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 3.2:	5 January 2026
Version 3.2.1:	12 January 2026
Version 3.2.2:	13 March 2026
Version 4.0:	26 June 2026

(compiled June 28, 2026)

DRAFT DOCUMENT

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

Participants

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

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

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

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

Contributors

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

The following members of the individual Standards Association balloting group voted on this draft. Balloters may have voted for approval, disapproval, or abstention.

[To be supplied by IEEE.]

When the IEEE SA Standards Board approved this Standard on <Date Approved>, it had the following membership:

[To be supplied by IEEE.]

Contents

1 Overview

1.1 Scope

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

1.2 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

2 Definitions and abbreviations

2.1 Definitions

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

bitwidth: the minimum number of bits required to encode a floating-point value in a given format, denoted K.

canonical form: of a nonzero finite floating-point datum X in format f, its representation as $\mathrm{sgn}(X) \times S \times 2^{1-\mathsf{P}} \times 2^{E}$ for minimal integer $E > -\mathsf{B}$, and nonnegative integer S, where $\mathsf{P} = \mathsf{PrecisionOf}(f)$ and $\mathsf{B} = \mathsf{ExponentBiasOf}(f)$. For zero, $S = 0$ and $E = 0$.

closed extended reals: set denoted $\mathbb{R}^{\omega}$, consisting of the real numbers augmented with positive infinity, negative infinity, and not-a-number: $\mathbb{R}^{\omega} := \mathbb{R} \cup \{-\infty, +\infty, \mathsf{NaN}\}$.

code point: integer in the range 0 to $2^{\mathsf{K}} - 1$ that encodes a floating-point datum occurring in a format of bitwidth K.

datum: see floating-point datum.

datum set: the subset of the closed extended reals representable in a given format f, denoted $\mathcal{D}_f$.

defined operation: operation whose signature and behavior are defined by this standard.

defined result: result of a defined operation for a given set of operands.

domain: format-defining parameter in $\{\mathsf{Finite}, \mathsf{Extended}\}$ that specifies whether the format's datum set includes infinities.

encoding: a format's unique mapping from its datum set to code points.

exponent: of a floating-point datum X expressed in canonical form as $\mathrm{sgn}(X) \times S \times 2^{1-\mathsf{P}} \times 2^{E}$, the integer E.

exponent bias: format-specific constant, denoted B, used in encoding and decoding the exponent field.

exponent field: the integer value of the biased exponent of a code point.

floating-point datum: a closed extended real value representable in a given format, an element of the datum set of the format.

floating-point value: a code point representing a floating-point datum in a given format.

format: binary floating-point representation parameterized by bitwidth, precision, signedness, and domain.

format-defining parameters: bitwidth (K), precision (P), signedness (Σ), and domain (Δ).

NaN (Not a Number): special non-numeric value used to represent undefined or unrepresentable results.

normal: finite floating-point datum whose significand S satisfies $2^{\mathsf{P}-1} \leq S < 2^{\mathsf{P}}$. A floating-point value is normal iff its datum is normal.

operand: an argument to an operation.

operation: a set of operation specializations parameterized over operation parameters, e.g., format, projection specification, and constants such as block size.

operation specialization: mapping from a tuple of operands to one or more results.

precision: the bitwidth of the trailing significand field plus one, denoted P.

projection specification: pair (rounding mode, saturation mode) that determines how a general closed extended real is projected to a datum in a format's datum set.

rounded value: a closed extended real value, the result of rounding (via $\omega\mathsf{RoundToPrecision}$, §4.7.4). If finite, it is of the form $S \times 2^E$ for integers S and E.

rounding mode: attribute of a projection specification that determines how a closed extended real value is rounded to a rounded value.

saturation mode: attribute of a projection specification that determines how rounded values outside the finite range of the target format are projected to a datum.

scale factor: multiplicative factor applied to all elements in a block.

significand: of a floating-point datum X expressed in canonical form as $\mathrm{sgn}(X) \times S \times 2^{1-\mathsf{P}} \times 2^E$, the integer S.

signed format: format that represents signed numbers and has signedness parameter Signed.

signedness: format-defining parameter in $\{\mathsf{Signed}, \mathsf{Unsigned}\}$ that specifies whether the format includes negative datums.

special value: floating-point values encoding zero, $\pm\infty$, and NaN.

subnormal: finite floating-point datum whose significand S satisfies $0 < S < 2^{\mathsf{P}-1}$. A floating-point value is subnormal iff its datum is subnormal.

trailing significand field: of a code point x in format f, the value $x \bmod 2^{\mathsf{P}-1}$ where P is the precision of the format f.

unsigned format: format that represents non-negative numbers and has signedness parameter $\mathsf{Unsigned}$.

value set: the set of floating-point values for a format.

2.2 Abbreviations

FAA	fused add–add
FMA	fused multiply–add
MLS	machine learning system
P3109	IEEE Working Group P3109 (Standard for Arithmetic Formats for Machine Learning)
IEEE-754	IEEE Std 754-2019™

2.3 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 $\#S$.

The signum function $\mathrm{sgn}(X)$ is defined as -1 for $X < 0$, 0 for $X = 0$, and $+1$ for $X > 0$.

Division of nonnegative integers is denoted by $x \div y$, modulo by $x \bmod y$.

Comments are right-justified and parenthesized. (A comment)

3 Floating-point formats

3.1 Formats

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

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

NOTE 1—This set contains a single NaN value (see Annex A.2). There is no negative zero in this set (see Annex A.3).

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

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

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

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

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

A *floating-point value* is a code point representing a floating-point datum in a given format. The code points for infinities are denoted Inf and -Inf.

NOTE 2—While a floating-point value is synonymous with a code point, the term is used in contexts where a format is associated with the value. For example, in an operation definition, a value might be declared "x: floating-point value, format f". In such contexts, the interpretation of a real constant, e.g., 2.0, as a floating-point value is to be read as $\omega\mathsf{Encode}_f(2.0)$. In cases where the format may be ambiguous, a real constant may be subscripted with its associated format, e.g., $X_f = \omega\mathsf{Encode}_f(X)$. For example, $2.0_{\mathsf{Binary8p4se}} = \omega\mathsf{Encode}_{\mathsf{Binary8p4se}}(2.0)$ represents the code point 0x48. Similarly, a format may be associated to infinity or NaN, e.g., $\mathsf{Inf}_{\mathsf{Binary8p4se}} = \infty_{\mathsf{Binary8p4se}}$ represents the code point 0x7F.

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

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

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

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

The *trailing significand* is the integer $S \bmod 2^{\mathsf{P}-1}$.

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

[1] See Annex C for rationale for $\mathsf{K} > 2$.
[2] See Annex A.4 for rationale for parameterization of domain.
[3] See Annex A.5 for rationale for this choice of exponent bias.
[4] See Annex A.6 for rationale for inclusion of subnormals.

3.2 Naming

Formats defined in this document shall be named $\mathsf{Binary}\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}$.

A shortened notation is also used to refer to specific formats: $\mathsf{Binary}\langle\varkappa\rangle\mathsf{p}\langle\psi\rangle\langle\sigma\rangle\langle\delta\rangle$ where the placeholders $\langle\varkappa\rangle$ and $\langle\psi\rangle$ are decimal representations of the bitwidth K and precision P, respectively; signedness $\langle\sigma\rangle \in \{\mathsf{s}, \mathsf{u}\}$, for signed and unsigned; and domain $\langle\delta\rangle \in \{\mathsf{e}, \mathsf{f}\}$ for extended and finite.

Examples:

- The format "Binary12p7se" is a 12-bit signed, extended format with precision 7.
- The format "Binary8p1uf" is an 8-bit unsigned, finite format with precision 1.

NOTE—The term "binary" indicates that the finite datums are integer multiples of (positive or negative) powers of two.

4 Operations

4.1 General

An *operation* is a parameterized mapping from a tuple of operands to one or more *results*. An operation is denoted by an identifier, followed by a comma-separated list of *operation parameter* names, either as subscripts or between angle brackets.

An *operation specialization* supplies values for the parameters of an operation. An operation specialization is denoted by an identifier, followed by a comma-separated list of operation parameter values, either as subscripts or between angle brackets.

Example:
Exp$<f_x, f_r, \rho>$ is an operation, and Exp<Binary8p4se, Binary8p3ue, (NearestTiesToEven, SatNone)> is a specialization of this operation.

An *auxiliary operation* may be defined which operates on the closed extended reals. Such operations are named with the prefix ω. These operations handle non-finite values explicitly and immediately, ensuring that all arithmetic and other mathematical operations are over (finite) real values.[5]

NOTE 1—Specifications operate on code points using integer arithmetic (e.g., divide and modulo) operations. These operations may be performed in any equivalent manner, for example bit shifting and masking.

NOTE 2—Operations on floating-point values are defined via conversion to the closed extended reals, on which the mathematical operation is performed, before projection into the floating-point datum set via rounding and saturation. The definitions in this document are specifications of behavior.

NOTE 3—The operation 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 result. Generally, NaN is returned when operands are out of domain (e.g., Log(-1.0)).

4.2 Projection specifications

A *projection specification* is a pair (rounding mode, saturation mode).

Rounding modes, as specified in §4.7.4, describe the mapping of closed extended real values to *rounded values* as follows:

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[A, B, C]	Stochastic rounding, with variants A, B, and C as defined in §4.7.4

Saturation modes, as specified in §4.7.5, describe the mapping of rounded values to datums as follows:

SatFinite	All rounded values are clamped to the representable finite range.
SatPropagate	Finite rounded values are clamped to the representable finite range. Infinite rounded values are preserved.
SatNone	Out-of-range rounded values are replaced with the extremal finite datum, positive or negative infinity, or NaN, as governed by the rounding direction and the signedness of the target format.

[5] Auxiliary operations are verified via formal specifications [14].

4.3 Operation definitions

4.3.1 Operation definition schema

Operations are defined according to the following schema.

Signature

$\mathsf{Operation}_{p_1,\ldots}(x_1, \ldots) \rightarrow r$

Parameters

p_1 : description of parameter 1
...

Operands

x_1 : description of operand 1
...

Result

r : description of result

Behavior

An ordered sequence of *pattern-matching declarations* (see §4.3.2)

Details

An optional section listing additional aspects of the operation's behavior.

4.3.2 Pattern-matching declarations

A pattern-matching declaration is of the form

$\mathsf{Operation}(x_1, \ldots) \rightarrow \ldots$

describing the result of the operation given operands $x_1, \ldots$.

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

The following notations facilitate concise specifications.

4.3.2.1 Exact pattern

Only the provided operands match.

Example:

$\mathsf{Operation}(\mathsf{NaN}, 0) \rightarrow \mathsf{NaN}$

4.3.2.2 Set inclusion

Match for all x values in the given set.

Example:

$\mathsf{Operation}(x \in \{\text{-}\mathsf{Inf}, \mathsf{Inf}\}, y) \rightarrow x$

4.3.2.3 Wildcard match

The $*$ symbol matches any value.

Example:

$\mathsf{Operation}(*, 0) \rightarrow 0$

4.3.2.4 Explicit parameters introduced in result expression

A reference to an operation specialization in the result expression uses explicit parameters.

Example:

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

Example:

The definition of the operation Log, taken from §4.10.9:

Signature

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

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Log}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Log}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(0) \rightarrow -\infty$
$\omega\mathsf{Log}(X) \rightarrow \log_e X$

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

4.4 Approximate implementations

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

An operation's *defined results* are the result values specified in the definitions in this document.

For numeric operations, a system that provides approximate implementations shall declare them as *κ-approximate* operations as follows.[6] A κ-approximate implementation shall compute values whose maximum difference from the defined results does not exceed κ *value steps*, as defined in this section.

A numeric operation a, for a given set of parameters, has a defined result $\hat{a}(x)$ for operands x. A κ-approximate implementation produces a floating-point value $\tilde{a}(x)$, which for some operands x has $\tilde{a}(x) \neq \hat{a}(x)$.

Where an approximate implementation does not "match on NaNs", that is where $\mathsf{IsNaN}(\hat{a}(x)) \neq \mathsf{IsNaN}(\tilde{a}(x))$ for any input x, then it shall declare $\kappa = \mathsf{NaN}$.

Define

$$\mathsf{MatchOnInfinity}(x, y) = \begin{cases} \mathsf{True} & \textbf{if } \mathsf{IsNaN}(x) \textbf{ and } \mathsf{IsNaN}(y) \\ \mathsf{True} & \textbf{if } \mathsf{IsFinite}(x) \textbf{ and } \mathsf{IsFinite}(y) \\ \mathsf{True} & \textbf{if } \mathsf{IsInfinite}(x) \textbf{ and } \mathsf{IsInfinite}(y) \textbf{ and } \mathsf{IsSignMinus}(x) = \mathsf{IsSignMinus}(y) \\ \mathsf{False} & \textbf{otherwise} \end{cases}$$

Where an approximate implementation matches on NaNs but does not match on infinities, that is where $\mathsf{MatchOnInfinity}(\hat{a}(x), \tilde{a}(x))$ is false for any input x, then it shall declare $\kappa = \infty$.

For operations which match on NaNs and infinities, the value of κ is defined as follows.

Let the set of operands producing finite results be I, so that for all $x \in I$ we have $\hat{a}(x) \in V$, where V is the result 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 specialization, for example a system may supply implementations of $\mathsf{Add}_{f_x, f_y, f_r, \rho}$ where κ depends on f_r.

For each κ-approximate operation specialization, κ shall be specified. Such a specification may be over I or over a covering union of disjoint subsets of I, written

$$\kappa \in \{ I_1 : \kappa_{I_1}, .., I_n : \kappa_{I_n} \} \quad \text{where} \quad I = \bigcup_i I_i \quad \text{and} \quad \forall i \neq j : I_i \cap I_j = \{\} \ .$$

This may be attested by any proof method, including direct computation.

Operations returning multiple floating-point values shall declare κ as follows, where $\kappa[m]$ is κ for the operation which returns the m^{th} return value, that is $\tilde{a}(x)[m]$. If any $\kappa[m]$ is NaN, $\kappa = \mathsf{NaN}$ shall be declared. Otherwise, if any $\kappa[m]$ is infinite, $\kappa = \infty$ shall be declared. If all $\kappa[m]$ are finite, $\kappa = \max_m \kappa[m]$ shall be declared.

Approximate implementations shall be named differently from the name of the operation that is approximated.

Additional accuracy declarations may be supplied.[7]

[6] See Annex D for worked examples of approximation declarations.

[7] See Annex E for examples.

4.5 Conforming implementations

An implementation shall provide the following operation specializations, where the two format sets $\mathcal{F}_4$ and $\mathcal{F}_8$ and the external format[8] set $\mathcal{F}_X$ are defined as

$$\begin{aligned} \mathcal{F}_8 &= \{\mathsf{Binary8p4se}, \mathsf{Binary8p3se}\} \\ \mathcal{F}_4 &= \{\mathsf{Binary4p2sf}\} \\ \{\} \neq \mathcal{F}_X &\subseteq \{\mathsf{binary32}, \mathsf{binary16}, \mathsf{BFloat16}\} \end{aligned}$$

with operation specializations as follows, for all f, f_1, f_2, and f_r in the indicated sets, and $\rho = (\mathsf{NearestTiesToEven}, \mathsf{SatNone})$.

Convert$<f, f_r, \rho>$	**where**	$\{f, f_r\} \subset \mathcal{F}_4 \cup \mathcal{F}_8 \cup \mathcal{F}_X$
Negate$<f, f, \rho>$	**where**	$f \in \mathcal{F}_4 \cup \mathcal{F}_8$
Abs$<f, f, \rho>$	**where**	$f \in \mathcal{F}_4 \cup \mathcal{F}_8$
Recip$<f, f_r, \rho>$	**where**	$\{f, f_r\} \subset \mathcal{F}_4 \cup \mathcal{F}_8 \cup \mathcal{F}_X$
Add$<f_1, f_2, f_r, \rho>$	**where**	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_8 \cup \mathcal{F}_X$
Subtract$<f_1, f_2, f_r, \rho>$	**where**	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_8 \cup \mathcal{F}_X$
Multiply$<f_1, f_2, f_r, \rho>$	**where**	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_8 \cup \mathcal{F}_X$
FMA$<f_1, f_2, f_r, f_r, \rho>$	**where**	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_X$
FAA$<f_1, f_2, f_r, f_r, \rho>$	**where**	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_X$
$MinmaxOp<f, f, f, \rho>$	**where**	$f \in \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$MinmaxOp \in \left\{ \begin{array}{ll} \mathsf{Minimum}, & \mathsf{Maximum}, \\ \mathsf{MinimumNumber}, & \mathsf{MaximumNumber}, \\ \mathsf{MinimumMagnitude}, & \mathsf{MaximumMagnitude}, \\ \mathsf{MinimumMagnitudeNumber}, & \mathsf{MaximumMagnitudeNumber}, \\ \mathsf{MinimumFinite}, & \mathsf{MaximumFinite} \end{array} \right\}$

For $CompareOp \in \{\mathsf{CompareLess}, \mathsf{CompareLessEqual}, \mathsf{CompareEqual}, \mathsf{CompareGreater}, \mathsf{CompareGreaterEqual}\}$ the following operation specializations shall be provided:

$CompareOp<f, f>$ **where** $f \in \mathcal{F}_4 \cup \mathcal{F}_8$.

For each format in $\mathcal{F}_4 \cup \mathcal{F}_8$, the following operation specializations shall be provided:

IsZero, IsOne, IsNaN, IsInfinite, IsFinite, IsSignMinus, IsNormal, IsSubnormal, NextGreaterThan, NextLessThan.

For each format in $\mathcal{F}_4 \cup \mathcal{F}_8 \cup \mathcal{F}_X$, the following format-level operation specializations shall be provided:

BitwidthOf, PrecisionOf, SignednessOf, DomainOf,
ExponentBitwidthOf, TrailingSignificandBitwidthOf, ExponentBiasOf,
MaxFiniteOf, MinFiniteOf, MinPositiveOf, MaxSubnormalOf, MinNormalOf.

[8] See §4.14 about external formats.

In addition, scaled operations, as defined in §5.5, shall be provided as follows, with $\mathcal{F}_s = \{\mathsf{Binary8p1uf}\}$:

ScaledAdd$<(f_s, f_1), (f_s, f_2), f_r, \rho>$	**where**	$f_s \in \mathcal{F}_s$
	and	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_8 \cup \mathcal{F}_X$
ScaledSubtract$<(f_s, f_1), (f_s, f_2), f_r, \rho>$	**where**	$f_s \in \mathcal{F}_s$
	and	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_8 \cup \mathcal{F}_X$
ScaledMultiply$<(f_s, f_1), (f_s, f_2), f_r, \rho>$	**where**	$f_s \in \mathcal{F}_s$
	and	$\{f_1, f_2\} \subset \mathcal{F}_4 \cup \mathcal{F}_8$
	and	$f_r \in \mathcal{F}_8 \cup \mathcal{F}_X$

Note that the choice of subset for $\mathcal{F}_X$ is implementation-defined.

4.6 Conformance declarations

Note that an implementation provides a subset of operation specializations for a subset of operations defined in this standard.

Where an operation specialization is supplied, the implementation shall compute the same result as does the defined operation specialization for all possible operand values. This may be attested by any proof method, including direct computation.

Where an approximate implementation of an operation specialization is supplied, the implementation shall declare κ as defined in §4.4. Such an implementation should use an implementation-specific identifier which indicates that the implementation is approximate.

It is recommended that an implementation supply a means whereby a user may query the presence of an implementation of a given operation specialization defined by a string representation.

Example:

An implementation that supports an exponentiation operation which operates on 8-bit, signed, extended floating-point values with a precision of 3 or 4, with result precision always 4, and saturation to finite or no saturation. These operation specializations could be declared as follows:

`exp_8.3.to.8.4.finite`: Exp<Binary8p3se, Binary8p4se, (NearestTiesToEven, SatFinite)>

`exp_8.4.to.8.4.finite`: Exp<Binary8p4se, Binary8p4se, (NearestTiesToEven, SatFinite)>

`exp_8.3.to.8.4.inf`: Exp<Binary8p3se, Binary8p4se, (NearestTiesToEven, SatNone)>

`exp_8.4.to.8.4.inf`: Exp<Binary8p4se, Binary8p4se, (NearestTiesToEven, SatNone)>

4.7 Internal functions

4.7.1 General

The following subclauses define auxiliary operations which are referenced in the definitions of operations, but which themselves are not required of a conforming implementation.

4.7.2 Decoding

Signature

$\omega\mathsf{Decode}_f(x) \to X$
$\omega\mathsf{DecodeAux}_f(\Sigma, \Delta, x) \to X$

Parameters

f : format of x

Operands

x : floating-point value, in format f

Result

X : floating-point datum, a closed extended real value in $\mathcal{D}_f$

Behavior

$\omega\mathsf{Decode}(x) \to \omega\mathsf{DecodeExternal}_f(x)$ **if** $f \in \{\mathsf{binary64}, \mathsf{binary32}, \mathsf{binary16}, \mathsf{BFloat16}\}$ [9]
$\omega\mathsf{Decode}(x) \to \omega\mathsf{DecodeAux}_f(\mathsf{SignednessOf}(f), \mathsf{DomainOf}(f), x)$

$\omega\mathsf{DecodeAux}(\mathsf{Signed}, *, 2^{\mathsf{K}-1}) \to \mathsf{NaN}$
$\omega\mathsf{DecodeAux}(\mathsf{Unsigned}, *, 2^{\mathsf{K}} - 1) \to \mathsf{NaN}$

$\omega\mathsf{DecodeAux}(\mathsf{Signed}, \mathsf{Extended}, 2^{\mathsf{K}-1} - 1) \to +\infty$
$\omega\mathsf{DecodeAux}(\mathsf{Signed}, \mathsf{Extended}, 2^{\mathsf{K}} - 1) \to -\infty$
$\omega\mathsf{DecodeAux}(\mathsf{Unsigned}, \mathsf{Extended}, 2^{\mathsf{K}} - 2) \to +\infty$

$\omega\mathsf{DecodeAux}(\mathsf{Signed}, \Delta, 2^{\mathsf{K}-1} < x < 2^{\mathsf{K}}) \to -\omega\mathsf{DecodeAux}_f(\mathsf{Signed}, \Delta, x - 2^{\mathsf{K}-1})$

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

where

$T = x \bmod 2^{\mathsf{P}-1}$ (Trailing significand)

$E_{\text{biased}} = x \div 2^{\mathsf{P}-1}$ (Biased exponent)

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

and

$\mathsf{P} = \mathsf{PrecisionOf}(f)$
$\mathsf{B} = \mathsf{ExponentBiasOf}(f)$

where

$\mathsf{K} = \mathsf{BitwidthOf}(f)$

[9] See §4.14 about external formats.

4.7.3 Projection

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

Signature

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

Parameters

f : target format
ρ : projection specification

Operands

X : closed extended real value

Result

x : floating-point value in format f

Behavior

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

where

$R = \omega\mathsf{RoundToPrecision}_{\mathsf{PrecisionOf}(f),\mathsf{ExponentBiasOf}(f),\mathsf{RoundOf}(\rho)}(X)$
$S = \omega\mathsf{Saturate}_{M^{lo},M^{hi}}(\mathsf{SatOf}(\rho), \mathsf{RoundOf}(\rho), R, \Sigma, \Delta)$
$x = \omega\mathsf{Encode}_f(S)$

and

$\Sigma = \mathsf{SignednessOf}(f)$
$\Delta = \mathsf{DomainOf}(f)$
$M^{lo} = \omega\mathsf{Decode}_f(\mathsf{MinFiniteOf}(f))$
$M^{hi} = \omega\mathsf{Decode}_f(\mathsf{MaxFiniteOf}(f))$

NOTE 1—In ωSaturate, all values above the maximum finite value M^{hi} are sent to either M^{hi} or $+\infty$. Nevertheless, for rounding to nearest, ωProject has the property, shared with IEEE-754, that values between M^{hi} and the number half a unit in the last place above M^{hi} will project to M^{hi} because ωRoundToPrecision precedes saturation.

NOTE 2—Under NearestTiesToEven rounding, subnormals are handled following IEEE-754, for example, values strictly between the smallest subnormal and its half are rounded to the smallest subnormal, while values of magnitude below or equal to half that of the smallest subnormal are rounded to zero.

4.7.4 Rounding to precision

Convert a closed extended real value to a closed extended real value expressible (where finite) as an integer multiple of a power of two.

NOTE— Where X is nonzero and finite, the returned value with a given precision P and exponent bias B, is of the form $\mathrm{sgn}(X) \times S \times 2^{1-\mathsf{P}} \times 2^{E}$, where $S \in \mathbb{Z}$, $0 \leq S < 2^{\mathsf{P}}$, and $E \in \mathbb{Z}$ with $-\mathsf{B} < E$.

Signature

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

Parameters

P : precision
B : exponent bias
μ : rounding mode

Operands

X : closed extended real value

Result

Z : closed extended real value

Behavior

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

where

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

$$\tilde{S} = |X| \times 2^{-Q} \qquad \text{(Real-valued significand, to be rounded to integer)}$$

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

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

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

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

and

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

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

Details

In stochastic rounding, random bits R are supplied as an unsigned integer in the range $0 \leq R < 2^N$, thus N is the number of random bits.

NOTE 1—The quality of the random bits is not specified in this document.

NOTE 2—The variants StochasticA, StochasticB, StochasticC offer a balance between accuracy and complexity [2].

NOTE 3—The notation $\mathsf{StochasticA}_{N,*}$ may be used to indicate that N random bits are supplied but not further specified.

NOTE 4—The intermediate value S 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 represented as the first number in the next binade.

4.7.5 Saturation

Saturate closed extended real to $\pm\infty$, or to maximum/minimum finite value.

Signature

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

Parameters

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

Operands

Sat : saturation mode
Round : rounding mode
X : closed extended real value
Σ : signedness in $\{\mathsf{Signed}, \mathsf{Unsigned}\}$
Δ : domain in $\{\mathsf{Finite}, \mathsf{Extended}\}$

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

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

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

NOTE—Whilst rounding precedes saturation in $\omega\mathsf{Project}$, the rounding mode is supplied to $\omega\mathsf{Saturate}$ in order to resolve cases where rounding direction requires a finite result, or where M^{hi} is even and rounding is to odd. Saturation does not round any value.

4.7.6 Encoding

Encode a closed extended real value to a code point in format f. $\omega\mathsf{Encode}$ is applied only to a value which is in the datum set of f. This value may be produced by $\omega\mathsf{RoundToPrecision}$ and $\omega\mathsf{Saturate}$, or the argument may be known to be in the datum set, for example, the absolute value of a number already in the set.

Signature

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

Parameters

f : target format

Operands

X : floating-point datum, a closed extended real value in the datum set of format f

Result

r : floating-point value, in format f

Behavior

$\omega\mathsf{Encode}(X) \to \omega\mathsf{EncodeExternal}_f(X) \quad \textbf{if } f \in \{\mathsf{binary64}, \mathsf{binary32}, \mathsf{binary16}, \mathsf{BFloat16}\}$

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

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

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

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

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

where

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

and

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

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

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

and

$\mathsf{K} = \mathsf{BitwidthOf}(f)$

$\mathsf{P} = \mathsf{PrecisionOf}(f)$

$\mathsf{B} = \mathsf{ExponentBiasOf}(f)$

$\Sigma = \mathsf{SignednessOf}(f)$

NOTE 1—From the precondition that X is in the datum set of format f, it follows that $S \in \mathbb{N}$, and that X is finite in finite formats.

NOTE 2—Similarly, $\omega\mathsf{Encode}$ is not called with a NaN operand by any operation in this specification, but may be called directly.

4.8 Decoding and encoding for external formats

The ωDecodeExternal and ωEncodeExternal functions convert between external formats and closed extended reals.

4.8.1 Decoding

The function ωDecodeExternal converts a floating-point value in an external format to a datum in the closed extended reals.

Signature

$\omega\mathsf{DecodeExternal}_f(x) \rightarrow X$

Parameters

f : format of x, in $\{\mathsf{binary64}, \mathsf{binary32}, \mathsf{binary16}, \mathsf{BFloat16}\}$

Operands

x : floating-point value, in format f

Result

X : floating-point datum, a closed extended real value in $\mathcal{D}_f$, the datum set of f

Behavior

$\omega\mathsf{DecodeExternal}(\text{Any NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{DecodeExternal}(\text{-}\mathsf{Inf}) \rightarrow -\infty$
$\omega\mathsf{DecodeExternal}(\mathsf{Inf}) \rightarrow +\infty$
$\omega\mathsf{DecodeExternal}(\text{-}0) \rightarrow 0$
$\omega\mathsf{DecodeExternal}(x) \rightarrow$ The extended real value encoded by x

4.8.2 Encoding

Encode floating-point datum to a floating-point value in external format f.

Signature

$\omega\mathsf{EncodeExternal}_f(X) \rightarrow r$

Parameters

f : target format, in $\{\mathsf{binary64}, \mathsf{binary32}, \mathsf{binary16}, \mathsf{BFloat16}\}$

Operands

X : floating-point datum, a closed extended real value in the datum set of format f

Result

r : floating-point value, in format f

Behavior

$\omega\mathsf{EncodeExternal}(\mathsf{NaN}) \rightarrow$ Any quiet NaN
$\omega\mathsf{EncodeExternal}(0) \rightarrow$ The code in f that decodes to the non-negative 0
$\omega\mathsf{EncodeExternal}(X) \rightarrow$ The code in f that decodes to X

Details

While an implementation may return any quiet NaN, it is recommended that the quiet NaN with zero payload is returned.

NOTE— The ωEncodeExternal function is only called with arguments in the datum set of format f, therefore the encoding is unambiguous and independent of rounding mode.

4.9 Converting between formats

4.9.1 Conversion

Convert a floating-point value to another format, with projection specification.

Signature

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

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Convert}(X) \rightarrow X$

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

4.10 Arithmetic operations

Arithmetic operations that take one or more floating-point values as operands and return one or more floating-point values are defined in this section.

4.10.1 Absolute value, Negation

Signature

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

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

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

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

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

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

4.10.2 Copying the sign

Copies the sign of a floating-point value onto another floating-point value.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

4.10.3 Addition, Subtraction

Addition and subtraction of two floating-point values, returning a floating-point value.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

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

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

4.10.4 Multiplication

Multiplication of two floating-point values, returning a floating-point value.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

4.10.5 Division

Division of two floating-point values, returning a floating-point value.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

NOTE 1—Divide($x, \pm$Inf) where x is finite yields 0.

NOTE 2—Divide($x, 0$) yields NaN. Returning Inf for finite x would imply $1/(1/-\infty) \rightarrow \infty$, an inconsistency.

4.10.6 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. The behavior of $\omega\mathsf{FMA}(X, Y, Z)$ is equivalent [14] to the behavior of $\omega\mathsf{Add}(\omega\mathsf{Multiply}(X, Y), Z)$.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point 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** $Y > 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(-\infty, +\infty, +\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, -\infty, +\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, +\infty, -\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(-\infty, -\infty, -\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{FMA}(+\infty, +\infty, *) \rightarrow +\infty$
$\omega\mathsf{FMA}(-\infty, -\infty, *) \rightarrow +\infty$
$\omega\mathsf{FMA}(+\infty, -\infty, *) \rightarrow -\infty$
$\omega\mathsf{FMA}(-\infty, +\infty, *) \rightarrow -\infty$
$\omega\mathsf{FMA}(*, *, +\infty) \rightarrow +\infty$
$\omega\mathsf{FMA}(*, *, -\infty) \rightarrow -\infty$
$\omega\mathsf{FMA}(X, -\infty, *)$ **if** $X > 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(X, -\infty, *)$ **if** $X < 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(X, +\infty, *)$ **if** $X > 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(X, +\infty, *)$ **if** $X < 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(+\infty, Y, *)$ **if** $Y > 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(+\infty, Y, *)$ **if** $Y < 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(-\infty, Y, *)$ **if** $Y > 0 \rightarrow -\infty$
$\omega\mathsf{FMA}(-\infty, Y, *)$ **if** $Y < 0 \rightarrow +\infty$
$\omega\mathsf{FMA}(X, Y, Z) \rightarrow X \times Y + Z$

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

4.10.7 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. The behavior of $\omega\mathsf{FAA}(X, Y, Z)$ is equivalent [14] to the behavior of $\omega\mathsf{Add}(\omega\mathsf{Add}(X, Y), Z)$, which is in turn equivalent to $\omega\mathsf{Add}(X, \omega\mathsf{Add}(Y, Z))$.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

4.10.8 Square root, Reciprocal, Reciprocal square root

Signature

$\mathsf{Sqrt}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Recip}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{RSqrt}_{f_x,f_r,\rho}(x) \rightarrow r$

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Sqrt}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sqrt}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sqrt}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sqrt}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Sqrt}(X) \rightarrow \sqrt{X}$

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

$\omega\mathsf{Recip}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Recip}(0) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Recip}(\pm\infty) \rightarrow 0$
$\omega\mathsf{Recip}(X) \rightarrow 1/X$

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

$\omega\mathsf{RSqrt}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{RSqrt}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{RSqrt}(X)$ **if** $X \leq 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{RSqrt}(+\infty) \rightarrow 0$
$\omega\mathsf{RSqrt}(X) \rightarrow 1/\sqrt{X}$

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

4.10.9 Logarithm, Exponentiation

Signature

$\mathsf{Exp}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Exp2}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Log}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Log2}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{LogOnePlus}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ExpMinusOne}_{f_x,f_r,\rho}(x) \rightarrow r$

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Exp}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Exp}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Exp}(-\infty) \rightarrow 0$
$\omega\mathsf{Exp}(X) \rightarrow e^X$

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

$\omega\mathsf{Exp2}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Exp2}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Exp2}(-\infty) \rightarrow 0$
$\omega\mathsf{Exp2}(X) \rightarrow 2^X$

$\mathsf{Exp2}(x) \rightarrow \omega\mathsf{Project}_{f_r,\rho}(\omega\mathsf{Exp2}(\omega\mathsf{Decode}_{f_x}(x)))$

$\omega\mathsf{Log}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Log}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log}(0) \rightarrow -\infty$
$\omega\mathsf{Log}(X) \rightarrow \log_e X$

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

$\omega\mathsf{Log2}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log2}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log2}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Log2}(X)$ **if** $X < 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Log2}(0) \rightarrow -\infty$
$\omega\mathsf{Log2}(X) \rightarrow \log_2 X$

$\mathsf{Log2}(x) \rightarrow \omega\mathsf{Project}_{f_r,\rho}(\omega\mathsf{Log2}(\omega\mathsf{Decode}_{f_x}(x)))$

$\omega\mathsf{LogOnePlus}(X) \rightarrow \omega\mathsf{Log}(\omega\mathsf{Add}(1, X))$

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

$\omega\mathsf{ExpMinusOne}(X) \rightarrow \omega\mathsf{Subtract}(\omega\mathsf{Exp}(X), 1)$

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

4.10.10 Trigonometric functions

Signature

$\mathsf{Sin}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Cos}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Tan}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcSin}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcCos}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcTan}_{f_x,f_r,\rho}(x) \rightarrow r$

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Sin}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sin}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sin}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sin}(X) \rightarrow \sin X$

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

$\omega\mathsf{ArcSin}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSin}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSin}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSin}(X)$ **if** $X < -1$ **or** $X > 1 \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSin}(X) \rightarrow \arcsin X$

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

$\omega\mathsf{Cos}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Cos}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Cos}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Cos}(X) \rightarrow \cos X$

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

$\omega\mathsf{ArcCos}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCos}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCos}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCos}(X)$ **if** $X < -1$ **or** $X > 1 \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCos}(X) \rightarrow \arccos X$

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

$\omega\mathsf{Tan}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Tan}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Tan}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Tan}(X)$ **if** $\cos X = 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{Tan}(X) \rightarrow \tan X$

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

$\omega\mathsf{ArcTan}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan}(+\infty) \rightarrow \frac{\pi}{2}$
$\omega\mathsf{ArcTan}(-\infty) \rightarrow -\frac{\pi}{2}$
$\omega\mathsf{ArcTan}(X) \rightarrow \arctan X$

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

NOTE— There are no datums X in the datum set of any format f for which $\cos X = 0$, as all floats are rational, but the special case is included for documentation, and to support formal verification.

4.10.11 Hyperbolic functions

Signature

$\mathsf{Sinh}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Cosh}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{Tanh}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcSinh}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcCosh}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcTanh}_{f_x,f_r,\rho}(x) \rightarrow r$

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Sinh}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Sinh}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Sinh}(-\infty) \rightarrow -\infty$
$\omega\mathsf{Sinh}(X) \rightarrow \sinh X$

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

$\omega\mathsf{ArcSinh}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSinh}(+\infty) \rightarrow +\infty$
$\omega\mathsf{ArcSinh}(-\infty) \rightarrow -\infty$
$\omega\mathsf{ArcSinh}(X) \rightarrow \operatorname{arcsinh} X$

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

$\omega\mathsf{Cosh}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Cosh}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Cosh}(-\infty) \rightarrow +\infty$
$\omega\mathsf{Cosh}(X) \rightarrow \cosh X$

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

$\omega\mathsf{ArcCosh}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosh}(+\infty) \rightarrow +\infty$
$\omega\mathsf{ArcCosh}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosh}(X)$ **if** $X < 1 \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosh}(X) \rightarrow \operatorname{arccosh} X$

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

$\omega\mathsf{Tanh}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Tanh}(+\infty) \rightarrow 1$
$\omega\mathsf{Tanh}(-\infty) \rightarrow -1$
$\omega\mathsf{Tanh}(X) \rightarrow \tanh X$

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

$\omega\mathsf{ArcTanh}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTanh}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTanh}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTanh}(1) \rightarrow +\infty$
$\omega\mathsf{ArcTanh}(-1) \rightarrow -\infty$
$\omega\mathsf{ArcTanh}(X)$ **if** $X < -1$ **or** $X > 1 \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTanh}(X) \rightarrow \operatorname{arctanh} X$

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

4.10.12 Trigonometric π-functions

Signature

$\mathsf{SinPi}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{CosPi}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{TanPi}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcSinPi}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcCosPi}_{f_x,f_r,\rho}(x) \rightarrow r$
$\mathsf{ArcTanPi}_{f_x,f_r,\rho}(x) \rightarrow r$

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{SinPi}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{SinPi}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{SinPi}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{SinPi}(X) \rightarrow \sin(X \times \pi)$

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

$\omega\mathsf{ArcSinPi}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSinPi}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSinPi}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSinPi}(X)$ **if** $X < -1$ **or** $X > 1 \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcSinPi}(X) \rightarrow \arcsin(X)/\pi$

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

$\omega\mathsf{CosPi}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{CosPi}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{CosPi}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{CosPi}(X) \rightarrow \cos(X \times \pi)$

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

$\omega\mathsf{ArcCosPi}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosPi}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosPi}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosPi}(X)$ **if** $X < -1$ **or** $X > 1 \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcCosPi}(X) \rightarrow \arccos(X)/\pi$

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

$\omega\mathsf{TanPi}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{TanPi}(+\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{TanPi}(-\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{TanPi}(X)$ **if** $\cos(X \times \pi) = 0 \rightarrow \mathsf{NaN}$
$\omega\mathsf{TanPi}(X) \rightarrow \tan(X \times \pi)$

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

$\omega\mathsf{ArcTanPi}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTanPi}(+\infty) \rightarrow 1/2$
$\omega\mathsf{ArcTanPi}(-\infty) \rightarrow -1/2$
$\omega\mathsf{ArcTanPi}(X) \rightarrow \arctan(X)/\pi$

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

4.10.13 Softplus

Signature

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

Parameters

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

Operands

x : floating-point value, format f_x

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{Softplus}(\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{Softplus}(+\infty) \rightarrow +\infty$
$\omega\mathsf{Softplus}(-\infty) \rightarrow 0$
$\omega\mathsf{Softplus}(X) \rightarrow \log_e(1 + e^X)$

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

4.10.14 Hypotenuse

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

4.10.15 Inverse tangent of two variables

Signature

$\mathsf{ArcTan2}_{f_y,f_x,f_r,\rho}(y,x) \rightarrow r$

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{ArcTan2}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2}(0, 0) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2}(-\infty, \pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2}(+\infty, \pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2}(*, +\infty) \rightarrow 0$
$\omega\mathsf{ArcTan2}(0, X)$ **if** $X > 0 \rightarrow 0$
$\omega\mathsf{ArcTan2}(+\infty, *) \rightarrow \frac{\pi}{2}$
$\omega\mathsf{ArcTan2}(Y, 0)$ **if** $Y > 0 \rightarrow \frac{\pi}{2}$
$\omega\mathsf{ArcTan2}(*, -\infty) \rightarrow \pi$
$\omega\mathsf{ArcTan2}(0, X)$ **if** $X < 0 \rightarrow \pi$
$\omega\mathsf{ArcTan2}(-\infty, *) \rightarrow -\frac{\pi}{2}$
$\omega\mathsf{ArcTan2}(Y, 0)$ **if** $Y < 0 \rightarrow -\frac{\pi}{2}$
$\omega\mathsf{ArcTan2}(Y, X)$ **if** $X > 0 \rightarrow \arctan(Y/X)$
$\omega\mathsf{ArcTan2}(Y, X)$ **if** $Y > 0$ **and** $X < 0 \rightarrow \arctan(Y/X) + \pi$
$\omega\mathsf{ArcTan2}(Y, X)$ **if** $Y < 0$ **and** $X < 0 \rightarrow \arctan(Y/X) - \pi$

$\mathsf{ArcTan2}(y, x) \rightarrow \omega\mathsf{Project}_{f_r,\rho}(\omega\mathsf{ArcTan2}(\omega\mathsf{Decode}_{f_y}(y), \omega\mathsf{Decode}_{f_x}(x)))$

NOTE 1—It would be inconsistent to define $\mathsf{ArcTan2}(0, 0)$ as 0, as the limit is indeterminate.

NOTE 2—$\mathsf{ArcTan2}(\pm\mathsf{Inf}, \pm\mathsf{Inf})$ yields NaN, which is consistent with $\mathsf{Divide}(\pm\mathsf{Inf}, \pm\mathsf{Inf})$.

4.10.16 Inverse tangent of two variables (π-variant)

Signature

$\mathsf{ArcTan2Pi}_{f_y,f_x,f_r,\rho}(y,x) \rightarrow r$

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

$\omega\mathsf{ArcTan2Pi}(\mathsf{NaN},*) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2Pi}(*,\mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2Pi}(0,0) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2Pi}(-\infty,\pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2Pi}(+\infty,\pm\infty) \rightarrow \mathsf{NaN}$
$\omega\mathsf{ArcTan2Pi}(*,+\infty) \rightarrow 0$
$\omega\mathsf{ArcTan2Pi}(0,X)$ **if** $X > 0 \rightarrow 0$
$\omega\mathsf{ArcTan2Pi}(+\infty,*) \rightarrow 1/2$
$\omega\mathsf{ArcTan2Pi}(Y,0)$ **if** $Y > 0 \rightarrow 1/2$
$\omega\mathsf{ArcTan2Pi}(*,-\infty) \rightarrow 1$
$\omega\mathsf{ArcTan2Pi}(0,X)$ **if** $X < 0 \rightarrow 1$
$\omega\mathsf{ArcTan2Pi}(-\infty,*) \rightarrow -1/2$
$\omega\mathsf{ArcTan2Pi}(Y,0)$ **if** $Y < 0 \rightarrow -1/2$
$\omega\mathsf{ArcTan2Pi}(Y,X)$ **if** $X > 0 \rightarrow \arctan(Y/X)/\pi$
$\omega\mathsf{ArcTan2Pi}(Y,X)$ **if** $Y > 0$ **and** $X < 0 \rightarrow (\arctan(Y/X)+\pi)/\pi$
$\omega\mathsf{ArcTan2Pi}(Y,X)$ **if** $Y < 0$ **and** $X < 0 \rightarrow (\arctan(Y/X)-\pi)/\pi$

$\mathsf{ArcTan2Pi}(y,x) \rightarrow \omega\mathsf{Project}_{f_r,\rho}(\omega\mathsf{ArcTan2Pi}(\omega\mathsf{Decode}_{f_y}(y),\omega\mathsf{Decode}_{f_x}(x)))$

NOTE 1—It would be inconsistent to define $\mathsf{ArcTan2Pi}(0,0)$ as 0, as the limit is indeterminate.

NOTE 2—$\mathsf{ArcTan2Pi}(\pm\mathsf{Inf},\pm\mathsf{Inf})$ yields NaN, which is consistent with $\mathsf{Divide}(\pm\mathsf{Inf},\pm\mathsf{Inf})$.

4.11 Extrema

4.11.1 Minimum and maximum, and number variants

Minimum and maximum, with or without propagation of NaN.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

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

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

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

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

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

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

4.11.2 Minimum and maximum magnitude, and number variants

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

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

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

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

$\omega\mathsf{MinimumMagnitudeNumber}(X, \mathsf{NaN}) \rightarrow X$
$\omega\mathsf{MinimumMagnitudeNumber}(\mathsf{NaN}, Y) \rightarrow Y$
$\omega\mathsf{MinimumMagnitudeNumber}(X, Y) \rightarrow \omega\mathsf{MinimumMagnitude}(X, Y)$

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

$\omega\mathsf{MaximumMagnitudeNumber}(X, \mathsf{NaN}) \rightarrow X$
$\omega\mathsf{MaximumMagnitudeNumber}(\mathsf{NaN}, Y) \rightarrow Y$
$\omega\mathsf{MaximumMagnitudeNumber}(X, Y) \rightarrow \omega\mathsf{MaximumMagnitude}(X, Y)$

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

4.11.3 Minimum and maximum finite variants

Minimum and maximum finite value.

Signature

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

Parameters

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

Operands

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

Result

r : floating-point value, format f_r

Behavior

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

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

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

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

4.11.4 Clamping

Signature

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

Parameters

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

Operands

x : floating-point value, format f_x
lo : floating-point value, format f_{lo}
hi : floating-point value, format f_{hi}

Result

r : floating-point value, format f_r

Behavior

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

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

4.12 Comparisons

Comparison operators take two operands and return a Boolean value.

Signature

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

Parameters

f_x : format of operand x
f_y : format of operand y

Operands

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

Result

b : boolean value

Behavior

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

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

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

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

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

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

[Comparisons continued on next page]

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

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

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

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

4.12.1 Total order predicate

Signature

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

Parameters

f_x : format of operand x
f_y : format of operand y

Operands

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

Result

b : Boolean value

Behavior

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

NOTE 1—The $\mathsf{TotalOrder}(x, y)$ predicate provides a total ordering over each format's value set.

NOTE 2—The above definition is consistent with the IEEE-754 definition of TotalOrder. There is a single NaN and it always compares as the most negative value.

4.12.2 Comparison operator symbols

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

Table 1—Comparison predicates and negations

Mathematical symbol	Predicate
$x = y$	CompareEqual(x, y)
$x > y$	CompareGreater(x, y)
$x \geq y,\ x$ >= y	CompareGreaterEqual(x, y)
$x < y$	CompareLess(x, y)
$x \leq y,\ x$ <= y	CompareLessEqual(x, y)
$x \neq y,\ x$!= y	**not** CompareEqual(x, y)

4.13 Predicates and classification

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

Signature

$\mathsf{Is}\mathit{Class}_f(x) \rightarrow b$

Parameters

f : format of operand x

Operands

x : floating-point value, in format f

Result

b : boolean value

Behavior

$\omega\mathsf{IsZero}(\mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{IsZero}(\pm\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{IsZero}(X) \rightarrow X = 0$

$\mathsf{IsZero}(x) \rightarrow \omega\mathsf{IsZero}(\omega\mathsf{Decode}_f(x))$

$\omega\mathsf{IsOne}(\mathsf{NaN}) \rightarrow \mathsf{False}$
$\omega\mathsf{IsOne}(\pm\infty) \rightarrow \mathsf{False}$
$\omega\mathsf{IsOne}(X) \rightarrow X = 1$

$\mathsf{IsOne}(x) \rightarrow \omega\mathsf{IsOne}(\omega\mathsf{Decode}_f(x))$

$\mathsf{IsNaN}(x) \rightarrow \omega\mathsf{Decode}_f(x) \in \{\mathsf{NaN}\}$

$\mathsf{IsInfinite}(x) \rightarrow \omega\mathsf{Decode}_f(x) \in \{-\infty, +\infty\}$

$\mathsf{IsFinite}(x) \rightarrow \mathbf{not}\ \mathsf{IsInfinite}(x)\ \mathbf{and\ not}\ \mathsf{IsNaN}(x)$

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

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

$\mathsf{IsNormal}(x) \rightarrow \mathsf{False}\quad \mathbf{if}\ (\mathsf{IsZero}(x)\ \mathbf{or}\ \mathsf{IsInfinite}(x)\ \mathbf{or}\ \mathsf{IsNaN}(x))$
$\mathsf{IsNormal}(x) \rightarrow |\omega\mathsf{Decode}_f(x)| \geq \omega\mathsf{Decode}_f(\mathsf{MinNormalOf}(f))$

$\mathsf{IsSubnormal}(x) \rightarrow \mathsf{False}\quad \mathbf{if}\ (\mathsf{IsZero}(x)\ \mathbf{or}\ \mathsf{IsInfinite}(x)\ \mathbf{or}\ \mathsf{IsNaN}(x))$
$\mathsf{IsSubnormal}(x) \rightarrow \mathbf{not}\ \mathsf{IsNormal}(x)$

4.13.1 Classifier operation

The classifier operation Class(x) tells which of the eight classes x falls into as defined by Table 2.

Signature

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

Parameters

f : format of operand x

Operands

x : floating-point value, format f

Result

c : enumeration

Behavior

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

Table 2—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 Format-level operations

Certain operations act on formats rather than values, for example, determining the precision of a format.[10]

The following operations return integers or enumerations:

Operation	Format					
	Binary$\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}$ $\Sigma = \mathsf{Signed}$	Binary$\{\mathsf{K}, \mathsf{P}, \Sigma, \Delta\}$ $\Sigma = \mathsf{Unsigned}$	binary64	binary32	binary16	BFloat16
BitwidthOf(f)	K	K	64	32	16	16
PrecisionOf(f)	P	P	53	24	11	8
SignednessOf(f)	Signed	Unsigned	Signed	Signed	Signed	Signed
DomainOf(f)	Δ	Δ	Extended	Extended	Extended	Extended
ExponentBitwidthOf(f)	$\mathsf{K} - \mathsf{P}$	$\mathsf{K} - \mathsf{P} + 1$	11	8	5	8
TrailingSignificandBitwidthOf(f)	$\mathsf{P} - 1$	$\mathsf{P} - 1$	52	23	10	7
ExponentBiasOf(f)	$2^{\mathsf{K}-\mathsf{P}-1}$	$2^{\mathsf{K}-\mathsf{P}}$	1023	127	15	127

The following operations return floating-point values, in the format f:

Operation	Description
MaxFiniteOf(f)	Maximum finite value representable in format f.
MinFiniteOf(f)	Minimum finite value representable in format f. In signed formats, MinFiniteOf(f) $= -$MaxFiniteOf(f). In unsigned formats, MinFiniteOf(f) $= 0$.
MinPositiveOf(f)	Minimum strictly positive value representable in format f.
MaxSubnormalOf(f)	Maximum positive subnormal value representable in format f, or NaN if no values are subnormal.
MinNormalOf(f)	Minimum positive normal value representable in format f.

4.15 Projection specification operations

These operations query projection specifications and return enumerations:

RoundOf(ρ)	Rounding mode of projection specification ρ, such that RoundOf((r, s)) $= r$.
SatOf(ρ)	Saturation mode of projection specification ρ, such that SatOf((r, s)) $= s$.

[10] The external formats binary64, binary32, binary16 are from [6], and BFloat16 from [7].

4.16 Next greater than and next less than

Signature

$\mathsf{NextGreaterThan}_f(x) \rightarrow r$
$\mathsf{NextLessThan}_f(x) \rightarrow r$

Parameters

f : format of operand x and result r

Operands

x : floating-point value, format f

Result

r : floating-point value, format f

Behavior

$\mathsf{NextGreaterThanAux}(*, *, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\mathsf{NextGreaterThanAux}(*, \mathsf{Extended}, \mathsf{Inf}) \rightarrow \mathsf{NaN}$
$\mathsf{NextGreaterThanAux}(*, \mathsf{Finite}, \mathit{MaxFinite}) \rightarrow \mathsf{NaN}$
$\mathsf{NextGreaterThanAux}(*, \mathsf{Extended}, \mathit{MaxFinite}) \rightarrow \mathsf{Inf}$
$\mathsf{NextGreaterThanAux}(\mathsf{Signed}, \mathsf{Extended}, \text{-}\mathsf{Inf}) \rightarrow \mathit{MinFinite}$
$\mathsf{NextGreaterThanAux}(\mathsf{Signed}, *, \mathit{SmallestNegative}) \rightarrow 0$
$\mathsf{NextGreaterThanAux}(\mathsf{Signed}, *, x)$ **if** $\mathsf{IsSignMinus}(x) \rightarrow x - 1$
$\mathsf{NextGreaterThanAux}(*, *, x) \rightarrow x + 1$ (See §4.1 for integer arithmetic on code points.)

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

$\mathsf{NextLessThanAux}(*, *, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\mathsf{NextLessThanAux}(*, \mathsf{Finite}, \mathit{MinFinite}) \rightarrow \mathsf{NaN}$
$\mathsf{NextLessThanAux}(\mathsf{Unsigned}, *, 0) \rightarrow \mathsf{NaN}$
$\mathsf{NextLessThanAux}(\mathsf{Signed}, \mathsf{Extended}, \text{-}\mathsf{Inf}) \rightarrow \mathsf{NaN}$
$\mathsf{NextLessThanAux}(\mathsf{Signed}, \mathsf{Extended}, \mathit{MinFinite}) \rightarrow \text{-}\mathsf{Inf}$
$\mathsf{NextLessThanAux}(*, \mathsf{Extended}, \mathsf{Inf}) \rightarrow \mathit{MaxFinite}$
$\mathsf{NextLessThanAux}(\mathsf{Signed}, *, 0) \rightarrow \mathit{SmallestNegative}$
$\mathsf{NextLessThanAux}(\mathsf{Signed}, *, x)$ **if** $\mathsf{IsSignMinus}(x) \rightarrow x + 1$
$\mathsf{NextLessThanAux}(*, *, x) \rightarrow x - 1$

$\mathsf{NextLessThan}(x) \rightarrow \mathsf{NextLessThanAux}(\mathsf{SignednessOf}(f), \mathsf{DomainOf}(f), x)$

where

$\mathit{MaxFinite} = \mathsf{MaxFiniteOf}(f)$
$\mathit{MinFinite} = \mathsf{MinFiniteOf}(f)$
$\mathit{SmallestNegative} = \omega\mathsf{Encode}_f(-\omega\mathsf{Decode}_f(\mathsf{MinPositiveOf}(f)))$ (The negative number of least magnitude)

NOTE 1—$\mathit{SmallestNegative}$ is not expanded for unsigned formats.

NOTE 2—These operations follow IEEE-754 nextUp and nextDown operations, with extensions for signedness and domain. The wording, using nextUp as an example, is extended from "least floating-point number that compares greater than" to "least floating-point number that compares greater than, or NaN". This requires that $\mathsf{Inf} \rightarrow \mathsf{NaN}$, while IEEE-754 defines $\mathbf{nextUp}(+\infty) = +\infty$, so new names are chosen in this document.

5 Block operations

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

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

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

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

5.1 Internal functions

5.1.1 Decoding

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 : scale factor, in format f_s
$\mathbf{x}$: sequence of floating-point values in format f_x

Result

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

Behavior

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

$$\textbf{where}$$

$$Z_i = \omega\mathsf{Multiply}(\omega\mathsf{Decode}_{f_s}(s), \omega\mathsf{Decode}_{f_x}(x_i))$$

5.1.2 Projection

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

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

Parameters

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

Operands

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

Result

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

Behavior

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

where

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

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

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

NOTE 1—If the scale factor s is zero, all non-NaN result elements are zero.

NOTE 2—If the scale factor s is infinite, all nonzero, non-NaN result elements are ± 1.

5.2 Conversion of blocks

5.2.1 Conversion from a block

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

Signature

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

Parameters

B : block size
f_s : format of scale factor operand s
f_x : format of operand elements x_i
f_r : format of result elements r
ρ : projection specification for result elements

Operands

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

Result

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

Behavior

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

5.2.2 Conversion to a block

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

Signature

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

Parameters

B : block size
f_x : format of operand elements x_i
f_s : format of result scale factor s
f_r : format of result elements r
ρ : projection specification for result elements

Operands

$\mathbf{x}$: sequence of floating-point values in format f_x
s : result scale factor, a floating-point value in format f_s

Result

s : result scale factor, a floating-point value in format f_s
$\mathbf{r}$: sequence of floating-point values in format f_r

Behavior

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

5.2.3 Conversion to a block with scale factor computation

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

Signature

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

Parameters

B : block size
f_x : format of operand elements x_i
f_s : format of result scale factor s
f_r : format of result elements r
ρ_s : projection specification for result scale factor
ρ : projection specification for result elements

Operands

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

Result

s : result scale factor, a floating-point value in format f_s
$\mathbf{r}$: sequence of floating-point values in format f_r

Behavior

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

NOTE 1—If all x_i are NaN, the result scale factor and elements will be NaN.

NOTE 2—If all x_i are infinite, the result scale factor will be Inf or $\mathsf{MaxFiniteOf}(f_s)$ and the elements will be ± 1 or $\pm\mathsf{MaxFiniteOf}(f_r)$.

NOTE 3—If some x_i are infinite, they are preserved (or saturated, according to ρ) in the result, and do not influence the scale of finite elements.

NOTE 4—If the maximum value S rounds to zero under ρ_s, the result scale factor will be zero, and all result elements will be zero.

NOTE 5—The projection specifications for the scale factor and elements allow 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

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

Parameters

B : block size
f_s : format of scale factor operand s
f_x : format of operand elements x_i
f_r : format of result r
ρ : projection specification

Operands

s : scale factor, a floating-point value in format f_s
$\mathbf{x}$: sequence of floating-point values in format f_x

Result

r : floating-point value in format f_r

Behavior

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

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

5.3.2 Dot product

Dot product of two blocks of floating-point values.

Signature

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

Parameters

B : block size
f_{sx} : format of scale factor operand s_x
f_x : format of operand elements x_i
f_{sy} : format of scale factor operand s_y
f_y : format of operand elements y_i
f_r : format of result r
ρ : projection specification

Operands

s_x : scale factor for x, a floating-point value in format f_{sx}
$\mathbf{x}$: sequence of floating-point values in format f_x
s_y : scale factor for y, a floating-point value in format f_{sy}
$\mathbf{y}$: sequence of floating-point values in format f_y

Result

r : floating-point value in format f_r

Behavior

$\mathsf{BlockDotProduct}((s_x,[x_1,...,x_B]),(s_y,[y_1,...,y_B])) = r$

where

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

5.4 Elementwise operations on blocks

Operations on blocks are declared according to the following schema, where BlockOp is defined in terms of ωOp.

Note that the scale factor of the result block s_r is supplied as an *input* operand. Implementations are free to supply any method of computation of this scale factor, and to supply an operation in which that computation is composed with the elementwise operation.

Valid substitutions for Op are:

- Unary ($n = 1$): Convert, Abs, Negate, Sqrt, RSqrt, Recip, Exp, Log, ExpMinusOne, LogOnePlus, Exp2, Log2, Sin, Cos, Tan, ArcSin, ArcCos, ArcTan, Sinh, Cosh, Tanh, ArcSinh, ArcCosh, ArcTanh, SinPi, CosPi, TanPi, ArcSinPi, ArcCosPi, ArcTanPi, Softplus;
- Binary ($n = 2$): CopySign, Add, Subtract, Multiply, Divide, Hypot, ArcTan2, ArcTan2Pi, Maximum, Minimum, MaximumNumber, MinimumNumber, MaximumMagnitude, MinimumMagnitude, MaximumMagnitudeNumber, MinimumMagnitudeNumber, MinimumFinite, MaximumFinite;
- Ternary ($n = 3$): FMA, FAA, Clamp.

Signature

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

Parameters

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

f_s : format of scale factor of result s_r
f_r : format of result elements r
ρ : projection specification for result elements

Operands

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

s_r : result scale factor, a floating-point value in format f_s

Result

s_r : result scale factor, copied from operand
$\mathbf{r}$: sequence of floating-point values in format f_r

Behavior

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

where

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

5.5 Scaled operations via block operations

A block size of $B = 1$ provides scaled operations.

An n-ary scaled operation is defined as follows:

$$\mathsf{Scaled}\,Op_{(f_{s1},f_{x1}),...,(f_{sn},f_{xn}),f_r,\rho}(s_1, x_1, ..., s_n, x_n) = r$$
$$\textbf{where}$$
$$(1, [r]) = \mathsf{Block}\,Op((s_1, [x_1]), ..., (s_n, [x_n]), 1)$$

NOTE—A choice of scale factor format with $\mathsf{P} = 1$ allows scale factors to be restricted to powers of two.

Appendices

Annex A

(informative)

Rationales and discussion

A.1 General

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

A.2 Not a number (NaN)

Datum sets include exactly one NaN.

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

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

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

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

A.3 Zero

Datum sets include exactly one zero.

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

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

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 it evident that the small hardware simplification has not been sufficient to balance the loss of one code point.

Another reason for including -0 in IEEE-754 was to allow the reciprocal of a signed infinity to be the corresponding signed zero. In this standard the reciprocal of either infinity is zero, and a corollary of that decision is that dividing by zero produces NaN, not an infinity.

The use of integer comparisons in sorting might argue against placing NaN at the negative zero code point. For example, the JAX machine learning framework is known to sort using integer comparison [4]. However, such sorting still requires $O(n)$ additional processing steps to enable the use of two's-complement integer comparison, and already has special treatment of NaN and -0. Eliminating -0 and placing NaN in the -0 position imposes negligible additional burden.

A.4 Infinities

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 using the finite and extended domains.

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

- Mask values, for example in transformer models in machine learning [12].
- Representation of overflow, for example to adjust dynamic loss scaling factors [15].

The following example shows that in certain machine learning computations, it is insufficient to substitute an infinity with the largest finite value.

[Continued on next page]

Example:

Consider the use of infinity in computation of attention masks. These values, assembled in a mask vector M with values $M_i \in \{0, -\infty\}$, are typically added to computed values A in a computation such as:

$$\log\left(\sum_i \exp(\tau \times (A_i + M_i))\right)$$

where $\tau > 0$ is a "temperature" or "base" parameter [3]. This calculation depends on the fact that $\exp(\tau \times (A_i - \infty)) = 0$. The operation sequence $\log(\sum_i \exp(v_i))$ is typically fused into a single block operation $\mathsf{logsumexp}(v)$.

For formats without infinities, $M_i = \infty$ will be replaced by a large float (e.g., the largest finite Binary8p4sf value is 240.0). 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 $\exp(1.0 - 240.0)$ is an extremely small number, sufficiently small to 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; instead, the implementation of logsumexp 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, as follows.

For example, compare a format using the finite domain where maxFinite=240, and a format using the extended domain. In both cases, the value 224.0 is representable, and the logsumexp calculation on a two-element vector might be, in a format with infinity:

$$\mathsf{logsumexp}(\tau \times [-224.0, -\infty]) \to \mathsf{logsumexp}(t * [0, -\infty]),$$

and in a format without infinity:

$$\mathsf{logsumexp}(\tau \times [-224.0, -240.0]) \to \mathsf{logsumexp}(t * [0, -16.0]).$$

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

A.5 Exponent bias

The exponent bias is derived from the format-defining parameters using the formula in §3.1.

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

The maximum exponent emax is

$$\mathsf{emax} = \begin{cases} 2^{\mathsf{K}-1} - 3 & \textbf{if } \mathsf{P} = 1, \Sigma = \mathsf{Unsigned}, \Delta = \mathsf{Extended} \\ 2^{\mathsf{K}-1} - 2 & \textbf{if } \mathsf{P} = 1, \Sigma = \mathsf{Unsigned}, \Delta = \mathsf{Finite} \\ 2^{\mathsf{K}-2} - 2 & \textbf{if } \mathsf{P} = 1, \Sigma = \mathsf{Signed}, \Delta = \mathsf{Extended} \\ 2^{\mathsf{K}-2} - 2 & \textbf{if } \mathsf{P} = 2, \Sigma = \mathsf{Unsigned}, \Delta = \mathsf{Extended} \\ 2^{\mathsf{K}-\mathsf{P}-1} - 1 & \textbf{otherwise if } \Sigma = \mathsf{Signed} \\ 2^{\mathsf{K}-\mathsf{P}} - 1 & \textbf{otherwise if } \Sigma = \mathsf{Unsigned} \end{cases}$$

A consequence of these specifications is that the datum 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 formats.

A.6 Subnormals

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, e.g., where model weights are initialized to small random values for training.

Subnormals are uniformly spaced around zero, and near-zero values are more probable under bell-shaped distributions. Formats with narrow exponent bitwidths necessarily have a limited range; subnormals extend this range by a power of two for every bit in the trailing significand.

The datum sets of formats with precision greater than one include subnormals; this does not preclude implementations of operations from flushing subnormals to zero, under the provision for approximate operations in §4.4.

Annex B
(informative)
Encoding

A K-bit floating-point value is encoded by an integer in the range 0 to $2^K - 1$. Some properties of this encoding are summarized here, while the detailed specification of the encoding is presented in §4.7.6.

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-754 uses for negative zero, that is 2^{K-1}. For unsigned formats, NaN is encoded at $2^K - 1$.

Extended formats contain one or two infinities. For signed formats, Inf is encoded at $2^{K-1} - 1$ and -Inf is encoded at $2^K - 1$. For unsigned formats, Inf is encoded at $2^K - 2$.

Table 3 summarizes the encodings of selected special values.

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

Table 3—Encodings of selected datums for given K, independent of P.

B.1 Value tables

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

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

Table 4—Value sets for $\mathsf{K} = 4$, signed formats with $1 \leq \mathsf{P} \leq 4$, extended domain.

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

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

Code point		Binary4p1ue	Binary4p2ue	Binary4p3ue	Binary4p4ue
0	0b0000	**Zero**	**Zero**	**Zero**	**Zero**
1	0b0001	≈0.0078	*0.0625*	*0.1250*	*0.1250*
2	0b0010	≈0.0156	0.1250	*0.2500*	*0.2500*
3	0b0011	≈0.0312	0.1875	*0.3750*	*0.3750*
4	0b0100	0.0625	0.2500	0.5000	*0.5000*
5	0b0101	0.1250	0.3750	0.6250	*0.6250*
6	0b0110	0.2500	0.5000	0.7500	*0.7500*
7	0b0111	0.5000	0.7500	0.8750	*0.8750*
8	0b1000	1.0000	1.0000	1.0000	1.0000
9	0b1001	2.0000	1.5000	1.2500	1.1250
10	0b1010	4.0000	2.0000	1.5000	1.2500
11	0b1011	8.0000	3.0000	1.7500	1.3750
12	0b1100	16.0000	4.0000	2.0000	1.5000
13	0b1101	32.0000	6.0000	2.5000	1.6250
14	0b1110	**Inf**	**Inf**	**Inf**	**Inf**
15	0b1111	**NaN**	**NaN**	**NaN**	**NaN**

Table 6—Value sets for $\mathsf{K} = 4$, unsigned formats with $1 \leq \mathsf{P} \leq 4$, extended domain.

Code point		Binary4p1uf	Binary4p2uf	Binary4p3uf	Binary4p4uf
0	0b0000	**Zero**	**Zero**	**Zero**	**Zero**
1	0b0001	≈0.0078	*0.0625*	*0.1250*	*0.1250*
2	0b0010	≈0.0156	0.1250	*0.2500*	*0.2500*
3	0b0011	≈0.0312	0.1875	*0.3750*	*0.3750*
4	0b0100	0.0625	0.2500	0.5000	*0.5000*
5	0b0101	0.1250	0.3750	0.6250	*0.6250*
6	0b0110	0.2500	0.5000	0.7500	*0.7500*
7	0b0111	0.5000	0.7500	0.8750	*0.8750*
8	0b1000	1.0000	1.0000	1.0000	1.0000
9	0b1001	2.0000	1.5000	1.2500	1.1250
10	0b1010	4.0000	2.0000	1.5000	1.2500
11	0b1011	8.0000	3.0000	1.7500	1.3750
12	0b1100	16.0000	4.0000	2.0000	1.5000
13	0b1101	32.0000	6.0000	2.5000	1.6250
14	0b1110	64.0000	8.0000	3.0000	1.7500
15	0b1111	**NaN**	**NaN**	**NaN**	**NaN**

Table 7—Value sets for $\mathsf{K} = 4$, unsigned formats with $1 \leq \mathsf{P} \leq 4$, finite domain.

Annex C

(informative)

Value tables for K=2

Table 8 shows the floating-point value sets for $\mathsf{K} = 2$ formats which follow the patterns in this document. Not all of these formats will typically be useful in practice, but they are included for completeness. As the table shows, certain peculiarities are evident:

- The formats Binary2p1ue and Binary2p2ue have the same datum set, albeit a different partitioning of subnormals.
- Similarly the formats Binary2p1uf and Binary2p2uf have the same datum set.
- The datum 1.0 is not present in Binary2p1se, Binary2p1ue, Binary2p2ue, hence is not encoded according to Table 3.
- Binary2p1se has no normal or subnormal values, hence $\mathsf{MinPositiveOf}(\mathsf{Binary2p1se}) = \mathsf{Inf}$.
- Binary2p1se and Binary2p2ue have no normal values, hence $\mathsf{MinNormalOf}(f) = \mathsf{NaN}$ for these formats.

Code point		Binary2p1se	Binary2p1sf	Binary2p1ue	Binary2p2ue	Binary2p1uf	Binary2p2uf
0	0b00	**Zero**	**Zero**	**Zero**	**Zero**	**Zero**	**Zero**
1	0b01	**Inf**	1.000	0.500	*0.500*	0.500	*0.500*
2	0b10	**NaN**	**NaN**	**Inf**	**Inf**	1.000	1.000
3	0b11	-**Inf**	-1.000	**NaN**	**NaN**	**NaN**	**NaN**

Table 8—Value sets for $\mathsf{K} = 2$. Subnormals are shown in *underlined italic*, special values in **bold**.

Annex D

(informative)

Examples of approximation declarations

D.1 Example: Approximate implementation of Exp

Consider an implementation of Exp<binary32, Binary8p4se, (NearestTiesToEven, SatNone)> which flushes subnormals in the result to zero or to the smallest normal, whichever is nearest, with ties going to zero. There are seven nonzero positive subnormals in Binary8p4se, of which four will be flushed to zero, and three will be returned as the smallest normal. Hence the value of κ over all inputs producing finite results is 4. Writing the cut points $a, b, c = \ln(2^{-11}), \ln(9 \cdot 2^{-11}), \ln(15 \cdot 2^{-11})$, the intervals are:

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

and the implementation would declare

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

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

D.2 Example: Approximate implementation of BlockDotProduct

Consider an implementation of dot product on blocks of size 8, with Binary8p4se elements and Binary8p1uf scale factors, with a binary32 result, rounding mode NearestTiesToEven and saturation mode SatNone. This is the operation specialization

BlockDotProduct<8, Binary8p1uf, Binary8p4se, Binary8p1uf, Binary8p4se, binary32, (NearestTiesToEven, SatNone)>

Suppose the nature of the implementation is such that it matches on NaNs and infinities, but has only the trivial bound $\kappa = 2^{32} - 2^{24} - 1$ over all inputs producing finite results. The implementation may choose to declare κ over a subset of the input space, for example by declaring

$$\begin{aligned} I_0 = \{(s_x, [x_1, ..., x_8], s_y, [y_1, ..., y_8]) \text{ such that} \\ & s_x = 1.0, \\ & s_y = 1.0, \\ & \text{Abs}(x_i) \leq 2.0 \text{ for } i \in \{1, 2\}, \\ & x_i = 0.0 \text{ for } i \in \{3 \ldots 8\}, \\ & \text{Abs}(y_i) \leq 2.0 \text{ for } i \in \{1, 2\}, \\ & y_i = 0.0 \text{ for } i \in \{3 \ldots 8\} \\ & \} \end{aligned}$$

and that $\kappa_{I_0} = 3$, while $\kappa_{I \setminus I_0}$ is the trivial bound as above.

This permits the implementation to provide a useful guarantee over a subset of the input space, while still providing a trivial guarantee over all inputs as required. Annex E shows an approach to characterization of accuracy for reduction operations.

Annex E

(informative)

Recommendations for reduction accuracy specifications

E.1 BlockReduceAdd

It is suggested that a finite error bound for the operation BlockReduceAdd be provided. Such a bound should apply at least whenever the output is finite. Examples of such bounds are given by Higham [5, §4.2] and by Blanchard et al. [1].

Example:

Consider a block of length $B = 8$, with arguments and output in Binary8p4se, where the accumulation is performed in binary16 using NearestTiesToEven, and the inputs are summed sequentially in any order. Let

$$S = X_1 + \cdots + X_8$$

denote the true sum, and let $\tilde{S}$ denote the implementation's approximate result. In the absence of overflow in $\tilde{S}$,

$$|S - \tilde{S}| \leq |S|/16 + 3.637 \times 10^{-3} \sum_{i=1}^{8} |X_i| \,.$$

The constant 3.637×10^{-3} is obtained by rounding up $(1 + 1/16)\big((1 + 2^{-11})^7 - 1\big)$, where the constants are obtained as follows: 11 is the accumulator precision, i.e., PrecisionOf(binary16); $7 = B - 1$ is the operation count; and $16 = 2^{\mathsf{PrecisionOf(Binary8p4se)}}$.

E.2 BlockReduceMultiply

It is suggested that a finite error bound for the operation BlockReduceMultiply be provided. Such a bound should apply at least whenever the output is finite. Examples of such bounds are given by Higham [5, §2.2].

Example:

Consider a block of length 8, with arguments and output in Binary8p4se, where the multiplications are performed in binary16 using NearestTiesToEven, sequentially in any order. Let

$$P = X_1 \times \cdots \times X_8$$

denote the true product, and let $\tilde{P}$ denote the implementation's approximate result. In the absence of underflow or overflow, both intermediate and final,

$$|P - \tilde{P}| \leq |P| \times 6.614 \times 10^{-2}.$$

The constant 6.614×10^{-2} is obtained by rounding up $(1 + 1/16)(1 + 2^{-11})^7 - 1$, where the constants are obtained as follows: 11 is the accumulator precision, i.e., PrecisionOf(binary16); $7 = B - 1$ is the operation count; and $16 = 2^{\mathsf{PrecisionOf(Binary8p4se)}}$.

E.3 BlockDotProduct

It is suggested that a finite error bound for the operation BlockDotProduct be provided. Such a bound should apply at least whenever the output is finite and the final rounding does not underflow. Examples of such bounds are given by Higham [5, §3.1] and by Blanchard et al. [1]. More generally applicable bounds may be complicated by implementation choices, for example when the precision used for the individual products varies during the computation.

Example:

Consider a block of length 8, with arguments and output in Binary8p4se, where the multiplications and accumulation are performed in binary16 using NearestTiesToEven, and the products are summed sequentially in any order. Let

$$D = X_1 \times Y_1 + \cdots + X_8 \times Y_8$$

denote the true dot product, and let $\tilde{D}$ denote the implementation's approximate result. In the absence of overflow, both intermediate and final, and if the final rounding does not underflow, then

$$|D - \tilde{D}| \leq |D|/16 + 3.637 \times 10^{-3} \sum_{i=1}^{8} |X_i \times Y_i|.$$

In this example the products $X_i \times Y_i$ are exact, while intermediate underflows, if they occur, are exact.

The constant 3.637×10^{-3} is obtained by rounding up $(1+1/16)\big((1+2^{-11})^7 - 1\big)$, where the constants are obtained as follows: 11 is the accumulator precision, i.e., PrecisionOf(binary16); $7 = B - 1$ is the number of additions; and $16 = 2^{\mathsf{PrecisionOf(Binary8p4se)}}$.

Annex F

(informative)

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 [10], describing hardware implementations including nVidia, Intel, and ARM.

AGQ: AMD, Graphcore, Qualcomm[11], implemented in Graphcore's C600 product, and AMD's gfx940.

TSL: Tesla Dojo Technology [13].

Format	Binary{K, P, Σ, Δ}			OCP			AGQ		TSL	
Subformat	k8p1uf	k8p3se	k8p4se	E8M0	E5M2	E4M3	E5M2	E4M3	E5M2	E4M3
Special values shared	Y			N			Y		N	
Exactly one NaN	Y			Y	N		Y		Y	
Include negative zero	N			N	Y		N		N	
Has infinity	N	Y		N	Y	N	N		N	
Max exponent emax	126	15	7	127	15	8	15	7	N/A	N/A

Max exponent for TSL is marked as N/A given the configurable bias.

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

Annex G

(informative)

Operation groups

Operations are defined in groups as follows

Group A (Core)

Convert<f_x, f_r, ρ>
Negate<f_x, f_r, ρ>
Abs<f_x, f_r, ρ>
Add<f_x, f_y, f_r, ρ>
Subtract<f_x, f_y, f_r, ρ>
Multiply<f_x, f_y, f_r, ρ>
FMA<f_x, f_y, f_z, f_r, ρ>
FAA<f_x, f_y, f_z, f_r, ρ>
Minimum<f_x, f_y, f_r, ρ>
Maximum<f_x, f_y, f_r, ρ>
MinimumNumber<f_x, f_y, f_r, ρ>
MaximumNumber<f_x, f_y, f_r, ρ>
MinimumMagnitude<f_x, f_y, f_r, ρ>
MaximumMagnitude<f_x, f_y, f_r, ρ>
MinimumMagnitudeNumber<f_x, f_y, f_r, ρ>
MaximumMagnitudeNumber<f_x, f_y, f_r, ρ>
MinimumFinite<f_x, f_y, f_r, ρ>
MaximumFinite<f_x, f_y, f_r, ρ>
Clamp<$f_x, f_{lo}, f_{hi}, f_r, \rho$>
CompareLess<f_x, f_y>
CompareLessEqual<f_x, f_y>
CompareEqual<f_x, f_y>
CompareGreater<f_x, f_y>
CompareGreaterEqual<f_x, f_y>
IsZero<f>
IsOne<f>
IsNaN<f>
IsInfinite<f>
IsFinite<f>
IsSignMinus<f>
IsNormal<f>
IsSubnormal<f>

Group KA (Block Core)

ConvertFromBlock<B, f_s, f_x, f_r, ρ>
ConvertToBlock<B, f_x, f_s, f_r, ρ>
ConvertToBlockMaxAbsFinite<$B, f_x, f_s, f_r, \rho_s, \rho$>
BlockConvert<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockNegate<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockAbs<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockAdd<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockSubtract<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMultiply<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockFMA<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_{s3}, f_{x3}, f_s, f_r, \rho$>
BlockFAA<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_{s3}, f_{x3}, f_s, f_r, \rho$>
BlockMinimum<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMaximum<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMinimumNumber<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMaximumNumber<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMinimumMagnitude<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMaximumMagnitude<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMinimumMagnitudeNumber<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMaximumMagnitudeNumber<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMinimumFinite<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockMaximumFinite<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockClamp<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_{s3}, f_{x3}, f_s, f_r, \rho$>

Group M (Meta)

BitwidthOf<f>
PrecisionOf<f>
SignednessOf<f>
DomainOf<f>
ExponentBitwidthOf<f>
TrailingSignificandBitwidthOf<f>
ExponentBiasOf<f>
MaxFiniteOf<f>
MinFiniteOf<f>
MinPositiveOf<f>
MaxSubnormalOf<f>
MinNormalOf<f>

Group B (Basic)

Divide<f_x, f_y, f_r, ρ>
Sqrt<f_x, f_r, ρ>
Recip<f_x, f_r, ρ>
RSqrt<f_x, f_r, ρ>
Exp<f_x, f_r, ρ>
Exp2<f_x, f_r, ρ>
ExpMinusOne<f_x, f_r, ρ>
Log<f_x, f_r, ρ>
Log2<f_x, f_r, ρ>
LogOnePlus<f_x, f_r, ρ>
Softplus<f_x, f_r, ρ>
NextGreaterThan<f>
NextLessThan<f>
CopySign<f_x, f_y, f_r, ρ>
TotalOrder<f_x, f_y>
Class<f>

Group C (Full)

Hypot<f_x, f_y, f_r, ρ>
ArcTan2<f_y, f_x, f_r, ρ>
ArcTan2Pi<f_y, f_x, f_r, ρ>
Sin<f_x, f_r, ρ>
Cos<f_x, f_r, ρ>
Tan<f_x, f_r, ρ>
ArcSin<f_x, f_r, ρ>
ArcCos<f_x, f_r, ρ>
ArcTan<f_x, f_r, ρ>
Sinh<f_x, f_r, ρ>
Cosh<f_x, f_r, ρ>
Tanh<f_x, f_r, ρ>
ArcSinh<f_x, f_r, ρ>
ArcCosh<f_x, f_r, ρ>
ArcTanh<f_x, f_r, ρ>
SinPi<f_x, f_r, ρ>
CosPi<f_x, f_r, ρ>
TanPi<f_x, f_r, ρ>
ArcSinPi<f_x, f_r, ρ>
ArcCosPi<f_x, f_r, ρ>
ArcTanPi<f_x, f_r, ρ>

Group KB (Block Basic)

BlockDivide<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockSqrt<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockRecip<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockRSqrt<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockExp<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockExp2<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockExpMinusOne<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockLog<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockLog2<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockLogOnePlus<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockSoftplus<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockCopySign<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>

Group KC (Block Full)

BlockHypot<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockArcTan2<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockArcTan2Pi<$B, f_{s1}, f_{x1}, f_{s2}, f_{x2}, f_s, f_r, \rho$>
BlockSin<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockCos<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockTan<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcSin<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcCos<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcTan<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockSinh<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockCosh<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockTanh<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcSinh<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcCosh<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcTanh<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockSinPi<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockCosPi<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockTanPi<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcSinPi<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcCosPi<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>
BlockArcTanPi<$B, f_{s1}, f_{x1}, f_s, f_r, \rho$>

Group E (Block Reductions)

BlockReduceAdd<B, f_s, f_x, f_r, ρ>
BlockReduceMultiply<B, f_s, f_x, f_r, ρ>
BlockDotProduct<$B, f_{sx}, f_x, f_{sy}, f_y, f_r, \rho$>

Annex H

(informative)

Bibliography

References

[1] P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh, "Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores," *SIAM Journal on Scientific Computing*, vol. 42, no. 3, pp. C124–C141, 2020.

[2] 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*. IEEE, 2025, pp. 133–140. [Online]. Available: https://doi.org/10.1109/ARITH64983.2025.00029

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

[4] Google, "Jax lax package: _float_to_int_for_sort ," https://github.com/google/jax/blob/fc5960f2b8/jax/_src/lax/lax.py#L3934.

[5] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. SIAM, 2002.

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

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "Cloud TPU: Machine learning accelerators for training and inference," *IEEE Micro*, vol. 38, no. 2, pp. 39–47, 2018.

[8] 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://www.arithmazium.org/library/lib/wk_branch_cuts_86.pdf.

[9] W. Kahan and J. W. Thomas, "Augmenting a programming language with complex arithmetic," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-92-667, 1991, https://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/6127.html.

[10] 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) revision 1.0," opencompute.org, Tech. Rep., 2023.

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

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

[13] 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.

[14] 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.*

IEEE, 2025, https://github.com/imandra-ai/ieee-p3109.

[15] R. Zhao, B. Vogel, and T. Ahmed, “Adaptive loss scaling for mixed precision training,” arXiv cs.LG, Tech. Rep., 2019, https://arxiv.org/abs/1910.12385.