

# 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  
(compiled 2026-01-05)

## DRAFT DOCUMENT

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

Copyright © 2025 by The Institute of Electrical and Electronics Engineers, Inc.  
Three Park Avenue  
New York, New York 10016-5997, USA  
All rights reserved.

This document is subject to change. USE AT YOUR OWN RISK! IEEE copyright statements SHALL NOT BE REMOVED from this draft, or modified in any way. Because this is an unapproved draft, this document must not be utilized for conformance / compliance purposes.

# Contents

<b>1 Contributors</b>	<b>5</b>
<b>2 Introduction</b>	<b>6</b>
2.1 Typographical conventions and notation . . . . .	6
2.2 Scope . . . . .	6
<b>3 Formats</b>	<b>7</b>
3.1 Format parameters and value sets . . . . .	7
3.1.1 Naming . . . . .	7
3.2 Exponent bias . . . . .	8
3.3 Subnormals . . . . .	8
3.4 Not a number (NaN) . . . . .	8
3.5 Zero . . . . .	9
3.6 Infinities . . . . .	9
3.7 Encoding . . . . .	10
<b>4 Operations</b>	<b>12</b>
4.1 Notation and definitions . . . . .	12
4.1.1 Mathematical notations . . . . .	12
4.1.2 The set of closed extended reals . . . . .	12
4.1.3 Use of integer arithmetic . . . . .	12
4.2 Projection specifications . . . . .	13
4.3 Scaled operations . . . . .	13
4.4 Non-requirement for operator side effects . . . . .	13
4.5 Format-level operations . . . . .	14
4.6 Approximate implementations . . . . .	15
4.7 Operation variants and superset specifications . . . . .	16
4.8 Operator definition template . . . . .	17
4.9 Functions . . . . .	18
4.9.1 Decode . . . . .	18
4.9.2 Project . . . . .	19
4.9.3 RoundToPrecision . . . . .	20
4.9.4 Saturate . . . . .	22
4.9.5 Encode . . . . .	24
4.10 Interconverting formats . . . . .	25
4.11 Arithmetic operations . . . . .	26
4.11.1 Absolute Value, Negation . . . . .	26
4.11.2 CopySign . . . . .	27
4.11.3 Addition, Subtraction . . . . .	28
4.11.4 Multiplication . . . . .	29
4.11.5 Division . . . . .	30
4.11.6 Fused Multiply-Add . . . . .	31
4.11.7 Fused Add-Add . . . . .	32
4.11.8 Square root, Reciprocal, Reciprocal square root . . . . .	33
4.11.9 Logarithm, Exponentiation . . . . .	34
4.11.10 Trigonometric functions . . . . .	35
4.11.11 Hyperbolic functions . . . . .	36
4.11.12 Trigonometric Pi-functions . . . . .	37
4.11.13 Softplus . . . . .	38
4.11.14 Hypotenuse . . . . .	39
4.11.15 Inverse tangent of two variables . . . . .	40
4.11.16 Inverse tangent of two variables (Pi-variant) . . . . .	41

4.12	Extrema . . . . .	42
4.12.1	Minimum, Maximum, MinimumNumber, and MaximumNumber . . . . .	42
4.12.2	MinimumMagnitude, MaximumMagnitude, and ‘Number’ variants . . . . .	43
4.12.3	MinimumFinite, MaximumFinite . . . . .	44
4.12.4	Clamp . . . . .	45
4.13	Comparisons . . . . .	46
4.14	Predicates and classification . . . . .	48
4.14.1	Classifier operation . . . . .	49
4.14.2	Total order predicate . . . . .	50
4.14.3	Next greater than and next less than . . . . .	51
4.14.4	Comparison predicates . . . . .	52
<b>5</b>	<b>Block operations</b>	<b>53</b>
5.1	Decode and Project Blocks . . . . .	53
5.1.1	BlockDecode . . . . .	53
5.1.2	BlockProject . . . . .	54
5.2	Convert Blocks . . . . .	55
5.2.1	ConvertFromBlock . . . . .	55
5.2.2	ConvertToBlock . . . . .	56
5.2.3	ConvertToBlockMaxAbsFinite . . . . .	57
5.3	Block reduction operations . . . . .	58
5.3.1	Sum and product . . . . .	58
5.3.2	Dot product . . . . .	59
5.4	Elementwise operations on blocks . . . . .	60
5.5	Scaled operations via block operations . . . . .	61
<b>6</b>	<b>Interconverting formats</b>	<b>62</b>
6.1	Conversion from IEEE Std 754 formats to P3109 . . . . .	63
6.2	Conversion from P3109 to IEEE Std 754 . . . . .	64
	<b>Appendices</b>	<b>65</b>
<b>A</b>	<b>Rationales</b>	<b>65</b>
A.1	Use of infinity in computation of attention masks . . . . .	65
<b>B</b>	<b>External Formats</b>	<b>66</b>
<b>C</b>	<b>Value Tables for K=2</b>	<b>66</b>
	<b>References</b>	<b>67</b>

# 1 Contributors

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

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

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

## *Editors*

Jeffrey Sarnoff (Editor in Chief), Andrew Fitzgibbon, Christoph M. Wintersteiger

## *Contributing Editors*

Amos Omondi, Guy Lemieux

## *Contributors*

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

## *Contacting The Working Group*

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

## 2 Introduction

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

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

**Need for this Work:** Machine Learning Systems have different arithmetic requirements from most other domains. Precisions tend to be lower, and accuracy is measured in dimensions other than just numerical (e.g. inference accuracy). Furthermore, machine learning systems are often integrated into mission-critical and safety-critical systems. With no standards specifically addressing these needs, Machine Learning Systems are built with inconsistent expectations and assumptions that hinder the compatibility and reuse of machine learning hardware, software, and training data.

**Stakeholders for the Standard:** System developers, vendors, and users of machine learning applications across many industries and interests including but not limited to computation, storage, medical, telecommunications, e-commerce, fleet management, automotive, robotics, and security.

### 2.1 Typographical conventions and notation

**Bold text** describes the decisions and specifications of this document.

### 2.2 Scope

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

Binary formats are parameterized by: bitwidth, precision, signedness, and domain.

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

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

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

## 3 Formats

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

The set of values in existing IEEE Std 754 floating-point usage is the finite reals, the non-finite values positive and negative infinity (Inf, -Inf), the value negative zero (-0), and not-a-number values (NaN, NaN<sub>1</sub>, ...).

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

Some values are considered “special”. The P3109 special values are zero, Inf, -Inf, and a single NaN.

### 3.1 Format parameters and value sets

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

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

**Formats shall be defined by the parameters of bitwidth, precision, signedness, and domain.**

**The bitwidth K shall be at least three.**

**The precision P shall be at least one.**

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

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

#### 3.1.1 Naming

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

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

Examples:

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

## 3.2 Exponent bias

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

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

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

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

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

## 3.3 Subnormals

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

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

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

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

## 3.4 Not a number (NaN)

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

Many other floating-point formats define several NaN values which are returned from operations with results outside the set of values, e.g. division of zero by zero, or addition of positive and negative infinities. Multiple NaN encodings are used in other formats to allow different exceptional conditions to be distinguished.

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

- Debugging of code running on accelerator hardware. In machine learning accelerators, exceptions may be difficult or expensive to convey back to user code, so it is common practice to allow NaN values to propagate through calculations to indicate that an error has occurred.
- Use as a sentinel value. In some datasets, for example, where individual element values may be missing or out of range, a sentinel may be used to record the position of these values. In many cases, this will require less memory than storing such information out-of-band, such as in a coordinate-list (COO) format array. In some cases,  $\pm\text{Inf}$  can be used as a missing value, but given the restricted range of P3109 formats, it is likely that infinity shall be used as a separate indicator of rounding from values outside of the finite range.

The use of multiple NaN payloads is known in statistical code (e.g. the R system has NaN and N/A), but it is not widely used. In the context of P3109, supporting multiple NaNs would reduce the already limited encoding space (e.g. occupying all code points where the exponent field is all ones, thereby reducing dynamic range) and would likely add additional hardware complexity.

## 3.5 Zero

**P3109 formats shall have exactly one zero.**

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

A key rationale for including  $-0$  in IEEE Std 754 was the consistent implementation of branch cuts in the ArcTan2 function and the complex trigonometric functions [6, 7]. The ArcTan2 function is rare, if not unknown, 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 Std 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, as described below.

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

## 3.6 Infinities

**Unsigned formats in the extended domain shall include positive infinity.**

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

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

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

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



### 3.7 Encoding

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

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

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

Formats in the extended domain 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 1 summarizes the encodings of selected special values.

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

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

Format	Minimum Positive	Minimum Normal	Maximum Finite
Binary8p1se	$1 \times 2^{-63}$	$1 \times 2^{-63}$	$1 \times 2^{62}$
Binary8p2se	$1 \times 2^{-32}$	$1 \times 2^{-31}$	$1 \times 2^{31}$
Binary8p3se	$1 \times 2^{-17}$	$1 \times 2^{-15}$	$3/2 \times 2^{15}$
Binary8p4se	$1 \times 2^{-10}$	$1 \times 2^{-7}$	$7/4 \times 2^7$
Binary8p5se	$1 \times 2^{-7}$	$1 \times 2^{-3}$	$15/8 \times 2^3$
Binary8p6se	$1 \times 2^{-6}$	$1 \times 2^{-1}$	$31/16 \times 2^1$
Binary8p7se	$1 \times 2^{-6}$	$1 \times 2^0$	$63/32 \times 2^0$

Table 2: Extremal values: examples for  $K = 8$ .

### 3.7.1 Value tables

Tables 3 and 4 show the complete value sets for  $K = 4$ .

Code point	Binary4p1se	Binary4p2se	Binary4p3se	Binary4p1sf	Binary4p2sf	Binary4p3sf
0 0b0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1 0b0001	0.1250	0.2500	0.2500	0.1250	0.2500	0.2500
2 0b0010	0.2500	0.5000	0.5000	0.2500	0.5000	0.5000
3 0b0011	0.5000	0.7500	0.7500	0.5000	0.7500	0.7500
4 0b0100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5 0b0101	2.0000	1.5000	1.2500	2.0000	1.5000	1.2500
6 0b0110	4.0000	2.0000	1.5000	4.0000	2.0000	1.5000
7 0b0111	Inf	Inf	Inf	8.0000	3.0000	1.7500
8 0b1000	NaN	NaN	NaN	NaN	NaN	NaN
9 0b1001	-0.1250	-0.2500	-0.2500	-0.1250	-0.2500	-0.2500
10 0b1010	-0.2500	-0.5000	-0.5000	-0.2500	-0.5000	-0.5000
11 0b1011	-0.5000	-0.7500	-0.7500	-0.5000	-0.7500	-0.7500
12 0b1100	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000
13 0b1101	-2.0000	-1.5000	-1.2500	-2.0000	-1.5000	-1.2500
14 0b1110	-4.0000	-2.0000	-1.5000	-4.0000	-2.0000	-1.5000
15 0b1111	-Inf	-Inf	-Inf	-8.0000	-3.0000	-1.7500

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

Code point	Bny4p1ue	Bny4p2ue	Bny4p3ue	Bny4p4ue	Bny4p1uf	Bny4p2uf	Bny4p3uf	Bny4p4uf
0 0b0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
1 0b0001	$\approx 0.0078$	0.0625	0.1250	0.1250	$\approx 0.0078$	0.0625	0.1250	0.1250
2 0b0010	$\approx 0.0156$	0.1250	0.2500	0.2500	$\approx 0.0156$	0.1250	0.2500	0.2500
3 0b0011	$\approx 0.0312$	0.1875	0.3750	0.3750	$\approx 0.0312$	0.1875	0.3750	0.3750
4 0b0100	0.0625	0.2500	0.5000	0.5000	0.0625	0.2500	0.5000	0.5000
5 0b0101	0.1250	0.3750	0.6250	0.6250	0.1250	0.3750	0.6250	0.6250
6 0b0110	0.2500	0.5000	0.7500	0.7500	0.2500	0.5000	0.7500	0.7500
7 0b0111	0.5000	0.7500	0.8750	0.8750	0.5000	0.7500	0.8750	0.8750
8 0b1000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9 0b1001	2.0000	1.5000	1.2500	1.1250	2.0000	1.5000	1.2500	1.1250
10 0b1010	4.0000	2.0000	1.5000	1.2500	4.0000	2.0000	1.5000	1.2500
11 0b1011	8.0000	3.0000	1.7500	1.3750	8.0000	3.0000	1.7500	1.3750
12 0b1100	16.0000	4.0000	2.0000	1.5000	16.0000	4.0000	2.0000	1.5000
13 0b1101	32.0000	6.0000	2.5000	1.6250	32.0000	6.0000	2.5000	1.6250
14 0b1110	Inf	Inf	Inf	Inf	64.0000	8.0000	3.0000	1.7500
15 0b1111	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 4: Value sets for  $K = 4$ , unsigned formats with  $1 \leq P \leq 4$ , extended and finite domains. Subnormals are shown in blue, special values in brown. For compactness, “Binary” has been abbreviated to “Bny”.

## 4 Operations

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

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

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

An implementation of any of the operations defined herein is conforming if it computes the same output as does the defined operation, for all possible inputs. This may be attested by any proof method, including direct computation. Section 4.6 discusses approximate implementations.

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

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

### 4.1 Notation and definitions

#### 4.1.1 Mathematical notations

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

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

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

Comments are introduced with an em-dash, and are right-justified.

—An example comment

#### 4.1.2 The set of closed extended reals

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

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

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

#### 4.1.3 Use of integer arithmetic

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

## 4.2 Projection specifications

Operations on P3109 values are defined via conversion to closed extended real values, on which the mathematical operation is performed, before conversion back to the appropriate P3109 value set. In general, operation results will not be exact P3109 values, and hence will be *projected* into the P3109 value set via rounding and overflow handling. A *projection specification* is a pair (rounding mode, saturation mode).

The defined rounding modes follow. The specifications of these modes are in the function  $\omega\text{RoundToPrecision}$  (§4.9.2):

NearestTiesToEven	Round to nearest, ties to even
NearestTiesToAway	Round to nearest, ties away from zero
TowardPositive	Round toward positive
TowardNegative	Round toward negative
TowardZero	Round toward zero
ToOdd	Round to odd
Stochastic[A, B, C]	Stochastic rounding, with variants A, B, and C as defined in §4.9.3

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

The defined saturation modes are as follows. Their specifications are given in the function  $\omega\text{Saturate}$  (§4.9.4):

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

## 4.3 Scaled operations

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

## 4.4 Non-requirement for operator side effects

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

## 4.5 Format-level operations

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

The following operations return integers or enumerations:

Operation	Description
$\text{BitwidthOf}(f)$	Width in bits of format $f$ , such that $\text{BitwidthOf}(\text{Binary}\{K, P, \Sigma, \Delta\}) = K$ .
$\text{PrecisionOf}(f)$	Precision of format $f$ , such that $\text{PrecisionOf}(\text{Binary}\{K, P, \Sigma, \Delta\}) = P$ .
$\text{SignednessOf}(f)$	Signedness of format $f$ , such that $\text{SignednessOf}(\text{Binary}\{K, P, \Sigma, \Delta\}) = \Sigma$ .
$\text{DomainOf}(f)$	Domain of format $f$ , such that $\text{DomainOf}(\text{Binary}\{K, P, \Sigma, \Delta\}) = \Delta$ .
$\text{ExponentBitsOf}(f)$	Number of exponent bits of format $f$ , such that $\text{ExponentBitsOf}(\text{Binary}\{K, P, \text{Signed}, \Delta\}) = K - P$ , $\text{ExponentBitsOf}(\text{Binary}\{K, P, \text{Unsigned}, \Delta\}) = K - P + 1$ .
$\text{TrailingBitsOf}(f)$	Number of trailing significand bits of format $f$ , such that $\text{TrailingBitsOf}(\text{Binary}\{K, P, \Sigma, \Delta\}) = P - 1$ .
$\text{ExponentBiasOf}(f)$	Exponent bias of format $f$ such that $\text{ExponentBiasOf}(\text{Binary}\{K, P, \text{Signed}, \Delta\}) = 2^{K-P-1}$ , $\text{ExponentBiasOf}(\text{Binary}\{K, P, \text{Unsigned}, \Delta\}) = 2^{K-P}$ .

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

Operation	Description
$\text{MaxFiniteOf}(f)$	Maximum finite value representable in format $f$ .
$\text{MinFiniteOf}(f)$	Minimum finite value representable in format $f$ . In signed formats, $\text{MinFiniteOf}(f) = -\text{MaxFiniteOf}(f)$ . In unsigned formats, $\text{MinFiniteOf}(f) = 0$ .
$\text{MinPositiveOf}(f)$	Minimum strictly positive value representable in format $f$ .
$\text{MinNormalOf}(f)$	Minimum positive normal value representable in format $f$ .

These operations query projection specifications:

$\text{RoundOf}(\rho)$	Rounding mode of projection specification $\rho$ , such that $\text{RoundOf}((r, s)) = r$ .
$\text{SatOf}(\rho)$	Saturation mode of projection specification $\rho$ , such that $\text{SatOf}((r, s)) = s$ .

## 4.6 Approximate implementations

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

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

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

The value of  $\kappa$  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,

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

The value of  $\kappa$  will in general be specific to each operation and parameters, for example a system may supply implementations of  $\text{Add}_{f_x, f_y, f_r, \rho}$  where  $\kappa$  depends on  $f_r$ . These implementations shall be named differently. For each  $\kappa$ -approximate implementation of an (operator, parameters) pair,  $\kappa$  shall be specified. Such a specification may be over  $I$  or over a covering union of disjoint subsets of  $I$ . This may be attested by any proof method, including direct computation.

## 4.7 Operation variants and superset specifications

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

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

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

For example, an implementation might declare as follows:

This implementation provides the following operations:

- $\text{Add}_{f_x, f_y, f_r, \rho}$ ,  
with  $\{f_x, f_y\} \subset \{\text{Binary8p3se}, \text{Binary8p3sf}, \text{Binary8p4se}, \text{Binary8p4sf}\}$   
and  $f_r \in \{\text{Binary8p3se}, \text{Binary8p4se}\}$
- $\text{BlockAdd}_{B, f_s, f_x, f_s, f_y, f_s, f_r, \rho}$ ,  
with  $B \in \{16, 32\}$   
and  $f_s \in \{\text{Binary8p1uf}, \text{Binary8p4uf}\}$   
and  $f_x \in \{\text{Binary6p3sf}, \text{Binary4p2sf}\}$   
and  $f_y \in \{\text{Binary6p3sf}, \text{Binary4p2sf}\}$   
and  $f_r \in \{\text{Binary8p3se}, \text{Binary8p4se}\}$
- $\text{FMA}_{f_x, f_y, f_z, f_r, \rho}$ ,  
with  $(f_x, f_y) \in \{(\text{Binary8p3se}, \text{Binary8p3se}), (\text{Binary8p3se}, \text{Binary8p4se}), (\text{Binary8p4se}, \text{Binary8p3se})\}$   
and  $f_z \in \{\text{Binary8p3se}, \text{Binary8p4se}\}$ ,  
and  $f_r = f_z$ ,  
with  $\text{PrecisionOf}(f_z) \geq \max(\text{PrecisionOf}(f_x), \text{PrecisionOf}(f_y))$ .
- $\text{ConvertFromIEEE754}_{\phi, f, \rho}$ ,  
with  $\phi \in \{\text{Binary16}, \text{Binary32}\}$   
and  $f \in \{\text{Binary8p3se}, \text{Binary8p4se}, \text{Binary8p5se}\}$

In all cases, projection specifications  $\rho$  obey  $\text{RoundOf}(\rho) \in \{\text{NearestTiesToEven}, \text{NearestTiesToAway}\}$  and  $\text{SatOf}(\rho) \in \{\text{SatFinite}, \text{OvfInf}\}$ .

$\text{ConvertFromIEEE754}$  implements  $\text{RoundOf}(\rho) \in \{\text{NearestTiesToEven}, \text{StochasticA}_{23,*}, \text{StochasticA}_{16,*}\}$ .

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

As an example, an implementation might declare the supply of an exponentiation operation which operates on 8-bit values with a precision of 3 or 4, with output precision always 4, and saturation or overflow to infinity, as follows:

The instruction `exp8.px.sat` has the following variants: `px` in `\{3,4\}` and `sat` in `\{finite,inf\}`.

These map to the P3109 Exp operation variants as follows:

```
exp8.3.finite:
  Exp{Binary{8,3,Signed,Extended}, Binary{8,4,Signed,Extended}, (NearestTiesToEven, SatFinite)}

exp8.4.finite:
  Exp{Binary{8,4,Signed,Extended}, Binary{8,4,Signed,Extended}, (NearestTiesToEven, SatFinite)}

exp8.3.inf:
  Exp{Binary{8,3,Signed,Extended}, Binary{8,4,Signed,Extended}, (NearestTiesToEven, OvfInf)}

exp8.4.inf:
  Exp{Binary{8,4,Signed,Extended}, Binary{8,4,Signed,Extended}, (NearestTiesToEven, OvfInf)}
```

## 4.8 Operator definition template

Operations are defined according to the following template:

### Signature

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

—Naming the operator, its parameters, operands and result.

### Parameters

—Parameters specify a family of related operations

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

### Operands

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

### Result

$r$  : result value, in format  $f_r$

—Result value

### Behavior

—An ordered sequence of pattern-matching declarations.

$\omega\text{Operator}(X, Y) \rightarrow \dots$

—Auxiliary definition, in the closed extended reals.

$\text{Operator}(\text{NaN}, 0) \rightarrow \text{NaN}$

—Exact pattern: only the provided operands match.

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

—Match for all  $x$  values in the given set.

$\text{Operator}(*, 0) \rightarrow 0$

—The  $*$  symbol matches any value.

$\text{Operator}(x, y) \rightarrow \text{Operator}_{f_x, f_y, f_r}(x, y)$

—Operators in pattern RHS have explicit parameters.

### Notes

Notes on the operation.

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

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

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

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

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



## 4.9 Internal functions

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

### 4.9.1 Decode

#### Signature

$$\begin{aligned}\omega\text{Decode}_f(x) &\rightarrow X \\ \omega\text{DecodeAux}_K(\sigma, \Delta, x) &\rightarrow X\end{aligned}$$

#### Parameters

$f$  : format of  $x$

#### Operands

$x$  : P3109 value, in format  $f$

#### Result

$X$  : closed extended real value

#### Behavior

$$\omega\text{Decode}(x) = \omega\text{DecodeAux}_{\text{BitwidthOf}(f)}(\text{SignednessOf}(f), \text{DomainOf}(f), x)$$

$$\omega\text{DecodeAux}(\text{Signed}, *, 2^{K-1}) \rightarrow \text{NaN}$$

$$\omega\text{DecodeAux}(\text{Unsigned}, *, 2^K - 1) \rightarrow \text{NaN}$$

$$\omega\text{DecodeAux}(\text{Signed}, \text{Extended}, 2^{K-1} - 1) \rightarrow +\infty$$

$$\omega\text{DecodeAux}(\text{Signed}, \text{Extended}, 2^K - 1) \rightarrow -\infty$$

$$\omega\text{DecodeAux}(\text{Unsigned}, \text{Extended}, 2^{K-1} - 2) \rightarrow +\infty$$

$$\omega\text{DecodeAux}(\text{Signed}, \Delta, 2^{K-1} < x < 2^K) \rightarrow -\omega\text{DecodeAux}(\text{Signed}, \Delta, x - 2^{K-1})$$

$$\omega\text{DecodeAux}(*, *, x) \rightarrow X$$

where

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

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

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

and

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

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

## 4.9.2 Project

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

### Signature

$$\omega\text{Project}_{f,\rho}(X) \rightarrow x$$

### Parameters

$f$  : target format

$\rho$  : projection specification

### Operands

$X$  : closed extended real value

### Result

$x$  : P3109 value, format  $f$

### Behavior

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

$$\omega\text{Project}(X) \rightarrow x$$

#### where

$$R = \omega\text{RoundToPrecision}_{\text{PrecisionOf}(f), \text{ExponentBiasOf}(f), \text{RoundOf}(\rho)}(X)$$

$$S = \omega\text{Saturate}_{\text{MaxFinite}}(\text{SatOf}(\rho), \text{RoundOf}(\rho), R, \sigma, \Delta)$$

$$x = \omega\text{Encode}_f(S)$$

#### and

$$\sigma = \text{SignednessOf}(f)$$

$$\Delta = \text{DomainOf}(f)$$

$$\text{MaxFinite} = \omega\text{Decode}_f(\text{MaxFiniteOf}(f))$$

### Notes

It is an error if  $\text{SatOf}(\rho) \neq \text{SatFinite}$  if  $\Delta = \text{Finite}$ .

In  $\omega\text{Saturate}$ , all values above  $M = \text{MaxFinite}$  are sent to either  $M$  or  $+\infty$ . Nevertheless, for rounding to nearest,  $\omega\text{Project}$  has the property, shared with IEEE Std 754, that values between  $M$  and the number half a unit in the last place above  $M$  will project to  $M$  because  $\omega\text{RoundToPrecision}$  precedes saturation.

Under NearestTiesToEven rounding, subnormals are handled following IEEE Std 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.

### 4.9.3 RoundToPrecision

Convert real value to real value representable with a given precision and unbounded exponent.

#### Signature

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

#### Parameters

$P$  : integer precision

$B$  : exponent bias

$\mu$  : rounding mode. For stochastic rounding, random bits  $R$ , an integer in the range  $0 \leq R < 2^N$ , are to be supplied as noted below.

#### Operands

$X$  : real value

#### Result

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

#### Behavior

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

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

where

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

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

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

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

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

$$\text{RoundAway}(\text{TowardZero}) = \text{False}$$

$$\text{RoundAway}(\text{TowardPositive}) = \eta > 0 \text{ and } X > 0$$

$$\text{RoundAway}(\text{TowardNegative}) = \eta > 0 \text{ and } X < 0$$

$$\text{RoundAway}(\text{NearestTiesToAway}) = \eta \geq 0.5$$

$$\text{RoundAway}(\text{NearestTiesToEven}) = \eta > 0.5 \text{ or } (\eta = 0.5 \text{ and not CodelsEven})$$

$$\text{RoundAway}(\text{ToOdd}) = \eta > 0 \text{ and CodelsEven}$$

$$\text{RoundAway}(\text{StochasticA}_{N,R}) = \lfloor \eta \times 2^N \rfloor + R \geq 2^N$$

$$\text{RoundAway}(\text{StochasticB}_{N,R}) = \lfloor \eta \times 2^{N+1} \rfloor + (2 \times R + 1) \geq 2^{N+1}$$

$$\text{RoundAway}(\text{StochasticC}_{N,R}) = \text{RNITE}(\eta \times 2^N) + R \geq 2^N$$

and

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

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

[continued on next page]

**Notes**

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. The quality of the random bits is not specified in this document. The notation  $\text{StochasticA}_{N,*}$  is used to indicate that  $N$  random bits are supplied but not further specified.

One or more variants of  $\text{StochasticA}$ ,  $\text{StochasticB}$ ,  $\text{StochasticC}$  may be supplied. Roughly, these variants are sorted in order of decreasing bias and increasing complexity [12].

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

#### 4.9.4 Saturate

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

##### Signature

$$\omega\text{Saturate}_M(\text{Sat}, \text{Round}, X, \sigma, \Delta) \rightarrow Z$$

##### Parameters

$M$  : maximum finite value

##### Operands

Sat : saturation mode

Round : rounding mode

$X$  : closed extended real value

$\sigma$  : signedness in {Signed, Unsigned}

$\Delta$  : domain {Finite, Extended}

##### Result

$Z$  : closed extended real value

##### Behavior

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

where

$$M^{lo} = \begin{cases} -M & \text{if } \sigma = \text{Signed} \\ 0 & \text{if } \sigma = \text{Unsigned} \end{cases}$$

$$M^{hi} = M \quad \text{—Note } M \text{ is a finite real}$$

### 4.9.5 Encode

Encode closed extended real value to P3109 format  $f$ .  $\omega\text{Encode}$  must be applied only to a value which is in the value set of format  $f$ . Such a value may be produced by, for example  $\omega\text{RoundToPrecision}$  and  $\omega\text{Saturate}$ , or the input may be known to be in the value set, for example, the absolute value of a value already in the set.

#### Signature

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

#### Parameters

$f$  : target format

#### Operands

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

#### Result

$r$  : P3109 value, format  $f$

#### Behavior

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

where

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

and

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

$$S = X \times 2^{-E} \times 2^{P-1} \quad \text{---}S \text{ is the significand}$$

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

and

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

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

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

$$\sigma = \text{SignednessOf}(f)$$

#### Notes

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

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

## 4.10 Interconverting formats

Convert a P3109 value to another P3109 format.

### Signature

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

### Parameters

$f_x$  : input format

$f_r$  : result format

$\rho$  : projection specification

### Operands

$x$  : P3109 value, format  $f_x$

### Result

$r$  : P3109 value, format  $f_r$

### Behavior

$\text{Convert}(\text{NaN}_{f_x}) \rightarrow \text{NaN}_{f_r}$

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

## 4.11 Arithmetic operations

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

### 4.11.1 Absolute Value, Negation

#### Signature

$$\begin{aligned}\text{Abs}_{f_x, f_r, \rho}(x) &\rightarrow r \\ \text{Negate}_{f_x, f_r, \rho}(x) &\rightarrow r\end{aligned}$$

#### Parameters

$f_x$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

#### Operands

$x$  : P3109 value, format  $f_x$

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

$$\begin{aligned}\omega\text{Abs}(\text{NaN}) &\rightarrow \text{NaN} \\ \omega\text{Abs}(-\infty) &\rightarrow +\infty \\ \omega\text{Abs}(+\infty) &\rightarrow +\infty \\ \omega\text{Abs}(X) &\rightarrow |X| \\ \text{Abs}(x) &\rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Abs}(\omega\text{Decode}_{f_x}(x)))\end{aligned}$$

$$\begin{aligned}\omega\text{Negate}(\text{NaN}) &\rightarrow \text{NaN} \\ \omega\text{Negate}(-\infty) &\rightarrow +\infty \\ \omega\text{Negate}(+\infty) &\rightarrow -\infty \\ \omega\text{Negate}(X) &\rightarrow -X \\ \text{Negate}(x) &\rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Negate}(\omega\text{Decode}_{f_x}(x)))\end{aligned}$$



### 4.11.2 CopySign

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

#### Signature

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

#### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_r$  : format of result  $r$   
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

$\omega\text{CopySign}(\text{NaN}, *) \rightarrow \text{NaN}$   
 $\omega\text{CopySign}(*, \text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{CopySign}(\pm\infty, +\infty) \rightarrow +\infty$   
 $\omega\text{CopySign}(\pm\infty, -\infty) \rightarrow -\infty$   
 $\omega\text{CopySign}(\pm\infty, Y) \text{ if } Y \geq 0 \rightarrow +\infty$   
 $\omega\text{CopySign}(\pm\infty, Y) \text{ if } Y < 0 \rightarrow -\infty$   
 $\omega\text{CopySign}(X, +\infty) \rightarrow |X|$   
 $\omega\text{CopySign}(X, -\infty) \rightarrow -|X|$   
 $\omega\text{CopySign}(X, Y) \text{ if } Y \geq 0 \rightarrow |X|$   
 $\omega\text{CopySign}(X, Y) \text{ if } Y < 0 \rightarrow -|X|$   
 $\text{CopySign}(X, Y) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{CopySign}(\omega\text{Decode}_{f_x}(x), \omega\text{Decode}_{f_y}(y)))$

### 4.11.3 Addition, Subtraction

Addition and subtraction of two P3109 values, returning a P3109 value.

#### Signature

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

#### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_r$  : format of  $r$   
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

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

#### 4.11.4 Multiplication

Multiplication of two P3109 values, returning a P3109 value.

##### Signature

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

##### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_r$  : format of  $r$   
 $\rho$  : projection specification

##### Operands

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

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

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

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

### 4.11.5 Division

Division of two P3109 values, returning a P3109 value.

$\text{Divide}(x, \pm\text{Inf})$  where  $x$  is finite yields 0.

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

#### Signature

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

#### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_r$  : format of  $r$   
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

### 4.11.6 Fused Multiply-Add

Compute  $R = X \times Y + Z$ , with computation in the reals.

Rounding and the determination of overflow and underflow are applied only on the result.

#### Signature

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

#### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_z$  : format of  $z$   
 $f_r$  : format of  $r$   
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

### 4.11.7 Fused Add-Add

Compute  $R = X + Y + Z$ , with computation in the reals.

Rounding and the determination of overflow and underflow are applied only on the result.

#### Signature

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

#### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_z$  : format of  $z$   
 $f_r$  : format of  $r$   
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

#### 4.11.8 Square root, Reciprocal, Reciprocal square root

##### Signature

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

##### Parameters

$f_x$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

##### Operands

$x$  : P3109 value, format  $f_x$

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

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

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

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

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

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

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

### 4.11.9 Logarithm, Exponentiation

#### Signature

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

#### Parameters

$f_x$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

#### Operands

$x$  : P3109 value, format  $f_x$

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

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

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

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

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

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

$\omega\text{LogOnePlus}(x) \rightarrow \omega\text{Log}(\omega\text{Add}(1, x))$

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

$\omega\text{ExpMinusOne}(x) \rightarrow \omega\text{Subtract}(\omega\text{Exp}(x), 1)$

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



#### 4.11.10 Trigonometric functions

##### Signature

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

##### Parameters

$f_x$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

##### Operands

$x$  : P3109 value, format  $f_x$

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

$\omega\text{Sin}(\text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{Sin}(+\infty) \rightarrow \text{NaN}$   
 $\omega\text{Sin}(-\infty) \rightarrow \text{NaN}$   
 $\omega\text{Sin}(X) \rightarrow \sin X$   
 $\text{Sin}(x) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Sin}(\omega\text{Decode}_{f_x}(x)))$

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

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

$\omega\text{Cos}(\text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{Cos}(+\infty) \rightarrow \text{NaN}$   
 $\omega\text{Cos}(-\infty) \rightarrow \text{NaN}$   
 $\omega\text{Cos}(X) \rightarrow \cos X$   
 $\text{Cos}(x) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Cos}(\omega\text{Decode}_{f_x}(x)))$

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

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

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

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

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

Note: There are no values  $X$  in the value 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 allow formal verification.

### 4.11.11 Hyperbolic functions

#### Signature

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

#### Parameters

$f_x$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

#### Operands

$x$  : P3109 value, format  $f_x$

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

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

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

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

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

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

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

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

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

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

### 4.11.12 Trigonometric Pi-functions

#### Signature

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

#### Parameters

$f_x$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

#### Operands

$x$  : P3109 value, format  $f_x$

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

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

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

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

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

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

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

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

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

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

### 4.11.13 Softplus

#### Signature

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

#### Parameters

$f_x$  : input format

$f_r$  : result format

$\rho$  : projection specification

#### Operands

$x$  : P3109 value, format  $f_x$

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

$\omega\text{Softplus}(X) \rightarrow \log_e(1 + e^X)$

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

#### 4.11.14 Hypotenuse

##### Signature

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

##### Parameters

$f_x$  : input format of  $x$   
 $f_y$  : input format of  $y$   
 $f_r$  : result format  
 $\rho$  : projection specification

##### Operands

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

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

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

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

$\omega\text{Hypot}(*, \pm\infty) \rightarrow +\infty$

$\omega\text{Hypot}(\pm\infty, *) \rightarrow +\infty$

$\omega\text{Hypot}(X, Y) \rightarrow \sqrt{|X|^2 + |Y|^2}$

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

#### 4.11.15 Inverse tangent of two variables

$\text{ArcTan2}(Y, X)$  computes the angle in radians between the positive x-axis and the point  $(X, Y)$ , in the range  $(-\pi, \pi]$ .  $\text{ArcTan2}(0, 0)$  yields NaN. It would be inconsistent to define this as 0, as the limit is undefined.

$\text{ArcTan2}(\pm\text{Inf}, \pm\text{Inf})$  yields NaN, which is consistent with  $\text{Divide}(\pm\text{Inf}, \pm\text{Inf})$ .

##### Signature

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

##### Parameters

$f_y$  : input format of  $y$   
 $f_x$  : input format of  $x$   
 $f_r$  : result format  
 $\rho$  : projection specification

##### Operands

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

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

$\omega\text{ArcTan2}(\text{NaN}, *) \rightarrow \text{NaN}$   
 $\omega\text{ArcTan2}(*, \text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{ArcTan2}(0, 0) \rightarrow \text{NaN}$   
 $\omega\text{ArcTan2}(-\infty, \pm\infty) \rightarrow \text{NaN}$   
 $\omega\text{ArcTan2}(+\infty, \pm\infty) \rightarrow \text{NaN}$   
 $\omega\text{ArcTan2}(*, +\infty) \rightarrow 0$   
 $\omega\text{ArcTan2}(0, X) \text{ if } X > 0 \rightarrow 0$   
 $\omega\text{ArcTan2}(+\infty, *) \rightarrow \frac{\pi}{2}$   
 $\omega\text{ArcTan2}(Y, 0) \text{ if } Y > 0 \rightarrow \frac{\pi}{2}$   
 $\omega\text{ArcTan2}(*, -\infty) \rightarrow \pi$   
 $\omega\text{ArcTan2}(0, X) \text{ if } X < 0 \rightarrow \pi$   
 $\omega\text{ArcTan2}(-\infty, *) \rightarrow -\frac{\pi}{2}$   
 $\omega\text{ArcTan2}(Y, 0) \text{ if } Y < 0 \rightarrow -\frac{\pi}{2}$   
 $\omega\text{ArcTan2}(Y, X) \text{ if } X > 0 \rightarrow \arctan(Y/X)$   
 $\omega\text{ArcTan2}(Y, X) \text{ if } Y > 0 \text{ and } X < 0 \rightarrow \arctan(Y/X) + \pi$   
 $\omega\text{ArcTan2}(Y, X) \text{ if } Y < 0 \text{ and } X < 0 \rightarrow \arctan(Y/X) - \pi$   
 $\text{ArcTan2}(y, x) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{ArcTan2}(\omega\text{Decode}_{f_y}(y), \omega\text{Decode}_{f_x}(x)))$

#### 4.11.16 Inverse tangent of two variables (Pi-variant)

$\text{ArcTan2}(Y, X)$  computes the angle in radians between the positive x-axis and the point  $(X, Y)$ , in the range  $(-1, +1]$ .

$\text{ArcTan2}(0, 0)$  yields NaN. It would be inconsistent to define this as 0, as the limit is undefined.

$\text{ArcTan2}(\pm\text{Inf}, \pm\text{Inf})$  yields NaN, which is consistent with  $\text{Divide}(\pm\text{Inf}, \pm\text{Inf})$ .

##### Signature

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

##### Parameters

$f_y$  : input format of  $y$

$f_x$  : input format of  $x$

$f_r$  : result format

$\rho$  : projection specification

##### Operands

$x$  : P3109 value, format  $f_x$

$y$  : P3109 value, format  $f_y$

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

$\omega\text{ArcTan2Pi}(\text{NaN}, *) \rightarrow \text{NaN}$

$\omega\text{ArcTan2Pi}(*, \text{NaN}) \rightarrow \text{NaN}$

$\omega\text{ArcTan2Pi}(0, 0) \rightarrow \text{NaN}$

$\omega\text{ArcTan2Pi}(-\infty, \pm\infty) \rightarrow \text{NaN}$

$\omega\text{ArcTan2Pi}(+\infty, \pm\infty) \rightarrow \text{NaN}$

$\omega\text{ArcTan2Pi}(*, +\infty) \rightarrow 0$

$\omega\text{ArcTan2Pi}(0, X) \text{ if } X > 0 \rightarrow 0$

$\omega\text{ArcTan2Pi}(+\infty, *) \rightarrow 1/2$

$\omega\text{ArcTan2Pi}(Y, 0) \text{ if } Y > 0 \rightarrow 1/2$

$\omega\text{ArcTan2Pi}(*, -\infty) \rightarrow 1$

$\omega\text{ArcTan2Pi}(0, X) \text{ if } X < 0 \rightarrow 1$

$\omega\text{ArcTan2Pi}(-\infty, *) \rightarrow -1/2$

$\omega\text{ArcTan2Pi}(Y, 0) \text{ if } Y < 0 \rightarrow -1/2$

$\omega\text{ArcTan2Pi}(Y, X) \text{ if } X > 0 \rightarrow \arctan(Y/X)/\pi$

$\omega\text{ArcTan2Pi}(Y, X) \text{ if } Y > 0 \text{ and } X < 0 \rightarrow (\arctan(Y/X) + \pi)/\pi$

$\omega\text{ArcTan2Pi}(Y, X) \text{ if } Y < 0 \text{ and } X < 0 \rightarrow (\arctan(Y/X) - \pi)/\pi$

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

## 4.12 Extrema

### 4.12.1 Minimum, Maximum, MinimumNumber, and MaximumNumber

Minimum and maximum, with or without propagation of NaN.

#### Signature

Operation <sub>$f_x, f_y, f_r, \rho$</sub> ( $x, y$ )  $\rightarrow r$

#### Parameters

$f_x$  : input format  
 $f_y$  : input format  
 $f_r$  : result format  
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

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

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

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

$\omega\text{MinimumNumber}(\text{NaN}, \text{NaN}) \rightarrow \text{NaN}$	$\omega\text{MaximumNumber}(\text{NaN}, \text{NaN}) \rightarrow \text{NaN}$
$\omega\text{MinimumNumber}(X, \text{NaN}) \rightarrow X$	$\omega\text{MaximumNumber}(X, \text{NaN}) \rightarrow X$
$\omega\text{MinimumNumber}(\text{NaN}, Y) \rightarrow Y$	$\omega\text{MaximumNumber}(\text{NaN}, Y) \rightarrow Y$
$\omega\text{MinimumNumber}(X, Y) \rightarrow \omega\text{Minimum}(X, Y)$	$\omega\text{MaximumNumber}(X, Y) \rightarrow \omega\text{Maximum}(X, Y)$

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

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



## 4.12.2 MinimumMagnitude, MaximumMagnitude, and ‘Number’ variants

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

### Signature

Operation <sub>$f_x, f_y, f_r, \rho$</sub> ( $x, y$ )  $\rightarrow r$

### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_r$  : format of result  $r$   
 $\rho$  : projection specification

### Operands

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

### Result

$r$  : P3109 value, format  $f_r$

### Behavior

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

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

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

$\omega\text{MinimumMagnitudeNumber}(x, \text{NaN}) \rightarrow x$

$\omega\text{MinimumMagnitudeNumber}(\text{NaN}, y) \rightarrow y$

$\omega\text{MinimumMagnitudeNumber}(x, y) \rightarrow \omega\text{MinimumMagnitude}(x, y)$

$\text{MinimumMagnitudeNumber}(x, y) \rightarrow$

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

$\omega\text{MaximumMagnitudeNumber}(x, \text{NaN}) \rightarrow x$

$\omega\text{MaximumMagnitudeNumber}(\text{NaN}, y) \rightarrow y$

$\omega\text{MaximumMagnitudeNumber}(x, y) \rightarrow \omega\text{MaximumMagnitude}(x, y)$

$\text{MaximumMagnitudeNumber}(x, y) \rightarrow$

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

### 4.12.3 MinimumFinite, MaximumFinite

Minimum and maximum finite value.

#### Signature

Operation <sub>$f_x, f_y, f_r, \rho$</sub> ( $x, y$ )  $\rightarrow r$

#### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$   
 $f_r$  : format of result  $r$   
 $\rho$  : projection specification

#### Operands

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

#### Result

$r$  : P3109 value, format  $f_r$

#### Behavior

$\omega\text{MinimumFinite}(\text{NaN}, \text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{MinimumFinite}(\text{NaN}, y) \rightarrow y$   
 $\omega\text{MinimumFinite}(x, \text{NaN}) \rightarrow x$   
 $\omega\text{MinimumFinite}(+\infty, +\infty) \rightarrow +\infty$   
 $\omega\text{MinimumFinite}(-\infty, -\infty) \rightarrow -\infty$   
 $\omega\text{MinimumFinite}(+\infty, -\infty) \rightarrow -\infty$   
 $\omega\text{MinimumFinite}(-\infty, +\infty) \rightarrow -\infty$   
 $\omega\text{MinimumFinite}(+\infty, Y) \rightarrow Y$   
 $\omega\text{MinimumFinite}(X, +\infty) \rightarrow X$   
 $\omega\text{MinimumFinite}(-\infty, Y) \rightarrow Y$   
 $\omega\text{MinimumFinite}(X, -\infty) \rightarrow X$   
 $\omega\text{MinimumFinite}(X, Y) \rightarrow$

if  $X < Y$  then  $X$  else  $Y$

$\omega\text{MaximumFinite}(\text{NaN}, \text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{MaximumFinite}(\text{NaN}, y) \rightarrow y$   
 $\omega\text{MaximumFinite}(x, \text{NaN}) \rightarrow x$   
 $\omega\text{MaximumFinite}(+\infty, +\infty) \rightarrow +\infty$   
 $\omega\text{MaximumFinite}(-\infty, -\infty) \rightarrow -\infty$   
 $\omega\text{MaximumFinite}(+\infty, -\infty) \rightarrow +\infty$   
 $\omega\text{MaximumFinite}(-\infty, +\infty) \rightarrow +\infty$   
 $\omega\text{MaximumFinite}(+\infty, Y) \rightarrow Y$   
 $\omega\text{MaximumFinite}(X, +\infty) \rightarrow X$   
 $\omega\text{MaximumFinite}(-\infty, Y) \rightarrow Y$   
 $\omega\text{MaximumFinite}(X, -\infty) \rightarrow X$   
 $\omega\text{MaximumFinite}(X, Y) \rightarrow$

if  $X < Y$  then  $Y$  else  $X$

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

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

#### 4.12.4 Clamp

Perform a clamp (also known as clip) operation.

##### Signature

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

##### Parameters

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

##### Operands

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

##### Result

$r$  : P3109 value, format  $f_r$

##### Behavior

$\omega\text{Clamp}(\text{NaN}, *, *) \rightarrow \text{NaN}$   
 $\omega\text{Clamp}(*, \text{NaN}, *) \rightarrow \text{NaN}$   
 $\omega\text{Clamp}(*, *, \text{NaN}) \rightarrow \text{NaN}$   
 $\omega\text{Clamp}(*, Lo, Hi) \text{ if } Lo > Hi \rightarrow \text{NaN}$   
 $\omega\text{Clamp}(*, +\infty, +\infty) \rightarrow +\infty$   
 $\omega\text{Clamp}(*, -\infty, -\infty) \rightarrow -\infty$   
 $\omega\text{Clamp}(*, *, -\infty) \rightarrow \text{NaN}$   
 $\omega\text{Clamp}(*, +\infty, *) \rightarrow \text{NaN}$   
 $\omega\text{Clamp}(+\infty, *, +\infty) \rightarrow +\infty$   
 $\omega\text{Clamp}(+\infty, *, Hi) \rightarrow Hi$   
 $\omega\text{Clamp}(-\infty, -\infty, *) \rightarrow -\infty$   
 $\omega\text{Clamp}(-\infty, Lo, *) \rightarrow Lo$   
 $\omega\text{Clamp}(X, -\infty, +\infty) \rightarrow X$   
 $\omega\text{Clamp}(X, -\infty, Hi) \text{ if } X \leq Hi \rightarrow X$   
 $\omega\text{Clamp}(X, -\infty, Hi) \text{ if } X > Hi \rightarrow Hi$   
 $\omega\text{Clamp}(X, Lo, +\infty) \text{ if } X \leq Lo \rightarrow Lo$   
 $\omega\text{Clamp}(X, Lo, +\infty) \text{ if } X > Lo \rightarrow X$   
 $\omega\text{Clamp}(X, Lo, Hi) \rightarrow \text{if } X \leq Lo \text{ then } Lo \text{ else if } X \geq Hi \text{ then } Hi \text{ else } X$   
 $\text{Clamp}(x, lo, hi) \rightarrow \omega\text{Project}_{f_r, \rho}(\omega\text{Clamp}(\omega\text{Decode}_{f_x}(x), \omega\text{Decode}_{f_{lo}}(lo), \omega\text{Decode}_{f_{hi}}(hi)))$

## 4.13 Comparisons

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

### Signature

$\text{CompareOp}_{f_x, f_y}(x, y) \rightarrow b$

### Parameters

$f_x$  : format of  $x$   
 $f_y$  : format of  $y$

### Operands

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

### Result

$b$  : Boolean value

### Behavior

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

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

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

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

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

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

[Comparisons continued on next page]

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

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

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

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

## 4.14 Predicates and classification

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

### Signature

$$\text{IsClass}_f(x) \rightarrow b$$

### Parameters

$f$  : format of  $x$

### Operands

$x$  : P3109 value, format  $f$

### Result

$b$  : Boolean value

### Behavior

$$\omega\text{IsZero}(\text{NaN}) \rightarrow \text{False}$$

$$\omega\text{IsZero}(\pm\infty) \rightarrow \text{False}$$

$$\omega\text{IsZero}(X) \rightarrow X = 0$$

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

$$\omega\text{IsOne}(\text{NaN}) \rightarrow \text{False}$$

$$\omega\text{IsOne}(\pm\infty) \rightarrow \text{False}$$

$$\omega\text{IsOne}(X) \rightarrow X = 1$$

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

$$\text{IsNaN}(\text{NaN}) \rightarrow \text{True}$$

$$\text{IsNaN}(x) \rightarrow \text{False}$$

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

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

$$\omega\text{IsSignMinus}(\text{NaN}) \rightarrow \text{False}$$

$$\omega\text{IsSignMinus}(+\infty) \rightarrow \text{False}$$

$$\omega\text{IsSignMinus}(-\infty) \rightarrow \text{True}$$

$$\omega\text{IsSignMinus}(X) \rightarrow X < 0$$

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

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

$$\text{IsNormal}(x) \rightarrow \begin{cases} (x \bmod 2^{K_f-1}) \div 2^{P_f-1} > 0 & \text{if } \sigma_f = \text{Signed} \\ x \div 2^{P_f-1} > 0 & \text{if } \sigma_f = \text{Unsigned} \end{cases}$$

$$\text{IsSubnormal}(x \in \{0, -\text{Inf}, \text{Inf}, \text{NaN}\}) \rightarrow \text{False}$$

$$\text{IsSubnormal}(x) \rightarrow \text{not IsNormal}(x)$$

### 4.14.1 Classifier operation

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

**Signature**
$$\text{Class}_f(x) \rightarrow c$$
**Parameters**
$$f : \text{format of } x$$
**Operands**
$$x : \text{P3109 value, format } f$$
**Result**
$$c : \text{enumeration}$$
**Behavior**
$$\text{Class}(x) \rightarrow \text{ClassEnum}$$

Table 5: Classifier operation

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

#### 4.14.2 Total order predicate

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

##### Signature

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

##### Parameters

$f_x$  : format of  $x$

$f_y$  : format of  $y$

##### Operands

$x$  : P3109 value, format  $f_x$

$y$  : P3109 value, format  $f_y$

##### Result

$b$  : Boolean value

##### Behavior

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

##### Note

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



### 4.14.3 Next greater than and next less than

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

#### Signature

$$\begin{aligned}\text{NextGreaterThan}_f(x) &\rightarrow r \\ \text{NextLessThan}_f(x) &\rightarrow r\end{aligned}$$

#### Parameters

$f$  : format of  $x$  and  $r$

#### Operands

$x$  : P3109 value, format  $f$

#### Result

$r$  : P3109 value, format  $f$

#### Behavior

$$\begin{aligned}\text{NextGreaterThanAux}(*, *, \text{NaN}) &\rightarrow \text{NaN} \\ \text{NextGreaterThanAux}(\text{Extended}, *, \text{Inf}) &\rightarrow \text{NaN} \\ \text{NextGreaterThanAux}(\text{Finite}, *, \text{MaxFinite}) &\rightarrow \text{NaN} \\ \text{NextGreaterThanAux}(\text{Extended}, *, \text{MaxFinite}) &\rightarrow \text{Inf} \\ \text{NextGreaterThanAux}(\text{Extended}, \text{Signed}, -\text{Inf}) &\rightarrow \text{MinFinite} \\ \text{NextGreaterThanAux}(*, \text{Signed}, \text{SmallestNegative}) &\rightarrow 0 \\ \text{NextGreaterThanAux}(*, \text{Signed}, x) \text{ if } \text{IsSignMinus}(x) &\rightarrow x - 1 \\ \text{NextGreaterThanAux}(*, *, x) &\rightarrow x + 1\end{aligned}$$

—See §4.1.3 for integer arithmetic on code points.

$$\text{NextGreaterThan}(x) \rightarrow \text{NextGreaterThanAux}(\text{DomainOf}(f), \text{SignednessOf}(f), x)$$
$$\begin{aligned}\text{NextLessThanAux}(*, *, \text{NaN}) &\rightarrow \text{NaN} \\ \text{NextLessThanAux}(\text{Finite}, *, \text{MinFinite}) &\rightarrow \text{NaN} \\ \text{NextLessThanAux}(*, \text{Unsigned}, 0) &\rightarrow \text{NaN} \\ \text{NextLessThanAux}(\text{Extended}, \text{Signed}, -\text{Inf}) &\rightarrow \text{NaN} \\ \text{NextLessThanAux}(\text{Extended}, \text{Signed}, \text{MinFinite}) &\rightarrow -\text{Inf} \\ \text{NextLessThanAux}(\text{Extended}, *, \text{Inf}) &\rightarrow \text{MaxFinite} \\ \text{NextLessThanAux}(*, \text{Signed}, 0) &\rightarrow \text{SmallestNegative} \\ \text{NextLessThanAux}(*, \text{Signed}, x) \text{ if } \text{IsSignMinus}(x) &\rightarrow x + 1 \\ \text{NextLessThanAux}(*, *, x) &\rightarrow x - 1\end{aligned}$$
$$\text{NextLessThan}(x) \rightarrow \text{NextLessThanAux}(\text{DomainOf}(f), \text{SignednessOf}(f), x)$$

#### where

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

#### 4.14.4 Comparison predicates

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

Table 6: Comparison predicates and negations

Math symbol	Predicate
$X = Y$	<code>CompareEqual(<math>X, Y</math>)</code>
$X > Y$	<code>CompareGreater(<math>X, Y</math>)</code>
$X \geq Y, X \geq Y$	<code>CompareGreaterEqual(<math>X, Y</math>)</code>
$X < Y$	<code>CompareLess(<math>X, Y</math>)</code>
$X \leq Y, X \leq Y$	<code>CompareLessEqual(<math>X, Y</math>)</code>
$X \neq Y, X \neq Y$	<code>not CompareEqual(<math>X, Y</math>)</code>

## 5 Block operations

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

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

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

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

## 5.1 Internal functions (block operations)

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

### 5.1.1 BlockDecode

#### Signature

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

#### Parameters

$B$  : block size

$f_s$  : format of scale factor  $s$

$f_x$  : format of elements  $x$

#### Operands

$s$  : P3109 scale, in format  $f_s$

$x$  : sequence of P3109 values in format  $f_x$

#### Result

$Z$  : sequence of closed extended real values

#### Behavior

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

where

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

### 5.1.2 BlockProject

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

#### Signature

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

#### Parameters

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

#### Operands

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

#### Result

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

#### Behavior

$$\begin{aligned} \omega\text{BlockProject}(s, [X_1, \dots, X_B]) &= [r_1, \dots, r_B] \\ \text{where} \\ S &= \omega\text{Decode}_{f_s}(s) \\ Z_i &= \begin{cases} \text{NaN} & \text{if } S \text{ is NaN or } X_i \text{ is NaN} \\ 0 & \text{if } S = 0 \\ 1 & \text{if } S = \pm\infty \\ \omega\text{Divide}(X_i, S) & \text{otherwise} \end{cases} \\ r_i &= \omega\text{Project}_{f_r, \rho_r}(Z_i) \end{aligned}$$

#### Notes

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

## 5.2 Convert Blocks

### 5.2.1 ConvertFromBlock

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

#### Signature

$\text{ConvertFromBlock}_{B, f_s, f_x, f_r, \rho_r}(s, [x_1, \dots, x_B]) \rightarrow [r_1, \dots, r_B]$

#### Parameters

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

#### Operands

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

#### Result

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

#### Behavior

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

**where**

$[Z_1, \dots, Z_B] = \omega\text{BlockDecode}(s, [x_1, \dots, x_B])$   
 $r_i = \omega\text{Project}_{f_r, \rho_r}(Z_i)$

### 5.2.2 ConvertToBlock

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

#### Signature

$$\text{ConvertToBlock}_{B, f_s, f_x, f_r, \rho_r}([x_1, \dots, x_B], s) \rightarrow (s, [r_1, \dots, r_B])$$

#### Parameters

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

#### Operands

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

#### Result

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

#### Behavior

$$\begin{aligned} \text{ConvertToBlock}([x_1, \dots, x_B], s) &= (s, [r_1, \dots, r_B]) \\ \text{where} \\ X_i &= \omega\text{Decode}_{f_x}(x_i) \\ [r_1, \dots, r_B] &= \omega\text{BlockProject}_{B, f_s, f_r, \rho_r}(s, [X_1, \dots, X_B]) \end{aligned}$$

### 5.2.3 ConvertToBlockMaxAbsFinite

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

#### Signature

$$\text{ConvertToBlockMaxAbsFinite}_{B, f_x, f_s, f_r, \rho_s, \rho_r}([x_1, \dots, x_B]) \rightarrow (s, [r_1, \dots, r_B])$$

#### Parameters

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

#### Operands

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

#### Result

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

#### Behavior

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

where

$$\begin{aligned} X_i &= \omega\text{Decode}_{f_x}(x_i) \\ M_i &= \omega\text{Abs}(X_i) \\ S &= \text{reduce}(\omega\text{MaximumFinite}, [0, M_1, \dots, M_B]) \\ s &= \omega\text{Project}_{f_s, \rho_s}(S) \\ [r_1, \dots, r_B] &= \omega\text{BlockProject}_{B, f_s, f_r, \rho_r}(s, [X_1, \dots, X_B]) \end{aligned}$$

#### Notes

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

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

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

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

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



## 5.3 Block reduction operations

### 5.3.1 Sum and product

#### Signature

$\text{BlockReduceAdd}_{B, f_s, f_x, f_r, \rho_r}((s, [x_1, \dots, x_B])) \rightarrow r$   
 $\text{BlockReduceMultiply}_{B, f_s, f_x, f_r, \rho_r}((s, [x_1, \dots, x_B])) \rightarrow r$

#### Parameters

$B$  : block size  
 $f_s$  : format of input scale factor  $s$   
 $f_x$  : format of input elements  $x$   
 $f_r$  : format of result  $r$   
 $\rho_r$  : projection specification

#### Operands

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

#### Result

$r$  : value in format  $f_r$

#### Behavior

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

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

#### Notes

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

### 5.3.2 Dot product

Dot product of two blocks of P3109 values.

#### Signature

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

#### Parameters

$B$  : block size  
 $f_{sx}$  : format of first input scale factor  $s_x$   
 $f_x$  : format of first input elements  $x$   
 $f_{sy}$  : format of second input scale factor  $s_y$   
 $f_y$  : format of second input elements  $y$   
 $f_r$  : format of result  $r$   
 $\rho_r$  : projection specification

#### Operands

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

#### Result

$r$  : value in format  $f_r$

#### Behavior

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

#### Notes

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

## 5.4 Elementwise operations on blocks

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

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

Permitted substitutions for  $Op$  are:

- Unary ( $n = 1$ ): 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.

### Signature

$$\text{BlockOp}_{B, (f_{s1}, f_{x1}), \dots, (f_{sn}, f_{xn}), f_s, f_r, \rho_r}((s_1, [x_{11}, \dots, x_{1B}]), \dots, (s_n, [x_{n1}, \dots, x_{nB}]), s_r) \rightarrow (s_r, [r_1, \dots, r_B])$$

### Parameters

$B$  : block size  
 $f_{s1}$  : format of first input scale factor  $s_1$   
 $f_{x1}$  : format of first input elements  $x_{1i}$   
 $\dots$  :  
 $f_{sn}$  : format of  $n^{\text{th}}$  input scale factor  $s_n$   
 $f_{xn}$  : format of  $n^{\text{th}}$  input elements  $x_{ni}$   
  
 $f_s$  : format of scale factor of output  $s_r$   
 $f_r$  : format of output elements  $r$   
 $\rho_r$  : projection specification for output elements

### Operands

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

### Result

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

### Behavior

$$\text{BlockOp}((s_1, [x_{11}, \dots, x_{1B}]), \dots, (s_n, [x_{n1}, \dots, x_{nB}]), s_r) = (s_r, [r_1, \dots, r_B])$$

where

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

## 5.5 Scaled operations via block operations

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

$$\begin{aligned} \text{ScaledOp}(s_1, x_1, s_2, x_2) &= r \\ \text{where} \\ [1, r] &= \text{BlockOp}((s_1, [x_1]), (s_2, [x_2]), 1) \end{aligned}$$

where the two-operand  $\text{BlockOp}$  expands to:

$$\begin{aligned} \text{BlockOp}((s_1, [x_1]), (s_2, [x_2]), 1) &= (s_r, [r]) \\ \text{where} \\ X_1 &= \omega \text{BlockDecode}_{B, f_{s_1}, f_{x_1}}(s_1, [x_1]) \\ X_2 &= \omega \text{BlockDecode}_{B, f_{s_2}, f_{x_2}}(s_2, [x_2]) \\ Z &= \omega \text{Op}(X_1, X_2) \\ r &= \omega \text{BlockProject}_{B, f_s, f_r, \rho_r}(1, Z) \end{aligned}$$

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

## 6 Interconverting formats

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

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

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

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

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

$M_\phi^{\text{hi}}$  : maximum finite value (e.g. 65504.0 for Binary16)  
 $P_\phi$  : precision (e.g. 11 for Binary16)  
 $B_\phi$  : exponent bias (e.g. 15 for Binary16)

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

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

### Signature

Encode754 $_\phi(X) \rightarrow r$

### Parameters

$\phi$  : target IEEE Std 754 format

### Operands

$X$  : closed extended real value, in the value set of format  $\phi$

### Result

$r$  : IEEE Std 754 value, format  $\phi$

### Behavior

Encode754(NaN)  $\rightarrow$  Any quiet IEEE Std 754 NaN  
Encode754( $X$ )  $\rightarrow$  The code in  $\phi$  that decodes to  $X$

### Notes

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

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

## 6.1 Conversion from IEEE Std 754 formats to P3109

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

### Signature

$$\text{ConvertFromIEEE754}_{\phi, f, \rho}(X) \rightarrow r$$

### Parameters

$\phi$  : source IEEE Std 754 format

$f$  : target format

$\rho$  : projection specification

### Operands

$X$  : IEEE-754 value, format  $\phi$

### Result

$r$  : P3109 value, format  $f$

### Behavior

$\text{ConvertFromIEEE754}(\text{Any IEEE Std 754 NaN}) \rightarrow \text{NaN}_f$

$\text{ConvertFromIEEE754}(X) \rightarrow \omega\text{Project}_{f, \rho}(\text{AsClosedExtendedReal}_{\phi}(X))$

## 6.2 Conversion from P3109 to IEEE Std 754

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

### Signature

$\text{ConvertToIEEE754}_{f,\rho,\phi}(x) \rightarrow X$

### Parameters

$f$  : input format

$\rho$  : projection specification

$\phi$  : IEEE Std 754 result format, precision  $P_\phi$ , bias  $B_\phi$ , maximum finite value  $M_\phi^{\text{hi}}$

### Operands

$x$  : P3109 value, format  $f$

### Result

$X$  : IEEE-754 value, format  $\phi$

### Behavior

$\text{ConvertToIEEE754}(\text{NaN}) \rightarrow \text{Encode754}_\phi(\text{NaN})$

$\text{ConvertToIEEE754}(x) \rightarrow \text{Encode754}_\phi(X)$

where

$Y = \omega\text{Decode}_f(x)$

$R = \omega\text{RoundToPrecision}_{P_\phi, B_\phi, \text{RoundOf}(\rho)}(Y)$

$X = \omega\text{Saturate}_{-M_\phi^{\text{hi}}, M_\phi^{\text{hi}}}(\text{SatOf}(\rho), \text{RoundOf}(\rho), R, \text{Signed}, \text{Extended})$

# Appendices

## A Rationales

### A.1 Use of infinity in computation of attention masks

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

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

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

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

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

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

$$\text{logsumexp}(v) \rightarrow \text{logsumexp}(v - \max(v)) + \max(v)$$

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

For example, in a format where  $\text{maxFinite}=240$  without Inf, and  $\text{maxFinite}=224$  with Inf:

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

while

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

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



## B Value Tables for K=2

Table 7 shows the value sets for  $K = 2$  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 value set, albeit a different partitioning of subnormals.
- Similarly the formats Binary2p1uf and Binary2p2uf have the same value set.
- The value 1.0 is not present in Binary2p1se, Binary2p2ue, Binary2p2ue, hence is not encoded according to Table 1.
- Binary2p1se has no normal or subnormal values, hence  $\text{MinPositiveOf}(\text{Binary2p1se}) = \text{Inf}$ .
- Binary2p1se and Binary2p2ue have no normal values, hence  $\text{MinNormalOf}(f) = \text{NaN}$  for these formats.

Code point	Binary2p1se	Binary2p1sf	Binary2p1ue	Binary2p2ue	Binary2p1uf	Binary2p2uf
0 0b00	0.000	0.000	0.000	0.000	0.000	0.000
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 7: Value sets for  $K = 2$ . Subnormals are shown in blue, special values in brown.

## C External Formats

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

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

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

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

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

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

## References

- [1] “IEEE standard for floating-point arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.
- [2] Posit Working Group, “Standard for Posit™ arithmetic (2022),” tech. rep., [posithub.org](https://posithub.org/docs/posit_standard-2.pdf), 2022.  
[https://posithub.org/docs/posit\\_standard-2.pdf](https://posithub.org/docs/posit_standard-2.pdf).
- [3] J. L. Gustafson, “Posit arithmetic,” tech. rep., [posithub.org](https://posithub.org/docs/Posits4.pdf), 2017.  
<https://posithub.org/docs/Posits4.pdf>.
- [4] L. Hunhold, “Beating Posits at their own game: Takum arithmetic,” tech. rep., arXiv math.NA, 2024.  
<https://arxiv.org/abs/2404.18603>.
- [5] Y. Luo, Z. Zhang, R. Wu, H. Liu, Y. Jin, K. Zheng, M. Wang, Z. He, G. Hu, L. Chen, T. Hu, J. Wang, M. Chen, M. Dmitry, K. Vladimir, B. Maxim, Y. Hu, G. Chen, and Z. Huang, “Ascend HiFloat8 format for deep learning,” tech. rep., arXiv cs.LG, 2024.  
<https://arxiv.org/abs/2409.16626>.
- [6] W. Kahan, “Branch cuts for complex elementary functions or much ado about nothing’s sign bit,” *Institute of Mathematics and its Applications Conference*, 1987.  
<https://people.freebsd.org/~das/kahan86branch.pdf>.
- [7] W. Kahan and J. W. Thomas, “Augmenting a programming language with complex arithmetic,” tech. rep., EECS Department, University of California, Berkeley, 1991.
- [8] Google, “Jax lax package: `_float_to_int_for_sort .`” [https://github.com/google/jax/blob/fc5960f2b8b7a0ef74dbae4e27c5c08ff1564cff/jax/\\_src/lax/lax.py#L3934](https://github.com/google/jax/blob/fc5960f2b8b7a0ef74dbae4e27c5c08ff1564cff/jax/_src/lax/lax.py#L3934).
- [9] R. Zhao, B. Vogel, and T. Ahmed, “Adaptive loss scaling for mixed precision training,” tech. rep., arXiv cs.LG, 2019.  
<https://arxiv.org/abs/1910.12385>.
- [10] C. M. Wintersteiger, “Formal verification of the IEEE P3109 standard for binary floating-point formats for machine learning,” in *IEEE 32nd Symposium on Computer Arithmetic, ARITH 2025, El Paso, TX, USA, May 4-7, 2025*, IEEE, 2025.
- [11] C. M. Wintersteiger, “Formal verification of the IEEE P3109 standard,” 2025. <https://github.com/imandra-ai/ieee-p3109>.
- [12] A. W. Fitzgibbon and S. Felix, “On stochastic rounding with few random bits,” in *IEEE 32nd Symposium on Computer Arithmetic, ARITH 2025, El Paso, TX, USA, May 4-7, 2025*, pp. 133–140, IEEE, 2025.
- [13] PyTorch authors, “Pytorch torchtext package: `_t5_multi_head_attention_forward .`” <https://github.com/pytorch/text/blob/a933cbe5a008bc2cb61d985cf5864069194157eb/torchtext/prototype/models/t5/modules.py#L236>.
- [14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ch. 6.2.2.3 Softmax Units for Multinoulli Output Distributions, pp. 180–184. MIT Press, 2016.
- [15] P. Micikevicius, S. Oberman, P. Dubey, M. Cornea, A. Rodriguez, I. Bratt, R. Grisenthwaite, N. Jouppi, C. Chou, A. Huffman, M. Schulte, R. Wittig, D. Jani, and S. Deng, “OCP 8-bit floating point specification (OFP8),” tech. rep., [opencompute.org](https://opencompute.org), 2023.  
<https://www.opencompute.org/documents/ocp-8-bit-floating-point-specification-ofp8-revision-1-0-2>
- [16] B. Nouné, P. Jones, D. Justus, D. Masters, and C. Luschi, “8-bit numerical formats for deep neural networks,” tech. rep., arXiv cs.LG, 2022.  
<https://arxiv.org/abs/2206.02915>.
- [17] Tesla, Inc., “Tesla Dojo Technology: A guide to Tesla’s configurable floating point formats and arithmetic,” 2023.  
[https://web.archive.org/web/20230503235751/https://tesla-cdn.thron.com/static/MXMU3S\\_tesla-dojo-technology\\_1WDVZN.pdf](https://web.archive.org/web/20230503235751/https://tesla-cdn.thron.com/static/MXMU3S_tesla-dojo-technology_1WDVZN.pdf).