IEEE Working Group P3109 Interim Report on Binary Floating-point Formats for Machine Learning

Initial release: 18 September 2023
Version 2.0: 29 October 2024
Version 3.0: 21 July 2025
Version 3.1: 7 November 2025

(compiled 2025-11-07)

DRAFT DOCUMENT

To cite this report please see the CITATION.cff file found at https://github.com/P3109/Public.

Contents

1 Contributors

Here are the active members of the Arithmetic Formats for Machine Learning working group.

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

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

Contacting The Working Group

To reach us, email `FP-FOR-ML-STUDY-GROUP@listserv.ieee.org`.

2 Introduction

This document represents ongoing discussions and current matters of consensus from IEEE Working Group P3109, "Standard for Arithmetic Formats for Machine Learning". The Project Authorization Request (PAR) for P3109 defines the scope, need, and stakeholders as follows:

> **Scope of proposed standard**: This standard defines a binary arithmetic and data format for machine learning-optimized domains. It also specifies the default handling of exceptions that occur in this arithmetic. This standard provides a consistent and flexible arithmetic framework optimized for Machine Learning Systems (MLS) in hardware and/or software implementations to minimize the work required to make MLS interoperable with each other, as well as other dependent systems. This standard is aligned with IEEE Std 754-2019 for Floating-Point Arithmetic.
>
> **Need for this Work**: Machine Learning Systems have different arithmetic requirements from most other domains. Precisions tend to be lower, and accuracy is measured in dimensions other than just numerical (e.g. inference accuracy). Furthermore, machine learning systems are often integrated into mission-critical and safety-critical systems. With no standards specifically addressing these needs, Machine Learning Systems are built with inconsistent expectations and assumptions that hinder the compatibility and reuse of machine learning hardware, software, and training data.
>
> **Stakeholders for the Standard**: System developers, vendors, and users of machine learning applications across many industries and interests including but not limited to computation, storage, medical, telecommunications, e-commerce, fleet management, automotive, robotics, and security.

2.1 Typographical conventions and notation

Bold text describes the decisions and specifications of this document.

2.2 Scope

This document specifies compact binary floating-point interchange formats and associated operations.

Binary formats are parameterized by: width in bits, precision, signedness, and domain.

This version of the interim report covers interchange formats and scalar operations including rounding with saturation modes and conversion between P3109 and IEEE Std 754 [1] formats.

This document defines a large set of floating-point formats, and several arithmetic and other operations on those formats. It is not expected that any implementation will define efficient implementations of all operations on all formats. It is expected that where an implementation declares the provision of an operation defined in this document, and the formats on which it is defined, that the behavior of the operation on those formats follows the definition in this document.

This document does not address tapered and related arithmetic formats [2, 3, 4, 5].

3 Formats

This section describes P3109 formats, and the set of values represented by each of these formats.

The universe of values in existing IEEE Std 754 floating-point usage is the finite reals, the non-finite values positive and negative infinity (Inf, -Inf), the value negative zero (-0), and not-a-number values (NaN, $\mathsf{NaN}_1, \ldots$).

A K-bit binary floating-point format $\mathcal{F}$ is comprised of its *value set*, a subset $\mathcal{V}_\mathcal{F}$ of the universe of values; and its unique *encoding*, a mapping from integers $0 \ldots 2^\mathsf{K} - 1$ to $\mathcal{V}_\mathcal{F}$.

Values are considered either "special" or "ordinary". The ordinary values consist of the normal and subnormal values and zero. The P3109 special values are Inf, -Inf, and a single NaN.

3.1 Format parameters and value sets

The finite floating-point numbers representable with a binary format are determined by four *format-defining* parameters:

- Storage width K, the total size of the format in bits;
- Precision P, the number of bits in the significand including the implicit leading bit;
- Signedness Σ, in $\{\mathsf{Signed}, \mathsf{Unsigned}\}$.
- Domain of the value set Δ, in $\{\mathsf{Finite}, \mathsf{Extended}\}$, encoding finite reals or extended reals (including infinities);

Formats shall be defined by the parameters of storage width, precision, signedness, and domain.

The width K shall be greater than one.

The precision P shall be greater than zero.

The precision P shall be less than K for signed formats; less than or equal to K for unsigned formats.

Other parameters are derived from the format-defining parameters. The trailing significand field (T) is given as $\mathsf{P} - 1$. For the definition of exponent bias in terms of K, P, and Σ (see §3.2).

3.1.1 Naming

Formats defined in this document shall be generically named $\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}$, where $\Sigma \in \{\mathsf{Signed}, \mathsf{Unsigned}\}$ and $\Delta \in \{\mathsf{Extended}, \mathsf{Finite}\}$.

A shortened notation is also used to refer to specific formats: $\mathsf{Binary}\langle\kappa\rangle\mathsf{p}\langle\psi\rangle\langle\sigma\rangle\langle\delta\rangle$ where the placeholders κ and ψ are decimal representations of the width K and precision P, respectively; signedness $\sigma \in \{\mathsf{s}, \mathsf{u}\}$, for signed and unsigned; and domain $\delta \in \{\mathsf{e}, \mathsf{f}\}$ for extended and finite. By convention, and without ambiguity, the characters s and e may be elided.

Examples:

- The format "Binary12p7se", which may also be written "Binary12p7", is a 12-bit signed format in the extended domain, with 1 sign bit, 5 exponent bits and 7 bits of precision (of which only 6 need to be explicitly represented).
- The format "Binary8p1uf", is an 8-bit unsigned format in the finite domain, with no sign bit, 8 exponent bits, and

1 bit of precision (requiring zero bits to be explicitly represented).

3.2 Exponent bias

In IEEE Std 754-2019, the exponent bias is determined by consideration of the exponent of the largest finite value emax, which is chosen to be $2^{\mathsf{K}-\mathsf{P}-1}-1$. The defining parameter is emax, while the bias is a derived parameter. In this document, in contrast, bias is chosen as the defining parameter, in order to maintain consistency between finite and extended formats at narrow bitwidths and low precisions. The bias is chosen in order to maintain the IEEE Std 754-2019 convention that $\mathsf{emax} = 2^{\mathsf{K}-\mathsf{P}-1}-1$ for the majority of formats, yielding the following specification:

For signed formats ($\mathsf{P} < \mathsf{K}$), the exponent bias shall be $2^{\mathsf{K}-\mathsf{P}-1}$.

For unsigned formats, the removal of the sign bit dictates a choice of bias so that $\mathsf{emax} = 2^{\mathsf{K}-\mathsf{P}}-1$

For unsigned formats ($\mathsf{P} \leq \mathsf{K}$), the exponent bias shall be $2^{\mathsf{K}-\mathsf{P}}$.

A consequence of these specifications is that the value 1.0 encodes consistently to the midway code point, that is $2^{\mathsf{K}-2}$ for signed formats and $2^{\mathsf{K}-1}$ for unsigned.

3.3 Subnormals

P3109 value sets of precision greater than 1 shall include subnormals.

IEEE Std 754 value sets include subnormals. A value with trailing significand field T and exponent field E is interpreted as $1.T \times 2^{E-\mathsf{bias}}$ except when all bits of the exponent bitfield E are zero, in which case the value is $0.T \times 2^{1-\mathsf{bias}}$.

For precision $\mathsf{P} = 1$ there are $\mathsf{P}-1 = 0$ trailing significand bits. There is no nonzero trailing significand field from which to construct subnormals, so all non-zero finite values are normal.

Subnormal numbers extend the dynamic range of floating-point values and induce equal quantization steps close to zero. This is useful when training models, where it is common for gradients to have near-zero non-zero values. Subnormals can also be useful to represent random values drawn from certain distributions. For example, model weights are initialized to small random values at training. Subnormals are uniformly spaced around zero, and near-zero values are more probable under bell-shaped distributions. Finally, formats with narrow exponent bitwidths necessarily have a limited range; subnormals extend this range by a power of 2 for every bit in the trailing significand.

3.4 Not a number (NaN)

P3109 value sets shall include exactly one NaN, which shall not signal.

Many other floating-point formats define several NaN values which are returned from operations with results outside the set of values, e.g. division of zero by zero, or addition of positive and negative infinities. Multiple NaN encodings are used in other formats 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, $\pm$Inf can be used as a missing value, but given the restricted range of P3109 formats, it is likely that infinity shall 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. the R system has NaN and N/A), but it is not widely used. In the context of P3109, supporting multiple NaNs would reduce the already limited encoding space (e.g. occupying all code points where the exponent field is all ones, thereby reducing dynamic range) and would likely add additional hardware complexity.

3.5 Zero

P3109 formats shall have exactly one zero. This zero value is non-negative.

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 where IEEE Std 754 encodes negative zero enables the strictly positive and strictly negative number ranges to be symmetric for signed formats.

A key rationale for including -0 in IEEE Std 754 was the consistent implementation of branch cuts in the atan2 function and the complex trigonometric functions [6, 7]. The atan2 function is rare, if not unknown, in deep learning applications. The related atan function is common in deep learning, however it is generally used as an activation function, rather than a trigonometric operation.

A secondary reason for providing -0 is the hardware simplification offered by its presence in the implementation of sign/magnitude arithmetic. However, current practice has made evident that the small hardware simplification has not been sufficient to balance the loss of one code point.

It might be considered that the use of integer comparisons in sorting would argue against placing NaN at the negative zero code point. For example, the JAX machine learning framework is known to sort using integer comparison [8]. However, such sorting still requires $O(n)$ preprocessing and postprocessing steps to enable the use of two's-complement integer comparison, and already has special treatment of NaN and -0, so eliminating -0 and placing NaN in the -0 position imposes negligible additional burden. Although sorting using comparison operations is undefined in the presence of NaNs, existing practice is to sort NaNs using TotalOrder.

3.6 Infinities

Unsigned formats in the extended domain shall include positive infinity.
Signed formats in the extended domain shall include positive and negative infinity.

Infinite values are used widely in machine learning systems. Examples of such usage are:

- Mask values, for example in transformer models. See §A.1 for more detailed discussion on this usage.
- Representation of overflow, for example, to adjust dynamic loss scaling factors [9].

Representing infinite values requires two code points in a signed format, and for narrow formats, the reduction in number of finite values may be significant. Hence formats are defined over the finite and extended domains.

3.7 Encoding

A K-bit P3109 floating-point value is encoded by an integer in the range 0 to $2^{\mathsf{K}} - 1$. Some properties of this encoding are summarized here, while the detailed specification of the encoding is presented in §4.9.5.

All formats contain a single zero, encoded by the integer 0.

All formats contain a single NaN. For signed formats, NaN is encoded at the code point which IEEE Std 754 uses for negative zero, that is $2^{\mathsf{K}-1}$. For unsigned formats, NaN is encoded at $2^{\mathsf{K}} - 1$.

Formats in the extended domain contain one or more infinities. For signed formats, Inf is encoded at $2^{\mathsf{K}-1} - 1$ and -Inf is encoded at $2^{\mathsf{K}} - 1$. For unsigned formats, Inf is encoded at $2^{\mathsf{K}} - 2$.

It is practical to list extremal finite values defined by the defined formats. For example: $\mathsf{MaxFinite}(\mathsf{Binary8p4se}) = 7/4 \times 2^7$, $\mathsf{MinNormal}(\mathsf{Binary8p5se}) = 1 \times 2^{-3}$. Table 2 shows the extremal values for $\mathsf{K} = 8$ and $1 \leq \mathsf{P} \leq 7$.

Tables 3 and 4 show the complete value sets for $\mathsf{K} = 4$.

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

Table 1: Encodings of selected values for given K, independent of P.

Format	MinPositive	MinNormal	MaxFinite
Binary8p1es	1×2^{-63}	1×2^{-63}	1×2^{62}
Binary8p2es	1×2^{-32}	1×2^{-31}	1×2^{31}
Binary8p3es	1×2^{-17}	1×2^{-15}	$3/2 \times 2^{15}$
Binary8p4es	1×2^{-10}	1×2^{-7}	$7/4 \times 2^{7}$
Binary8p5es	1×2^{-7}	1×2^{-3}	$15/8 \times 2^{3}$
Binary8p6es	1×2^{-6}	1×2^{-1}	$31/16 \times 2^{1}$
Binary8p7es	1×2^{-6}	1×2^{0}	$63/32 \times 2^{0}$

Table 2: Extremal values: examples for $\mathsf{K} = 8$.

Code point	k4p1se	k4p2se	k4p3se	k4p1sf	k4p2sf	k4p3sf
0x00	0.000	0.000	0.000	0.000	0.000	0.000
0x01	0.125	0.250	0.250	0.125	0.250	0.250
0x02	0.250	0.500	0.500	0.250	0.500	0.500
0x03	0.500	0.750	0.750	0.500	0.750	0.750
0x04	1.000	1.000	1.000	1.000	1.000	1.000
0x05	2.000	1.500	1.250	2.000	1.500	1.250
0x06	4.000	2.000	1.500	4.000	2.000	1.500
0x07	Inf	Inf	Inf	8.000	3.000	1.750
0x08	NaN	NaN	NaN	NaN	NaN	NaN
0x09	-0.125	-0.250	-0.250	-0.125	-0.250	-0.250
0x0A	-0.250	-0.500	-0.500	-0.250	-0.500	-0.500
0x0B	-0.500	-0.750	-0.750	-0.500	-0.750	-0.750
0x0C	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
0x0D	-2.000	-1.500	-1.250	-2.000	-1.500	-1.250
0x0E	-4.000	-2.000	-1.500	-4.000	-2.000	-1.500
0x0F	-Inf	-Inf	-Inf	-8.000	-3.000	-1.750

Table 3: Value sets for K $= 4$, signed formats with $1 \leq$ P < 4, extended and finite domains. Subnormals are shown in blue, special values in brown.

Code point	k4p1ue	k4p2ue	k4p3ue	k4p4ue	k4p1uf	k4p2uf	k4p3uf	k4p4uf
0x00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0x01	0.008	0.062	0.125	0.125	0.008	0.062	0.125	0.125
0x02	0.016	0.125	0.250	0.250	0.016	0.125	0.250	0.250
0x03	0.031	0.188	0.375	0.375	0.031	0.188	0.375	0.375
0x04	0.062	0.250	0.500	0.500	0.062	0.250	0.500	0.500
0x05	0.125	0.375	0.625	0.625	0.125	0.375	0.625	0.625
0x06	0.250	0.500	0.750	0.750	0.250	0.500	0.750	0.750
0x07	0.500	0.750	0.875	0.875	0.500	0.750	0.875	0.875
0x08	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0x09	2.000	1.500	1.250	1.125	2.000	1.500	1.250	1.125
0x0A	4.000	2.000	1.500	1.250	4.000	2.000	1.500	1.250
0x0B	8.000	3.000	1.750	1.375	8.000	3.000	1.750	1.375
0x0C	16.000	4.000	2.000	1.500	16.000	4.000	2.000	1.500
0x0D	32.000	6.000	2.500	1.625	32.000	6.000	2.500	1.625
0x0E	Inf	Inf	Inf	Inf	64.000	8.000	3.000	1.750
0x0F	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 4: Value sets for K $= 4$, unsigned formats with $1 \leq$ P ≤ 4, extended and finite domains. Subnormals are shown in blue, special values in brown.

4 Operations

This section defines the *behavior* (with no constraint as to the implementation) of operations which P3109 systems may provide. Such operations include:

- conversion between P3109 formats and between P3109 formats and IEEE-754 formats;
- comparison, classification, and informational operations;
- arithmetic operations such as addition, multiplication and fused multiply-add;
- operations on blocks comprising a scale factor and sequence of values.

In the definitions of operations, certain mathematical *functions* are also defined. The definitions in this document are specifications of behavior, they are not to be interpreted as constraints on implementation.

An implementation of any of the operations defined herein is conforming if it computes the same output as does the defined operation, for all possible inputs. This may be attested by any proof method, including direct computation.

An operation is a *numeric operation* if one or more of its outputs is a floating-point value.

An operation's *defined outputs* are the (exact) output values specified in the definitions in this document.

4.1 Notation and definitions

4.1.1 Mathematical notations

$\mathsf{IsOdd}(I)$ is true if integer I is odd, false otherwise. $\mathsf{IsEven}(I)$ is true if integer I is even, false otherwise.

The number of elements in a set S is denoted $\#S$.

Integer division is denoted by $x \div y$, modulo by $x \mod y$.

Comments are introduced with an em-dash, and are right-justified —An example comment

4.1.2 The set of closed extended reals

The set of *closed extended reals* is the set $\Omega := \mathbb{R} \cup \{-\infty, \infty, \mathsf{NaN}\}$ of reals augmented with positive and negative infinity and NaN. In common with existing mathematical treatments, there is no negative zero in this set.

Functions which operate on the closed extended reals are marked with a prefixed ω, e.g. $\omega\mathsf{Add}$.

Operation specifications will, in general, convert floating-point operands to closed extended real values in order to define their behaviors. Behavior on any inputs with non-real operands is explicitly specified in this document.

4.1.3 Use of integer arithmetic

As defined above, a K-bit P3109 floating-point value (referred to in general as a *P3109 value*) is encoded by an integer in the range 0 to $2^{\mathsf{K}} - 1$. Specifications operate on such values using integer arithmetic (e.g. divide and modulo) operations. Implementations may perform these operations in any equivalent manner, for example bit shifting and masking.

4.2 Projection specifications

Operations on P3109 values are defined via conversion to closed extended real values, on which the mathematical operation is performed, before conversion back to the appropriate P3109 value set. In general, operation results will not be exact P3109 values, and hence will be *projected* into the P3109 value set via rounding and overflow handling. A *projection specification* is a pair (rounding mode, saturation mode). For a given projection specification π, these are written $\mathsf{Rnd}_\pi, \mathsf{Sat}_\pi$.

The defined rounding modes are as follows. The precise specifications of these modes are in the function ωRoundToPrecision (§4.9.2):

NearestTiesToEven	Round to nearest, ties to even
NearestTiesToAway	Round to nearest, ties away from zero
TowardPositive	Round toward positive
TowardNegative	Round toward negative
TowardZero	Round toward zero
ToOdd	Round to odd
Stochastic[ABC]	Stochastic rounding, with variants A, B, and C as defined in §4.9.2

Values are first rounded to the target precision, with exponent unbounded above. Those which are then outside the maximum value in the target format are treated according to the saturation mode.

The defined saturation modes are as follows. Their precise specifications are given in the function ωSaturate (§4.9.4):

SatFinite	All return values are clamped to the representable finite range.
SatPropagate	Finite return values are clamped to the representable finite range. Infinite return values are preserved.
OvfInf	As in IEEE Std 754-2019, out-of-range values are replaced with: the extremal finite value, positive or negative infinity, as indicated by the projection specification, and the signedness of the target format.

4.3 Scaled operations

It is known in machine learning applications to provide operation variants which apply a scale factor to one or more operands, or to an output value. For example, a scaled multiplication operation might be defined as $\mathrm{ScaledMultiply}(x, y, l) = 2^l \times (x \times y)$, where l is an integer log-scale factor. In this document, such scaled operations may be defined using block operations, as described in §5.5.

4.4 Non-requirement for operator side effects

The operator definitions herein describe no side effects, such as the setting of flags, or the triggering of interrupts, and do not return values other than the defined return value. Typically NaN is returned when input values are out of domain (e.g. $\log(-1.0)$). When saturation is specified, there is no direct mechanism to distinguish overflowed values from values which round to the format's maximum value. An implementation which has side effects will still conform to this specification providing its return value on all inputs matches the definitions herein.

4.5 Format-level operations

Certain operations are defined at the format level, rather than the value level, for example determining the precision of a format. These operations are:

Operation	Description
$\mathsf{Width}(f)$	Width in bits of format f, such that $\mathsf{Width}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}) = \mathsf{K}$.
$\mathsf{Precision}(f)$	Precision of format f, such that $\mathsf{Precision}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}) = \mathsf{P}$.
$\mathsf{Signedness}(f)$	Signedness of format f, such that $\mathsf{Signedness}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}) = \Sigma$.
$\mathsf{Domain}(f)$	Domain of format f, such that $\mathsf{Domain}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}) = \Delta$.
$\mathsf{ExponentBits}(f)$	Number of exponent bits of format f, such that $\mathsf{ExponentBits}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \mathsf{Signed}, \Delta\}) = \mathsf{K} - \mathsf{P}$ and $\mathsf{ExponentBits}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \mathsf{Unsigned}, \Delta\}) = \mathsf{K} - \mathsf{P} + 1$.
$\mathsf{TrailingBits}(f)$	Number of trailing significand bits of format f, such that $\mathsf{TrailingBits}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}) = \mathsf{P} - 1$.
$\mathsf{ExponentBias}(f)$	Exponent bias of format f such that $\mathsf{ExponentBias}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \mathsf{Signed}, \Delta\}) = 2^{\mathsf{K}-\mathsf{P}-1}$ and $\mathsf{ExponentBias}(\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \mathsf{Unsigned}, \Delta\}) = 2^{\mathsf{K}-\mathsf{P}}$.
$\mathsf{MaxFinite}(f)$	Maximum finite value representable in format f.
$\mathsf{MinFinite}(f)$	Minimum finite value representable in format f.
$\mathsf{MinPositive}(f)$	Minimum strictly positive value representable in format f.
$\mathsf{MinNormal}(f)$	Minimum positive normal value representable in format f.
$\mathsf{Rnd}(\pi)$	Rounding mode of projection specification π, such that $\mathsf{Rnd}((\mathsf{Rnd}_\pi, \mathsf{Sat}_\pi)) = \mathsf{Rnd}_\pi$.
$\mathsf{Sat}(\pi)$	Saturation mode of projection specification π, such that $\mathsf{Sat}((\mathsf{Rnd}_\pi, \mathsf{Sat}_\pi)) = \mathsf{Sat}_\pi$.

4.6 Approximate implementations

For numeric operations, in addition to an exact implementation, which must be provided, a system may additionally provide κ-*approximate* implementations. Such an implementation shall compute values whose maximum difference from the defined outputs, over all operands producing finite outputs, does not exceed κ value steps, defined as follows.

A numeric operation a, for a given set of parameters, has defined (exact) output $\hat{a}(x)$ for operands x. A κ-approximate implementation produces a floating-point value $\tilde{a}(x)$, which for some operands x has $\tilde{a}(x) \neq \hat{a}(x)$. For all operands producing NaN, -Inf, or Inf, a κ-approximate implementation shall produce the same value.

Let the set of operands producing finite outputs be I, so that for all $x \in I$ we have $\hat{a}(x) \in V$, where V is the output format's finite value set. For all $x \in I$, $\tilde{a}(x)$ shall be in V.

The value of κ will be the maximum over all operands $x \in I$ of the number of values in V between $\tilde{a}(x)$ and $\hat{a}(x)$ inclusive of the former, exclusive of the latter. Formally,

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

The value of κ will in general be specific to each operation and parameters, for example a system may supply implementations of $\mathsf{Add}_{f_x, f_y, f_r, \pi}$ where κ depends on f_r. These implementations shall be named differently. For each

κ-approximate implementation of an (operator, parameters) pair, κ shall be specified. Such a specification may be over I or over a covering union of disjoint subsets of I. This may be attested by any proof method, including direct computation.

4.7 Operation variants and superset specifications

Operation definitions in this document are *parameterized*. Example parameters include input and result formats, and projection specifications. The set of *variants* so defined for a given operation is called a *superset specification*.

Conforming implementations that use operation names defined by this document shall specify precisely the variants provided.

Operation names defined by this document shall be used only for variants which exactly follow the specifications in this document.

For example, an implementation might declare as follows:

> "This implementation conforms to P3109, providing the following operation variants:
>
> - $\mathsf{Add}_{f_x,f_y,f_r,\pi}$, with $\{f_x, f_y\} \subset \{\mathsf{Binary8p3s}\delta, \mathsf{Binary8p4s}\delta\}$ and $f_r \in \{\mathsf{Binary8p3se}, \mathsf{Binary8p4se}\}$
> - $\mathsf{BlockAdd}_{B,f_s,f_x,f_s,f_y,f_s,f_r,\pi}$, with $B \in \{16, 32\}$ and $f_s \in \{\mathsf{Binary8p1uf}, \mathsf{Binary8p4uf}\}$ and $f_x \in \{\mathsf{Binary6p3sf}, \mathsf{Binary4p2sf}\}$ and $f_y \in \{\mathsf{Binary6p3sf}, \mathsf{Binary4p2sf}\}$ and $f_r \in \{\mathsf{Binary8p3se}, \mathsf{Binary8p4se}\}$
> - $\mathsf{FMA}_{f_x,f_y,f_z,f_r,\pi}$, with $f_x \in \{\mathsf{Binary8p3se}, \mathsf{Binary8p4se}\}$ and $f_y = f_x$, and $f_z \in \{\mathsf{Binary8p3se}, \mathsf{Binary8p4se}\}$, and $f_r = f_z$, with $\mathsf{Precision}(f_z) > \mathsf{Precision}(f_x)$.
> - $\mathsf{ConvertFromIEEE754}_{\phi,f,\pi}$, with ϕ in {Binary16, Binary32} and f in {Binary8p3se, Binary8p4se, Binary8p5se}
>
> In all cases, projection modes π obey $\mathsf{Rnd}_\pi \in \{\mathsf{NearestTiesToEven},\ \mathsf{NearestTiesToAway}\}$ and $\mathsf{Sat}_\pi \in \{\mathsf{SatFinite}, \mathsf{OvfInf}\}$".
> In addition, ConvertFromIEEE754 implements $\mathsf{Sat}_\pi \in \{\mathsf{StochasticA}_R \mid 0 \leq R < 2^{23}\}$.

This example is truncated—realistic compliance declarations might run to many pages.

Such declarations may be compressed by defining common *profiles*, outside of the scope of this current document.

When referring to operations and formats in machine-readable formats, subscripts may be replaced with braces, for example:

```
Add{fx,fy,fr,pi} with {fx,fy} in {
   Binary{8,3,Signed,$Delta}, Binary{8,4,Signed,$Delta}
}
and fr in {
   Binary{8,3,Signed,Extended}, Binary{8,4,Signed,Extended}
}
for $Delta in {Finite, Extended}
```

4.8 Operator definition template

Operations are defined according to the following template:

Signature

$\mathsf{Operator}_{f_x,f_y,f_r}(x, y) \rightarrow r$ —Naming the operator, its parameters, operands and result.

Parameters —Parameters specify a family of related operations

f_x : input format
f_y : input format
f_r : result format

Operands

x : P3109 value, in format f_x
y : P3109 value, in format f_y

Result

r : result value —Result value

Behavior —An ordered sequence of pattern-matching declarations.

$\omega\mathsf{Operator}(X, Y) \rightarrow$ ... —Auxiliary definition, in the closed extended reals.

$\mathsf{Operator}(\mathsf{NaN}, 0) \rightarrow \mathsf{NaN}$ —Exact pattern: only the provided operands match.
$\mathsf{Operator}(x \in \{\text{-}\mathsf{Inf}, \mathsf{Inf}\}, y) \rightarrow x$ —Match for all x values in the given set.
$\mathsf{Operator}(*, 0) \rightarrow 0$ —The $*$ symbol matches any value.
$\mathsf{Operator}(x, y) \rightarrow \mathsf{Operator}_{f_x,f_y,f_r}(x, y)$ —Operators in pattern RHS have explicit parameters.

Notes

Notes on the operation.

In a sequence of pattern-matching declarations, the first matching pattern in the order presented in this document defines the behavior for a given operand sequence.

In pattern matching, certain shorthand notations are used, as follows. Consider an operation which takes a P3109 value x in format f_x, and returns a P3109 value in format f_y. One matching pattern might be $\mathsf{Operator}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$, where the input and result NaN values are in different formats, although this is not explicitly marked. A more explicit presentation, such as $\mathsf{Operator}(\mathsf{NaN}_{f_x}) \rightarrow \mathsf{NaN}_{f_y}$ was considered to increase difficulty of comprehension without a compensatory reduction in ambiguity.

Similarly, parameter subscripts are generally elided on the left-hand-side of patterns where they are common to all, but are generally written explicitly on the right.

Ancillary information, typically parameters, may include information such as an operand's format or rounding behaviors. An implementation may choose to provide that information using any appropriate mechanism. For example, available mechanisms in a hardware implementation might include passing the information in a hardware register, or as additional bits in an operation's opcode.

An auxiliary operation may be defined which operates on the closed extended reals. Such operations are by convention named by the prefix ω. Any such definition uses only real arithmetic, explicitly handing any cases which consume or produce infinities or NaN. Auxiliary operations are in general verified via formal specifications[10, 11].

4.9 Functions

This section defines mathematical functions which are referenced in the later definitions of operations, but which themselves will not be provided in a conforming implementation.

4.9.1 Decode

Signature

$\omega\mathsf{Decode}_f(x) \to X$
$\omega\mathsf{DecodeAux}_{\mathsf{K},\mathsf{P},\mathsf{B},\sigma,\Delta}(x) \to X$

Parameters

f : format, width K_f, precision P_f, bias B_f, signedness σ_f, domain Δ_f

K : width
P : precision
B : exponent bias
σ : signedness in $\{\mathsf{Signed}, \mathsf{Unsigned}\}$
Δ : domain in $\{\mathsf{Extended}, \mathsf{Finite}\}$

Operands

x : P3109 value, in format f

Result

X : closed extended real value

Behavior

$\omega\mathsf{Decode}(x) = \omega\mathsf{DecodeAux}_{\mathsf{K}_f,\mathsf{P}_f,\mathsf{B}_f,\sigma_f,\Delta_f}(x)$

$\omega\mathsf{DecodeAux}(2^{\mathsf{K}-1}\quad) \textbf{ if } \sigma = \mathsf{Signed} \to \mathsf{NaN}$
$\omega\mathsf{DecodeAux}(2^{\mathsf{K}}\quad - 1) \textbf{ if } \sigma = \mathsf{Unsigned} \to \mathsf{NaN}$

$\omega\mathsf{DecodeAux}(2^{\mathsf{K}-1} - 1) \textbf{ if } \Delta = \mathsf{Extended} \textbf{ and } \sigma = \mathsf{Signed} \to +\infty$
$\omega\mathsf{DecodeAux}(2^{\mathsf{K}} - 1) \textbf{ if } \Delta = \mathsf{Extended} \textbf{ and } \sigma = \mathsf{Signed} \to -\infty$
$\omega\mathsf{DecodeAux}(2^{\mathsf{K}} - 2) \textbf{ if } \Delta = \mathsf{Extended} \textbf{ and } \sigma = \mathsf{Unsigned} \to \infty$

$\omega\mathsf{DecodeAux}(2^{\mathsf{K}-1} < x < 2^{\mathsf{K}}) \textbf{ if } \sigma = \mathsf{Signed} \to -\omega\mathsf{DecodeAux}(x - 2^{\mathsf{K}-1})$

$\omega\mathsf{DecodeAux}(x) \to X$
where

$$X = \begin{cases} (0 + T \times 2^{1-\mathsf{P}}) \times 2^{1-\mathsf{B}} & \textbf{if } E = 0 \quad \text{—Subnormal} \\ (1 + T \times 2^{1-\mathsf{P}}) \times 2^{E-\mathsf{B}} & \textbf{otherwise} \quad \text{—Normal} \end{cases}$$

$T = x \mod 2^{\mathsf{P}-1}$
$E = x \div 2^{\mathsf{P}-1}$

Note

The Finite cases are all handled in the final clause, so it does not need to be mentioned explicitly in pattern matching.

4.9.2 Project

Project closed extended real value to P3109 format f, applying specified rounding and saturation.

Signature

$\omega\mathsf{Project}_{f,\pi}(X) \to x$

Parameters

f : target format,
precision P_f, domain Δ_f, signedness σ_f, exponent bias b_f, minimum/maximum finite value $M_f^{\mathrm{lo}}/M_f^{\mathrm{hi}}$

π : projection specification: rounding mode Rnd_π, saturation Sat_π

Operands

X : closed extended real value

Result

x : P3109 value, format f

Behavior

$\omega\mathsf{Project}(\mathsf{NaN}) \to \mathsf{NaN}$

$\omega\mathsf{Project}(X) \to x$

$$\begin{aligned}&\textbf{where}\\ &\quad R = \omega\mathsf{RoundToPrecision}_{\mathsf{P}_f,b_f,\mathsf{Rnd}_\pi}(X)\\ &\quad S = \omega\mathsf{Saturate}_{M_f^{\mathrm{lo}},M_f^{\mathrm{hi}}}(\mathsf{Sat}_\pi,\mathsf{Rnd}_\pi,R,\sigma_f,\Delta_f)\\ &\quad x = \omega\mathsf{Encode}_f(S)\end{aligned}$$

Notes

It is an error if $\mathsf{Sat}_\pi \neq \mathsf{SatFinite}$ if $\Delta_f = \mathsf{Finite}$.

In ωSaturate, all values above M^{hi} are sent to either M^{hi} or ∞. Nevertheless, ωProject has the property that values between M^{hi} and $M^{\mathrm{hi}} + \frac{1}{2}\,\mathrm{ulp}(M^{\mathrm{hi}})$ will project to M^{hi} because ωRoundToPrecision precedes saturation.

Under NearestTiesToEven rounding, subnormals are handled following IEEE Std 754-2019, for example, values strictly between the smallest subnormal and its half are rounded to the smallest subnormal, while positive values below or equal to half of the smallest subnormal are rounded to zero.

4.9.3 RoundToPrecision

Convert real value to real value representable with a given precision and unbounded exponent (unbounded above, bounded below by $2 - \mathsf{P} - \mathsf{B}$).

Signature

$\omega\mathsf{RoundToPrecision}_{\mathsf{P},\mathsf{B},\mathsf{Rnd}}(X) \rightarrow Z$

Parameters

P : integer precision
B : exponent bias
Rnd : rounding mode. For stochastic rounding, random bits R are to be supplied as noted below.

Operands

X : real value

Result

Z : real value, of the form $N \times 2^E$, where $N \in \mathbb{Z}$, $0 \leq |N| < 2^{\mathsf{P}}$, and $2 - \mathsf{P} - \mathsf{B} \leq E$.

Behavior

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

where

$E = \max(\lfloor \log_2(|X|) \rfloor, 1 - \mathsf{B}) - \mathsf{P} + 1$ —Subnormals handled by $\max(\cdot, 1 - \mathsf{B})$

$S = |X| \times 2^{-E}$ —Real-valued significand, to be rounded to integer

$$I = \begin{cases} \lfloor S \rfloor + 1 & \textbf{if } \mathsf{RoundAway}(\mathsf{Rnd}) \\ \lfloor S \rfloor & \textbf{otherwise} \end{cases}$$

$Z = \mathrm{sign}(X) \times I \times 2^E$

and, for $\Delta = S - \lfloor S \rfloor$:

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

and

$$\mathsf{CodeIsEven} = \begin{cases} \mathsf{IsEven}(\lfloor S \rfloor) & \textbf{if } \mathsf{P} > 1 \\ (\lfloor S \rfloor = 0) \textbf{ or } \mathsf{IsEven}(E + \mathsf{B}) & \textbf{if } \mathsf{P} = 1 \end{cases}$$

[continued on next page]

$$\mathsf{RNITE}(X) = \begin{cases} \lfloor X \rfloor & \textbf{if } (X < \lfloor X \rfloor + 0.5) \textbf{ or } (X = \lfloor X \rfloor + 0.5 \textbf{ and } \mathsf{IsEven}(\lfloor X \rfloor)) \\ \lfloor X \rfloor + 1 & \textbf{otherwise} \end{cases}$$

Notes

In stochastic rounding, random bits R are supplied as an unsigned integer in the range $0 \leq R < 2^{\#R}$, hence $\#R$ is the number of random bits. The quality of the random bits is not specified in this document.

The intermediate value I may be set to 2^{P}, which might appear to preclude its representation in $\mathsf{P} - 1$ bits of explicit significand, but the computed real value is then $(1 + 0) \times 2^{E+1+\mathsf{P}}$, representable as the first number in the next binade.

One or more variants of StochasticA, StochasticB, StochasticC may be supplied. Roughly, these variants are sorted in order of decreasing bias and increasing implementation cost [12].

4.9.4 Saturate

Saturate closed extended real to $\pm\infty$, or to maximum finite value, according to projection specification parameters Sat, Rnd.

Signature

$\omega\mathsf{Saturate}_{M^{\mathrm{lo}},M^{\mathrm{hi}}}(\mathsf{Sat}, \mathsf{Rnd}, X, \sigma, \Delta) \rightarrow Z$

Parameters

M^{hi} : real maximum value
M^{lo} : real minimum value

Operands

Sat : saturation mode
Rnd : rounding mode
X : closed extended real value
σ : signedness in {Signed, Unsigned}
Δ : domain {Finite, Extended}

Result

Z : closed extended real value

Behavior

$\omega\mathsf{Saturate}(*, *, \mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Saturate}(*, *, X, *, *)$ **if** $M^{lo} \leq X$ **and** $X \leq M^{hi} \rightarrow X$
$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, +\infty, *, *) \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, -\infty, *, *) \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, X, *, *)$ **if** $X \leq M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatFinite}, *, X, *, *)$ **if** $X \geq M^{hi} \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, +\infty, *, \mathsf{Extended}) \rightarrow +\infty$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, +\infty, *, *) \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, -\infty, \mathsf{Signed}, \mathsf{Extended}) \rightarrow -\infty$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, -\infty, *, *) \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, X, *, *)$ **if** $X \leq M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{SatPropagate}, *, X, *, *)$ **if** $X \geq M^{hi} \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, +\infty, *, \mathsf{Extended}) \rightarrow +\infty$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, +\infty, *, *) \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, -\infty, \mathsf{Signed}, \mathsf{Extended}) \rightarrow -\infty$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, -\infty, *, *) \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, \mathsf{TowardZero} \vee \mathsf{TowardPositive}, X, *, *)$ **if** $X \leq M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, \mathsf{TowardZero} \vee \mathsf{TowardNegative}, X, *, *)$ **if** $X \geq M^{hi} \rightarrow M^{hi}$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, X, \mathsf{Signed}, \mathsf{Extended})$ **if** $X \leq M^{lo} \rightarrow -\infty$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, X, *, *)$ **if** $X \leq M^{lo} \rightarrow M^{lo}$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, X, *, \mathsf{Extended})$ **if** $X \geq M^{hi} \rightarrow +\infty$
$\omega\mathsf{Saturate}(\mathsf{OvfInf}, *, X, *, *)$ **if** $X \geq M^{hi} \rightarrow M^{hi}$

4.9.5 Encode

Encode closed extended real value to P3109 format f. ωEncode must be applied only to a value which is in the value set of format f. Such a value may be produced by, for example ωRoundToPrecision and ωSaturate, or the input may be known to be in the value set, for example, the absolute value of a value already in the set.

Signature

$\omega\mathsf{Encode}_f(X) \to r$

Parameters

f : target format, width K_f, precision P_f, signedness σ_f, exponent bias b_f

Operands

X : closed extended real value, in the value set of format f

Result

r : P3109 value, format f

Behavior

$$\omega\mathsf{Encode}(\mathsf{NaN}) \to \begin{cases} 2^{\mathsf{K}_f-1} & \textbf{if } \sigma_f = \mathsf{Signed} \\ 2^{\mathsf{K}_f}-1 & \textbf{if } \sigma_f = \mathsf{Unsigned} \end{cases}$$

$$\omega\mathsf{Encode}(\infty) \to \begin{cases} 2^{\mathsf{K}_f-1}-1 & \textbf{if } \sigma_f = \mathsf{Signed} \\ 2^{\mathsf{K}_f}-2 & \textbf{if } \sigma_f = \mathsf{Unsigned} \end{cases}$$

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

$$\omega\mathsf{Encode}(0) \to 0$$

$$\omega\mathsf{Encode}(X > 0) \to r$$

where

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

and

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

$$S = X \times 2^{-E} \times 2^{\mathsf{P}_f-1}$$ —S is the significand

$$T = S \mod 2^{\mathsf{P}_f-1}$$

Note

Because of the precondition that X is in the value set of format f, it follows that $S \in \mathbb{N}$, and that $X \neq \infty$ if $\Delta_f = \mathsf{Finite}$.

Similarly, ωEncode is not called with a NaN operand by any operation in this specification, but may be called directly.

4.10 Interconverting formats

Convert a P3109 value to another P3109 format.

Signature

$\mathsf{Convert}_{f_x, f_r, \pi}(x) \to r$

Parameters

f_x : input format
f_r : result format
π : projection specification

Operands

x : P3109 value, format f_x

Result

r : P3109 value, format f_r

Behavior

$\mathsf{Convert}(\mathsf{NaN}_{f_x}) \to \mathsf{NaN}_{f_r}$
$\mathsf{Convert}(x) \to \omega\mathsf{Project}_{f_r, \pi}(\omega\mathsf{Decode}_{f_x}(x))$

4.11 Arithmetic operations

Arithmetic operations which take one or more P3109 values as arguments, and return one or more P3109 values are defined in this section. Arithmetic operations which mix IEEE Std 754 and P3109 values are described in §6.

4.11.1 Absolute Value, Negation

Signature

$\mathsf{Abs}_{f_x,f_r}(x) \to r$
$\mathsf{Negate}_{f_x,f_r}(x) \to r$

Parameters

f_x : input format
f_r : result format
π : projection specification

Operands

x : P3109 value, format f_x

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Abs}(\mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Abs}(-\infty) \to +\infty$
$\omega\mathsf{Abs}(+\infty) \to +\infty$
$\omega\mathsf{Abs}(X) \to |X|$

$\mathsf{Abs}(x) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Abs}(\omega\mathsf{Decode}_{f_x}(x)))$

$\omega\mathsf{Negate}(\mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Negate}(-\infty) \to +\infty$
$\omega\mathsf{Negate}(+\infty) \to -\infty$
$\omega\mathsf{Negate}(X) \to -X$

$\mathsf{Negate}(x) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Negate}(\omega\mathsf{Decode}_{f_x}(x)))$

4.11.2 CopySign

Copies the sign of a P3109 value onto another P3109 value.

Signature

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

Parameters

f_x : format of x
f_y : format of y
f_r : format of result r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{CopySign}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{CopySign}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{CopySign}(\pm\infty, +\infty) \rightarrow +\infty$
$\omega\mathsf{CopySign}(\pm\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{CopySign}(\pm\infty, Y)$ **if** $Y \geq 0 \rightarrow +\infty$
$\omega\mathsf{CopySign}(\pm\infty, Y)$ **if** $Y < 0 \rightarrow -\infty$
$\omega\mathsf{CopySign}(X, +\infty) \rightarrow |X|$
$\omega\mathsf{CopySign}(X, -\infty) \rightarrow -|X|$
$\omega\mathsf{CopySign}(X, Y)$ **if** $Y \geq 0 \rightarrow |X|$
$\omega\mathsf{CopySign}(X, Y)$ **if** $Y < 0 \rightarrow -|X|$

$\mathsf{CopySign}(X, Y) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{CopySign}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

4.11.3 Addition, Subtraction

These operations take two P3109 values as input, and return a P3109 value. The definitions allow for all input and output formats to differ, a given implementation may supply any subset of the defined operations.

Signature

$\mathsf{Add}_{f_x,f_y,f_r,\pi}(x,y) \to r$
$\mathsf{Subtract}_{f_x,f_y,f_r,\pi}(x,y) \to r$

Parameters

f_x : format of x
f_y : format of y
f_r : format of r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Add}(\mathsf{NaN},*) \to \mathsf{NaN}$
$\omega\mathsf{Add}(*,\mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Add}(+\infty,-\infty) \to \mathsf{NaN}$
$\omega\mathsf{Add}(-\infty,+\infty) \to \mathsf{NaN}$
$\omega\mathsf{Add}(+\infty,*) \to +\infty$
$\omega\mathsf{Add}(*,+\infty) \to +\infty$
$\omega\mathsf{Add}(-\infty,*) \to -\infty$
$\omega\mathsf{Add}(*,-\infty) \to -\infty$
$\omega\mathsf{Add}(X,Y) \to X+Y$

$\mathsf{Add}(x,y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Add}(\omega\mathsf{Decode}_{f_x}(x),\omega\mathsf{Decode}_{f_y}(y)))$

$\omega\mathsf{Subtract}(\mathsf{NaN},*) \to \mathsf{NaN}$
$\omega\mathsf{Subtract}(*,\mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Subtract}(+\infty,+\infty) \to \mathsf{NaN}$
$\omega\mathsf{Subtract}(-\infty,-\infty) \to \mathsf{NaN}$
$\omega\mathsf{Subtract}(*,+\infty) \to -\infty$
$\omega\mathsf{Subtract}(+\infty,*) \to +\infty$
$\omega\mathsf{Subtract}(*,-\infty) \to +\infty$
$\omega\mathsf{Subtract}(-\infty,*) \to -\infty$
$\omega\mathsf{Subtract}(X,Y) \to X-Y$

$\mathsf{Subtract}(x,y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Subtract}(\omega\mathsf{Decode}_{f_x}(x),\omega\mathsf{Decode}_{f_y}(y)))$

4.11.4 Multiplication

This operation takes two P3109 values as input, and return a P3109 value. The definition allows for all input and result formats to differ, a given implementation may supply any subset of the defined operations.

Signature

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

Parameters

f_x : format of x
f_y : format of y
f_r : format of r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Multiply}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\omega\mathsf{Multiply}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Multiply}(+\infty, +\infty) \to +\infty$
$\omega\mathsf{Multiply}(-\infty, -\infty) \to +\infty$
$\omega\mathsf{Multiply}(-\infty, +\infty) \to -\infty$
$\omega\mathsf{Multiply}(+\infty, -\infty) \to -\infty$
$\omega\mathsf{Multiply}(+\infty, Y)$ **if** $Y > 0 \to +\infty$
$\omega\mathsf{Multiply}(+\infty, Y)$ **if** $Y = 0 \to \mathsf{NaN}$
$\omega\mathsf{Multiply}(+\infty, Y)$ **if** $Y < 0 \to -\infty$
$\omega\mathsf{Multiply}(X, +\infty)$ **if** $X > 0 \to +\infty$
$\omega\mathsf{Multiply}(X, +\infty)$ **if** $X = 0 \to \mathsf{NaN}$
$\omega\mathsf{Multiply}(X, +\infty)$ **if** $X < 0 \to -\infty$
$\omega\mathsf{Multiply}(-\infty, Y)$ **if** $Y > 0 \to -\infty$
$\omega\mathsf{Multiply}(-\infty, Y)$ **if** $Y = 0 \to \mathsf{NaN}$
$\omega\mathsf{Multiply}(-\infty, Y)$ **if** $Y < 0 \to +\infty$
$\omega\mathsf{Multiply}(X, -\infty)$ **if** $X > 0 \to -\infty$
$\omega\mathsf{Multiply}(X, -\infty)$ **if** $X = 0 \to \mathsf{NaN}$
$\omega\mathsf{Multiply}(X, -\infty)$ **if** $X < 0 \to +\infty$
$\omega\mathsf{Multiply}(X, Y) \to X \times Y$

$\mathsf{Multiply}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Multiply}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

4.11.5 Division

This operation takes two P3109 values as input, and returns a P3109 value. The definition allows for all input and result formats to differ, a given implementation may supply any subset of the defined operations.

Divide($x, \pm\mathsf{Inf}$) where x is finite yields 0.
Divide($x, 0$) yields NaN. It would be inconsistent to return Inf for finite x, as this would imply $1/(1/\text{-}\mathsf{Inf}) \rightarrow \mathsf{Inf}$.

Signature

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

Parameters

f_x : format of x
f_y : format of y
f_r : format of r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Divide}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(\pm\infty, \pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(*, Y)$ **if** $Y = 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Divide}(+\infty, Y)$ **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** $Y < 0 \rightarrow +\infty$
$\omega\mathsf{Divide}(*, \pm\infty) \rightarrow 0$
$\omega\mathsf{Divide}(X, Y) \rightarrow X/Y$

$\mathsf{Divide}(x, y) \rightarrow \omega\mathsf{Project}_{f_r, \pi}(\omega\mathsf{Divide}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

4.11.6 Fused Multiply-Add

Compute $R = X \times Y + Z$, with computation in the reals.
Rounding and the determination of overflow and underflow are applied only on the result.

Signature

$\mathsf{FMA}_{f_x,f_y,f_z,f_r,\pi}(x, y, z) \rightarrow r$

Parameters

f_x : format of x
f_y : format of y
f_z : format of z
f_r : format of r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y
z : P3109 value, format f_z

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{FMA}(\mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(*, \mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(*, *, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(0, \pm\infty, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(\pm\infty, 0, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(X, +\infty, +\infty)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(X, -\infty, +\infty)$ **if** $X > 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, Y, +\infty)$ **if** $Y < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(-\infty, Y, +\infty)$ **if** $Y > 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(X, -\infty, -\infty)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(X, +\infty, -\infty)$ **if** $X > 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(-\infty, Y, -\infty)$ **if** $Y < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, Y, -\infty)$ **if** $Y > 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(-\infty, +\infty, +\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, -\infty, +\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, +\infty, -\infty) \rightarrow \mathsf{NaN}$

$\omega\mathsf{FMA}(-\infty, -\infty, -\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, +\infty, *) \rightarrow +\infty$
$\omega\mathsf{FMA}(-\infty, -\infty, *) \rightarrow +\infty$
$\omega\mathsf{FMA}(+\infty, -\infty, *) \rightarrow -\infty$
$\omega\mathsf{FMA}(-\infty, +\infty, *) \rightarrow -\infty$
$\omega\mathsf{FMA}(*, *, -\infty) \rightarrow -\infty$
$\omega\mathsf{FMA}(X, -\infty, Z)$ **if** $X \geq 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(X, -\infty, Z)$ **if** $X < 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(*, *, +\infty) \rightarrow +\infty$
$\omega\mathsf{FMA}(X, +\infty, Z)$ **if** $X \geq 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(X, +\infty, Z)$ **if** $X < 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(+\infty, Y, Z)$ **if** $Y \geq 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(+\infty, Y, Z)$ **if** $Y < 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(-\infty, Y, Z)$ **if** $Y \geq 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(-\infty, Y, Z)$ **if** $Y < 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(X, Y, Z) \rightarrow X \times Y + Z$

$\mathsf{FMA}(x, y, z) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{FMA}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y), \omega\mathsf{Decode}_{f_z}(z)))$

4.11.7 Fused Add-Add

Compute $R = X + Y + Z$, with computation in the reals.
Rounding and the determination of overflow and underflow are applied only on the result.

Signature

$\mathsf{FAA}_{f_x,f_y,f_z,f_r,\pi}(x, y, z) \rightarrow r$

Parameters

f_x : format of x
f_y : format of y
f_z : format of z
f_r : format of r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y
z : P3109 value, format f_z

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{FAA}(\mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(*, \mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(*, *, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(+\infty, -\infty, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(-\infty, +\infty, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(+\infty, *, -\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(-\infty, *, +\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(*, +\infty, -\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(*, -\infty, +\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FAA}(+\infty, +\infty, +\infty) \rightarrow +\infty$
$\omega\mathsf{FAA}(-\infty, -\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{FAA}(*, +\infty, +\infty) \rightarrow +\infty$

$\omega\mathsf{FAA}(+\infty, *, +\infty) \rightarrow +\infty$
$\omega\mathsf{FAA}(+\infty, +\infty, *) \rightarrow +\infty$
$\omega\mathsf{FAA}(*, -\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{FAA}(-\infty, *, -\infty) \rightarrow -\infty$
$\omega\mathsf{FAA}(-\infty, -\infty, *) \rightarrow -\infty$
$\omega\mathsf{FAA}(+\infty, *, *) \rightarrow +\infty$
$\omega\mathsf{FAA}(*, +\infty, *) \rightarrow +\infty$
$\omega\mathsf{FAA}(*, *, +\infty) \rightarrow +\infty$
$\omega\mathsf{FAA}(-\infty, *, *) \rightarrow -\infty$
$\omega\mathsf{FAA}(*, -\infty, *) \rightarrow -\infty$
$\omega\mathsf{FAA}(*, *, -\infty) \rightarrow -\infty$
$\omega\mathsf{FAA}(X, Y, Z) \rightarrow X + Y + Z$

$\mathsf{FAA}(x, y, z) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{FAA}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y), \omega\mathsf{Decode}_{f_z}(z)))$

4.11.8 Square root, Logarithm, Exponentiation, Reciprocal

Signature

$\mathsf{Sqrt}_{f_x,f_r,\pi}(x) \rightarrow r$
$\mathsf{RSqrt}_{f_x,f_r,\pi}(x) \rightarrow r$
$\mathsf{Exp}_{f_x,f_r,\pi}(x) \rightarrow r$
$\mathsf{Log}_{f_x,f_r,\pi}(x) \rightarrow r$
$\mathsf{Exp2}_{f_x,f_r,\pi}(x) \rightarrow r$
$\mathsf{Log2}_{f_x,f_r,\pi}(x) \rightarrow r$
$\mathsf{Recip}_{f_x,f_r,\pi}(x) \rightarrow r$

Parameters

f_x : input format
f_r : result format
π : projection specification

Operands

x : P3109 value, format f_x

Result

r : P3109 value, format f_r

Behavior

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

$\omega\mathsf{RSqrt}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{RSqrt}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{RSqrt}(X)$ **if** $X \leq 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{RSqrt}(+\infty) \rightarrow 0$
$\omega\mathsf{RSqrt}(X) \rightarrow 1/\sqrt{X}$
$\mathsf{RSqrt}(x) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{RSqrt}(\omega\mathsf{Decode}_{f_x}(x)))$

$\omega\mathsf{Exp}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Exp}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Exp}(-\infty) \rightarrow 0$
$\omega\mathsf{Exp}(X) \rightarrow e^X$
$\mathsf{Exp}(x) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Exp}(\omega\mathsf{Decode}_{f_x}(x)))$

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

$\omega\mathsf{Exp2}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Exp2}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Exp2}(-\infty) \rightarrow 0$
$\omega\mathsf{Exp2}(X) \rightarrow 2^X$
$\mathsf{Exp2}(x) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Exp2}(\omega\mathsf{Decode}_{f_x}(x)))$

$\omega\mathsf{Log2}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log2}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log2}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Log2}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log2}(X)$ **if** $X = 0 \rightarrow -\infty$
$\omega\mathsf{Log2}(X) \rightarrow \log_2 X$
$\mathsf{Log2}(x) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Log2}(\omega\mathsf{Decode}_{f_x}(x)))$

$\omega\mathsf{Recip}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Recip}(X)$ **if** $X = 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Recip}(\pm\infty) \rightarrow 0$
$\omega\mathsf{Recip}(X) \rightarrow 1/X$
$\mathsf{Recip}(x) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Recip}(\omega\mathsf{Decode}_{f_x}(x)))$

4.11.9 Hypotenuse

Signature

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

Parameters

f_x : input format of x
f_y : input format of y
f_r : result format
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Hypot}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Hypot}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Hypot}(*, \pm\infty) \rightarrow +\infty$
$\omega\mathsf{Hypot}(\pm\infty, *) \rightarrow +\infty$
$\omega\mathsf{Hypot}(X, Y) \rightarrow \sqrt{|X|^2 + |Y|^2}$

$\mathsf{Hypot}(x) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Hypot}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

4.12 Extrema

4.12.1 Minimum, Maximum, MinimumNumber, and MaximumNumber

Minimum and maximum, with or without propagation of NaN.

Signature

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

Parameters

f_x : input format
f_y : input format
f_r : result format
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Minimum}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\omega\mathsf{Minimum}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Minimum}(+\infty, +\infty) \to +\infty$
$\omega\mathsf{Minimum}(-\infty, -\infty) \to -\infty$
$\omega\mathsf{Minimum}(+\infty, -\infty) \to -\infty$
$\omega\mathsf{Minimum}(-\infty, +\infty) \to -\infty$
$\omega\mathsf{Minimum}(+\infty, Y) \to Y$
$\omega\mathsf{Minimum}(X, +\infty) \to X$
$\omega\mathsf{Minimum}(-\infty, *) \to -\infty$
$\omega\mathsf{Minimum}(*, -\infty) \to -\infty$
$\omega\mathsf{Minimum}(X, Y) \to \textbf{ if } X < Y \textbf{ then } X \textbf{ else } Y$

$\omega\mathsf{Maximum}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\omega\mathsf{Maximum}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{Maximum}(+\infty, +\infty) \to +\infty$
$\omega\mathsf{Maximum}(-\infty, -\infty) \to -\infty$
$\omega\mathsf{Maximum}(+\infty, -\infty) \to +\infty$
$\omega\mathsf{Maximum}(-\infty, +\infty) \to +\infty$
$\omega\mathsf{Maximum}(+\infty, *) \to +\infty$
$\omega\mathsf{Maximum}(*, +\infty) \to +\infty$
$\omega\mathsf{Maximum}(-\infty, Y) \to Y$
$\omega\mathsf{Maximum}(X, -\infty) \to X$
$\omega\mathsf{Maximum}(X, Y) \to \textbf{ if } X < Y \textbf{ then } Y \textbf{ else } X$

$\mathsf{Minimum}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Minimum}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

$\mathsf{Maximum}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Maximum}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

$\omega\mathsf{MinimumNumber}(\mathsf{NaN}, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{MinimumNumber}(X, \mathsf{NaN}) \to X$
$\omega\mathsf{MinimumNumber}(\mathsf{NaN}, Y) \to Y$
$\omega\mathsf{MinimumNumber}(X, Y) \to \omega\mathsf{Minimum}(X, Y)$

$\omega\mathsf{MaximumNumber}(\mathsf{NaN}, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{MaximumNumber}(X, \mathsf{NaN}) \to X$
$\omega\mathsf{MaximumNumber}(\mathsf{NaN}, Y) \to Y$
$\omega\mathsf{MaximumNumber}(X, Y) \to \omega\mathsf{Maximum}(X, Y)$

$\mathsf{MinimumNumber}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MinimumNumber}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

$\mathsf{MaximumNumber}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MaximumNumber}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

4.12.2 MinimumMagnitude, MaximumMagnitude, and 'Number' variants

Minimum and maximum by magnitude, with or without propagation of NaN.

Signature

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

Parameters

f_x : format of x
f_y : format of y
f_r : format of result r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{MinimumMagnitude}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\omega\mathsf{MinimumMagnitude}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{MinimumMagnitude}(+\infty, +\infty) \to +\infty$
$\omega\mathsf{MinimumMagnitude}(-\infty, -\infty) \to -\infty$
$\omega\mathsf{MinimumMagnitude}(+\infty, -\infty) \to -\infty$
$\omega\mathsf{MinimumMagnitude}(-\infty, +\infty) \to -\infty$
$\omega\mathsf{MinimumMagnitude}(\pm\infty, Y) \to Y$
$\omega\mathsf{MinimumMagnitude}(X, \pm\infty) \to X$
$\omega\mathsf{MinimumMagnitude}(X, Y)$ **if** $|X| < |Y| \to X$
$\omega\mathsf{MinimumMagnitude}(X, Y)$ **if** $|X| > |Y| \to Y$
$\omega\mathsf{MinimumMagnitude}(X, Y)$ **if** $|X| = |Y| \to$
if $X < Y$ **then** X **else** Y

$\omega\mathsf{MaximumMagnitude}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\omega\mathsf{MaximumMagnitude}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\omega\mathsf{MaximumMagnitude}(+\infty, *) \to +\infty$
$\omega\mathsf{MaximumMagnitude}(*, +\infty) \to +\infty$
$\omega\mathsf{MaximumMagnitude}(-\infty, *) \to -\infty$
$\omega\mathsf{MaximumMagnitude}(*, -\infty) \to -\infty$
$\omega\mathsf{MaximumMagnitude}(X, Y)$ **if** $|X| > |Y| \to X$
$\omega\mathsf{MaximumMagnitude}(X, Y)$ **if** $|X| < |Y| \to Y$
$\omega\mathsf{MaximumMagnitude}(X, Y)$ **if** $|X| = |Y| \to$
if $X < Y$ **then** Y **else** X

$$\mathsf{MinimumMagnitude}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MinimumMagnitude}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$$

$$\mathsf{MaximumMagnitude}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MaximumMagnitude}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$$

$\omega\mathsf{MinimumMagnitudeNumber}(x, \mathsf{NaN}) \to x$
$\omega\mathsf{MinimumMagnitudeNumber}(\mathsf{NaN}, y) \to y$
$\omega\mathsf{MinimumMagnitudeNumber}(x, y) \to \omega\mathsf{MinimumMagnitude}(x, y)$

$$\mathsf{MinimumMagnitudeNumber}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MinimumMagnitudeNumber}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$$

$\omega\mathsf{MaximumMagnitudeNumber}(x, \mathsf{NaN}) \to x$
$\omega\mathsf{MaximumMagnitudeNumber}(\mathsf{NaN}, y) \to y$
$\omega\mathsf{MaximumMagnitudeNumber}(x, y) \to \omega\mathsf{MaximumMagnitude}(x, y)$

$$\mathsf{MaximumMagnitudeNumber}(x, y) \to \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MaximumMagnitudeNumber}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$$

4.12.3 MinimumFinite, MaximumFinite

Minimum and maximum finite value.

Signature

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

Parameters

f_x : format of x
f_y : format of y
f_r : format of result r
π : projection specification

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{MinimumFinite}(\mathsf{NaN}, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{MinimumFinite}(\mathsf{NaN}, y) \rightarrow y$
$\omega\mathsf{MinimumFinite}(x, \mathsf{NaN}) \rightarrow x$
$\omega\mathsf{MinimumFinite}(+\infty, +\infty) \rightarrow +\infty$
$\omega\mathsf{MinimumFinite}(-\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{MinimumFinite}(+\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{MinimumFinite}(-\infty, +\infty) \rightarrow -\infty$
$\omega\mathsf{MinimumFinite}(+\infty, Y) \rightarrow Y$
$\omega\mathsf{MinimumFinite}(X, +\infty) \rightarrow X$
$\omega\mathsf{MinimumFinite}(-\infty, Y) \rightarrow Y$
$\omega\mathsf{MinimumFinite}(X, -\infty) \rightarrow X$
$\omega\mathsf{MinimumFinite}(X, Y) \rightarrow$ **if** $X < Y$ **then** X **else** Y

$\omega\mathsf{MaximumFinite}(\mathsf{NaN}, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{MaximumFinite}(\mathsf{NaN}, y) \rightarrow y$
$\omega\mathsf{MaximumFinite}(x, \mathsf{NaN}) \rightarrow x$
$\omega\mathsf{MaximumFinite}(+\infty, +\infty) \rightarrow +\infty$
$\omega\mathsf{MaximumFinite}(-\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{MaximumFinite}(+\infty, -\infty) \rightarrow +\infty$
$\omega\mathsf{MaximumFinite}(-\infty, +\infty) \rightarrow +\infty$
$\omega\mathsf{MaximumFinite}(+\infty, Y) \rightarrow Y$
$\omega\mathsf{MaximumFinite}(X, +\infty) \rightarrow X$
$\omega\mathsf{MaximumFinite}(-\infty, Y) \rightarrow Y$
$\omega\mathsf{MaximumFinite}(X, -\infty) \rightarrow X$
$\omega\mathsf{MaximumFinite}(X, Y) \rightarrow$ **if** $X < Y$ **then** Y **else** X

$\mathsf{MinimumFinite}(x, y) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MinimumFinite}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

$\mathsf{MaximumFinite}(x, y) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{MaximumFinite}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y)))$

4.12.4 Clamp

Perform a clamp (also known as clip) operation.

Signature

$\mathsf{Clamp}_{f_x,f_{lo},f_{hi},f_r,\pi}(x, lo, hi) \rightarrow r$

Parameters

f_x : format of x
f_{lo} : format of lower bound lo
f_{hi} : format of upper bound hi
f_r : format of result r
π : projection specification

Operands

x : P3109 value, format f_x
lo : P3109 value, format f_{lo}
hi : P3109 value, format f_{hi}

Result

r : P3109 value, format f_r

Behavior

$\omega\mathsf{Clamp}(\mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Clamp}(*, \mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Clamp}(*, *, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Clamp}(*, Lo, Hi) \ \textbf{if} \ Lo > Hi \rightarrow \mathsf{NaN}$
$\omega\mathsf{Clamp}(*, +\infty, +\infty) \rightarrow +\infty$
$\omega\mathsf{Clamp}(*, -\infty, -\infty) \rightarrow -\infty$
$\omega\mathsf{Clamp}(*, *, -\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Clamp}(*, +\infty, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Clamp}(+\infty, *, +\infty) \rightarrow +\infty$
$\omega\mathsf{Clamp}(+\infty, *, Hi) \rightarrow Hi$
$\omega\mathsf{Clamp}(-\infty, -\infty, *) \rightarrow -\infty$
$\omega\mathsf{Clamp}(-\infty, Lo, *) \rightarrow Lo$
$\omega\mathsf{Clamp}(X, -\infty, +\infty) \rightarrow X$
$\omega\mathsf{Clamp}(X, -\infty, Hi) \ \textbf{if} \ X \leq Hi \rightarrow X$
$\omega\mathsf{Clamp}(X, -\infty, Hi) \ \textbf{if} \ X > Hi \rightarrow Hi$
$\omega\mathsf{Clamp}(X, Lo, +\infty) \ \textbf{if} \ X \leq Lo \rightarrow Lo$
$\omega\mathsf{Clamp}(X, Lo, +\infty) \ \textbf{if} \ X > Lo \rightarrow X$
$\omega\mathsf{Clamp}(X, Lo, Hi) \rightarrow \ \textbf{if} \ X \leq Lo \ \textbf{then} \ Lo \ \textbf{else} \ \textbf{if} \ X \geq Hi \ \textbf{then} \ Hi \ \textbf{else} \ X$

$\mathsf{Clamp}(x, lo, hi) \rightarrow \omega\mathsf{Project}_{f_r,\pi}(\omega\mathsf{Clamp}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_{lo}}(lo), \omega\mathsf{Decode}_{f_{hi}}(hi)))$

4.13 Comparisons

Comparison operators take two P3109 operands and return a Boolean value. Any NaN operand yields the result False.

Signature

$\mathsf{Compare}Op_{f_x,f_y}(x, y) \rightarrow b$

Parameters

f_x : format of x
f_y : format of y

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

b : Boolean value

Behavior

$\omega\mathsf{CompareLess}(\mathsf{NaN}, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLess}(*, \mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLess}(+\infty, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLess}(*, +\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareLess}(-\infty, -\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLess}(-\infty, *) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareLess}(*, -\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLess}(X, Y) \rightarrow X < Y$

$\mathsf{CompareLess}(x, y) \rightarrow \omega\mathsf{CompareLess}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y))$

$\omega\mathsf{CompareLessEqual}(\mathsf{NaN}, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLessEqual}(*, \mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLessEqual}(*, +\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareLessEqual}(-\infty, *) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareLessEqual}(+\infty, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLessEqual}(*, -\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareLessEqual}(X, Y) \rightarrow X \leq Y$

$\mathsf{CompareLessEqual}(x, y) \rightarrow \omega\mathsf{CompareLessEqual}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y))$

[Comparisons continued on next page]

$\omega\mathsf{CompareEqual}(\mathsf{NaN}, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareEqual}(*, \mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareEqual}(+\infty, +\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareEqual}(-\infty, -\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareEqual}(+\infty, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareEqual}(-\infty, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareEqual}(*, -\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareEqual}(*, +\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareEqual}(X, Y) \rightarrow X = Y$

$\mathsf{CompareEqual}(x, y) \rightarrow \omega\mathsf{CompareEqual}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y))$

$\omega\mathsf{CompareGreaterEqual}(\mathsf{NaN}, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreaterEqual}(*, \mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreaterEqual}(+\infty, *) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareGreaterEqual}(-\infty, -\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareGreaterEqual}(-\infty, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreaterEqual}(*, +\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreaterEqual}(*, -\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareGreaterEqual}(X, Y) \rightarrow X \geq Y$

$\mathsf{CompareGreaterEqual}(x, y) \rightarrow \omega\mathsf{CompareGreaterEqual}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y))$

$\omega\mathsf{CompareGreater}(\mathsf{NaN}, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreater}(*, \mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreater}(*, +\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreater}(-\infty, -\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreater}(*, -\infty) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareGreater}(+\infty, *) \rightarrow \mathsf{True}$
$\omega\mathsf{CompareGreater}(-\infty, *) \rightarrow \mathsf{False}$
$\omega\mathsf{CompareGreater}(X, Y) \rightarrow X > Y$

$\mathsf{CompareGreater}(x, y) \rightarrow \omega\mathsf{CompareGreater}(\omega\mathsf{Decode}_{f_x}(x), \omega\mathsf{Decode}_{f_y}(y))$

4.14 Predicates and classification

Conforming implementations shall provide the classification predicates and the classifier operation defined below.

The classification operations comprise: 1) a set of predicate functions with a Boolean return value, taking a single P3109 value as input; 2) a classifier operation $\mathsf{Class}(x)$ that returns a single value of enumeration type, describing the input value's properties.

Signature

$\mathsf{Is}Class_f(x) \to b$

Parameters

f : format of x

Operands

x : P3109 value, format f

Result

b : Boolean value

Behavior

$\omega\mathsf{IsZero}(\mathsf{NaN}) \to \mathsf{False}$
$\omega\mathsf{IsZero}(\pm\infty) \to \mathsf{False}$
$\omega\mathsf{IsZero}(X) \to X = 0$
$\mathsf{IsZero}(x) \to \omega\mathsf{IsZero}(\omega\mathsf{Decode}_f(x))$

$\omega\mathsf{IsOne}(\mathsf{NaN}) \to \mathsf{False}$
$\omega\mathsf{IsOne}(\pm\infty) \to \mathsf{False}$
$\omega\mathsf{IsOne}(X) \to X = 1$
$\mathsf{IsOne}(x) \to \omega\mathsf{IsOne}(\omega\mathsf{Decode}_f(x))$

$\mathsf{IsNaN}(\mathsf{NaN}) \to \mathsf{True}$
$\mathsf{IsNaN}(x) \to \mathsf{False}$

$\mathsf{IsFinite}(x) \to x \notin \{\text{-}\mathsf{Inf}, \mathsf{Inf}, \mathsf{NaN}\}$

$\mathsf{IsInfinite}(x) \to x \in \{\text{-}\mathsf{Inf}, \mathsf{Inf}\}$

$\omega\mathsf{IsSignMinus}(\mathsf{NaN}) \to \mathsf{False}$
$\omega\mathsf{IsSignMinus}(+\infty) \to \mathsf{False}$
$\omega\mathsf{IsSignMinus}(-\infty) \to \mathsf{True}$
$\omega\mathsf{IsSignMinus}(X) \to X < 0$

$\mathsf{IsSignMinus}(x) \to \omega\mathsf{IsSignMinus}(\omega\mathsf{Decode}_f(x))$

$\mathsf{IsNormal}(x \in \{0, \text{-}\mathsf{Inf}, \mathsf{Inf}, \mathsf{NaN}\}) \to \mathsf{False}$

$$\mathsf{IsNormal}(x) \to \begin{cases} (x \mod 2^{\mathsf{K}_f - 1}) \div 2^{\mathsf{P}_f - 1} > 0 & \textbf{if } \sigma_f = \mathsf{Signed} \\ x \div 2^{\mathsf{P}_f - 1} > 0 & \textbf{if } \sigma_f = \mathsf{Unsigned} \end{cases}$$

$\mathsf{IsSubnormal}(x \in \{0, \text{-}\mathsf{Inf}, \mathsf{Inf}, \mathsf{NaN}\}) \to \mathsf{False}$
$\mathsf{IsSubnormal}(x) \to \textbf{not } \mathsf{IsNormal}(x)$

4.14.1 Classifier operation

The Classifier operation Class(x) tells which of the eight classes x falls into as defined by Table 5.

Signature

$\mathsf{Class}_f(x) \rightarrow c$

Parameters

f : format of x

Operands

x : P3109 value, format f

Result

c : enumeration

Behavior

$\mathsf{Class}(x) \rightarrow \mathrm{ClassEnum}$

Table 5: Classifier operation

ClassEnum	*Condition*
ClsNaN	IsNaN(x)
ClsNegativeInfinity	IsInfinite(x) **and** IsSignMinus(x)
ClsNegativeNormal	IsNormal(x) **and** IsSignMinus(x)
ClsNegativeSubnormal	IsSubnormal(x) **and** IsSignMinus(x)
ClsZero	IsZero(x)
ClsPositiveSubnormal	IsSubnormal(x) **and not** IsSignMinus(x)
ClsPositiveNormal	IsNormal(x) **and not** IsSignMinus(x)
ClsPositiveInfinity	IsInfinite(x) **and not** IsSignMinus(x)

4.14.2 Total order predicate

The $\mathsf{TotalOrder}(x, y)$ predicate provides a total ordering over each P3109 format's value set.

Signature

$\mathsf{TotalOrder}_{f_x, f_y}(x, y) \to b$

Operands

f_x : format of x
f_y : format of y

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y

Result

b : Boolean value

Behavior

$\mathsf{TotalOrder}(\mathsf{NaN}, x) \to \mathsf{True}$
$\mathsf{TotalOrder}(x, \mathsf{NaN}) \to \mathsf{False}$
$\mathsf{TotalOrder}(x, y) \to \mathsf{CompareLessEqual}_{f_x, f_y}(x, y)$

Note

The above definition is consistent with the IEEE Std 754-2019 definition of TotalOrder for signed extended formats. In particular, among P3109 formats, there is a single NaN and it always compares as the most negative value.

4.14.3 Comparison predicates

This section defines the mapping between comparison operator symbols that a system may make available with P3109 values as operands.

Table 6: Comparison predicates and negations

Math symbol	Predicate
$X = Y$	$\mathsf{CompareEqual}(X, Y)$
$X > Y$	$\mathsf{CompareGreater}(X, Y)$
$X \geq Y,\ X{>=}Y$	$\mathsf{CompareGreaterEqual}(X, Y)$
$X < Y$	$\mathsf{CompareLess}(X, Y)$
$X \leq Y,\ X{<=}Y$	$\mathsf{CompareLessEqual}(X, Y)$
$X \neq Y,\ X\,!{=}Y$	$\mathbf{not}\ \mathsf{CompareEqual}(X, Y)$

5 Block operations

Operations on blocks (sequences) of P3109 values. A block is a pair $(s, [x_1, ..., x_B])$ comprising scale factor s in P3109 format f_s and a sequence of one or more elements x_i, each in P3109 format f_x.

These operations make use of the function reduce, defined as follows, for binary function f, and values $x_1, \ldots, x_n$:

$$\begin{aligned} \mathsf{reduce}(f, [x_1, x_2]) &= f(x_1, x_2) \\ \mathsf{reduce}(f, [x_1, \ldots, x_n]) &= \mathsf{reduce}(f, [f(x_1, x_2), x_3, \ldots, x_n]) \quad \text{for } n \geq 3 \end{aligned}$$

Scaled operations are available as block operations using blocks of one element (§5.5).

5.1 Decode and Project Blocks

The functions in this section are used in the definitions of block operations, and are not operations exposed by this standard.

5.1.1 BlockDecode

Signature

$\omega\mathsf{BlockDecode}_{B,f_s,f_x}(s, [x_1, ..., x_B]) \rightarrow [Z_1, ..., Z_B]$

Parameters

B : block size
f_s : format of scale factor s
f_x : format of elements x

Operands

s : P3109 scale, in format f_s
$\mathbf{x}$: sequence of P3109 values in format f_x

Result

$\mathbf{Z}$: sequence of closed extended real values

Behavior

$\omega\mathsf{BlockDecode}(s, [x_1, ..., x_B]) \rightarrow [Z_1, ..., Z_B]$
where
$Z_i = \omega\mathsf{Multiply}(\omega\mathsf{Decode}_{f_s}(s), \omega\mathsf{Decode}_{f_x}(x_i))$

5.1.2 BlockProject

Convert a sequence of closed extended reals $X_{1..B}$ to a block $(s, [r_{1..B}])$, with scale factor s supplied as an operand.

Signature

$\omega\mathsf{BlockProject}_{B,f_s,f_r,\pi_r}(s, [X_1, ..., X_B]) \rightarrow [r_1, ..., r_B]$

Parameters

B : block size
f_s : format of output scale factor s
f_r : format of output elements r
π_r : projection specification for elements

Operands

s : output scale factor in format f_s
$\mathbf{X}$: sequence of closed extended reals

Result

$\mathbf{r}$: sequence of values in format f_r

Behavior

$$\omega\mathsf{BlockProject}(s, [X_1, ..., X_B]) = [r_1, ..., r_B]$$

where

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

$$Z_i = \begin{cases} \mathsf{NaN} & \textbf{if } S \text{ is } \mathsf{NaN} \textbf{ or } X_i \text{ is } \mathsf{NaN} \\ 0 & \textbf{if } S = 0 \\ 1 & \textbf{if } S = \pm\infty \\ \omega\mathsf{Divide}(X_i, S) & \textbf{otherwise} \end{cases}$$

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

Notes

If the scale factor s is zero, all output elements shall be zero.
If the scale factor s is infinite, all non-NaN output elements shall be 1.

5.2 Convert Blocks

5.2.1 ConvertFromBlock

Convert a block $(s, [x_{1..B}])$ to a sequence of values $r_{1..B}$.

Signature

$\mathsf{ConvertFromBlock}_{B,f_s,f_x,f_r,\pi_r}(s, [x_1, ..., x_B]) \rightarrow [r_1, ..., r_B]$

Parameters

B : block size
f_x : format of inputs x
f_s : format of input scale factor s
f_r : format of output elements r
π_r : projection specification for elements

Operands

s : input scale factor in format f_s
$\mathbf{x}$: sequence of values in format f_x

Result

$\mathbf{r}$: sequence of values in format f_r

Behavior

$$\mathsf{ConvertFromBlock}(s, [x_1, ..., x_B]) = [r_1, ..., r_B]$$
$$\textbf{where}$$
$$[Z_1, ..., Z_B] = \omega\mathsf{BlockDecode}(s, [x_1, ..., x_B])$$
$$r_i = \omega\mathsf{Project}_{f_r,\pi_r}(Z_i)$$

5.2.2 ConvertToBlock

Convert a sequence of values $x_{1..B}$ to a block $(s, [r_{1..B}])$, with scale factor supplied as an operand.

Signature

$\mathsf{ConvertToBlock}_{B,f_s,f_x,f_r,\pi_r}([x_1, ..., x_B], s) \rightarrow (s, [r_1, ..., r_B])$

Parameters

B : block size
f_x : format of inputs x
f_s : format of output scale factor s
f_r : format of output elements r
π_r : projection specification for elements

Operands

$\mathbf{x}$: sequence of values in format f_x
s : output scale factor in format f_s

Result

s : output scale factor in format f_s
$\mathbf{r}$: sequence of values in format f_r

Behavior

$$\mathsf{ConvertToBlock}([x_1, ..., x_B], s) = (s, [r_1, ..., r_B])$$
$$\textbf{where}$$
$$X_i = \omega\mathsf{Decode}_{f_x}(x_i)$$
$$[r_1, ..., r_B] = \omega\mathsf{BlockProject}_{B,f_s,f_r,\pi_r}(s, [X_1, ..., X_B])$$

5.2.3 ConvertToBlockMaxAbsFinite

Convert a sequence of values $x_{1..B}$ to a block $(s, [r_{1..B}])$, computing scale factor as a maximum over absolute values of x_i.

Signature

$\mathsf{ConvertToBlockMaxAbsFinite}_{B,f_x,f_s,f_r,\pi_s,\pi_r}([x_1, ..., x_B]) \rightarrow (s, [r_1, ..., r_B])$

Parameters

B : block size
f_x : format of inputs x
f_s : format of output scale factor s
f_r : format of output elements r
π_s : projection specification for scale factor
π_r : projection specification for elements

Operands

$\mathbf{x}$: sequence of values in format f_x

Result

s : output scale factor in format f_s
$\mathbf{r}$: sequence of values in format f_r

Behavior

$$\mathsf{ConvertToBlockMaxAbsFinite}([x_1, ..., x_B]) = (s, [r_1, ..., r_B])$$
$$\textbf{where}$$
$$X_i = \omega\mathsf{Decode}_{f_x}(x_i)$$
$$M_i = \omega\mathsf{Abs}(X_i)$$
$$S = \mathsf{reduce}(\omega\mathsf{MaximumFinite}, [0, M_1, ..., M_B])$$
$$s = \omega\mathsf{Project}_{f_s,\pi_s}(S)$$
$$[r_1, ..., r_B] = \omega\mathsf{BlockProject}_{B,f_s,f_r,\pi_r}(s, [X_1, ..., X_B])$$

Notes

If all x_i are NaN, the output scale and elements will be NaN.

If all x_i are infinite, the output scale will be Inf and the elements will be 1.

If some x_i are infinite, they are preserved (or saturated, according to π_r) in the output, and do not influence the scale of finite elements.

If the maximum value S rounds to zero under π_s, the output scale will be zero, and all output elements will be zero.

The projection specifications for the scale factor and elements allows implementation of some common strategies, e.g. rounding s using TowardPositive and saturating r.

5.3 Block reduction operations

5.3.1 Sum and product

Signature

$\mathsf{BlockReduceAdd}_{B,\ f_s,f_x,\ f_r,\pi_r}((s,[x_1,...,x_B])) \rightarrow r$
$\mathsf{BlockReduceMultiply}_{B,\ f_s,f_x,\ f_r,\pi_r}((s,[x_1,...,x_B])) \rightarrow r$

Parameters

B : block size
f_s : format of input scale factor s
f_x : format of input elements x

f_r : format of result r
π_r : projection specification

Operands

s : scale in format f_s
$\mathbf{x}$: sequence of values in format f_x

Result

r : value in format f_r

Behavior

$\mathsf{BlockReduceAdd}((s_x,[x_1,...,x_B])) = r$
where
$[X_1,...,X_B] = \omega\mathsf{BlockDecode}_{B,f_{sx},f_x}(s_x,[x_1,...,x_B])$
$R = \mathsf{reduce}(\omega\mathsf{Add},[0,X_1,...,X_B])$
$r = \omega\mathsf{Project}_{f_r,\pi_r}(R)$

$\mathsf{BlockReduceMultiply}((s_x,[x_1,...,x_B])) = r$
where
$[X_1,...,X_B] = \omega\mathsf{BlockDecode}_{B,f_{sx},f_x}(s_x,[x_1,...,x_B])$
$R = \mathsf{reduce}(\omega\mathsf{Multiply},[1,X_1,...,X_B])$
$r = \omega\mathsf{Project}_{f_r,\pi_r}(R)$

Notes

While accumulation order is left-to-right in this definition, an implementation may choose any order yielding the same result, noting that this standard defines only exact results. A κ-approximate implementation may similarly choose any accumulation order. The way partial sums and products are developed can reduce internal overflow and underflow.

5.3.2 Dot product

Dot product of two blocks of P3109 values.

Signature

$\mathsf{BlockDotProduct}_{B,\ f_{sx},f_x,\ f_{sy},f_y,\ f_r,\pi_r}((s_x,[x_1,...,x_B]),(s_y,[y_1,...,y_B])) \to r$

Parameters

B : block size
f_{sx} : format of first input scale factor s_x
f_x : format of first input elements x
f_{sy} : format of second input scale factor s_y
f_y : format of second input elements y

f_r : format of result r
π_r : projection specification

Operands

s_x : scale in format f_{sx}
$\mathbf{x}$: sequence of values in format f_x
s_y : scale in format f_{sy}
$\mathbf{y}$: sequence of values in format f_y

Result

r : value in format f_r

Behavior

$$\mathsf{BlockDotProduct}((s_x,[x_1,...,x_B]),(s_y,[y_1,...,y_B])) = r$$
$$\textbf{where}$$
$$[X_1,...,X_B] = \omega\mathsf{BlockDecode}_{B,f_{sx},f_x}(s_x,[x_1,...,x_B])$$
$$[Y_1,...,Y_B] = \omega\mathsf{BlockDecode}_{B,f_{sy},f_y}(s_y,[y_1,...,y_B])$$
$$P_i = \omega\mathsf{Multiply}(X_i,Y_i)$$
$$R = \mathsf{reduce}(\omega\mathsf{Add},[0,P_1,...,P_B])$$
$$r = \omega\mathsf{Project}_{f_r,\pi_r}(R)$$

Notes

While accumulation order is left-to-right in this definition, an implementation may choose any order yielding the same result, noting that this standard defines only exact results. A κ-approximate implementation may similarly choose any accumulation order. The way partial sums and products are developed can reduce internal overflow and underflow.

5.4 Elementwise operations on blocks

Operations on blocks may be declared according to this template, where BlockOp is defined in terms of $\omega\,Op$.

Note that the scale factor of the output block s_r is supplied as an *input* operand. Implementations are free to supply any method of computation of this scale factor, and free to supply an operation in which that computation is composed with the elementwise operation; this standard does not specify that computation.

Permitted substitutions for Op are:

- Unary ($n = 1$): Negate, Abs, Exp, Sqrt, RSqrt, Exp2, Log2, Log, Recip;
- Binary ($n = 2$): CopySign, Add, Subtract, Multiply, Divide, Hypot,
  Minimum, MinimumNumber, MinimumMagnitude, MinimumMagnitudeNumber,
  Maximum, MaximumNumber, MaximumMagnitude, MaximumMagnitudeNumber;
- Ternary ($n = 3$): FMA, FAA.

Signature

$$\mathsf{Block}Op_{B,\ (f_{s1},f_{x1}),\ldots,\ (f_{sn},f_{xn}),\ f_s,f_r,\pi_r}((s_1,[x_{11},...,x_{1B}]),...,(s_n,[x_{n1},...,x_{nB}]),s_r) \rightarrow\ (s_r,[r_1,...,r_B])$$

Parameters

B : block size
f_{s1} : format of first input scale factor s_1
f_{x1} : format of first input elements x_{1i}
... : ...
f_{sn} : format of n^{th} input scale factor s_n
f_{xn} : format of n^{th} input elements x_{ni}

f_s : format of scale factor of output s_r
f_r : format of output elements r
π_r : projection specification for output elements

Operands

s_1 : scale, in format f_{s1}
$\mathbf{x}_1$: sequence of values in format f_{x1}
... :
s_n : scale, in format f_{sn}
$\mathbf{x}_n$: sequence of values in format f_{xn}

s_r : scale of output, in format f_s

Result

s_r : scale of output, copied from input
$\mathbf{r}$: sequence of values in format f_r

Behavior

$$\mathsf{Block}Op((s_1,[x_{11},...,x_{1B}]),...,(s_n,[x_{n1},...,x_{nB}]),s_r) = (s_r,[r_1,...,r_B])$$

where

$$[X_{11},...,X_{1B}] = \omega\mathsf{BlockDecode}_{B,f_{s1},f_{x1}}(s_1,[x_{11},...,x_{1B}])$$
$$...$$
$$[X_{n1},...,X_{nB}] = \omega\mathsf{BlockDecode}_{B,f_{sn},f_{xn}}(s_n,[x_{n1},...,x_{nB}])$$
$$Z_i = \omega\,Op(X_{1i},...,X_{ni}) \quad \forall i \in \{1,...,B\}$$
$$[r_1,...,r_B] = \omega\mathsf{BlockProject}_{B,f_s,f_r,\pi_r}(s_r,[Z_1,...,Z_B])$$

5.5 Scaled operations via block operations

It is observed that a block size of 1 is permitted, yielding scaled binary or ternary operations. As an example, a binary scaled operation is defined as follows:

$$\begin{aligned}&\mathsf{Scaled}\,Op(s_1, x_1, s_2, x_2) = r\\ &\qquad\qquad\textbf{where}\\ &\qquad\qquad\quad [1, r] = \mathsf{Block}\,Op((s_1, [x_1]), (s_2, [x_2]), 1)\end{aligned}$$

where the two-operand $\mathsf{Block}\,Op$ expands to:

$$\begin{aligned}&\mathsf{Block}\,Op((s_1, [x_1]), (s_2, [x_2]), 1) = (s_r, [r])\\ &\qquad\qquad\textbf{where}\\ &\qquad\qquad\quad X_1 = \omega\mathsf{BlockDecode}_{B,f_{s1},f_{x1}}(s_1, [x_1])\\ &\qquad\qquad\quad X_2 = \omega\mathsf{BlockDecode}_{B,f_{s2},f_{x2}}(s_2, [x_2])\\ &\qquad\qquad\quad Z = \omega\,Op(X_1, X_2)\\ &\qquad\qquad\quad r = \omega\mathsf{BlockProject}_{B,f_s,f_r,\pi_r}(1, Z)\end{aligned}$$

An implementation may choose to provide direct implementations of scaled operations, and may, for example, choose a $\mathsf{P} = 1$ format for the scale factors to restrict scale factors to powers of two.

6 Interconverting formats

This section describes operations which take a mix of IEEE Std 754 and P3109 operands and return IEEE Std 754 values. An implementation is free to define additional fused operations in which P3109 operands are upconverted to IEEE Std 754 before operating.

This section addresses operations that cannot be expressed as a fused sequence of operations. For instance, it is not possible to upconvert all values from Binary8p1se to Binary16. Moreover, the operations described here may include saturation and rounding modes not available in definitions based solely on upconversion.

Like the operations defined previously, these are superset specifications. An implementation might provide any subset of the parameterized set of operations, accompanied by a statement specifying which subset is implemented. An example might be

$$\mathsf{ConvertFromIEEE754}\{\mathsf{Binary16}, \mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \mathsf{Signed}, \mathsf{Extended}\}, (\mathsf{NearestTiesToEven}, \mathsf{OvfInf})\}$$
$$\text{for } (\mathsf{K}, \mathsf{P}) \in \{(8,4), (8,3), (6,3), (4,3)\}.$$

In specifying IEEE Std 754 formats, the symbol ϕ is used, from which format parameters are extracted as needed:

M_ϕ^{hi} : maximum finite value (e.g. 65504.0 for Binary16)
P_ϕ : precision (e.g. 11 for Binary16)
B_ϕ : exponent bias (e.g. 15 for Binary16)

We will make use of the functions AsClosedExtendedReal and Encode754, defined as follows:

$\mathsf{AsClosedExtendedReal}_\phi$ is a function which converts an encoded finite IEEE Std 754 value to a value in the closed extended reals. There is no negative zero in P3109 or the closed extended reals, so $\mathsf{AsClosedExtendedReal}(\text{-}0) \to 0$.

Signature

$\mathsf{Encode754}_\phi(X) \to r$

Parameters

ϕ : target IEEE Std 754 format

Operands

X : closed extended real value, in the value set of format ϕ

Result

r : IEEE Std 754 value, format ϕ

Behavior

$\mathsf{Encode754}(\mathsf{NaN}) \to$ Any quiet IEEE Std 754 NaN
$\mathsf{Encode754}(X) \to$ The code in ϕ that decodes to X

Notes

In this document, the Encode754 function is only called with arguments in the value set of format ϕ, therefore encoding is unambiguous and independent of rounding mode. Conversely, it is an error in this document if this condition is not guaranteed.

An implementation may return any quiet NaN. It is recommended that the quiet NaN with zero payload is returned.

6.1 Conversion from IEEE Std 754 formats to P3109

Convert a value in an IEEE Std 754 format to the corresponding value in a given P3109 format, considering rounding and saturation.

Signature

$\mathsf{ConvertFromIEEE754}_{\phi,f,\pi}(X) \rightarrow r$

Parameters

ϕ : source IEEE Std 754 format
f : target format
π : projection specification

Operands

X : IEEE-754 value, format ϕ

Result

r : P3109 value, format f

Behavior

$\mathsf{ConvertFromIEEE754}(\text{Any IEEE Std 754 NaN}) \rightarrow \mathsf{NaN}_f$
$\mathsf{ConvertFromIEEE754}(X) \rightarrow \omega\mathsf{Project}_{f,\pi}(\mathsf{AsClosedExtendedReal}_{\phi}(X))$

6.2 Conversion from P3109 to IEEE Std 754

Convert a P3109 value to a value in an IEEE Std 754 format.

Signature

$\mathsf{ConvertToIEEE754}_{f,\pi,\phi}(x) \rightarrow X$

Parameters

f : input format
π : projection specification
ϕ : IEEE Std 754 result format, precision P_ϕ, bias B_ϕ, maximum value M_ϕ^{hi}

Operands

x : P3109 value, format f

Result

X : IEEE-754 value, format ϕ

Behavior

$\mathsf{ConvertToIEEE754}(\mathsf{NaN}) \rightarrow \mathsf{Encode754}_\phi(\mathsf{NaN})$
$\mathsf{ConvertToIEEE754}(x) \rightarrow \mathsf{Encode754}_\phi(X)$
where
$Y = \omega\mathsf{Decode}_f(x)$
$R = \omega\mathsf{RoundToPrecision}_{P_\phi,B_\phi,\mathsf{Rnd}_\pi}(Y)$
$X = \omega\mathsf{Saturate}_{-M_\phi^{\text{hi}},M_\phi^{\text{hi}}}(\mathsf{Sat}_\pi, \mathsf{Rnd}_\pi, R, \mathsf{Signed}, \mathsf{Extended})$

Appendices

A Rationales

A.1 Use of infinity in computation of attention masks

This section expands the rationale in §3.6. A common use for ∞ is to create masks, for example, in Transformer models in machine learning [13].

These values, assembled in mask matrix M with values $M_{ij} \in \{0, -\infty\}$ are typically added to computed values A, in a computation such as:

$$\log\left(\sum \exp(\tau \times (A + M))\right)$$

where τ is a "temperature" or "base" parameter [14]. This calculation depends on the property $\exp(\tau \times (A_{ij} - \infty)) = 0$.

If a floating-point encoding does not provide infinity, then instead M_{ij} will be replaced by a large float (e.g. 224 is the largest finite Binary8p4 value). This is not in itself a difficulty: if all the A values are bounded (e.g. the results of a softmax operation are bounded above by 1.0), then $\exp(1.0 - 224.0)$ is an extremely small number, which will certainly round to zero. Therefore, an explicit representation of infinity is *not* needed in order for this computation to yield its desired value.

However, careful implementations do not execute the calculation as written, and instead fuse the $\log(\sum_i \exp(v_i))$ operation into a single operation $\mathsf{logsumexp}(v)$, whose implementation makes use of the identity transformation

$$\mathsf{logsumexp}(v) \to \mathsf{logsumexp}(v - \max(v)) + \max(v)$$

Without the "sticky" properties of Inf, this would produce incorrect answers.

For example, in a format where maxFinite=240 without Inf, and maxFinite=224 with Inf:

$$\mathsf{logsumexp}(t * [-224, -\infty]) \to \mathsf{logsumexp}(t * [0, -\infty])$$

while

$$\mathsf{logsumexp}(t * [-224, -240]) \to \mathsf{logsumexp}(t * [0, -16])$$

If $t = 1$ and all calculations are done in 8-bit floating-point, then the answer will be the same, because $\exp(-16) \approx 1.1 \times 10^{-7}$, which will round to zero in all precisions $\mathsf{P} > 2$; but if t is small, or calculations are done in mixed precision, as is common with 8-bit floating-point, the loss of "stickiness" will silently yield unexpected answers. It is not expected that the full calculation shall be done in 8-bit floating-point, but the subtraction of the maximum value (and computation of the maximum) might reasonably be in 8-bit floating-point.

B External Formats

This table summarizes the points of agreement and of difference between a number of existing format families, some of which have hardware implementations, and their P3109 analogs.

OCP: Open Compute Platform [15], describing hardware implementations including nVidia, Intel, and ARM.

AGQ: AMD, Graphcore, Qualcomm[16], implemented in Graphcore's C600 product, and AMD's gfx940.

TSL: Tesla Dojo Technology [17], A Guide to Tesla's Configurable floating-point Formats & Arithmetic

Format	P3109			OCP			AGQ		TSL	
Subformat	k8p3se	k8p4se	k8p1uf	E8M0	E5M2	E4M3	E5M2	E4M3	E5M2	E4M3
Special values shared	Y			N			Y		N	
Exactly one NaN	Y			Y	N		Y		Y	
Positive and negative ∞	Y		N	N	Y	N	N		N	
Include negative zero	N			N	Y		N		N	
Max exponent emax	15	7	126	127	15	8	15	7	N/A	N/A

"Special values shared" means that format families within an implementation with the same signedness and domain share the encodings of special values.

References

[1] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[2] Posit Working Group, "Standard for Posit™ arithmetic (2022)," tech. rep., posithub.org, 2022. https://posithub.org/docs/posit_standard-2.pdf.

[3] J. L. Gustafson, "Posit arithmetic," tech. rep., posithub.org, 2017. https://posithub.org/docs/Posits4.pdf.

[4] L. Hunhold, "Beating Posits at their own game: Takum arithmetic," tech. rep., arXiv math.NA, 2024. https://arxiv.org/abs/2404.18603.

[5] Y. Luo, Z. Zhang, R. Wu, H. Liu, Y. Jin, K. Zheng, M. Wang, Z. He, G. Hu, L. Chen, T. Hu, J. Wang, M. Chen, M. Dmitry, K. Vladimir, B. Maxim, Y. Hu, G. Chen, and Z. Huang, "Ascend HiFloat8 format for deep learning," tech. rep., arXiv cs.LG, 2024. https://arxiv.org/abs/2409.16626.

[6] W. Kahan, "Branch cuts for complex elementary functions or much ado about nothing's sign bit," *Institute of Mathematics and its Applications Conference*, 1987. https://people.freebsd.org/~das/kahan86branch.pdf.

[7] W. Kahan and J. W. Thomas, "Augmenting a programming language with complex arithmetic," tech. rep., EECS Department, University of California, Berkeley, 1991.

[8] Google, "Jax lax package: _float_to_int_for_sort ." https://github.com/google/jax/blob/fc5960f2b8b7a0ef74dbae4e27c5c08ff1564cff/jax/_src/lax/lax.py#L3934.

[9] R. Zhao, B. Vogel, and T. Ahmed, "Adaptive loss scaling for mixed precision training," tech. rep., arXiv cs.LG, 2019. https://arxiv.org/abs/1910.12385.

[10] C. M. Wintersteiger, "Formal verification of the IEEE P3109 standard for binary floating-point formats for machine learning," in *IEEE 32nd Symposium on Computer Arithmetic, ARITH 2025, El Paso, TX, USA, May 4-7, 2025*, IEEE, 2025.

[11] C. M. Wintersteiger, "Formal verification of the IEEE P3109 standard," 2025. https://github.com/imandra-ai/ieee-p3109.

[12] A. W. Fitzgibbon and S. Felix, "On stochastic rounding with few random bits," in *IEEE 32nd Symposium on Computer Arithmetic, ARITH 2025, El Paso, TX, USA, May 4-7, 2025*, pp. 133–140, IEEE, 2025.

[13] PyTorch authors, "Pytorch torchtext package: _t5_multi_head_attention_forward ." https://github.com/pytorch/text/blob/a933cbe5a008bc2cb61d985cf5864069194157eb/torchtext/prototype/models/t5/modules.py#L236.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ch. 6.2.2.3 Softmax Units for Multinoulli Output Distributions, pp. 180–184. MIT Press, 2016.

[15] P. Micikevicius, S. Oberman, P. Dubey, M. Cornea, A. Rodriguez, I. Bratt, R. Grisenthwaite, N. Jouppi, C. Chou, A. Huffman, M. Schulte, R. Wittig, D. Jani, and S. Deng, "OCP 8-bit floating point specification (OFP8)," tech. rep., opencompute.org, 2023. https://www.opencompute.org/documents/ocp-8-bit-floating-point-specification-ofp8-revision-1-0-2

[16] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "8-bit numerical formats for deep neural networks," tech. rep., arXiv cs.LG, 2022. https://arxiv.org/abs/2206.02915.

[17] Tesla, Inc., "Tesla Dojo Technology: A guide to Tesla's configurable floating point formats and arithmetic," 2023. https://web.archive.org/web/20230503235751/https://tesla-cdn.thron.com/static/MXMU3S_tesla-dojo-technology_1WDVZN.pdf.