$ for unsigned formats
- Signedness Σ , in $\{\text{Signed}, \text{Unsigned}\}$
- Domain Δ , in $\{\text{Finite}, \text{Extended}\}$

Short names: e.g. binary8p4sf, analogous to OCP E4M3.

Case insensitive, but the document uses CamelCase.

Exponent bias

For signed formats, the exponent bias shall be $B = 2^{K-P-1}$.

For unsigned formats, the exponent bias shall be $B = 2^{K-P}$.

This differs from IEEE-754, where the exponent bias is defined in terms of e_{max} , the exponent of the largest finite value, which is in turn derived from the IEEE-754 interchange format parameters k and p .

For the majority of P3109 formats, the connection is the same as in IEEE-754.

FP4 Value Tables: Extended Domain

Code point	k4p1se	k4p2se	k4p3se
0 0b0000	Zero	Zero	Zero
1 0b0001	0.1250	<u>0.2500</u>	<u>0.2500</u>
2 0b0010	0.2500	0.5000	<u>0.5000</u>
3 0b0011	0.5000	0.7500	<u>0.7500</u>
4 0b0100	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.5000	1.2500
6 0b0110	4.0000	2.0000	1.5000
7 0b0111	Inf	Inf	Inf
8 0b1000	NaN	NaN	NaN
9 0b1001	-0.1250	<u>-0.2500</u>	<u>-0.2500</u>
10 0b1010	-0.2500	<u>-0.5000</u>	<u>-0.5000</u>
11 0b1011	-0.5000	<u>-0.7500</u>	<u>-0.7500</u>
12 0b1100	-1.0000	-1.0000	-1.0000
13 0b1101	-2.0000	-1.5000	-1.2500
14 0b1110	-4.0000	-2.0000	-1.5000
15 0b1111	-Inf	-Inf	-Inf

FP4 Value Tables: Extended Domain

Code point	k4p1se	k4p2se	k4p3se
0 0b0000	Zero	Zero	Zero
1 0b0001	0.1250	<u>0.2500</u>	<u>0.2500</u>
2 0b0010	0.2500	0.5000	<u>0.5000</u>
3 0b0011	0.5000	0.7500	<u>0.7500</u>
4 0b0100	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.5000	1.2500
6 0b0110	4.0000	2.0000	1.5000
7 0b0111	Inf	Inf	Inf
8 0b1000	NaN	NaN	NaN
9 0b1001	-0.1250	<u>-0.2500</u>	<u>-0.2500</u>
10 0b1010	-0.2500	<u>-0.5000</u>	<u>-0.5000</u>
11 0b1011	-0.5000	<u>-0.7500</u>	<u>-0.7500</u>
12 0b1100	-1.0000	-1.0000	-1.0000
13 0b1101	-2.0000	-1.5000	-1.2500
14 0b1110	-4.0000	-2.0000	-1.5000
15 0b1111	-Inf	-Inf	-Inf

Code point	k4p1ue	k4p2ue	k4p3ue	k4p4ue
0 0b0000	Zero	Zero	Zero	Zero
1 0b0001	≈0.0078	<u>0.0625</u>	<u>0.1250</u>	<u>0.1250</u>
2 0b0010	≈0.0156	0.1250	<u>0.2500</u>	<u>0.2500</u>
3 0b0011	≈0.0312	0.1875	<u>0.3750</u>	<u>0.3750</u>
4 0b0100	0.0625	0.2500	0.5000	<u>0.5000</u>
5 0b0101	0.1250	0.3750	0.6250	<u>0.6250</u>
6 0b0110	0.2500	0.5000	0.7500	<u>0.7500</u>
7 0b0111	0.5000	0.7500	0.8750	<u>0.8750</u>
8 0b1000	1.0000	1.0000	1.0000	1.0000
9 0b1001	2.0000	1.5000	1.2500	1.1250
10 0b1010	4.0000	2.0000	1.5000	1.2500
11 0b1011	8.0000	3.0000	1.7500	1.3750
12 0b1100	16.0000	4.0000	2.0000	1.5000
13 0b1101	32.0000	6.0000	2.5000	1.6250
14 0b1110	Inf	Inf	Inf	Inf
15 0b1111	NaN	NaN	NaN	NaN

FP4 Value Tables: Finite Domain

Code point	k4p1sf	k4p2sf	k4p3sf
0 0b0000	Zero	Zero	Zero
1 0b0001	0.1250	<u>0.2500</u>	<u>0.2500</u>
2 0b0010	0.2500	0.5000	<u>0.5000</u>
3 0b0011	0.5000	0.7500	<u>0.7500</u>
4 0b0100	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.5000	1.2500
6 0b0110	4.0000	2.0000	1.5000
7 0b0111	8.0000	3.0000	1.7500
8 0b1000	NaN	NaN	NaN
9 0b1001	-0.1250	<u>-0.2500</u>	<u>-0.2500</u>
10 0b1010	-0.2500	<u>-0.5000</u>	<u>-0.5000</u>
11 0b1011	-0.5000	<u>-0.7500</u>	<u>-0.7500</u>
12 0b1100	-1.0000	-1.0000	-1.0000
13 0b1101	-2.0000	-1.5000	-1.2500
14 0b1110	-4.0000	-2.0000	-1.5000
15 0b1111	-8.0000	-3.0000	-1.7500

FP4 Value Tables: Finite Domain

Code point	k4p1sf	k4p2sf	k4p3sf
0 0b0000	Zero	Zero	Zero
1 0b0001	0.1250	<u>0.2500</u>	<u>0.2500</u>
2 0b0010	0.2500	0.5000	<u>0.5000</u>
3 0b0011	0.5000	0.7500	<u>0.7500</u>
4 0b0100	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.5000	1.2500
6 0b0110	4.0000	2.0000	1.5000
7 0b0111	8.0000	3.0000	1.7500
8 0b1000	NaN	NaN	NaN
9 0b1001	-0.1250	<u>-0.2500</u>	<u>-0.2500</u>
10 0b1010	-0.2500	<u>-0.5000</u>	<u>-0.5000</u>
11 0b1011	-0.5000	<u>-0.7500</u>	<u>-0.7500</u>
12 0b1100	-1.0000	-1.0000	-1.0000
13 0b1101	-2.0000	-1.5000	-1.2500
14 0b1110	-4.0000	-2.0000	-1.5000
15 0b1111	-8.0000	-3.0000	-1.7500

Code point	k4p1uf	k4p2uf	k4p3uf	k4p4uf
0 0b0000	Zero	Zero	Zero	Zero
1 0b0001	≈0.0078	<u>0.0625</u>	<u>0.1250</u>	<u>0.1250</u>
2 0b0010	≈0.0156	0.1250	<u>0.2500</u>	<u>0.2500</u>
3 0b0011	≈0.0312	0.1875	<u>0.3750</u>	<u>0.3750</u>
4 0b0100	0.0625	0.2500	0.5000	<u>0.5000</u>
5 0b0101	0.1250	0.3750	0.6250	<u>0.6250</u>
6 0b0110	0.2500	0.5000	0.7500	<u>0.7500</u>
7 0b0111	0.5000	0.7500	0.8750	<u>0.8750</u>
8 0b1000	1.0000	1.0000	1.0000	1.0000
9 0b1001	2.0000	1.5000	1.2500	1.1250
10 0b1010	4.0000	2.0000	1.5000	1.2500
11 0b1011	8.0000	3.0000	1.7500	1.3750
12 0b1100	16.0000	4.0000	2.0000	1.5000
13 0b1101	32.0000	6.0000	2.5000	1.6250
14 0b1110	64.0000	8.0000	3.0000	1.7500
15 0b1111	NaN	NaN	NaN	NaN

FP4 Value Tables: Finite Domain

Code point	k4p1sf	k4p2sf	k4p3sf
0 0b0000	Zero	Zero	Zero
1 0b0001	0.1250	<u>0.2500</u>	<u>0.2500</u>
2 0b0010	0.2500	0.5000	<u>0.5000</u>
3 0b0011	0.5000	0.7500	<u>0.7500</u>
4 0b0100	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.5000	1.2500
6 0b0110	4.0000	2.0000	1.5000
7 0b0111	8.0000	3.0000	1.7500
8 0b1000	NaN	NaN	NaN
9 0b1001	-0.1250	<u>-0.2500</u>	<u>-0.2500</u>
10 0b1010	-0.2500	<u>-0.5000</u>	<u>-0.5000</u>
11 0b1011	-0.5000	<u>-0.7500</u>	<u>-0.7500</u>
12 0b1100	-1.0000	-1.0000	-1.0000
13 0b1101	-2.0000	-1.5000	-1.2500
14 0b1110	-4.0000	-2.0000	-1.5000
15 0b1111	-8.0000	-3.0000	-1.7500

Code point	E2M1
0 0b0000	Zero
1 0b0001	<u>0.5000</u>
2 0b0010	1.0000
3 0b0011	1.5000
4 0b0100	2.0000
5 0b0101	3.0000
6 0b0110	4.0000
7 0b0111	6.0000
8 0b1000	NaN
9 0b1001	-0.5000
10 0b1010	-1.0000
11 0b1011	-1.5000
12 0b1100	-2.0000
13 0b1101	-3.0000
14 0b1110	-4.0000
15 0b1111	-6.0000

OCP E2M1 is a factor-of-two
rescaling of Binary4p2sf

Encodings of Selected Values

Datum	Symbol	Signed extended	Signed finite	Unsigned extended	Unsigned finite
Zero	0.0	0	0	0	0
One	1.0	$2^{K-2} - 0$	$2^{K-2} - 0$	$2^{K-1} - 0$	$2^{K-1} - 0$
Not a Number	NaN	$2^{K-1} - 0$	$2^{K-1} - 0$	$2^{K-0} - 1$	$2^{K-0} - 1$
Positive Infinity	Inf	$2^{K-1} - 1$	N/A	$2^{K-0} - 2$	N/A
Negative Infinity	-Inf	$2^{K-0} - 1$	N/A	N/A	N/A

Takeaway

Zero is always code point 0. Signed formats place the single NaN at the IEEE negative-zero slot; unsigned extended formats use the top two code points for $+\infty$ and NaN.

Encodings of Selected Values

Datum	Symbol	Signed extended	Signed finite	Unsigned extended	Unsigned finite
Zero	0.0	0x00	0x00	0x00	0x00
One	1.0	0x40	0x40	0x80	0x80
Not a Number	NaN	0x80	0x80	0xff	0xff
Positive Infinity	Inf	0x7f	N/A	0xfe	N/A
Negative Infinity	-Inf	0xff	N/A	N/A	N/A

Takeaway

Zero is always code point 0. Signed formats place the single NaN at the IEEE negative-zero slot; unsigned extended formats use the top two code points for $+\infty$ and NaN.

Concrete example for $K = 8$.

Floating-point datum

Define the set of **closed extended reals** to be the reals augmented with positive and negative infinity and NaN:

$$\mathbb{R}^\omega := \mathbb{R} \cup \{-\infty, +\infty, \text{NaN}\}$$

There is no negative zero in this set.

A **floating-point format** f comprises a **datum set** $\mathcal{D}_f \subset \mathbb{R}^\omega$ and an **encoding**.

The **encoding** is a unique bijective mapping from \mathcal{D}_f to integer **code points** $0 \dots 2^K - 1$, where K is the bitwidth of the format.

A **floating-point datum** in a format f is an element of \mathcal{D}_f , i.e., a closed extended real that encodes to a code point in f .

Floating-point datum and canonical form

A finite floating-point datum is a real number whose absolute value has the form

$$S \times 2^{1-P} \times 2^E,$$

The **exponent** E is an integer such that $E > -B$, where B is the exponent bias (later)

The **significand** S , is an integer $0 \leq S < 2^P$.

This decomposition is not unique: if S is small, so that $2S$ is still less than 2^P , then

$$S \times 2^{1-P} \times E = (2S) \times 2^{1-P} \times 2^{E-1}$$

It is made canonical by choosing the smallest $E > -B$, and hence the largest $S < 2^P$.

Floating-point datum and canonical form

A finite floating-point datum is a real number whose absolute value has the form

$$S \times 2^{1-P} \times 2^E,$$

The **exponent** E is an integer such that $E > -B$, where B is the exponent bias (later)

The **significand** S , is an integer $0 \leq S < 2^P$.

This decomposition is not unique: if S is small, so that $2S$ is still less than 2^P , then

$$S \times 2^{1-P} \times E = (2S) \times 2^{1-P} \times 2^{E-1}$$

It is made canonical by choosing the smallest $E > -B$, and hence the largest $S < 2^P$.

The datum is **subnormal** if $0 < S < 2^{P-1}$.

Operations and Operation Specializations

Operators are liberally **parameterized**, over operand and result formats, rounding, and saturation.

Systems do not supply operations, they supply **operation specializations**, e.g.

$\text{Log}\langle \text{Binary8p4se}, \text{BFloat16}, (\text{ToOdd}, \text{SatNone}) \rangle$

Signature

$$\text{Log}_{f_x, f_r, \rho}(x) \rightarrow r$$

Parameters

f_x : format of operand x

f_r : format of result r

ρ : projection specification

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$$\omega \text{Log}(\text{NaN}) \rightarrow \text{NaN}$$

$$\omega \text{Log}(-\infty) \rightarrow \text{NaN}$$

$$\omega \text{Log}(+\infty) \rightarrow +\infty$$

$$\omega \text{Log}(X) \text{ if } X < 0 \rightarrow \text{NaN}$$

$$\omega \text{Log}(0) \rightarrow -\infty$$

$$\omega \text{Log}(X) \rightarrow \log_e X$$

$$\text{Log}(x) \rightarrow \omega \text{Project}_{f_r, \rho}(\omega \text{Log}(\omega \text{Decode}_{f_x}(x)))$$

Operations and Operation Specializations

Operations operate on **floating-point values**.

A floating-point value “is a code point representing a floating-point datum in a given format”

A real constant may be subscripted with its associated format, e.g., $X_f = \omega\text{Encode}_f(X)$.

For example,

$$2.0_{\text{Binary8p4se}} = \omega\text{Encode}_{\text{Binary8p4se}}(2.0) = 0x48.$$

Signature

$$\text{Log}_{f_x, f_r, \rho}(x) \rightarrow r$$

Parameters

f_x : format of operand x

f_r : format of result r

ρ : projection specification

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$$\omega\text{Log}(\text{NaN}) \rightarrow \text{NaN}$$

$$\omega\text{Log}(-\infty) \rightarrow \text{NaN}$$

$$\omega\text{Log}(+\infty) \rightarrow +\infty$$

$$\omega\text{Log}(X) \text{ if } X < 0 \rightarrow \text{NaN}$$

$$\omega\text{Log}(0) \rightarrow -\infty$$

$$\omega\text{Log}(X) \rightarrow \log_e X$$

$$\text{Log}(x) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Log}(\omega\text{Decode}_{f_x}(x)))$$

Operations and Operation Specializations

Decode, omega-Op, Project

Recall $\mathbb{R}^\omega := \mathbb{R} \cup \{-\infty, +\infty, \text{NaN}\}$

General pattern of definitions

$X = \omega\text{Decode}(x)$ where $X \in \mathbb{R}^\omega$

$R = \omega\text{Log}(X)$ where $R \in \mathbb{R}^\omega$

$r = \omega\text{Project}_{f,\rho}(R)$ result $r \in \mathbb{Z}$, a code point

Pattern matching proceeds top to bottom.

Special cases handled explicitly.

Signature

$\text{Log}_{f_x, f_r, \rho}(x) \rightarrow r$

Parameters

f_x : format of operand x

f_r : format of result r

ρ : projection specification

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\text{Log}(\text{NaN}) \rightarrow \text{NaN}$

$\omega\text{Log}(-\infty) \rightarrow \text{NaN}$

$\omega\text{Log}(+\infty) \rightarrow +\infty$

$\omega\text{Log}(X)$ if $X < 0 \rightarrow \text{NaN}$

$\omega\text{Log}(0) \rightarrow -\infty$

$\omega\text{Log}(X) \rightarrow \log_e X$

$\text{Log}(x) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Log}(\omega\text{Decode}_{f_x}(x)))$

Project: Rounding and Saturation

General pattern:

$$X = \omega\text{Decode}(x) \quad \text{where } X \in \mathbb{R}^\omega$$

$$R = \omega\text{Log}(X) \quad \text{where } R \in \mathbb{R}^\omega$$

$$r = \omega\text{Project}_{f,\rho}(R) \quad \text{result } r \in \mathbb{Z}$$

$\omega\text{Project}_{f,\rho}(R)$ is defined as:

$$R_r = \omega\text{RoundToPrecision}(R)$$

$$R_s = \omega\text{Saturate}(R_r)$$

$$r = \omega\text{Encode}(R_s)$$

RoundToPrecision parameters

P : integer precision

B : exponent bias

μ : rounding mode

Operand and Result

X : closed extended real value

Z : closed extended real value

Behavior

$\omega\text{RoundToPrecision}(X \in \{0, -\infty, \infty, \text{NaN}\}) \rightarrow X$

$\omega\text{RoundToPrecision}(X) \rightarrow Z$

where

$$Q = \max(\lfloor \log_2(|X|) \rfloor, 1 - B) - P + 1$$

$$\tilde{S} = |X| \times 2^{-Q}$$

$$S = \begin{cases} \lfloor \tilde{S} \rfloor + 1 & \text{if RoundAway}(\mu) \\ \lfloor \tilde{S} \rfloor & \text{otherwise} \end{cases}$$

$$Z = \text{sgn}(X) \times S \times 2^Q$$

Project: Rounding and Saturation

RoundToPrecision parameters

P : integer precision

B : exponent bias

μ : rounding mode

Operand and Result

X : closed extended real value

Z : closed extended real value

Behavior

$\omega\text{RoundToPrecision}(X \in \{0, -\infty, \infty, \text{NaN}\}) \rightarrow X$

$\omega\text{RoundToPrecision}(X) \rightarrow Z$

where

$$Q = \max(\lfloor \log_2(|X|) \rfloor, 1 - B) - P + 1$$

$$\tilde{S} = |X| \times 2^{-Q}$$

$$S = \begin{cases} \lfloor \tilde{S} \rfloor + 1 & \text{if RoundAway}(\mu) \\ \lfloor \tilde{S} \rfloor & \text{otherwise} \end{cases}$$

$$Z = \text{sgn}(X) \times S \times 2^Q$$

RoundAway, for $\nu = \tilde{S} - \lfloor \tilde{S} \rfloor$:

RoundAway(TowardZero) = False

RoundAway(TowardPositive) = $\nu > 0$ and $X > 0$

RoundAway(TowardNegative) = $\nu > 0$ and $X < 0$

RoundAway(NearestTiesToAway) = $\nu \geq 0.5$

RoundAway(NearestTiesToEven) =

$\nu > 0.5$ or ($\nu = 0.5$ and not CodelsEven)

RoundAway(ToOdd) = $\nu > 0$ and CodelsEven

RoundAway(StochasticA_{N,R}) = $\lfloor \nu \times 2^N \rfloor + R \geq 2^N$

RoundAway(StochasticB_{N,R}) =

$\lfloor \nu \times 2^{N+1} \rfloor + (2 \times R + 1) \geq 2^{N+1}$

RoundAway(StochasticC_{N,R}) = RNITE($\nu \times 2^N$) + R $\geq 2^N$

Project: Rounding and Saturation

General pattern:

$$X = \omega\text{Decode}(x)$$

where $X \in \mathbb{R}^\omega$

$$R = \omega\text{Log}(X)$$

where $R \in \mathbb{R}^\omega$

$$r = \omega\text{Project}_{f,\rho}(R) \quad \text{result } r \in \mathbb{Z}$$

$\omega\text{Project}_{f,\rho}(R)$ is defined as:

$$R_r = \omega\text{RoundToPrecision}(R)$$

$$R_s = \omega\text{Saturate}(R_r)$$

$$r = \omega\text{Encode}(R_s)$$

Saturate Parameters

M^{lo} : minimum finite value

M^{hi} : maximum finite value

Operands

Sat : saturation mode

Round : rounding mode

X : closed extended real value

Σ : signedness in {Signed, Unsigned}

Δ : domain in {Finite, Extended}

Behavior

$\omega\text{Saturate}(*, *, \text{NaN}, *, *) \rightarrow \text{NaN}$

$\omega\text{Saturate}(*, *, X, *, *)$ if $M^{lo} \leq X$ and $X \leq M^{hi} \rightarrow X$

$\omega\text{Saturate}(\text{SatFinite}, *, +\infty, *, *) \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatFinite}, *, -\infty, *, *) \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatFinite}, *, X, *, *)$ if $X < M^{lo} \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatFinite}, *, X, *, *)$ if $X > M^{hi} \rightarrow M^{hi}$

Project: Rounding and Saturation

General pattern:

$X = \omega\text{Decode}(x)$ where $X \in \mathbb{R}^\omega$

$R = \omega\text{Log}(X)$ where $R \in \mathbb{R}^\omega$

$r = \omega\text{Project}_{f,\rho}(R)$ result $r \in \mathbb{Z}$

$\omega\text{Project}_{f,\rho}(R)$ is defined as:

$R_r = \omega\text{RoundToPrecision}(R)$

$R_s = \omega\text{Saturate}(R_r)$

$r = \omega\text{Encode}(R_s)$

Saturate Parameters

M^{lo} : minimum finite value

M^{hi} : maximum finite value

Operands

Sat : saturation mode

Round : rounding mode

X : closed extended real value

Σ : signedness in {Signed, Unsigned}

Δ : domain in {Finite, Extended}

Behavior

$\omega\text{Saturate}(*, *, \text{NaN}, *, *) \rightarrow \text{NaN}$

$\omega\text{Saturate}(*, *, X, *, *)$ if $M^{lo} \leq X$ and $X \leq M^{hi} \rightarrow X$

$\omega\text{Saturate}(\text{SatPropagate}, *, +\infty, *, \text{Extended}) \rightarrow +\infty$

$\omega\text{Saturate}(\text{SatPropagate}, *, +\infty, *, *) \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatPropagate}, *, -\infty, \text{Signed}, \text{Extended}) \rightarrow -\infty$

$\omega\text{Saturate}(\text{SatPropagate}, *, -\infty, *, *) \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatPropagate}, *, X, *, *)$ if $X < M^{lo} \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatPropagate}, *, X, *, *)$ if $X > M^{hi} \rightarrow M^{hi}$

Project: Rounding and Saturation

General pattern:

$X = \omega\text{Decode}(x)$ where $X \in \mathbb{R}^\omega$

$R = \omega\text{Log}(X)$ where $R \in \mathbb{R}^\omega$

$r = \omega\text{Project}_{f,\rho}(R)$ result $r \in \mathbb{Z}$

$\omega\text{Project}_{f,\rho}(R)$ is defined as:

$R_r = \omega\text{RoundToPrecision}(R)$

$R_s = \omega\text{Saturate}(R_r)$

$r = \omega\text{Encode}(R_s)$

Behavior

$\omega\text{Saturate}(*, *, \text{NaN}, *, *) \rightarrow \text{NaN}$

$\omega\text{Saturate}(*, *, X, *, *)$ **if** $M^{lo} \leq X$ **and** $X \leq M^{hi} \rightarrow X$

$\omega\text{Saturate}(\text{SatNone}, *, +\infty, *, \text{Extended}) \rightarrow +\infty$

$\omega\text{Saturate}(\text{SatNone}, *, +\infty, *, *) \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatNone}, *, -\infty, \text{Signed}, \text{Extended}) \rightarrow -\infty$

$\omega\text{Saturate}(\text{SatNone}, *, -\infty, \text{Unsigned}, *) \rightarrow \text{NaN}$

$\omega\text{Saturate}(\text{SatNone}, *, -\infty, *, *) \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatNone}, \text{ToOdd}, X, \text{Unsigned}, \text{Extended})$ **if** $X > M^{hi} \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatNone}, \text{TowardZero}, X, *, *)$ **if** $X > M^{hi} \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatNone}, \text{TowardZero}, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatNone}, \text{TowardNegative}, X, *, *)$ **if** $X > M^{hi} \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatNone}, \text{TowardPositive}, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatNone}, *, X, \text{Signed}, \text{Extended})$ **if** $X < M^{lo} \rightarrow -\infty$

$\omega\text{Saturate}(\text{SatNone}, *, X, \text{Unsigned}, *)$ **if** $X < M^{lo} \rightarrow \text{NaN}$

$\omega\text{Saturate}(\text{SatNone}, *, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatNone}, *, X, *, \text{Extended})$ **if** $X > M^{hi} \rightarrow +\infty$

$\omega\text{Saturate}(\text{SatNone}, *, X, *, *)$ **if** $X > M^{hi} \rightarrow M^{hi}$

How do I know if r is exact under ToOdd?

Fmt (extended)	sat	finite r < MAX	r=MAX	r=inf	cf 754 at MAX, Inf
754++ ToOdd	-	IsEven(r) => Exact	?	yes	
Signed	SatNone	IsEven(r)	yes	?	more, less
Signed	SatProp	IsEven(r)	?	yes	same, same
Signed	SatFinite	IsEven(r)	?	n/a	same, n/a
Unsigned	SatNone	IsEven(r)	yes	?	more, less
Unsigned	SatProp	IsEven(r)	?	yes	same, same
Unsigned	SatFinite	IsEven(r)	?	n/a	same, n/a

$\omega\text{Saturate}(\text{SatNone}, \text{TowardZero} \vee \text{TowardPositive}, X, *, *)$ if $X < M^{lo} \rightarrow M^{lo}$

$\omega\text{Saturate}(\text{SatNone}, \text{TowardZero} \vee \text{TowardNegative}, X, *, *)$ if $X \geq M^{hi} \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatNone}, \text{ToOdd}, X, \text{Unsigned}, *)$ if $X \geq M^{hi} \rightarrow M^{hi}$

$\omega\text{Saturate}(\text{SatNone}, *, X, \text{Signed}, \text{Extended})$ if $X < M^{lo} \rightarrow -\infty$

$\omega\text{Saturate}(\text{SatNone}, *, X, *, *)$ if $X < M^{lo} \rightarrow M^{lo}$

Project: Encode

General pattern:

$$X = \omega\text{Decode}(x) \quad \text{where } X \in \mathbb{R}^\omega$$

$$R = \omega\text{Log}(X) \quad \text{where } R \in \mathbb{R}^\omega$$

$$r = \omega\text{Project}_{f,\rho}(R) \quad \text{result } r \in \mathbb{Z}$$

$\omega\text{Project}_{f,\rho}(R)$ is defined as:

$$R_r = \omega\text{RoundToPrecision}(R)$$

$$R_s = \omega\text{Saturate}(R_r)$$

$$r = \omega\text{Encode}(R_s)$$

Note: integer multiply and modulo rather than bitshift/masking.

Behavior of ωEncode :

$$\omega\text{Encode}(\text{NaN}) \rightarrow \begin{cases} 2^{K-1} & \text{if } \Sigma = \text{Signed} \\ 2^K - 1 & \text{if } \Sigma = \text{Unsigned} \end{cases}$$
$$\omega\text{Encode}(+\infty) \rightarrow \begin{cases} 2^{K-1} - 1 & \text{if } \Sigma = \text{Signed} \\ 2^K - 2 & \text{if } \Sigma = \text{Unsigned} \end{cases}$$

$$\omega\text{Encode}(X < 0) \rightarrow \omega\text{Encode}_f(-X) + 2^{K-1}$$

$$\omega\text{Encode}(0) \rightarrow 0$$

$$\omega\text{Encode}(X > 0) \rightarrow r$$

where

$$r = \begin{cases} T & \text{if } S < 2^{P-1} \\ T + (E + B) \times 2^{P-1} & \text{otherwise} \end{cases}$$

and

$$E = \max(\lfloor \log_2(X) \rfloor, 1 - B)$$

$$S = X \times 2^{-E} \times 2^{P-1}$$

$$T = S \bmod 2^{P-1}$$

Formal specs

Divide demonstrates:

- Returns NaN on divide by zero, to avoid the inconsistency $1/(1/\infty) = +\infty$.
- Operation definitions are formally specified and machine-checked.

Signature

$\text{Divide}_{f_x, f_y, f_r, \rho}(x, y) \rightarrow r$

Operands

x : floating-point value, format f_x

y : floating-point value, format f_y

Result

r : floating-point value, format f_r

Behavior

$\omega\text{Divide}(\text{NaN}, *) \rightarrow \text{NaN}$

$\omega\text{Divide}(*, \text{NaN}) \rightarrow \text{NaN}$

$\omega\text{Divide}(+\infty, \pm\infty) \rightarrow \text{NaN}$

$\omega\text{Divide}(-\infty, \pm\infty) \rightarrow \text{NaN}$

$\omega\text{Divide}(*, 0) \rightarrow \text{NaN}$

$\omega\text{Divide}(+\infty, Y) \text{ if } Y > 0 \rightarrow +\infty$

$\omega\text{Divide}(+\infty, Y) \text{ if } Y < 0 \rightarrow -\infty$

$\omega\text{Divide}(-\infty, Y) \text{ if } Y > 0 \rightarrow -\infty$

$\omega\text{Divide}(-\infty, Y) \text{ if } Y < 0 \rightarrow +\infty$

$\omega\text{Divide}(*, \pm\infty) \rightarrow 0$

$\omega\text{Divide}(X, Y) \rightarrow X/Y$

$\text{Divide}(x, y) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Divide}(X, Y))$

where

$X = \omega\text{Decode}_{f_x}(x)$

$Y = \omega\text{Decode}_{f_y}(y)$

Formal specs

```
(** 4.11.5 Division *)

let wDivide (x : CER.t) (y : CER.t) :
  (CER.t, string) Result.t =
  let open CER in
  match x, y with
  | NaN, _          -> Ok NaN
  | _, NaN          -> Ok NaN
  | PInf, (PInf | NInf) -> Ok NaN
  | NInf, (PInf | NInf) -> Ok NaN
  | _, R 0.0        -> Ok NaN
  | PInf, R y when y >. 0.0 -> Ok PInf
  | PInf, R y when y <. 0.0 -> Ok NInf
  | NInf, R y when y >. 0.0 -> Ok NInf
  | NInf, R y when y <. 0.0 -> Ok PInf
  | _, (PInf | NInf) -> Ok (R 0.0)
  | R x, R y -> Ok (R (x /. y))
  | _ -> Error "undefined"

let divide f_x f_y f_r rho x y =
  match
    wDivide (wDecode f_x x) (wDecode f_y y)
  with
  | Ok r -> wProject f_r rho r
  | Error e -> Error e

(** Theorem: "divide never returns an error" *)
theorem divide_ok f_x f_y f_r rho x y =
  Result.is_ok (divide f_x f_y f_r rho x y)
```

Signature

$\text{Divide}_{f_x, f_y, f_r, \rho}(x, y) \rightarrow r$

Operands

x : floating-point value, format f_x

y : floating-point value, format f_y

Result

r : floating-point value, format f_r

Behavior

$\omega\text{Divide}(\text{NaN}, *) \rightarrow \text{NaN}$

$\omega\text{Divide}(*, \text{NaN}) \rightarrow \text{NaN}$

$\omega\text{Divide}(+\infty, \pm\infty) \rightarrow \text{NaN}$

$\omega\text{Divide}(-\infty, \pm\infty) \rightarrow \text{NaN}$

$\omega\text{Divide}(*, 0) \rightarrow \text{NaN}$

$\omega\text{Divide}(+\infty, Y)$ if $Y > 0 \rightarrow +\infty$

$\omega\text{Divide}(+\infty, Y)$ if $Y < 0 \rightarrow -\infty$

$\omega\text{Divide}(-\infty, Y)$ if $Y > 0 \rightarrow -\infty$

$\omega\text{Divide}(-\infty, Y)$ if $Y < 0 \rightarrow +\infty$

$\omega\text{Divide}(*, \pm\infty) \rightarrow 0$

$\omega\text{Divide}(X, Y) \rightarrow X/Y$

$\text{Divide}(x, y) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Divide}(X, Y))$

where

$X = \omega\text{Decode}_{f_x}(x)$

$Y = \omega\text{Decode}_{f_y}(y)$

Formal Specs and Generated Text

IML formal specification → Machine-checked proofs → Generated standards text

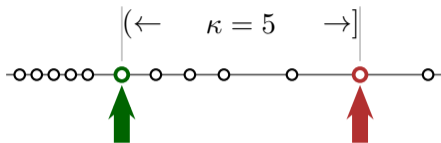
What is proved

- completeness and consistency of generated operation rules
- support lemmas for `Encode`, `Decode`, and projection
- around 500 theorems already in the formalization

And more:

FLoPS: Semantics, Operations, and Properties of P3109 Floating-Point Representations in Lean
Tung-Che Chang, Sehyeok Park, Jay P Lim, and Santosh Nagarakatte
<https://github.com/rutgers-apl/FLoPS>

Kappa-Approximate Implementations

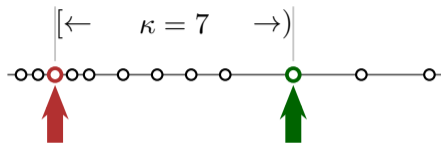


$$\hat{r}(x) \in V$$

Correctly rounded result

$$\tilde{r}(x) \in V$$

Approximate result



$$\tilde{r}(x) \in V$$

Approximate result

$$\hat{r}(x) \in V$$

Correctly rounded result

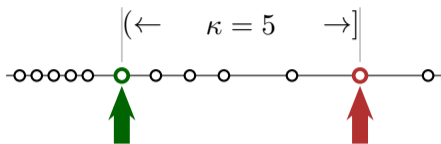
$$\kappa_{\text{above}}(x) = \#([\hat{r}(x), \tilde{r}(x)] \cap V)$$

$$\kappa_{\text{below}}(x) = \#([\tilde{r}(x), \hat{r}(x)] \cap V)$$

Interpretation

κ counts the representable finite values between $\tilde{r}(x)$ and $\hat{r}(x)$, measured in codepoints. It's like ulps, but well defined across binades, and near normal/subnormal boundaries.

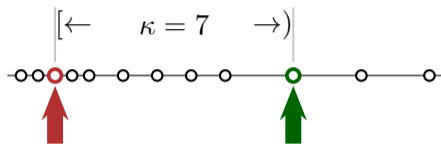
Kappa-Approximate Implementations



$\hat{r}(x) \in V$

$\tilde{r}(x) \in V$

Correctly rounded result Approximate result



$\tilde{r}(x) \in V$

$\hat{r}(x) \in V$

Approximate result Correctly rounded result

$$\kappa_{\text{above}}(x) = \#((\hat{r}(x), \tilde{r}(x)] \cap V)$$

$$\kappa_{\text{below}}(x) = \#([\tilde{r}(x), \hat{r}(x)) \cap V)$$

Formula

$$\kappa = \max_{x \in I} \# \left(\left((\hat{r}(x), \tilde{r}(x)] \cup [\tilde{r}(x), \hat{r}(x)) \right) \cap V \right).$$

Kappa-Approximation example

An implementer of `Exp<binary32, Binary8p4se, (NearestTiesToEven, SatNone)>` flushes subnormals in the result to zero or to the smallest normal, whichever is nearest, with ties going to zero.

There are seven nonzero positive subnormals in `Binary8p4se`, of which four will be flushed to zero, and three will be returned as the smallest normal.

The value of κ over all inputs producing finite results is 4.

So the implementation could declare $\kappa = 4$.

Kappa-Approximation example

An implementer of `Exp<binary32, Binary8p4se, (NearestTiesToEven, SatNone)>` flushes subnormals in the result to zero or to the smallest normal, whichever is nearest, with ties going to zero.

There are seven nonzero positive subnormals in `Binary8p4se`, of which four will be flushed to zero, and three will be returned as the smallest normal.

Writing the cut points $a, b, c = \ln(2^{-11}), \ln(9 \cdot 2^{-11}), \ln(15 \cdot 2^{-11})$, the intervals are:

$$I_0 = \mathcal{D}_{\text{binary32}} \cap ((-\infty, a) \cup (c, +\infty)) \quad \kappa_{I_0} = 0$$

$$I_4 = \mathcal{D}_{\text{binary32}} \cap (a, b) \quad \kappa_{I_4} = 4$$

$$I_3 = \mathcal{D}_{\text{binary32}} \cap (b, c) \quad \kappa_{I_3} = 3$$

and the implementation could have the more precise declaration

$$\kappa \in \{I_0 : 0, I_3 : 3, I_4 : 4\}.$$

Note that as a, b, c are irrational, the sets I_0, I_3, I_4 are disjoint and cover all inputs producing finite results.

Block Operations

Key observation: most block operations take block scale as input.

$\text{BlockExp}((s, [x_1, \dots, x_B]), s_r) \rightarrow (s_r, [r_1, \dots, r_B])$ $\text{BlockProject}(s, [X_1, \dots, X_B]) \rightarrow [r_1, \dots, r_B]$ where

where

$$S = \omega\text{Decode}_{f_s}(s)$$

$$X_i = \omega\text{Multiply}(S, \omega\text{Decode}_{f_x}(x_i))$$

$$Z_i = \omega\text{Exp}(X_i)$$

$$[r_1, \dots, r_B] = \text{BlockProject}(s_r, [Z_1, \dots, Z_B])$$

$$S = \omega\text{Decode}_{f_s}(s)$$

$$Z_i = \begin{cases} \text{NaN} & \text{if } S \text{ is NaN or } X_i \text{ is NaN} \\ 0 & \text{if } S = 0 \\ \text{sgn}(X_i) \times \text{sgn}(S) & \text{if } S = \pm\infty \\ \omega\text{Divide}(X_i, S) & \text{otherwise} \end{cases}$$

$$r_i = \omega\text{Project}_{f_r, \rho_r}(Z_i)$$

ConvertToBlock: absmax scaling

ConvertToBlockMaxAbsFinite

$\text{CTBMAF}([x_1, \dots, x_B]) \rightarrow (s, [r_1, \dots, r_B])$

$X_i = \omega\text{Decode}_{f_x}(x_i)$

$M_i = \omega\text{Abs}(X_i)$

$S = \text{reduce}(\omega\text{MaximumFinite}, [\text{NaN}, M_1, \dots, M_B])$

$s = \omega\text{Project}_{f_s, \rho_s}(S)$

$[r_1, \dots, r_B] = \text{BlockProject}_{B, f_s, f_r, \rho}(s, [X_1, \dots, X_B])$

Interpretation

Reduction starting from NaN gives sensible behaviors for all-NaN, or all-Inf blocks.

NextGreaterThan and NextLessThan

These operations follow IEEE-754 nextUp and nextDown operations, with extensions for signedness and domain. The wording, using nextUp as an example, is extended from “least floating-point number that compares greater than” to “least floating-point number that compares greater than, or NaN”.

This requires that $\text{Inf} \rightarrow \text{NaN}$, while IEEE-754 defines $\text{nextUp}(+\infty) = +\infty$, so new names are chosen.

$\text{NextGreaterThanAux}(*, *, \text{NaN}) \rightarrow \text{NaN}$

$\text{NextGreaterThanAux}(*, \text{Extended}, \text{Inf}) \rightarrow \text{NaN}$

$\text{NextGreaterThanAux}(*, \text{Finite}, \text{MaxFiniteOf}(f)) \rightarrow \text{NaN}$

$\text{NextGreaterThanAux}(*, \text{Extended}, \text{MaxFiniteOf}(f)) \rightarrow \text{Inf}$

$\text{NextGreaterThanAux}(\text{Signed}, \text{Extended}, -\text{Inf}) \rightarrow \text{MinFiniteOf}(f)$

$\text{NextGreaterThanAux}(\text{Signed}, *, \text{SmallestNegative}) \rightarrow 0$

$\text{NextGreaterThanAux}(\text{Signed}, *, x) \text{ if } \text{IsSignMinus}(x) \rightarrow x - 1$

$\text{NextGreaterThanAux}(*, *, x) \rightarrow x + 1$

$\text{NextGreaterThan}(x) \rightarrow \text{NextGreaterThanAux}(\text{SignednessOf}(f), \text{DomainOf}(f), x)$

where $\text{SmallestNegative} = \omega \text{Encode}_f(-\omega \text{Decode}_f(\text{MinPositiveOf}(f)))$

Conclusion

- One standard describes a family of floating-point formats.
- Arithmetic is defined for all inputs through explicit decode, closed-real computation, and projection.
- Approximate implementations get a portable, representation-level accuracy language: κ .
- Block operations are part of the semantic model.
- Formal models generate text, proofs, and executable oracles together.

Fundamentally

“Describe, don’t Prescribe”

See https://github.com/P3109/Public/IEEE_P3109_WG_Interim_Report.pdf

MaximumFinite

Wrapper: decode operands, apply $\omega\text{MaximumFinite}$, then project to f_r .

$$\omega\text{MaximumFinite}(\text{NaN}, Y) = Y,$$

$$\omega\text{MaximumFinite}(+\infty, -\infty) = +\infty,$$

$$\omega\text{MaximumFinite}(+\infty, Y) = Y,$$

$$\omega\text{MaximumFinite}(-\infty, Y) = Y,$$

$$\omega\text{MaximumFinite}(\text{NaN}, \text{NaN}) = \text{NaN}$$

$$\omega\text{MaximumFinite}(X, \text{NaN}) = X$$

$$\omega\text{MaximumFinite}(+\infty, +\infty) = +\infty$$

$$\omega\text{MaximumFinite}(-\infty, -\infty) = -\infty$$

$$\omega\text{MaximumFinite}(-\infty, +\infty) = +\infty$$

$$\omega\text{MaximumFinite}(X, +\infty) = X$$

$$\omega\text{MaximumFinite}(X, -\infty) = X$$

$$\omega\text{MaximumFinite}(X, Y) = \begin{cases} Y & \text{if } X < Y \\ X & \text{otherwise} \end{cases}$$