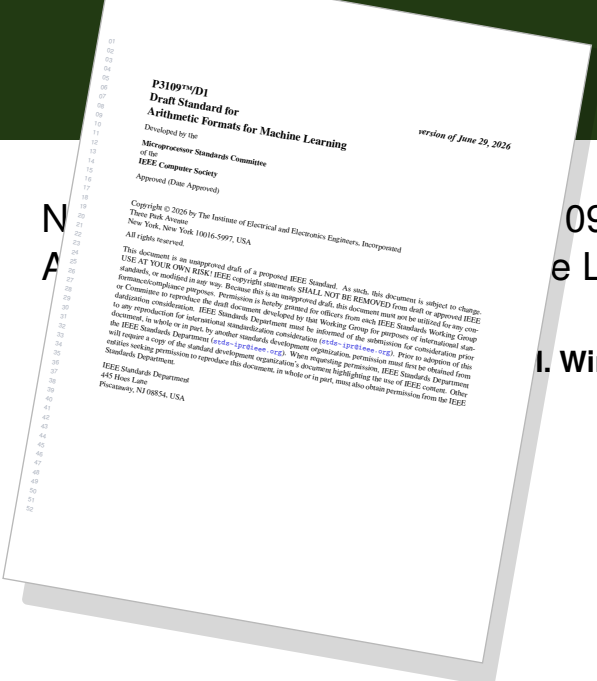
Novel Aspects of IEEE SA P3109 Arithmetic Formats for Machine Learning

Andrew Fitzgibbon
Graphcore

Christoph M. Wintersteiger
Imandra

Jeffrey Sarnoff
IEEE



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)

09

e Learning

I. Wintersteiger **Jeffrey Sarnoff**
IEEE

N

A

earning

Participants

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

Kiran Gunnam, *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 Omondi, Guy Lemieux

Contributors

Nima Badizadegan	Michel Hack	Michael Overton
Paul Balanca	Dandan Huang	Katherine Parry
David Chen	Laslo Hunhold	Nathalie Revol
Marco Cococcioni	Seokbum Ko	Jason Riedy
George Constantinides	Carlo Luschi	Ali Sazegari
Marius Cornea	David Lutz	Eric Schwarz
Mike Cowlishaw	Tue Ly	Olivier Sentieys
James Davenport	Albert Martin	Michael Siu
James Demmel	Mantas Mikaitis	Gil Tabak
Kenneth Dockser	Aaftab Munshi	Julio Villalba
Massimiliano Fasi	Santosh Nagarakatte	Kristopher Wong
Silviu-Ioan Filip	Aisha Naseer	Thomas Yeh
John Gustafson	Badreddine Noune	Aleksandr Zakharchenko
	Stuart 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.

ntersteiger

Jeffrey Sarnoff
IEEE

Contents

8

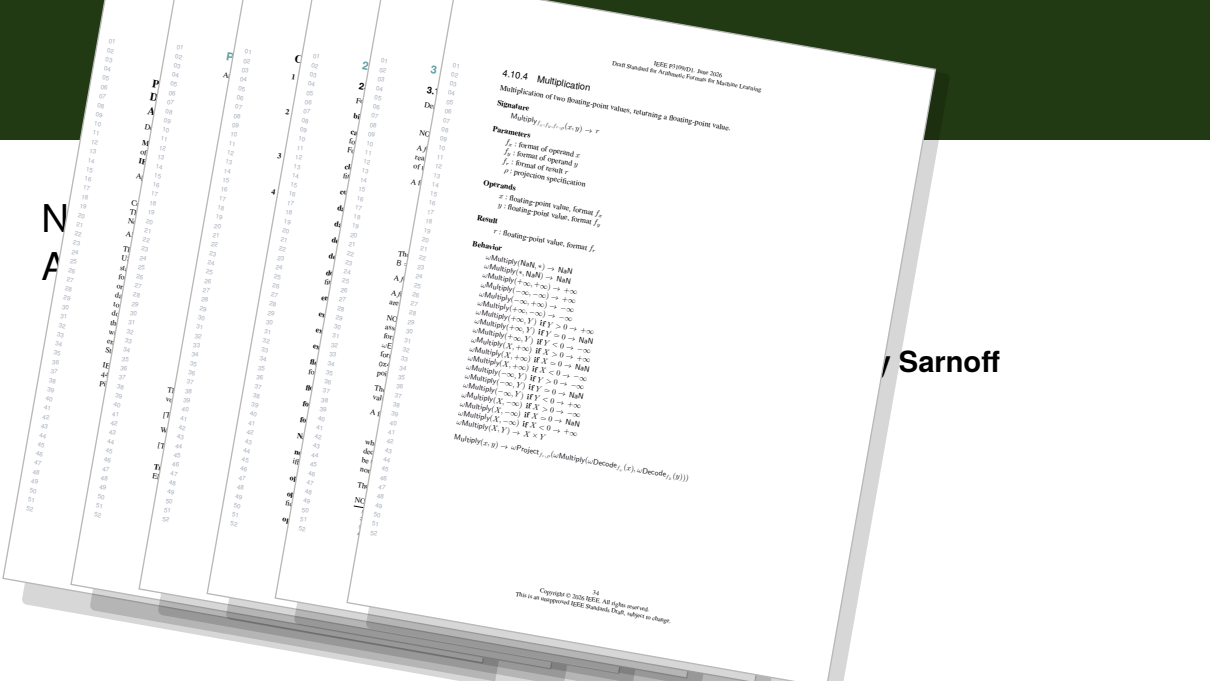
ng

eiger **Jeffrey Sarnoff**
IEEE

Jeffrey Sarnoff
IEEE

N
A

effrey Sarnoff
EEE


IEEE P1990/D1, June 2026
Draft Standard for Arithmetic Formats for Machine Learning
4.10.4 Multiplication
Multiplication of two floating-point values, returning a floating-point value.
Signature
Parameters
Operands
Result
Behavior
Copyright © 2026 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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 NaR.

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, ±inf 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 NA), but it is not widely used in large-scale machine learning systems.

For our 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 $\mathrm{Arc}\mathrm{Tan}2$ function and the complex trigonometric functions [(8, 9)]. The $\mathrm{Arc}\mathrm{Tan}2$ function is rarely used in deep learning applications. The related $\mathrm{Arc}\mathrm{Tan}$ function is common in deep learning, however it is generally used as an activation function, rather than a trigonometric operation.

B.1 Value tables

Tables 4, 5, 6, and 7 show the complete value sets for $K = 4$. Subnormals are shown in *underflayed italic*, special values are shown in **bold**.

Code point	Binary4p1se	Binary4p2se	Binary4p3se
0 0b0000	**Zero**	**Zero**	**Zero**
1 0b0001	0.1250	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	0.2500
3 0b0011	0.5000	0.7500	*0.5000*
4 0b0100	1.0000	1.0000	0.7500
5 0b0101	2.0000	1.2500	1.0000
6 0b0110	4.0000	1.5000	1.2500
7 0b0111	**Inf**	2.0000	1.5000
8 0b1000	**NaN**	**Inf**	**Inf**
9 0b1001	-0.1250	**NaN**	**NaN**
10 0b1010	-0.2500	*-0.2500*	*-0.2500*
11 0b1011	-0.5000	-0.5000	-0.5000
12 0b1100	-1.0000	-0.7500	*-0.5000*
13 0b1101	-2.0000	-1.0000	-0.7500
14 0b1110	-4.0000	-1.2500	-1.0000
15 0b1111	**-Inf**	-1.5000	-1.2500
		-2.0000	-1.5000
		-Inf	**-Inf**

Table 4—Value sets for $K = 4$, signed formats with $1 \le P \le 4$, extended domain.

Code point	Binary4p1sf	Binary4p2sf	Binary4p3sf
0 0b0000	**Zero**	**Zero**	**Zero**
1 0b0001	0.1250	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	*0.5000*
3 0b0011	0.5000	0.7500	*0.7500*
4 0b0100	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.2500	1.2500
6 0b0110	4.0000	1.5000	1.5000
7 0b0111	8.0000	2.0000	1.7500
8 0b1000	**NaN**	3.0000	**NaN**
9 0b1001	-0.1250	**NaN**	*-0.2500*
10 0b1010	-0.2500	*-0.2500*	-0.5000
11 0b1011	-0.5000	-0.5000	*-0.7500*
12 0b1100	-1.0000	-0.7500	-1.0000
13 0b1101	-2.0000	-1.0000	-1.2500
14 0b1110	-4.0000	-1.2500	-1.5000
15 0b1111	-8.0000	-1.5000	-1.7500
		-2.0000	
		-3.0000	

Table 5—Value sets for $K = 4$, signed formats with $1 \le P \le 4$, finite domain.

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

Describe, don't prescribe

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

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.

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	OCP		AGQ		TSL		P3109	
Subformat	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 emax	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$\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}$

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

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

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

This differs from IEEE-754, where the exponent bias is defined in terms of emax, 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	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	*0.5000*
3 0b0011	0.5000	0.7500	*0.7500*
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	*-0.2500*	*-0.2500*
10 0b1010	-0.2500	-0.5000	*-0.5000*
11 0b1011	-0.5000	-0.7500	*-0.7500*
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	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	*0.5000*
3 0b0011	0.5000	0.7500	*0.7500*
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	*-0.2500*	*-0.2500*
10 0b1010	-0.2500	-0.5000	*-0.5000*
11 0b1011	-0.5000	-0.7500	*-0.7500*
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	*0.0625*	*0.1250*	*0.1250*
2 0b0010	≈0.0156	0.1250	*0.2500*	*0.2500*
3 0b0011	≈0.0312	0.1875	*0.3750*	*0.3750*
4 0b0100	0.0625	0.2500	0.5000	*0.5000*
5 0b0101	0.1250	0.3750	0.6250	*0.6250*
6 0b0110	0.2500	0.5000	0.7500	*0.7500*
7 0b0111	0.5000	0.7500	0.8750	*0.8750*
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	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	*0.5000*
3 0b0011	0.5000	0.7500	*0.7500*
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	*-0.2500*	*-0.2500*
10 0b1010	-0.2500	-0.5000	*-0.5000*
11 0b1011	-0.5000	-0.7500	*-0.7500*
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	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	*0.5000*
3 0b0011	0.5000	0.7500	*0.7500*
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	*-0.2500*	*-0.2500*
10 0b1010	-0.2500	-0.5000	*-0.5000*
11 0b1011	-0.5000	-0.7500	*-0.7500*
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	*0.0625*	*0.1250*	*0.1250*
2 0b0010	≈0.0156	0.1250	*0.2500*	*0.2500*
3 0b0011	≈0.0312	0.1875	*0.3750*	*0.3750*
4 0b0100	0.0625	0.2500	0.5000	*0.5000*
5 0b0101	0.1250	0.3750	0.6250	*0.6250*
6 0b0110	0.2500	0.5000	0.7500	*0.7500*
7 0b0111	0.5000	0.7500	0.8750	*0.8750*
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	*0.2500*	*0.2500*
2 0b0010	0.2500	0.5000	*0.5000*
3 0b0011	0.5000	0.7500	*0.7500*
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	*-0.2500*	*-0.2500*
10 0b1010	-0.2500	-0.5000	*-0.5000*
11 0b1011	-0.5000	-0.7500	*-0.7500*
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	*0.5000*
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:

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

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** $0 \ldots 2^{\mathsf{K}} - 1$, 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

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

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

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

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

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

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

Floating-point datum and canonical form

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

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

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

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

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

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

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

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

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.

Log<Binary8p4se, BFloat16, (ToOdd, SatNone)>

Signature

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

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

$\omega\mathsf{Log}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Log}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(0) \rightarrow -\infty$
$\omega\mathsf{Log}(X) \rightarrow \log_e X$
$\mathsf{Log}(x) \rightarrow \omega\mathsf{Project}_{f_r, \rho}(\omega\mathsf{Log}(\omega\mathsf{Decode}_{f_x}(x)))$

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., $X_f = \omega\mathsf{Encode}_f(X)$.

For example,
$2.0_{\mathsf{Binary8p4se}} = \omega\mathsf{Encode}_{\mathsf{Binary8p4se}}(2.0) = \mathtt{0x48}$.

Signature

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

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

$\omega\mathsf{Log}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Log}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(0) \rightarrow -\infty$
$\omega\mathsf{Log}(X) \rightarrow \log_e X$
$\mathsf{Log}(x) \rightarrow \omega\mathsf{Project}_{f_r,\rho}(\omega\mathsf{Log}(\omega\mathsf{Decode}_{f_x}(x)))$

Operations and Operation Specializations

Decode, omega-Op, Project

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

General pattern of definitions

$$
\begin{aligned}
X &= \omega\text{Decode}(x) && \text{where } X \in \mathbb{R}^\omega \\
R &= \omega\text{Log}(X) && \text{where } R \in \mathbb{R}^\omega \\
r &= \omega\text{Project}_{f,\rho}(R) && \text{result } r \in \mathbb{Z}\text{, a code point}
\end{aligned}
$$

Pattern matching proceeds top to bottom.

Special cases handled explicitly.

Signature

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

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

$\omega\text{Log}(\text{NaN}) \to \text{NaN}$
$\omega\text{Log}(-\infty) \to \text{NaN}$
$\omega\text{Log}(+\infty) \to +\infty$
$\omega\text{Log}(X) \text{ if } X < 0 \to \text{NaN}$
$\omega\text{Log}(0) \to -\infty$
$\omega\text{Log}(X) \to \log_e X$
$\text{Log}(x) \to \omega\text{Project}_{f_r,\rho}(\omega\text{Log}(\omega\text{Decode}_{f_x}(x)))$

Project: Rounding and Saturation

General pattern:

$$X = \omega\mathsf{Decode}(x) \qquad \text{where } X \in \mathbb{R}^{\omega}$$
$$R = \omega\mathsf{Log}(X) \qquad \text{where } R \in \mathbb{R}^{\omega}$$
$$r = \omega\mathsf{Project}_{f,\rho}(R) \qquad \text{result } r \in \mathbb{Z}$$

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

$$R_r = \omega\mathsf{RoundToPrecision}(R)$$
$$R_s = \omega\mathsf{Saturate}(R_r)$$
$$r = \omega\mathsf{Encode}(R_s)$$

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

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

where

$$Q = \max(\lfloor \log_2(|X|) \rfloor, 1 - \mathsf{B}) - \mathsf{P} + 1$$
$$\tilde{S} = |X| \times 2^{-Q}$$
$$S = \begin{cases} \lfloor \tilde{S} \rfloor + 1 & \textbf{if } \mathsf{RoundAway}(\mu) \\ \lfloor \tilde{S} \rfloor & \textbf{otherwise} \end{cases}$$
$$Z = \mathrm{sgn}(X) \times S \times 2^{Q}$$

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

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

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

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

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

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

$\text{RoundAway}(\text{TowardZero}) = \text{False}$
$\text{RoundAway}(\text{TowardPositive}) = \nu > 0 \textbf{ and } X > 0$
$\text{RoundAway}(\text{TowardNegative}) = \nu > 0 \textbf{ and } X < 0$
$\text{RoundAway}(\text{NearestTiesToAway}) = \nu \geq 0.5$
$\text{RoundAway}(\text{NearestTiesToEven}) =$
$\quad \nu > 0.5 \textbf{ or } \left(\nu = 0.5 \textbf{ and not } \text{CodeIsEven}\right)$
$\text{RoundAway}(\text{ToOdd}) = \nu > 0 \textbf{ and } \text{CodeIsEven}$
$\text{RoundAway}(\text{StochasticA}_{N,R}) = \lfloor \nu \times 2^N \rfloor + R \geq 2^N$
$\text{RoundAway}(\text{StochasticB}_{N,R}) =$
$\quad \lfloor \nu \times 2^{N+1} \rfloor + (2 \times R + 1) \geq 2^{N+1}$
$\text{RoundAway}(\text{StochasticC}_{N,R}) = \text{RNITE}(\nu \times 2^N) + R \geq 2^N$

Project: Rounding and Saturation

General pattern:

$$X = \omega\mathsf{Decode}(x) \quad \text{where } X \in \mathbb{R}^{\omega}$$
$$R = \omega\mathsf{Log}(X) \quad \text{where } R \in \mathbb{R}^{\omega}$$
$$r = \omega\mathsf{Project}_{f,\rho}(R) \quad \text{result } r \in \mathbb{Z}$$

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

$$R_r = \omega\mathsf{RoundToPrecision}(R)$$
$$R_s = \omega\mathsf{Saturate}(R_r)$$
$$r = \omega\mathsf{Encode}(R_s)$$

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 $\{\mathsf{Signed}, \mathsf{Unsigned}\}$
Δ : domain in $\{\mathsf{Finite}, \mathsf{Extended}\}$

Behavior

$$\omega\mathsf{Saturate}(*, *, \mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$$
$$\omega\mathsf{Saturate}(*, *, X, *, *) \textbf{ if } M^{lo} \leq X \textbf{ and } X \leq M^{hi} \rightarrow X$$
$$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, +\infty, *, *) \rightarrow M^{hi}$$
$$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, -\infty, *, *) \rightarrow M^{lo}$$
$$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, X, *, *) \textbf{ if } X < M^{lo} \rightarrow M^{lo}$$
$$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, X, *, *) \textbf{ if } X > M^{hi} \rightarrow M^{hi}$$

Project: Rounding and Saturation

General pattern:

$$X = \omega\mathsf{Decode}(x) \qquad \text{where } X \in \mathbb{R}^{\omega}$$
$$R = \omega\mathsf{Log}(X) \qquad \text{where } R \in \mathbb{R}^{\omega}$$
$$r = \omega\mathsf{Project}_{f,\rho}(R) \qquad \text{result } r \in \mathbb{Z}$$

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

$$R_r = \omega\mathsf{RoundToPrecision}(R)$$
$$R_s = \omega\mathsf{Saturate}(R_r)$$
$$r = \omega\mathsf{Encode}(R_s)$$

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

$\omega\mathsf{Saturate}(*, *, \mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Saturate}(*, *, X, *, *)$ **if** $M^{lo} \leq X$ **and** $X \leq M^{hi} \rightarrow X$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, +\infty, *, \mathsf{Extended}) \rightarrow +\infty$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, +\infty, *, *) \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, -\infty, \mathsf{Signed}, \mathsf{Extended}) \rightarrow -\infty$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, -\infty, *, *) \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, X, *, *)$ **if** $X > M^{hi} \rightarrow M^{hi}$

Project: Rounding and Saturation

General pattern:

$$X = \omega\mathsf{Decode}(x) \qquad \text{where } X \in \mathbb{R}^{\omega}$$
$$R = \omega\mathsf{Log}(X) \qquad \text{where } R \in \mathbb{R}^{\omega}$$
$$r = \omega\mathsf{Project}_{f,\rho}(R) \qquad \text{result } r \in \mathbb{Z}$$

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

$$R_r = \omega\mathsf{RoundToPrecision}(R)$$
$$R_s = \omega\mathsf{Saturate}(R_r)$$
$$r = \omega\mathsf{Encode}(R_s)$$

Behavior

$\omega\mathsf{Saturate}(*, *, \mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Saturate}(*, *, X, *, *)$ **if** $M^{lo} \leq X$ **and** $X \leq M^{hi} \rightarrow X$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, +\infty, *, \mathsf{Extended}) \rightarrow +\infty$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, +\infty, *, *) \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, -\infty, \mathsf{Signed}, \mathsf{Extended}) \rightarrow -\infty$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, -\infty, \mathsf{Unsigned}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, -\infty, *, *) \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, \mathsf{ToOdd}, X, \mathsf{Unsigned}, \mathsf{Extended})$ **if** $X > M^{hi} \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, \mathsf{TowardZero}, X, *, *)$ **if** $X > M^{hi} \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, \mathsf{TowardZero}, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, \mathsf{TowardNegative}, X, *, *)$ **if** $X > M^{hi} \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, \mathsf{TowardPositive}, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, X, \mathsf{Signed}, \mathsf{Extended})$ **if** $X < M^{lo} \rightarrow -\infty$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, X, \mathsf{Unsigned}, *)$ **if** $X < M^{lo} \rightarrow \mathsf{NaN}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, X, *, *)$ **if** $X < M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, X, *, \mathsf{Extended})$ **if** $X > M^{hi} \rightarrow +\infty$
$\omega\mathsf{Saturate}(\mathsf{SatNone}, *, X, *, *)$ **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

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

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

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

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

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

Project: Encode

General pattern:

$$
\begin{aligned}
X &= \omega\mathsf{Decode}(x) && \text{where } X \in \mathbb{R}^{\omega} \\
R &= \omega\mathsf{Log}(X) && \text{where } R \in \mathbb{R}^{\omega} \\
r &= \omega\mathsf{Project}_{f,\rho}(R) && \text{result } r \in \mathbb{Z}
\end{aligned}
$$

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

$$
\begin{aligned}
R_r &= \omega\mathsf{RoundToPrecision}(R) \\
R_s &= \omega\mathsf{Saturate}(R_r) \\
r &= \omega\mathsf{Encode}(R_s)
\end{aligned}
$$

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

Behavior of ωEncode:

$$
\begin{aligned}
&\omega\mathsf{Encode}(\mathsf{NaN}) \rightarrow \begin{cases} 2^{\mathsf{K}-1} & \textbf{if } \Sigma = \mathsf{Signed} \\ 2^{\mathsf{K}} - 1 & \textbf{if } \Sigma = \mathsf{Unsigned} \end{cases} \\
&\omega\mathsf{Encode}(+\infty) \rightarrow \begin{cases} 2^{\mathsf{K}-1} - 1 & \textbf{if } \Sigma = \mathsf{Signed} \\ 2^{\mathsf{K}} - 2 & \textbf{if } \Sigma = \mathsf{Unsigned} \end{cases} \\
&\omega\mathsf{Encode}(X < 0) \rightarrow \omega\mathsf{Encode}_f(-X) + 2^{\mathsf{K}-1} \\
&\omega\mathsf{Encode}(0) \rightarrow 0 \\
&\omega\mathsf{Encode}(X > 0) \rightarrow r
\end{aligned}
$$

where

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

and

$$
\begin{aligned}
E &= \max(\lfloor \log_2(X) \rfloor, 1 - \mathsf{B}) \\
S &= X \times 2^{-E} \times 2^{\mathsf{P}-1} \\
T &= S \bmod 2^{\mathsf{P}-1}
\end{aligned}
$$

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

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Divide}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(+\infty, \pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(-\infty, \pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(*, 0) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(+\infty, Y)$ **if** $Y > 0 \rightarrow +\infty$
$\omega\mathsf{Divide}(+\infty, Y)$ **if** $Y < 0 \rightarrow -\infty$
$\omega\mathsf{Divide}(-\infty, Y)$ **if** $Y > 0 \rightarrow -\infty$
$\omega\mathsf{Divide}(-\infty, Y)$ **if** $Y < 0 \rightarrow +\infty$
$\omega\mathsf{Divide}(*, \pm\infty) \rightarrow 0$
$\omega\mathsf{Divide}(X, Y) \rightarrow X/Y$
$\mathsf{Divide}(x, y) \rightarrow \omega\mathsf{Project}_{f_r, \rho}(\omega\mathsf{Divide}(X, Y))$
where
$X = \omega\mathsf{Decode}_{f_x}(x)$
$Y = \omega\mathsf{Decode}_{f_y}(y)$

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

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

Operands

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

Result

r : floating-point value, format f_r

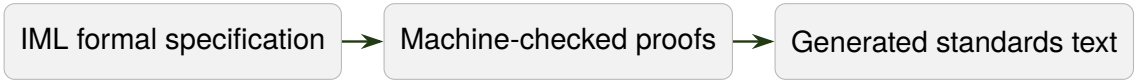
Behavior

$\omega\mathsf{Divide}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\omega\mathsf{Divide}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Divide}(+\infty, \pm\infty) \to \mathsf{NaN}$
$\omega\mathsf{Divide}(-\infty, \pm\infty) \to \mathsf{NaN}$
$\omega\mathsf{Divide}(*, 0) \to \mathsf{NaN}$
$\omega\mathsf{Divide}(+\infty, Y)$ **if** $Y > 0 \to +\infty$
$\omega\mathsf{Divide}(+\infty, Y)$ **if** $Y < 0 \to -\infty$
$\omega\mathsf{Divide}(-\infty, Y)$ **if** $Y > 0 \to -\infty$
$\omega\mathsf{Divide}(-\infty, Y)$ **if** $Y < 0 \to +\infty$
$\omega\mathsf{Divide}(*, \pm\infty) \to 0$
$\omega\mathsf{Divide}(X, Y) \to X/Y$
$\mathsf{Divide}(x, y) \to \omega\mathsf{Project}_{f_r, \rho}(\omega\mathsf{Divide}(X, Y))$

where

$X = \omega\mathsf{Decode}_{f_x}(x)$
$Y = \omega\mathsf{Decode}_{f_y}(y)$

Formal Specs and Generated Text



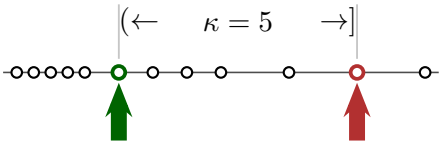
What is proved

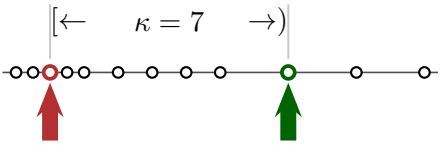
- completeness and consistency of generated operation rules
- support lemmas for Encode, Decode, and projection
- around 500 theorems already in the formalization

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





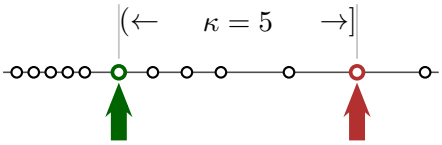
$$\kappa_{\text{above}}(x) = \#\big((\hat{r}(x), \tilde{r}(x)] \cap V\big)$$

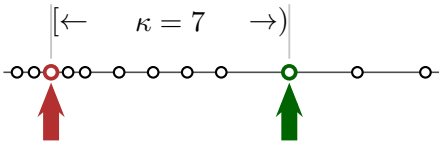
$$\kappa_{\text{below}}(x) = \#\big([\tilde{r}(x), \hat{r}(x)) \cap V\big)$$

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





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

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

Formula

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

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:

$$
\begin{aligned}
I_0 &= \mathcal{D}_{\text{binary32}} \cap ((-\infty, a) \cup (c, +\infty)) && \kappa_{I_0} = 0 \\
I_4 &= \mathcal{D}_{\text{binary32}} \cap (a, b) && \kappa_{I_4} = 4 \\
I_3 &= \mathcal{D}_{\text{binary32}} \cap (b, c) && \kappa_{I_3} = 3
\end{aligned}
$$

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.

$$\mathsf{BlockExp}((s,[x_1,...,x_B]),s_r) \to (s_r,[r_1,...,r_B])$$

where

$$S = \omega\mathsf{Decode}_{f_s}(s)$$
$$X_i = \omega\mathsf{Multiply}(S, \omega\mathsf{Decode}_{f_x}(x_i))$$
$$Z_i = \omega\mathsf{Exp}(X_i)$$
$$[r_1,...,r_B] = \mathsf{BlockProject}(s_r,[Z_1,...,Z_B])$$

$$\mathsf{BlockProject}(s,[X_1,...,X_B]) \to [r_1,...,r_B] \quad \text{where}$$

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

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

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

ConvertToBlock: absmax scaling

ConvertToBlockMaxAbsFinite

$$
\begin{aligned}
\mathsf{CTBMAF}([x_1, \ldots, x_B]) \rightarrow\ & (s, [r_1, \ldots, r_B]) \\
X_i &= \omega\mathsf{Decode}_{f_x}(x_i) \\
M_i &= \omega\mathsf{Abs}(X_i) \\
S &= \mathsf{reduce}(\omega\mathsf{MaximumFinite}, [\mathsf{NaN}, M_1, ..., M_B]) \\
s &= \omega\mathsf{Project}_{f_s, \rho_s}(S) \\
[r_1, ..., r_B] &= \mathsf{BlockProject}_{B, f_s, f_r, \rho}(s, [X_1, ..., X_B])
\end{aligned}
$$

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

NextGreaterThanAux$(*, *, \text{NaN}) \rightarrow$ NaN
NextGreaterThanAux$(*, \text{Extended}, \text{Inf}) \rightarrow$ NaN
NextGreaterThanAux$(*, \text{Finite}, \text{MaxFiniteOf}(f)) \rightarrow$ NaN
NextGreaterThanAux$(*, \text{Extended}, \text{MaxFiniteOf}(f)) \rightarrow$ Inf
NextGreaterThanAux$(\text{Signed}, \text{Extended}, \text{-Inf}) \rightarrow \text{MinFiniteOf}(f)$
NextGreaterThanAux$(\text{Signed}, *, \mathit{SmallestNegative}) \rightarrow 0$
NextGreaterThanAux$(\text{Signed}, *, x)$ **if** IsSignMinus$(x) \rightarrow x - 1$
NextGreaterThanAux$(*, *, x) \rightarrow x + 1$

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

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

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 $\omega\mathsf{MaximumFinite}$, then project to f_r.

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

$$\omega\mathsf{MaximumFinite}(\mathsf{NaN}, Y) = Y, \qquad \omega\mathsf{MaximumFinite}(X,\mathsf{NaN}) = X$$

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

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

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

$$\omega\mathsf{MaximumFinite}(+\infty, Y) = Y, \qquad \omega\mathsf{MaximumFinite}(X,+\infty) = X$$

$$\omega\mathsf{MaximumFinite}(-\infty, Y) = Y, \qquad \omega\mathsf{MaximumFinite}(X,-\infty) = X$$

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