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

Initial release: 18 September 2023
Version 1.0: 29 October 2024
Version 2.0: 21 July 2025

Version 3.0: 2025-07-29
(compiled 2025-07-29)

DRAFT DOCUMENT

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

Contents

1 Contributors

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

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

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

Editors

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

Contributing Editors

Amos Omondi, Guy Lemieux

Contributors

Nima Badizadegan	Laslo Hunhold	Michael Overton
Paul Balanca	Faizan Khattak	Katherine Parry
David Chen	Simon Knowles	Valentina Popescu
Eric Chung	Seok Ko	Nathalie Revol
Marco Cococcioni	Sudhanva Kulkarni	Jason Riedy
Marius Cornea	Ada Li	Ali Sazegari
Mike Cowlishaw	Carlo Luschi	Eric Schwarz
James Davenport	David Lutz	Olivier Sentieys
James Demmel	Tue Ly	Michael Siu
Kenneth Dockser	Albert Martin	Gil Tabak
Massimiliano Fasi	Mantas Mikaitis	Julio Villalba
Silvio-loan Filip	Aaftab Munshi	Kristopher Wong
Jeff Gonion	Santosh Nagarakatte	Thomas Yeh
John Gustafson	Aisha Naseer	Chao Yu
Michel Hack	Badreddine Noune	Aleksandr Zakharchenko

Contacting The Working Group

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

2 Introduction

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

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

2.1 Typographical conventions and notation

Bold text describes the decisions and specifications of this document.

2.2 Scope

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

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

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

3 Formats

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

The universe of values in existing IEEE Std 754 floating-point usage is the finite reals, the non-finite values positive and negative infinity (Inf, -Inf), the value negative zero (-0), and not-a-number values (NaN, NaN_1, ...).

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

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

3.1 Format parameters and value sets

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

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

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

The width K shall be greater than one, and less than sixteen.

The precision P shall be greater than zero.

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

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

3.1.1 Naming

Formats defined in this document shall be generically named $\mathsf{binary}\langle\kappa\rangle\mathsf{p}\langle\psi\rangle\langle\sigma\rangle\langle\delta\rangle$ where the placeholders κ and ψ are decimal digit strings, $\sigma \in \{\mathsf{s}, \mathsf{u}\}$, and $\delta \in \{\mathsf{e}, \mathsf{f}\}$.

By convention, and without ambiguity, the characters s and e (for signed, and extended) may be elided.

Examples:

- The format "binary12p7se", which may also be written "binary12p7", is a 12-bit signed format in the extended domain, with 1 sign bit, 5 exponent bits and 7 bits of precision (of which only 6 need to be explicitly represented).
- The format "binary8p1uf", is an 8-bit unsigned format in the finite domain, with no sign bit, 8 exponent bits, and 1 bit of precision (requiring zero bits to be explicitly represented).

3.2 Exponent bias

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

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

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

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

A consequence of these specifications is that the value 1.0 encodes consistently to the midway code point, that is 2^{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 $1.T \times 2^{E-\mathsf{bias}}$ except when all bits of the exponent bitfield E are zero, in which case the value is $0.T \times 2^{-\mathsf{bias}}$.

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

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

3.4 Not a number (NaN)

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

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

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

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

can be used as a missing value, but given the restricted range of P3109 formats, it is likely that infinity shall be used as a separate indicator of rounding from values outside of the finite range.

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

3.5 Zero

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

The inclusion of negative zero (-0) would incur the cost of an additional code point. Given the decision to encode only a single NaN, placing that NaN at where IEEE Std 754 encodes negative zero enables the strictly positive and strictly negative number ranges to be symmetric for signed formats.

A key rationale for including -0 in IEEE Std 754 was the consistent implementation of branch cuts in the atan2 function and the complex trigonometric functions [1, 2]. The atan2 function is rare, if not unknown, in deep learning applications. The related atan function is common in deep learning, however it is generally used as an activation function, rather than a trigonometric operation.

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

It might be considered that the use of integer comparisons in sorting would argue against placing NaN at the negative zero code point. For example, the JAX machine learning framework is known to sort using integer comparison [3]. However, such sorting still requires $O(n)$ preprocessing and postprocessing steps to enable the use of twos-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 §B.1 for more detailed discussion on this usage.
- Representation of overflow, for example, to adjust dynamic loss scaling factors [4].

Representing infinite values requires two codepoints 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 §4.7.5.

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 more 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 $2^K - 2$.

p3109_k4p3es	p3109_k4p3fs	p3109_k4p3eu	p3109_k4p3fu
0x00 = 0_0_00 = 0.0 = 0.0	0x00 = 0_0_00 = 0.0 = 0.0	0x00 = 00_00 = 0.0 = 0.0	0x00 = 00_00 = 0.0 = 0.0
0x01 = 0_0_01 = +0b0.01*2^0 = 0.25	0x01 = 0_0_01 = +0b0.01*2^0 = 0.25	0x01 = 00_01 = +0b0.01*2^-1 = 0.125	0x01 = 00_01 = +0b0.01*2^-1 = 0.125
0x02 = 0_0_10 = +0b0.10*2^0 = 0.5	0x02 = 0_0_10 = +0b0.10*2^0 = 0.5	0x02 = 00_10 = +0b0.10*2^-1 = 0.25	0x02 = 00_10 = +0b0.10*2^-1 = 0.25
0x03 = 0_0_11 = +0b0.11*2^0 = 0.75	0x03 = 0_0_11 = +0b0.11*2^0 = 0.75	0x03 = 00_11 = +0b0.11*2^-1 = 0.375	0x03 = 00_11 = +0b0.11*2^-1 = 0.375
0x04 = 0_1_00 = +0b1.00*2^0 = 1.0	0x04 = 0_1_00 = +0b1.00*2^0 = 1.0	0x04 = 01_00 = +0b1.00*2^-1 = 0.5	0x04 = 01_00 = +0b1.00*2^-1 = 0.5
0x05 = 0_1_01 = +0b1.01*2^0 = 1.25	0x05 = 0_1_01 = +0b1.01*2^0 = 1.25	0x05 = 01_01 = +0b1.01*2^-1 = 0.625	0x05 = 01_01 = +0b1.01*2^-1 = 0.625
0x06 = 0_1_10 = +0b1.10*2^0 = 1.5	0x06 = 0_1_10 = +0b1.10*2^0 = 1.5	0x06 = 01_10 = +0b1.10*2^-1 = 0.75	0x06 = 01_10 = +0b1.10*2^-1 = 0.75
0x07 = 0_1_11 = inf = inf	0x07 = 0_1_11 = +0b1.11*2^0 = 1.75	0x07 = 01_11 = +0b1.11*2^-1 = 0.875	0x07 = 01_11 = +0b1.11*2^-1 = 0.875
0x08 = 1_0_00 = nan = nan	0x08 = 1_0_00 = nan = nan	0x08 = 10_00 = +0b1.00*2^0 = 1.0	0x08 = 10_00 = +0b1.00*2^0 = 1.0
0x09 = 1_0_01 = -0b0.01*2^0 = -0.25	0x09 = 1_0_01 = -0b0.01*2^0 = -0.25	0x09 = 10_01 = +0b1.01*2^0 = 1.25	0x09 = 10_01 = +0b1.01*2^0 = 1.25
0x0a = 1_0_10 = -0b0.10*2^0 = -0.5	0x0a = 1_0_10 = -0b0.10*2^0 = -0.5	0x0a = 10_10 = +0b1.10*2^0 = 1.5	0x0a = 10_10 = +0b1.10*2^0 = 1.5
0x0b = 1_0_11 = -0b0.11*2^0 = -0.75	0x0b = 1_0_11 = -0b0.11*2^0 = -0.75	0x0b = 10_11 = +0b1.11*2^0 = 1.75	0x0b = 10_11 = +0b1.11*2^0 = 1.75
0x0c = 1_1_00 = -0b1.00*2^0 = -1.0	0x0c = 1_1_00 = -0b1.00*2^0 = -1.0	0x0c = 11_00 = +0b1.00*2^1 = 2.0	0x0c = 11_00 = +0b1.00*2^1 = 2.0
0x0d = 1_1_01 = -0b1.01*2^0 = -1.25	0x0d = 1_1_01 = -0b1.01*2^0 = -1.25	0x0d = 11_01 = +0b1.01*2^1 = 2.5	0x0d = 11_01 = +0b1.01*2^1 = 2.5
0x0e = 1_1_10 = -0b1.10*2^0 = -1.5	0x0e = 1_1_10 = -0b1.10*2^0 = -1.5	0x0e = 11_10 = inf = inf	0x0e = 11_10 = +0b1.10*2^1 = 3.0
0x0f = 1_1_11 = -inf = -inf	0x0f = 1_1_11 = -0b1.11*2^0 = -1.75	0x0f = 11_11 = nan = nan	0x0f = 11_11 = nan = nan

Table 1: Value sets for $K = 4, P = 3$: signed extended, signed finite, unsigned extended, unsigned finite.

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

Format	minSubnormal	maxSubnormal	minNormal	maxNormal
binary8p1es	N/A	N/A	1×2^{-63}	1×2^{62}
binary8p2es	1×2^{-32}	1×2^{-32}	1×2^{-31}	1×2^{31}
binary8p3es	1×2^{-17}	$3/2 \times 2^{-16}$	1×2^{-15}	$3/2 \times 2^{15}$
binary8p4es	1×2^{-10}	$7/4 \times 2^{-8}$	1×2^{-7}	$7/4 \times 2^{7}$
binary8p5es	1×2^{-7}	$15/8 \times 2^{-4}$	1×2^{-3}	$15/8 \times 2^{3}$
binary8p6es	1×2^{-6}	$31/16 \times 2^{-2}$	1×2^{-1}	$31/16 \times 2^{1}$
binary8p7es	1×2^{-6}	$63/32 \times 2^{-1}$	1×2^{0}	$63/32 \times 2^{0}$

Table 3: Extremal values. Some examples for $K \in \{4, 8\}$.

It is practical to list extremal finite values defined by the defined formats. Following IEEE Std 754-2019 naming patterns, we adopt: $\mathsf{maxNormal}(\tau), \mathsf{minNormal}(\tau), \mathsf{minSubnormal}(\tau)$ where τ is a format.

For example: $\mathsf{maxNormal}(\mathsf{binary8p4se}) = 7/4 \times 2^7$ and $\mathsf{minNormal}(\mathsf{binary8p5se}) = 1 \times 2^{-3}$.

Table 3 shows the extremal values for $K = 8$ and $1 \leq P \leq 7$. A few 8-bit value tables are provided in section D.

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

In the definitions of operations, certain mathematical *functions* are also defined. We emphasize that the definitions in this document are specifications of behavior, not implementation. All arithmetic represented as operating on extended real values is to be interpreted as defined in Appendix A.

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

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

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

4.1 Notation and definitions

4.1.1 Mathematical notations

We denote by $\mathbb{1}[b]$ the value 1 if $b = \mathsf{True}$, or 0 if $b = \mathsf{False}$.

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

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

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

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

4.1.2 The set of extended reals

The set of *extended reals* is the set $\mathbb{R} \cup \{-\infty, \infty\}$ of real numbers augmented with positive and negative infinity, also known as the "affinely extended real numbers". In common with existing mathematical treatments, there is no negative zero in this set. In the extended reals, certain operations, such as $\infty - \infty$, are *undefined*. Following convention with the extended reals, operations produce and consume infinities as appropriate, e.g. $-\infty + -\infty = -\infty$, $\log 0 = -\infty$, $\exp(-\infty) = 0$. The definitions in this specification are constructed such that no operation on extended real quantities produces an undefined result. Appendix A defines all operations on extended reals used in this document. Operation specifications will, in general, convert floating-point operands to extended real values in order to define their behaviors.

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 $2^K - 1$. Operation 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 extended real values, on which the mathematical operation is performed, before conversion back to the appropriate P3109 value set. In general, operation results will not be exact P3109 values, and hence will be *projected* into the P3109 value set via rounding and overflow handling. A *projection specification* is a pair (rounding mode, saturation mode). For a given projection specification π, these are written $\mathsf{Rnd}_\pi, \mathsf{Sat}_\pi$.

The defined rounding modes are as follows. The precise specifications of these modes are in the function RoundToPrecision (§4.7.2):

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

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 then treated according to the saturation mode.

The defined saturation modes are as follows. Their precise specifications are given in the function Saturate (§4.7.4):

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

4.3 Approximate implementations

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

A numeric operation a has defined (exact) output $\hat{a}(x)$ for operands x. Let the set of operands producing finite outputs be I_a, so that $\hat{a}(x) \in V$ for all $x \in I_a$, where V is the output format's finite value set. For all other operands, i.e. those producing NaN, -Inf, or Inf, an α-approximate implementation shall produce the same value.

An α-approximate implementation will produce a value $\tilde{a}(x) \in V$, which for some operands x has $\tilde{a}(x) \neq \hat{a}(x)$.

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

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

The value of α will in general be specific to each operation and parameters, for example a system may supply implementations of $\mathsf{Add}_{f_x,f_y,f_z,\pi}$ where α depends on f_z. These implementations shall be named differently. For each α-approximate implementation of an (operator, parameters) pair, α shall be specified. This may be attested by any proof method, including direct computation.

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 Operation variants and superset specifications

Each operation definition in this document is *parameterized*. Example parameters include input and output formats, and projection specifications. In addition operand types may be specified as if to arbitrary precision (e.g. s in the template definition below is defined as a general integer, while a given implementation will accept only a finite set of integers). The set of *variants* so defined for a given operation is called a *superset specification*.

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

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

For example, an implementation might declare as follows:

> "This implementation conforms to P3109, providing the following operation variants:
>
> - ConvertToP3109, ϕ in {Binary16, Binary32}, f in {binary8p3se, binary8p4se, binary8p5se}
> - AddScaled, with $s_x = 0$, $s_y \in \{-128 \dots 127\}$, $\{f_x, f_y, f_z\} \subset$ {binary8p3se, binary8p4se, binary8p5se}
> - MultiplyScaled, with $s \in \{-32 \dots 32\}$
>
> In all cases, provided projection modes π obey $\mathsf{Rnd}_\pi \in \{\mathsf{NearestTiesToEven},\ \mathsf{NearestTiesToAway}\}$ and $\mathsf{Sat}_\pi \in \{\mathsf{SatPropagate}, \mathsf{OvfInf}\}$"

This example is truncated—realistic compliance declarations might run to many pages. Such declarations may be compressed by defining common *profiles*, outside of the scope of this current document.

4.6 Operator definition template

Operations are defined according to the following template:

Signature

$\mathsf{Operator}_f(x, s) \rightarrow Z$ —Naming the operator, its parameters, operands and result.

Parameters —Parameters specify a family of related operations

f : format, precision P_f

Operands

x : P3109 value, in format f
s : Integer scale

Output

Z : extended real value or NaN —Result value and type

Behavior —An ordered sequence of pattern-matching declarations.

$\mathsf{Operator}(\mathsf{NaN}, 0) \rightarrow \mathsf{NaN}$ —Exact pattern: only the provided operands match.
$\mathsf{Operator}(x \in \{\text{-}\mathsf{Inf}, \mathsf{Inf}\}, 0) \rightarrow x$ —Match for all x values in the given set.
$\mathsf{Operator}(*, 0) \rightarrow 0$ —The $*$ symbol matches any value.
$\mathsf{Operator}(x, s < 0) \rightarrow -\mathsf{Operator}_f(x, -s)$

Notes

Notes on the operation.

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

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

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

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

4.7 Functions

This section defines mathematical functions which are referenced in the later definitions of operations, but which themselves need not (and, in cases where the inputs or outputs are real, cannot) be provided in a conforming implementation.

4.7.1 Decode

Signature

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

Parameters

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

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

Operands

x : P3109 value, in format f

Output

X : extended real value or NaN

Behavior

$\mathsf{Decode}(x) = \mathsf{DecodeAux}_{K_f,P_f,b_f,\sigma_f,D_f}(x)$

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

$\mathsf{DecodeAux}(2^{\mathsf{K}-1} - 1) \to +\infty$ **if** $D = \mathsf{Extended} \wedge \sigma = \mathsf{Signed}$
$\mathsf{DecodeAux}(2^{\mathsf{K}} - 1) \to -\infty$ **if** $D = \mathsf{Extended} \wedge \sigma = \mathsf{Signed}$
$\mathsf{DecodeAux}(2^{\mathsf{K}} - 2) \to \infty$ **if** $D = \mathsf{Extended} \wedge \sigma = \mathsf{Unsigned}$

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

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

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

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

Note

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

4.7.2 Project

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

Signature

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

Parameters

f : target format,
precision P_f, domain Δ_f, signedness σ_f, exponent bias b_f, minimum/maximum finite value $M_f^{\mathrm{lo}}/M_f^{\mathrm{hi}}$
π : projection specification: rounding mode Rnd_π, saturation Sat_π

Operands

X : extended real value

Output

x : P3109 value, format f

Behavior

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

where

$R = \mathsf{RoundToPrecision}_{\mathsf{P}_f, b_f, \mathsf{Rnd}_\pi}(X)$

$S = \mathsf{Saturate}_{M_f^{\mathrm{lo}}, M_f^{\mathrm{hi}}, \sigma_f}(\mathsf{Sat}_\pi, \mathsf{Rnd}_\pi, R)$

$x = \mathsf{Encode}_f(S)$

Notes

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

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

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

4.7.3 RoundToPrecision

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

Signature

$\mathsf{RoundToPrecision}_{\mathsf{P},\mathsf{Rnd}}(X) \to Z$

Parameters

P : integer precision
b : exponent bias
Rnd : rounding mode

Operands

X : extended real value

Output

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

Behavior

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

where

$E = \max(\lfloor \log_2(|X|) \rfloor, 1 - b) - \mathsf{P} + 1$ —Subnormals handled by $\max(\cdot, 1 - b)$
$S = |X| \times 2^{-E}$ —Real-valued significand, to be rounded to integer
$I = \lfloor S \rfloor + \mathbb{1}[\mathsf{RoundAway}(\mathsf{Rnd})]$
$Z = \mathrm{sign}(X) \times I \times 2^E$

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

$\mathsf{RoundAway}(\mathsf{TowardZero}) = \mathsf{False}$
$\mathsf{RoundAway}(\mathsf{TowardPositive}) = \Delta > 0 \wedge X > 0$
$\mathsf{RoundAway}(\mathsf{TowardNegative}) = \Delta > 0 \wedge X < 0$
$\mathsf{RoundAway}(\mathsf{NearestTiesToAway}) = \Delta \geq 0.5$
$\mathsf{RoundAway}(\mathsf{NearestTiesToEven}) =$

$$\Delta > 0.5 \vee \left(\Delta = 0.5 \wedge \begin{cases} \mathsf{IsEven}(\lfloor S \rfloor + 1) & \textbf{if } \mathsf{P} > 1 \\ \mathsf{IsEven}(E + b + 1) \wedge (\lfloor S \rfloor \neq 0) & \textbf{if } \mathsf{P} = 1 \end{cases} \right)$$

4.7.4 Saturate

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

Signature

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

Parameters

M^{hi} : real maximum value
M^{lo} : real minimum value
σ : signedness in $\{\mathsf{Signed}, \mathsf{Unsigned}\}$

Operands

Sat : saturation mode
Rnd : rounding mode
X : extended real value

Output

Z : extended real value

Behavior

$\mathsf{Saturate}(*, *, X \in [M^{\mathrm{lo}}, M^{\mathrm{hi}}]) \rightarrow X$

$\mathsf{Saturate}(\mathsf{SatFinite}, *, X \leq M^{\mathrm{lo}}) \rightarrow M^{\mathrm{lo}}$
$\mathsf{Saturate}(\mathsf{SatFinite}, *, X \geq M^{\mathrm{hi}}) \rightarrow M^{\mathrm{hi}}$

$\mathsf{Saturate}(\mathsf{SatPropagate}, *, X \in \{\pm\infty\}) \rightarrow X$
$\mathsf{Saturate}(\mathsf{SatPropagate}, *, X \leq M^{\mathrm{lo}}) \rightarrow M^{\mathrm{lo}}$
$\mathsf{Saturate}(\mathsf{SatPropagate}, *, X \geq M^{\mathrm{hi}}) \rightarrow M^{\mathrm{hi}}$

$\mathsf{Saturate}(\mathsf{OvfInf}, *, X \in \{\pm\infty\}) \rightarrow X$
$\mathsf{Saturate}(\mathsf{OvfInf}, \mathsf{TowardZero} \vee \mathsf{TowardPositive}, X \leq M^{\mathrm{lo}}) \rightarrow M^{\mathrm{lo}}$
$\mathsf{Saturate}(\mathsf{OvfInf}, \mathsf{TowardZero} \vee \mathsf{TowardNegative}, X \geq M^{\mathrm{hi}}) \rightarrow M^{\mathrm{hi}}$
$\mathsf{Saturate}(\mathsf{OvfInf}, *, X \leq M^{\mathrm{lo}}) \rightarrow \begin{cases} -\infty & \textbf{if } \sigma = \mathsf{Signed} \\ M^{\mathrm{lo}} & \textbf{if } \sigma = \mathsf{Unsigned} \end{cases}$
$\mathsf{Saturate}(\mathsf{OvfInf}, *, X \geq M^{\mathrm{hi}}) \rightarrow \infty$

4.7.5 Encode

Encode extended real value or NaN to P3109 format f. 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 RoundToPrecision and Saturate, or the input may be known to be in the value set, for example, in the negation of a value already in the set.

Signature

$\mathsf{Encode}_f(X) \to z$

Parameters

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

Operands

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

Output

z : P3109 value, format f

Behavior

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

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

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

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

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

where

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

and

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

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

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

Note

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

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

4.8 Conversion operations

4.8.1 Conversion from P3109 to P3109

Convert a P3109 value to another P3109 format.

Signature

$\mathsf{ConvertP3109ToP3109}_{f_x, f_z, \pi}(x) \to z$

Parameters

f_x : input format
f_z : target format
π : projection specification

Operands

x : P3109 value, format f_x

Output

z : P3109 value, format f_z

Behavior

$\mathsf{ConvertP3109ToP3109}(\mathsf{NaN}_{f_x}) \to \mathsf{NaN}_{f_z}$
$\mathsf{ConvertP3109ToP3109}(x) \to \mathsf{Project}_{f_z, \pi}(X) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x)$

4.9 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 §5.

4.9.1 Absolute Value, Negation

Operations which take P3109 values as input, and whose outputs are guaranteed to be exact values in the same P3109 format. These operations do not permit the output format to be different from the input format, and are defined only for signed formats.

Signature

$\mathsf{Operation}_f(x) \to z$

Parameters

f : input and output format, signedness Signed

Operands

x : P3109 value, format f

Output

z : P3109 value, format f

Behavior

$$\mathsf{Abs}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{Encode}_f(|X|) & \text{Otherwise} \end{cases} \quad \textbf{where} \quad X = \mathsf{Decode}_f(x)$$

$$\mathsf{Negate}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{Encode}_f(-X) & \text{Otherwise} \end{cases} \quad \textbf{where} \quad X = \mathsf{Decode}_f(x)$$

Note

The extended reals do not have negative zero, hence $-0 = 0$ and $\mathsf{Abs}(0) = 0$.

4.9.2 CopySign

Copies the sign of a P3109 value onto another one.

Signature

$\mathsf{Operation}_{f_{io},f_y}(x, y) \to z$

Parameters

f_{io} : input and output formats, must be signed
f_y : format of y, must be signed

Operands

x : P3109 value, format f_{io}
y : P3109 value, format f_y

Output

z : P3109 value, format f_{io}

Behavior

$$\mathsf{CopySign}(x, y) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \vee \mathsf{isNaN}(y) \\ \mathsf{Project}_{f_{io},\pi}(|X|) & Y \geq 0 \\ \mathsf{Project}_{f_{io},\pi}(-|X|) & Y < 0 \end{cases}$$
$$\textbf{where} \quad X = \mathsf{Decode}_{f_{io}}(x), Y = \mathsf{Decode}_{f_y}(y)$$

4.9.3 Addition, Subtraction, Multiplication, Division

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

Signature

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

Parameters

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

Operands

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

Output

z : P3109 value, format f_z

Behavior

$\mathsf{Add}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{Add}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{Add}(\text{-}\mathsf{Inf}, \mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Add}(\mathsf{Inf}, \text{-}\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Add}(x, y) \to \mathsf{Project}_{f_z,\pi}(X + Y) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_y}(y)$

$\mathsf{Subtract}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{Subtract}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{Subtract}(\mathsf{Inf}, \mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Subtract}(\text{-}\mathsf{Inf}, \text{-}\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Subtract}(x, y) \to \mathsf{Project}_{f_z,\pi}(X - Y) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_y}(y)$

$\mathsf{Multiply}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{Multiply}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{Multiply}(0, \pm\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Multiply}(\pm\mathsf{Inf}, 0) \to \mathsf{NaN}$
$\mathsf{Multiply}(x, y) \to \mathsf{Project}_{f_z,\pi}(X \times Y) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_y}(y)$

$\mathsf{Divide}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{Divide}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{Divide}(\pm\mathsf{Inf}, \pm\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{Divide}(*, 0) \to \mathsf{NaN}$
$\mathsf{Divide}(x, y) \to \mathsf{Project}_{f_z,\pi}(X/Y) \quad \textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_y}(y)$

Notes

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

4.9.4 Fused Multiply Add

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

Signature

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

Parameters

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

Operands

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

Output

r : P3109 value, format f_r

Behavior

$$\mathsf{FMA}(\mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(*, \mathsf{NaN}, *) \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(*, *, \mathsf{NaN}) \rightarrow \mathsf{NaN}$$

$$\mathsf{FMA}(0, \pm\mathsf{Inf}, *) \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(\pm\mathsf{Inf}, 0, *) \rightarrow \mathsf{NaN}$$

$$\mathsf{FMA}(x, +\mathsf{Inf}, +\mathsf{Inf}) \quad \textbf{if } x < 0 \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(x, -\mathsf{Inf}, +\mathsf{Inf}) \quad \textbf{if } x > 0 \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(+\mathsf{Inf}, y, +\mathsf{Inf}) \quad \textbf{if } y < 0 \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(-\mathsf{Inf}, y, +\mathsf{Inf}) \quad \textbf{if } y > 0 \rightarrow \mathsf{NaN}$$

$$\mathsf{FMA}(x, -\mathsf{Inf}, -\mathsf{Inf}) \quad \textbf{if } x < 0 \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(x, +\mathsf{Inf}, -\mathsf{Inf}) \quad \textbf{if } x > 0 \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(-\mathsf{Inf}, y, -\mathsf{Inf}) \quad \textbf{if } y < 0 \rightarrow \mathsf{NaN}$$
$$\mathsf{FMA}(+\mathsf{Inf}, y, -\mathsf{Inf}) \quad \textbf{if } y > 0 \rightarrow \mathsf{NaN}$$

$$\mathsf{FMA}(x, y, z) \rightarrow \mathsf{Project}_{f_r,\pi}(X \times Y + Z)$$
$$\textbf{where}$$
$$X = \mathsf{Decode}_{f_x}(x)$$
$$Y = \mathsf{Decode}_{f_y}(y)$$
$$Z = \mathsf{Decode}_{f_z}(z)$$

4.9.5 Fused Add Add

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

Signature

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

Parameters

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

Operands

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

Output

r : P3109 value, format f_r

Behavior

$\mathsf{FAA}(\mathsf{NaN}, *, *) \to \mathsf{NaN}$
$\mathsf{FAA}(*, \mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{FAA}(*, *, \mathsf{NaN}) \to \mathsf{NaN}$

$\mathsf{FAA}(+\mathsf{Inf}, -\mathsf{Inf}, *) \to \mathsf{NaN}$
$\mathsf{FAA}(-\mathsf{Inf}, +\mathsf{Inf}, *) \to \mathsf{NaN}$
$\mathsf{FAA}(+\mathsf{Inf}, *, -\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{FAA}(-\mathsf{Inf}, *, +\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{FAA}(*, +\mathsf{Inf}, -\mathsf{Inf}) \to \mathsf{NaN}$
$\mathsf{FAA}(*, -\mathsf{Inf}, +\mathsf{Inf}) \to \mathsf{NaN}$

$\mathsf{FAA}(x, y, z) \to \mathsf{Project}_{f_r,\pi}(X + Y + Z)$
where
$X = \mathsf{Decode}_{f_x}(x)$
$Y = \mathsf{Decode}_{f_y}(y)$
$Z = \mathsf{Decode}_{f_z}(z)$

4.9.6 Scaled addition

Compute $X \times 2^{s_x} + Y \times 2^{s_y}$, and return a P3109 value. Scaling is applied in the extended reals, before projection to the target format.

Signature

$\mathsf{AddScaled}_{f_x, f_y, f_z, \pi}(x, s_x, y, s_y) \to z$

Parameters

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

Operands

x : P3109 value, format f_x
s_x : integer log-scale factor for x
y : P3109 value, format f_y
s_y : integer log-scale factor for y

Output

z : P3109 value, format f_z

Behavior

$\mathsf{AddScaled}(\mathsf{NaN}, *, *, *) \to \mathsf{NaN}$
$\mathsf{AddScaled}(*, *, \mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{AddScaled}(\text{-}\mathsf{Inf}, *, \mathsf{Inf}, *) \to \mathsf{NaN}$
$\mathsf{AddScaled}(\mathsf{Inf}, *, \text{-}\mathsf{Inf}, *) \to \mathsf{NaN}$
$\mathsf{AddScaled}(x, s_x, y, s_y) \to \mathsf{Project}_{f_z, \pi}(Z)$

where

$Z = X \times 2^{s_x} + Y \times 2^{s_y}$
$X = \mathsf{Decode}_{f_x}(x)$
$Y = \mathsf{Decode}_{f_y}(y)$

Note

The valid range of the log-scale factors s_x and s_y is not specified. Implementers should declare the provision of AddScaled with specified constraints on legal values for these operands (see §4.5).

Example declarations might include

"AddScaled, with log-scale factors in $\{-32 \ldots 31\}$"
"AddScaled, with $s_x = 0$, $s_y \in \{-128 \ldots 127\}$"
"AddScaled, with $s_x \in \{-128 \ldots 127\}$, $s_y = s_x$"

See §4.5 for further discussion.

4.9.7 Scaled multiplication

Compute $X \times Y \times 2^s$, and return a P3109 value. Scaling is applied in the extended reals, before projection to the target format.

Signature

$\mathsf{MultiplyScaled}_{f_x, f_y, f_z, \pi}(x, y, s) \to z$

Parameters

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

Operands

x : P3109 value, format f_x
y : P3109 value, format f_y
s : integer log-scale factor

Output

z : P3109 value, format f_z

Behavior

$\mathsf{MultiplyScaled}(*, \mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{MultiplyScaled}(\mathsf{NaN}, *, *) \to \mathsf{NaN}$
$\mathsf{MultiplyScaled}(0, \pm\mathsf{Inf}, *) \to \mathsf{NaN}$
$\mathsf{MultiplyScaled}(\pm\mathsf{Inf}, 0, *) \to \mathsf{NaN}$
$\mathsf{MultiplyScaled}(x, y, s) \to \mathsf{Project}_{f_z, \pi}(Z)$

$$\textbf{where}$$
$$Z = X \times Y \times 2^s$$
$$X = \mathsf{Decode}_{f_x}(x)$$
$$Y = \mathsf{Decode}_{f_y}(y)$$

Note

The valid range of the log-scale factor s is not specified. Implementations should declare the provision of MultiplyScaled with specified constraints on legal values for these operands (see §4.5).

An example such declaration might be

> "MultiplyScaled, with log-scale factor in $\{-32 \ldots 31\}$"

See §4.5 for further discussion.

4.9.8 Square root, Logarithm, Exponentiation

Signature

$\mathsf{Operation}_{f_x,f_z,\pi}(x) \to z$

Parameters

f_x : input format
f_z : output format
π : projection specification

Operands

x : P3109 value, format f_x

Output

z : P3109 value, format f_z

Behavior

$$\mathsf{Sqrt}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{NaN} & X < 0 \\ \mathsf{Project}_{f_z,\pi}(\sqrt{X}) & X \geq 0 \end{cases}$$

$$\mathsf{RSqrt}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{NaN} & X \leq 0 \\ \mathsf{Project}_{f_z,\pi}(1/\sqrt{X}) & X > 0 \end{cases}$$

$$\mathsf{Exp}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{Project}_{f_z,\pi}(e^X) & \mathsf{Otherwise} \end{cases}$$

$$\mathsf{Exp2}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{Project}_{f_z,\pi}(2^X) & \mathsf{Otherwise} \end{cases}$$

$$\mathsf{Log}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{NaN} & X < 0 \\ \mathsf{Project}_{f_z,\pi}(\log(X)) & X \geq 0 \end{cases}$$

$$\mathsf{Log2}(x) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{NaN} & X < 0 \\ \mathsf{Project}_{f_z,\pi}(\log_2(X)) & X \geq 0 \end{cases}$$

$$\textbf{where} \quad X = \mathsf{Decode}_{f_x}(x)$$

4.9.9 Hypotenuse

Signature

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

Parameters

f_x : input format of x
f_y : input format of y
f_z : output format
π : projection specification

Operands

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

Output

z : P3109 value, format f_z

Behavior

$$\mathsf{Hypot}(x, y) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \\ \mathsf{Project}_{f_z,\pi}\left(\sqrt{X^2 + Y^2}\right) & \mathsf{Otherwise} \end{cases}$$
$$\textbf{where} \quad X = \mathsf{Decode}_{f_x}(x), Y = \mathsf{Decode}_{f_x}(y)$$

4.10 Comparisons, predicates, and classification

4.10.1 Minimum, Maximum, MinimumNumber, and MaximumNumber

Minimum and maximum, with or without propagation of NaN. Operands and return value are in the same format.

Signature

$\mathsf{Operation}_f(x, y) \rightarrow z$

Parameters

f : format

Operands

x : P3109 value, format f
y : P3109 value, format f

Output

z : P3109 value, format f

Behavior

$\mathsf{Minimum}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\mathsf{Minimum}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\mathsf{Minimum}(x, y) \rightarrow \mathsf{Encode}_f(\min(X, Y))$
where
$X = \mathsf{Decode}_f(x)$
$Y = \mathsf{Decode}_f(y)$

$\mathsf{Maximum}(*, \mathsf{NaN}) \rightarrow \mathsf{NaN}$
$\mathsf{Maximum}(\mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\mathsf{Maximum}(x, y) \rightarrow \mathsf{Encode}_f(\max(X, Y))$
where
$X = \mathsf{Decode}_f(x)$
$Y = \mathsf{Decode}_f(y)$

$\mathsf{MinimumNumber}(x, \mathsf{NaN}) \rightarrow x$
$\mathsf{MinimumNumber}(\mathsf{NaN}, y) \rightarrow y$
$\mathsf{MinimumNumber}(x, y) \rightarrow \mathsf{Minimum}_f(x, y)$

$\mathsf{MaximumNumber}(x, \mathsf{NaN}) \rightarrow x$
$\mathsf{MaximumNumber}(\mathsf{NaN}, y) \rightarrow y$
$\mathsf{MaximumNumber}(x, y) \rightarrow \mathsf{Maximum}_f(x, y)$

4.10.2 MinimumMagnitude, MaximumMagnitude, and 'Number' variants

Minimum and maximum by magnitude.

Signature

$\mathsf{Operation}_f(x, y) \to z$

Parameters

f : format

Operands

x : P3109 value, format f
y : P3109 value, format f

Output

z : P3109 value, format f

Behavior

$\mathsf{MinimumMagnitude}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{MinimumMagnitude}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{MinimumMagnitude}(x, y) \to \mathsf{Encode}_f(R)$

$$\textbf{where}$$
$$X = \mathsf{Decode}_f(x)$$
$$Y = \mathsf{Decode}_f(y)$$
$$R = \begin{cases} X & \textbf{if } |X| < |Y| \\ Y & \textbf{if } |X| > |Y| \\ \min(X, Y) & \textbf{if } |X| = |Y| \end{cases}$$

$\mathsf{MaximumMagnitude}(*, \mathsf{NaN}) \to \mathsf{NaN}$
$\mathsf{MaximumMagnitude}(\mathsf{NaN}, *) \to \mathsf{NaN}$
$\mathsf{MaximumMagnitude}(x, y) \to \mathsf{Encode}_f(R)$

$$\textbf{where}$$
$$X = \mathsf{Decode}_f(x)$$
$$Y = \mathsf{Decode}_f(y)$$
$$R = \begin{cases} X & \textbf{if } |X| > |Y| \\ Y & \textbf{if } |X| < |Y| \\ \max(X, Y) & \textbf{if } |X| = |Y| \end{cases}$$

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

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

4.10.3 Clamp

Perform a clamp (also known as clip) operation.

Signature

$\mathsf{Clamp}_f(x, lo, hi) \to z$

Parameters

f : format

Operands

x : P3109 value, format f
lo : P3109 value, format f
hi : P3109 value, format f

Output

z : P3109 value, format f

Behavior

$$\mathsf{Clamp}(x, lo, hi) \to \begin{cases} \mathsf{NaN} & \mathsf{isNaN}(x) \vee \mathsf{isNaN}(lo) \vee \mathsf{isNaN}(hi) \vee (LO > HI) \\ lo & X \leq LO \\ hi & X \geq HI \\ x & \text{Otherwise} \end{cases}$$

$$\textbf{where} \quad X = \mathsf{Decode}_f(x), LO = \mathsf{Decode}_f(lo), HI = \mathsf{Decode}_f(hi)$$

4.10.4 Comparisons

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

The below specifications are expressed with the following substitutions for compareOp and *CompareExpr*:

compareOp	*CompareExpr*
compareLess	$X < Y$
compareLessEqual	$X \leq Y$
compareEqual	$X = Y$
compareGreaterEqual	$X > Y$
compareGreater	$X \geq Y$

Signature

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

Parameters

f_x : format of x
f_y : format of y

Operands

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

Output

b : Boolean value

Behavior

compare$Op(\text{NaN}, \text{NaN}) \rightarrow \text{False}$
compare$Op(\text{NaN}, y) \rightarrow \text{False}$
compare$Op(x, \text{NaN}) \rightarrow \text{False}$
compare$Op(x, y) \rightarrow$ *CompareExpr*

where

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

4.10.5 Predicates and classification

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

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

Signature

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

Parameters

f : format of x

Operands

x : P3109 value, format f

Output

b : Boolean value

Behavior

$\mathsf{isZero}(\mathsf{NaN}) \to \mathsf{False}$
$\mathsf{isZero}(x) \to \mathsf{Decode}_f(x) = 0$

$\mathsf{isOne}(\mathsf{NaN}) \to \mathsf{False}$
$\mathsf{isOne}(x) \to \mathsf{Decode}_f(x) = 1$

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

$\mathsf{isSignMinus}(\mathsf{NaN}) \to \mathsf{True}$
$\mathsf{isSignMinus}(x) \to \mathsf{Decode}_f(x) < 0$

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

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

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

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

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

4.10.6 Classifier operation

The Classifier operation $\mathsf{class}(x)$ tells which of the eight classes x falls into as defined by Table 4.

Signature

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

Parameters

f : format of x

Operands

x : P3109 value, format f

Output

c : enumeration

Behavior

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

Table 4: Classifier operation

ClassEnum	*Condition*
clsNaN	isNaN(x)
clsNegativeInfinity	isInfinite(x) $\wedge$ isSignMinus(x)
clsNegativeNormal	isNormal(x) $\wedge$ isSignMinus(x)
clsNegativeSubnormal	isSubnormal(x) $\wedge$ isSignMinus(x)
clsZero	isZero(x)
clsPositiveSubnormal	isSubnormal(x) $\wedge \neg$isSignMinus(x)
clsPositiveNormal	isNormal(x) $\wedge \neg$isSignMinus(x)
clsPositiveInfinity	isInfinite(x) $\wedge \neg$isSignMinus(x)

4.10.7 Total order predicate

The totalOrder(x, y) predicate provides a total ordering over each P3109 format's value set.

Signature

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

Operands

f_x : format of x
f_y : format of y

Operands

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

Output

b : boolean value

Behavior

totalOrder(NaN, x) $\rightarrow$ True
totalOrder(x, NaN) $\rightarrow$ False
totalOrder$(x, y) \rightarrow$ compareLessEqual$_{f_x, f_y}(x, y)$

Note

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

4.10.8 Comparison predicates

Conforming implementations shall provide the comparison predicates defined by Table 5 and the totalOrder(x, y) predicate.

Comparison operations are two-argument predicates, and their negations, that return True or False. Comparisons may be ordered or unordered. A comparison is considered unordered iff either argument is NaN. All other comparisons are ordered.

For $\{=, >, \geq, <, \leq, \lessgtr\}$, if any argument is NaN, the result is False.

For $\{\neq, \not>, \not\geq, \not<, \not\leq, \not\lessgtr\}$, if any argument is NaN, the result is True.

Otherwise, the result of a comparison shall match the mathematical result.

Table 5: Comparison predicates and negations

Math symbol	Predicate *true relations*	Math symbol	Negation of predicate *true relations*
$=$	compareEqual *equal*	$\neq$, NOT $=$	compareNotEqual *less than, greater than, unordered*
$>$	compareGreater *greater than*	$\not>$, NOT $>$	compareNotGreater *less than, equal, unordered*
$\geq$	compareGreaterEqual *greater than, equal*	$\not\geq$, NOT $\geq$	compareLessUnordered *less than, unordered*
$<$	compareLess *less than*	$\not<$, NOT $<$	compareNotLess *greater than, equal, unordered*
$\leq$	compareLessEqual *less than, equal*	$\not\leq$, NOT $\leq$	compareGreaterUnordered *greater than, unordered*
$\lessgtr$	compareOrdered *less than, equal, greater than*	$\not\lessgtr$, NOT $\lessgtr$	compareUnordered *unordered*

5 Mixed IEEE Std 754 and P3109 operations

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.

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

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

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

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

Signature

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

Parameters

ϕ : target IEEE Std 754 format

Operands

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

Output

z : IEEE Std 754 value, format ϕ

Behavior

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

Notes

In this document, the Encode754 function is only called with arguments in the value set of format ϕ, therefore encoding is unambiguous and independent of rounding mode. This constraint on the arguments shall hold in any conforming implementation.

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

5.1 Operations

5.1.1 Conversion from IEEE Std 754 formats to P3109

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

Signature

$\mathsf{ConvertToP3109}_{\phi,f,\pi}(X) \rightarrow z$

Parameters

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

Operands

X : IEEE-754 value, format ϕ

Output

z : P3109 value, format f

Behavior

$\mathsf{ConvertToP3109}(\text{Any IEEE Std 754 NaN}) \rightarrow \mathsf{NaN}_f$
$\mathsf{ConvertToP3109}(X) \rightarrow \mathsf{Project}_{f,\pi}(X') \quad \textbf{where} \quad X' = \mathsf{AsExtendedReal}_{\phi}(X)$

5.1.2 Conversion from P3109 to IEEE Std 754

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

Signature

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

Parameters

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

Operands

x : P3109 value, format f

Output

X : IEEE-754 value, format ϕ

Behavior

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

5.1.3 Scaled Fused Multiply Add

Compute $Z = A \times 2^{s_a} + X \times Y \times 2^{s}$, with A and Z in the same format. Scaling is applied in the extended reals, before projection to the target format.

Signature

$\mathsf{ScaledFMA}_{\phi,f_x,f_y,\pi}(a, s_a, x, y, s) \rightarrow z$

Parameters

ϕ : IEEE Std 754 format of a and target, precision P_ϕ, bias B_ϕ, maximum value M_ϕ^{hi}
f_x : format of x
f_y : format of y
π : projection specification

Operands

a : IEEE Std 754 value, format ϕ
s_a : integer log-scale factor
x : P3109 value, format f_x
y : P3109 value, format f_y
s : integer log-scale factor

Output

z : IEEE Std 754 value, format ϕ

Behavior

$\mathsf{ScaledFMA}(\mathsf{NaN}, *, *, *, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(*, *, \mathsf{NaN}, *, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(*, *, *, \mathsf{NaN}, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(*, *, \pm\mathsf{Inf}, 0, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(*, *, 0, \pm\mathsf{Inf}, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(-\mathsf{Inf}, *, +\mathsf{Inf}, y > 0, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(-\mathsf{Inf}, *, -\mathsf{Inf}, y < 0, *) \rightarrow \mathsf{NaN}$
$\mathsf{ScaledFMA}(a, s_a, x, y, s) \rightarrow z$

where

$A = \mathsf{AsExtendedReal}_\phi(a)$
$X = \mathsf{Decode}_{f_x}(x)$
$Y = \mathsf{Decode}_{f_y}(y)$
$Z = A \times 2^{s_a} + X \times Y \times 2^{s}$
$Z' = \mathsf{RoundToPrecision}_{P_\phi, B_\phi, \mathsf{Rnd}_\pi}(Z)$
$Z'' = \mathsf{Saturate}_{-M_\phi^{\text{hi}}, M_\phi^{\text{hi}}, \mathsf{Signed}}(\mathsf{Sat}_\pi, \mathsf{Rnd}_\pi, Z')$
$z = \mathsf{Encode754}_\phi(Z'')$ —Z'' guaranteed representable in format ϕ

Note

Implementations shall declare the provision of ScaledFMA with specified constraints on legal operand values: "ScaledFMA, with log-scale factors in $\{-32 \ldots 31\}$, $f_x = f_y$ in $\{\mathsf{binary8p3}, \mathsf{binary8}p4\}$, ϕ in $\{\mathsf{Binary16}, \mathsf{Binary32}\}$"

Appendices

A Operations on Extended Reals

For reference, we include a definition of all relevant operations on extended reals based on the corresponding operations on non-extended reals, explicitly stating undefined cases.

Absolute value $|x| =$

$$
\begin{aligned}
|\pm\infty| &= +\infty \\
\text{otherwise } &|x|
\end{aligned}
$$

Sign $\text{sign}(x)$:

$$
\begin{aligned}
\text{sign}(+\infty) &= 1 \\
\text{sign}(-\infty) &= -1 \\
\text{sign}(x) &= -1 \text{ if } x < 0 \\
\text{otherwise } &1
\end{aligned}
$$

Negation $-x$:

$$
\begin{aligned}
--\infty &= +\infty \\
-+\infty &= -\infty \\
\text{otherwise } &-x
\end{aligned}
$$

Addition $x + y$:

$$
\begin{aligned}
-\infty + +\infty = +\infty + -\infty &= \text{undefined} \\
-\infty + * = * + -\infty &= -\infty \\
+\infty + * = * + +\infty &= +\infty \\
\text{otherwise } x + y &
\end{aligned}
$$

Subtraction $x - y$:

$$
\begin{aligned}
-\infty - -\infty = +\infty - +\infty &= \text{undefined} \\
-\infty - * = * - +\infty &= -\infty \\
+\infty - * = * - -\infty &= +\infty \\
\text{otherwise } x - y &
\end{aligned}
$$

Natural logarithm $\log_e(x)$:

$$
\begin{aligned}
\log_e(-\infty) = \log_e(+\infty) &= \text{undefined} \\
\log_e(x) &= \text{undefined} \quad \text{if } x < 0 \\
\log_e(x) &= -\infty \quad \text{if } x = 0 \\
\text{otherwise : } &\log_e(x)
\end{aligned}
$$

Multiplication $x \times y$:

$$
\begin{aligned}
\pm\infty \times 0 = 0 \times \pm\infty &= \text{undefined} \\
-\infty \times -\infty = +\infty \times +\infty &= +\infty \\
-\infty \times +\infty = +\infty \times -\infty &= -\infty \\
+\infty \times y = x \times +\infty &= +\infty \quad \text{if } x, y > 0 \\
+\infty \times y = x \times +\infty &= -\infty \quad \text{if } x, y < 0 \\
-\infty \times y = x \times -\infty &= -\infty \quad \text{if } x, y > 0 \\
-\infty \times y = x \times -\infty &= +\infty \quad \text{if } x, y < 0 \\
\text{otherwise } x \times y &
\end{aligned}
$$

Division x/y:

$$
\begin{aligned}
\pm\infty/\pm\infty = */0 &= \text{undefined} \\
x/-\infty = x/+\infty &= 0 \\
+\infty/* &= +\infty \quad \text{if } y > 0 \\
+\infty/* &= -\infty \quad \text{if } y < 0 \\
-\infty/* &= -\infty \quad \text{if } y > 0 \\
-\infty/* &= +\infty \quad \text{if } y < 0 \\
-\infty/+\infty = +\infty/-\infty &= -\infty \\
\text{otherwise } x/y &
\end{aligned}
$$

Square root $\sqrt{x}$:

$$
\begin{aligned}
\sqrt{-\infty} &= \text{undefined} \\
\sqrt{x} &= \text{undefined} \quad \text{if } x < 0 \\
\sqrt{+\infty} &= +\infty \\
\text{otherwise } \sqrt{x} &
\end{aligned}
$$

Binary logarithm $\log_2(x)$:

$$
\begin{aligned}
\log_2(-\infty) = \log_2(+\infty) &= \text{undefined} \\
\log_2(x) &= \text{undefined} \quad \text{if } x < 0 \\
\log_2(x) &= -\infty \quad \text{if } x = 0 \\
\text{otherwise } &\log_2(x)
\end{aligned}
$$

Natural exponential e^x:

$e^{+\infty} = +\infty$

$e^{-\infty} = 0$

otherwise e^x

Natural logarithm $\log_e(x)$:

$\log_e(-\infty) = \log_e(+\infty)$ = undefined

$\log_e(x)$ = undefined **if** $x < 0$

$\log_e(x)$ $= -\infty$ **if** $x = 0$

otherwise : $\log_e(x)$

Binary exponential 2^x:

$2^{+\infty} = +\infty$

$2^{-\infty} = 0$

otherwise 2^x

Less-than $x < y$:

$-\infty < -\infty \quad = +\infty < +\infty = +\infty < -\infty \quad = false$

$-\infty < +\infty \quad = true$

$-\infty < * \quad = * < +\infty \quad = true$

$+\infty < * \quad = * < -\infty \quad = false$

otherwise $x < y$

B Rationales

B.1 Use of infinity in computation of attention masks

This section expands the rationale in §3.6. A common use for ∞ is to create masks, for example, in Transformer models in machine learning [5].

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

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

where τ is a "temperature" or "base" parameter [6]. 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 binary8p4value). This is not in itself a difficulty: if all the A values are bounded (e.g. the results of a softmax operation are bounded above by 1.0), then $\exp(1.0 - 224.0)$ is an extremely small number, which will certainly round to zero. Therefore, an explicit representation of infinity is *not* needed in order for this computation to yield its desired value.

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

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

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

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

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

while

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

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

C External Formats

This table summarizes the points of agreement and of difference between the 8-bit formats proposed in this document and a number of existing format families, some of which have hardware implementations.

OCP: Open Compute Platform [7], describing hardware implementations including nVidia, Intel, and ARM.

AGQ: AMD, Graphcore, Qualcomm[8], implemented in Graphcore's C600 product, and AMD's gfx940.

TSL: Tesla Dojo Technology [9], A Guide to Tesla's Configurable floating-point Formats & Arithmetic

Format	P3109			OCP			AGQ		TSL	
Subformat	k8p3es	k8p4es	k8p1fu	E8M0	E5M2	E4M3	E5M2	E4M3	E5M2	E4M3
Special values shared	Y			N			Y		N	
Exactly one NaN	Y			Y	N		Y		Y	
Positive and negative ∞	Y		N	N	Y	N	N		N	
Include negative zero	N			N	Y		N		N	
Max exponent emax	15	7	126	127	15	8	15	7	N/A	N/A

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

D Value Tables

Sample value tables mapping bit strings to value sets are provided in this section.

A typical entry is of the form:

```
HEX  = BINARY      = BINARY_FLOAT   = DECIMAL
0x0b = 0_0001_011 = +0b1.011×2^-7 = 0.0107421875
```

Where the fields are interpreted as follows:

`HEX`	Hexadecimal encoding of the code point
`BINARY`	Binary expansion of the code point, underscores separate `< sign >_< exponent >_< significand >`
`BINARY_FLOAT`	The precise float value as a binary fraction followed by `2^`e with exponent e
`DECIMAL`	A decimal expansion of the value. If the expansion is not an exact representation of the precise float value, the equals sign is replaced by "approximately equals" ($\approx$).

In addition, entries for subnormal and special values are rendered in color as follows:

`0x05 = 0_0000_101 = +0b0.101×2^-7 = 0.0048828125`	Subnormal value
`0x80 = 1_0000_000 = NaN`	Special value (NaN, +Inf, -Inf)

D.1 binary8p3se, emin $= -15$, emax $= 15$

0x00 = 0_00000_00 = 0.0
0x01 = 0_00000_01 = +0b0.01×2^-15 ≈ 7.6293945E-06
0x02 = 0_00000_10 = +0b0.10×2^-15 ≈ 1.5258789E-05
0x03 = 0_00000_11 = +0b0.11×2^-15 ≈ 2.2888184E-05
0x04 = 0_00001_00 = +0b1.00×2^-15 ≈ 3.0517578E-05
0x05 = 0_00001_01 = +0b1.01×2^-15 ≈ 3.8146973E-05
0x06 = 0_00001_10 = +0b1.10×2^-15 ≈ 4.5776367E-05
0x07 = 0_00001_11 = +0b1.11×2^-15 ≈ 5.3405762E-05
0x08 = 0_00010_00 = +0b1.00×2^-14 ≈ 6.1035156E-05
0x09 = 0_00010_01 = +0b1.01×2^-14 ≈ 7.6293945E-05
0x0a = 0_00010_10 = +0b1.10×2^-14 ≈ 9.1552734E-05
0x0b = 0_00010_11 = +0b1.11×2^-14 ≈ 0.00010681152
0x0c = 0_00011_00 = +0b1.00×2^-13 ≈ 0.00012207031
0x0d = 0_00011_01 = +0b1.01×2^-13 ≈ 0.00015258789
0x0e = 0_00011_10 = +0b1.10×2^-13 ≈ 0.00018310547
0x0f = 0_00011_11 = +0b1.11×2^-13 ≈ 0.00021362305
0x10 = 0_00100_00 = +0b1.00×2^-12 = 0.000244140625
0x11 = 0_00100_01 = +0b1.01×2^-12 ≈ 0.00030517578
0x12 = 0_00100_10 = +0b1.10×2^-12 ≈ 0.00036621094
0x13 = 0_00100_11 = +0b1.11×2^-12 ≈ 0.00042724609
0x14 = 0_00101_00 = +0b1.00×2^-11 = 0.0004882812 5
0x15 = 0_00101_01 = +0b1.01×2^-11 ≈ 0.00061035156
0x16 = 0_00101_10 = +0b1.10×2^-11 = 0.000732421875
0x17 = 0_00101_11 = +0b1.11×2^-11 ≈ 0.00085449219
0x18 = 0_00110_00 = +0b1.00×2^-10 = 0.0009765625
0x19 = 0_00110_01 = +0b1.01×2^-10 = 0.001220703125
0x1a = 0_00110_10 = +0b1.10×2^-10 = 0.00146484375
0x1b = 0_00110_11 = +0b1.11×2^-10 = 0.001708984375
0x1c = 0_00111_00 = +0b1.00×2^-9 = 0.001953125
0x1d = 0_00111_01 = +0b1.01×2^-9 = 0.00244140625
0x1e = 0_00111_10 = +0b1.10×2^-9 = 0.0029296875
0x1f = 0_00111_11 = +0b1.11×2^-9 = 0.00341796875
0x20 = 0_01000_00 = +0b1.00×2^-8 = 0.00390625
0x21 = 0_01000_01 = +0b1.01×2^-8 = 0.0048828125
0x22 = 0_01000_10 = +0b1.10×2^-8 = 0.005859375
0x23 = 0_01000_11 = +0b1.11×2^-8 = 0.0068359375
0x24 = 0_01001_00 = +0b1.00×2^-7 = 0.0078125
0x25 = 0_01001_01 = +0b1.01×2^-7 = 0.009765625
0x26 = 0_01001_10 = +0b1.10×2^-7 = 0.01171875
0x27 = 0_01001_11 = +0b1.11×2^-7 = 0.013671875
0x28 = 0_01010_00 = +0b1.00×2^-6 = 0.015625
0x29 = 0_01010_01 = +0b1.01×2^-6 = 0.01953125
0x2a = 0_01010_10 = +0b1.10×2^-6 = 0.0234375
0x2b = 0_01010_11 = +0b1.11×2^-6 = 0.02734375
0x2c = 0_01011_00 = +0b1.00×2^-5 = 0.03125
0x2d = 0_01011_01 = +0b1.01×2^-5 = 0.0390625
0x2e = 0_01011_10 = +0b1.10×2^-5 = 0.046875
0x2f = 0_01011_11 = +0b1.11×2^-5 = 0.0546875
0x30 = 0_01100_00 = +0b1.00×2^-4 = 0.0625
0x31 = 0_01100_01 = +0b1.01×2^-4 = 0.078125
0x32 = 0_01100_10 = +0b1.10×2^-4 = 0.09375
0x33 = 0_01100_11 = +0b1.11×2^-4 = 0.109375
0x34 = 0_01101_00 = +0b1.00×2^-3 = 0.125
0x35 = 0_01101_01 = +0b1.01×2^-3 = 0.15625
0x36 = 0_01101_10 = +0b1.10×2^-3 = 0.1875
0x37 = 0_01101_11 = +0b1.11×2^-3 = 0.21875
0x38 = 0_01110_00 = +0b1.00×2^-2 = 0.25
0x39 = 0_01110_01 = +0b1.01×2^-2 = 0.3125
0x3a = 0_01110_10 = +0b1.10×2^-2 = 0.375
0x3b = 0_01110_11 = +0b1.11×2^-2 = 0.4375
0x3c = 0_01111_00 = +0b1.00×2^-1 = 0.5
0x3d = 0_01111_01 = +0b1.01×2^-1 = 0.625
0x3e = 0_01111_10 = +0b1.10×2^-1 = 0.75
0x3f = 0_01111_11 = +0b1.11×2^-1 = 0.875
0x40 = 0_10000_00 = +0b1.00×2^0 = 1.0
0x41 = 0_10000_01 = +0b1.01×2^0 = 1.25
0x42 = 0_10000_10 = +0b1.10×2^0 = 1.5
0x43 = 0_10000_11 = +0b1.11×2^0 = 1.75
0x44 = 0_10001_00 = +0b1.00×2^1 = 2.0
0x45 = 0_10001_01 = +0b1.01×2^1 = 2.5
0x46 = 0_10001_10 = +0b1.10×2^1 = 3.0
0x47 = 0_10001_11 = +0b1.11×2^1 = 3.5
0x48 = 0_10010_00 = +0b1.00×2^2 = 4.0
0x49 = 0_10010_01 = +0b1.01×2^2 = 5.0
0x4a = 0_10010_10 = +0b1.10×2^2 = 6.0
0x4b = 0_10010_11 = +0b1.11×2^2 = 7.0
0x4c = 0_10011_00 = +0b1.00×2^3 = 8.0
0x4d = 0_10011_01 = +0b1.01×2^3 = 10.0
0x4e = 0_10011_10 = +0b1.10×2^3 = 12.0
0x4f = 0_10011_11 = +0b1.11×2^3 = 14.0
0x50 = 0_10100_00 = +0b1.00×2^4 = 16.0
0x51 = 0_10100_01 = +0b1.01×2^4 = 20.0
0x52 = 0_10100_10 = +0b1.10×2^4 = 24.0
0x53 = 0_10100_11 = +0b1.11×2^4 = 28.0
0x54 = 0_10101_00 = +0b1.00×2^5 = 32.0
0x55 = 0_10101_01 = +0b1.01×2^5 = 40.0
0x56 = 0_10101_10 = +0b1.10×2^5 = 48.0
0x57 = 0_10101_11 = +0b1.11×2^5 = 56.0
0x58 = 0_10110_00 = +0b1.00×2^6 = 64.0
0x59 = 0_10110_01 = +0b1.01×2^6 = 80.0
0x5a = 0_10110_10 = +0b1.10×2^6 = 96.0
0x5b = 0_10110_11 = +0b1.11×2^6 = 112.0
0x5c = 0_10111_00 = +0b1.00×2^7 = 128.0
0x5d = 0_10111_01 = +0b1.01×2^7 = 160.0
0x5e = 0_10111_10 = +0b1.10×2^7 = 192.0
0x5f = 0_10111_11 = +0b1.11×2^7 = 224.0
0x60 = 0_11000_00 = +0b1.00×2^8 = 256.0
0x61 = 0_11000_01 = +0b1.01×2^8 = 320.0
0x62 = 0_11000_10 = +0b1.10×2^8 = 384.0
0x63 = 0_11000_11 = +0b1.11×2^8 = 448.0
0x64 = 0_11001_00 = +0b1.00×2^9 = 512.0
0x65 = 0_11001_01 = +0b1.01×2^9 = 640.0
0x66 = 0_11001_10 = +0b1.10×2^9 = 768.0
0x67 = 0_11001_11 = +0b1.11×2^9 = 896.0
0x68 = 0_11010_00 = +0b1.00×2^10 = 1024.0
0x69 = 0_11010_01 = +0b1.01×2^10 = 1280.0
0x6a = 0_11010_10 = +0b1.10×2^10 = 1536.0
0x6b = 0_11010_11 = +0b1.11×2^10 = 1792.0
0x6c = 0_11011_00 = +0b1.00×2^11 = 2048.0
0x6d = 0_11011_01 = +0b1.01×2^11 = 2560.0
0x6e = 0_11011_10 = +0b1.10×2^11 = 3072.0
0x6f = 0_11011_11 = +0b1.11×2^11 = 3584.0
0x70 = 0_11100_00 = +0b1.00×2^12 = 4096.0
0x71 = 0_11100_01 = +0b1.01×2^12 = 5120.0
0x72 = 0_11100_10 = +0b1.10×2^12 = 6144.0
0x73 = 0_11100_11 = +0b1.11×2^12 = 7168.0
0x74 = 0_11101_00 = +0b1.00×2^13 = 8192.0
0x75 = 0_11101_01 = +0b1.01×2^13 = 10240.0
0x76 = 0_11101_10 = +0b1.10×2^13 = 12288.0
0x77 = 0_11101_11 = +0b1.11×2^13 = 14336.0
0x78 = 0_11110_00 = +0b1.00×2^14 = 16384.0
0x79 = 0_11110_01 = +0b1.01×2^14 = 20480.0
0x7a = 0_11110_10 = +0b1.10×2^14 = 24576.0
0x7b = 0_11110_11 = +0b1.11×2^14 = 28672.0
0x7c = 0_11111_00 = +0b1.00×2^15 = 32768.0
0x7d = 0_11111_01 = +0b1.01×2^15 = 40960.0
0x7e = 0_11111_10 = +0b1.10×2^15 = 49152.0
0x7f = 0_11111_11 = +Inf
0x80 = 1_00000_00 = NaN
0x81 = 1_00000_01 = −0b0.01×2^-15 ≈ −7.6293945E-06
0x82 = 1_00000_10 = −0b0.10×2^-15 ≈ −1.5258789E-05
0x83 = 1_00000_11 = −0b0.11×2^-15 ≈ −2.2888184E-05
0x84 = 1_00001_00 = −0b1.00×2^-15 ≈ −3.0517578E-05
0x85 = 1_00001_01 = −0b1.01×2^-15 ≈ −3.8146973E-05
0x86 = 1_00001_10 = −0b1.10×2^-15 ≈ −4.5776367E-05
0x87 = 1_00001_11 = −0b1.11×2^-15 ≈ −5.3405762E-05
0x88 = 1_00010_00 = −0b1.00×2^-14 ≈ −6.1035156E-05
0x89 = 1_00010_01 = −0b1.01×2^-14 ≈ −7.6293945E-05
0x8a = 1_00010_10 = −0b1.10×2^-14 ≈ −9.1552734E-05
0x8b = 1_00010_11 = −0b1.11×2^-14 ≈ -0.00010681152
0x8c = 1_00011_00 = −0b1.00×2^-13 ≈ -0.00012207031
0x8d = 1_00011_01 = −0b1.01×2^-13 ≈ -0.00015258789
0x8e = 1_00011_10 = −0b1.10×2^-13 ≈ -0.00018310547
0x8f = 1_00011_11 = −0b1.11×2^-13 ≈ -0.00021362305
0x90 = 1_00100_00 = −0b1.00×2^-12 ≈ -0.00024414062
0x91 = 1_00100_01 = −0b1.01×2^-12 ≈ -0.00030517578
0x92 = 1_00100_10 = −0b1.10×2^-12 ≈ -0.00036621094
0x93 = 1_00100_11 = −0b1.11×2^-12 ≈ -0.00042724609
0x94 = 1_00101_00 = −0b1.00×2^-11 = -0.0004882812 5
0x95 = 1_00101_01 = −0b1.01×2^-11 ≈ -0.00061035156
0x96 = 1_00101_10 = −0b1.10×2^-11 ≈ -0.00073242188
0x97 = 1_00101_11 = −0b1.11×2^-11 ≈ -0.00085449219
0x98 = 1_00110_00 = −0b1.00×2^-10 = -0.0009765625
0x99 = 1_00110_01 = −0b1.01×2^-10 ≈ -0.0012207031
0x9a = 1_00110_10 = −0b1.10×2^-10 = -0.00146484375
0x9b = 1_00110_11 = −0b1.11×2^-10 ≈ -0.0017089844
0x9c = 1_00111_00 = −0b1.00×2^-9 = -0.001953125
0x9d = 1_00111_01 = −0b1.01×2^-9 = -0.00244140625
0x9e = 1_00111_10 = −0b1.10×2^-9 = -0.0029296875
0x9f = 1_00111_11 = −0b1.11×2^-9 = -0.00341796875
0xa0 = 1_01000_00 = −0b1.00×2^-8 = -0.00390625
0xa1 = 1_01000_01 = −0b1.01×2^-8 = -0.0048828125
0xa2 = 1_01000_10 = −0b1.10×2^-8 = -0.005859375
0xa3 = 1_01000_11 = −0b1.11×2^-8 = -0.0068359375
0xa4 = 1_01001_00 = −0b1.00×2^-7 = -0.0078125
0xa5 = 1_01001_01 = −0b1.01×2^-7 = -0.009765625
0xa6 = 1_01001_10 = −0b1.10×2^-7 = -0.01171875
0xa7 = 1_01001_11 = −0b1.11×2^-7 = -0.013671875
0xa8 = 1_01010_00 = −0b1.00×2^-6 = -0.015625
0xa9 = 1_01010_01 = −0b1.01×2^-6 = -0.01953125
0xaa = 1_01010_10 = −0b1.10×2^-6 = -0.0234375
0xab = 1_01010_11 = −0b1.11×2^-6 = -0.02734375
0xac = 1_01011_00 = −0b1.00×2^-5 = -0.03125
0xad = 1_01011_01 = −0b1.01×2^-5 = -0.0390625
0xae = 1_01011_10 = −0b1.10×2^-5 = -0.046875
0xaf = 1_01011_11 = −0b1.11×2^-5 = -0.0546875
0xb0 = 1_01100_00 = −0b1.00×2^-4 = -0.0625
0xb1 = 1_01100_01 = −0b1.01×2^-4 = -0.078125
0xb2 = 1_01100_10 = −0b1.10×2^-4 = -0.09375
0xb3 = 1_01100_11 = −0b1.11×2^-4 = -0.109375
0xb4 = 1_01101_00 = −0b1.00×2^-3 = -0.125
0xb5 = 1_01101_01 = −0b1.01×2^-3 = -0.15625
0xb6 = 1_01101_10 = −0b1.10×2^-3 = -0.1875
0xb7 = 1_01101_11 = −0b1.11×2^-3 = -0.21875
0xb8 = 1_01110_00 = −0b1.00×2^-2 = -0.25
0xb9 = 1_01110_01 = −0b1.01×2^-2 = -0.3125
0xba = 1_01110_10 = −0b1.10×2^-2 = -0.375
0xbb = 1_01110_11 = −0b1.11×2^-2 = -0.4375
0xbc = 1_01111_00 = −0b1.00×2^-1 = -0.5
0xbd = 1_01111_01 = −0b1.01×2^-1 = -0.625
0xbe = 1_01111_10 = −0b1.10×2^-1 = -0.75
0xbf = 1_01111_11 = −0b1.11×2^-1 = -0.875
0xc0 = 1_10000_00 = −0b1.00×2^0 = -1.0
0xc1 = 1_10000_01 = −0b1.01×2^0 = -1.25
0xc2 = 1_10000_10 = −0b1.10×2^0 = -1.5
0xc3 = 1_10000_11 = −0b1.11×2^0 = -1.75
0xc4 = 1_10001_00 = −0b1.00×2^1 = -2.0
0xc5 = 1_10001_01 = −0b1.01×2^1 = -2.5
0xc6 = 1_10001_10 = −0b1.10×2^1 = -3.0
0xc7 = 1_10001_11 = −0b1.11×2^1 = -3.5
0xc8 = 1_10010_00 = −0b1.00×2^2 = -4.0
0xc9 = 1_10010_01 = −0b1.01×2^2 = -5.0
0xca = 1_10010_10 = −0b1.10×2^2 = -6.0
0xcb = 1_10010_11 = −0b1.11×2^2 = -7.0
0xcc = 1_10011_00 = −0b1.00×2^3 = -8.0
0xcd = 1_10011_01 = −0b1.01×2^3 = -10.0
0xce = 1_10011_10 = −0b1.10×2^3 = -12.0
0xcf = 1_10011_11 = −0b1.11×2^3 = -14.0
0xd0 = 1_10100_00 = −0b1.00×2^4 = -16.0
0xd1 = 1_10100_01 = −0b1.01×2^4 = -20.0
0xd2 = 1_10100_10 = −0b1.10×2^4 = -24.0
0xd3 = 1_10100_11 = −0b1.11×2^4 = -28.0
0xd4 = 1_10101_00 = −0b1.00×2^5 = -32.0
0xd5 = 1_10101_01 = −0b1.01×2^5 = -40.0
0xd6 = 1_10101_10 = −0b1.10×2^5 = -48.0
0xd7 = 1_10101_11 = −0b1.11×2^5 = -56.0
0xd8 = 1_10110_00 = −0b1.00×2^6 = -64.0
0xd9 = 1_10110_01 = −0b1.01×2^6 = -80.0
0xda = 1_10110_10 = −0b1.10×2^6 = -96.0
0xdb = 1_10110_11 = −0b1.11×2^6 = -112.0
0xdc = 1_10111_00 = −0b1.00×2^7 = -128.0
0xdd = 1_10111_01 = −0b1.01×2^7 = -160.0
0xde = 1_10111_10 = −0b1.10×2^7 = -192.0
0xdf = 1_10111_11 = −0b1.11×2^7 = -224.0
0xe0 = 1_11000_00 = −0b1.00×2^8 = -256.0
0xe1 = 1_11000_01 = −0b1.01×2^8 = -320.0
0xe2 = 1_11000_10 = −0b1.10×2^8 = -384.0
0xe3 = 1_11000_11 = −0b1.11×2^8 = -448.0
0xe4 = 1_11001_00 = −0b1.00×2^9 = -512.0
0xe5 = 1_11001_01 = −0b1.01×2^9 = -640.0
0xe6 = 1_11001_10 = −0b1.10×2^9 = -768.0
0xe7 = 1_11001_11 = −0b1.11×2^9 = -896.0
0xe8 = 1_11010_00 = −0b1.00×2^10 = -1024.0
0xe9 = 1_11010_01 = −0b1.01×2^10 = -1280.0
0xea = 1_11010_10 = −0b1.10×2^10 = -1536.0
0xeb = 1_11010_11 = −0b1.11×2^10 = -1792.0
0xec = 1_11011_00 = −0b1.00×2^11 = -2048.0
0xed = 1_11011_01 = −0b1.01×2^11 = -2560.0
0xee = 1_11011_10 = −0b1.10×2^11 = -3072.0
0xef = 1_11011_11 = −0b1.11×2^11 = -3584.0
0xf0 = 1_11100_00 = −0b1.00×2^12 = -4096.0
0xf1 = 1_11100_01 = −0b1.01×2^12 = -5120.0
0xf2 = 1_11100_10 = −0b1.10×2^12 = -6144.0
0xf3 = 1_11100_11 = −0b1.11×2^12 = -7168.0
0xf4 = 1_11101_00 = −0b1.00×2^13 = -8192.0
0xf5 = 1_11101_01 = −0b1.01×2^13 = -10240.0
0xf6 = 1_11101_10 = −0b1.10×2^13 = -12288.0
0xf7 = 1_11101_11 = −0b1.11×2^13 = -14336.0
0xf8 = 1_11110_00 = −0b1.00×2^14 = -16384.0
0xf9 = 1_11110_01 = −0b1.01×2^14 = -20480.0
0xfa = 1_11110_10 = −0b1.10×2^14 = -24576.0
0xfb = 1_11110_11 = −0b1.11×2^14 = -28672.0
0xfc = 1_11111_00 = −0b1.00×2^15 = -32768.0
0xfd = 1_11111_01 = −0b1.01×2^15 = -40960.0
0xfe = 1_11111_10 = −0b1.10×2^15 = -49152.0
0xff = 1_11111_11 = −Inf

D.2 binary8p4se, emin $= -7$, emax $= 7$

0x00 = 0_0000_000 = 0.0
0x01 = 0_0000_001 = +0b0.001×2^-7 = 0.0009765625
0x02 = 0_0000_010 = +0b0.010×2^-7 = 0.001953125
0x03 = 0_0000_011 = +0b0.011×2^-7 = 0.0029296875
0x04 = 0_0000_100 = +0b0.100×2^-7 = 0.00390625
0x05 = 0_0000_101 = +0b0.101×2^-7 = 0.0048828125
0x06 = 0_0000_110 = +0b0.110×2^-7 = 0.005859375
0x07 = 0_0000_111 = +0b0.111×2^-7 = 0.0068359375
0x08 = 0_0001_000 = +0b1.000×2^-7 = 0.0078125
0x09 = 0_0001_001 = +0b1.001×2^-7 = 0.0087890625
0x0a = 0_0001_010 = +0b1.010×2^-7 = 0.009765625
0x0b = 0_0001_011 = +0b1.011×2^-7 = 0.0107421875
0x0c = 0_0001_100 = +0b1.100×2^-7 = 0.01171875
0x0d = 0_0001_101 = +0b1.101×2^-7 = 0.0126953125
0x0e = 0_0001_110 = +0b1.110×2^-7 = 0.013671875
0x0f = 0_0001_111 = +0b1.111×2^-7 = 0.0146484375
0x10 = 0_0010_000 = +0b1.000×2^-6 = 0.015625
0x11 = 0_0010_001 = +0b1.001×2^-6 = 0.017578125
0x12 = 0_0010_010 = +0b1.010×2^-6 = 0.01953125
0x13 = 0_0010_011 = +0b1.011×2^-6 = 0.021484375
0x14 = 0_0010_100 = +0b1.100×2^-6 = 0.0234375
0x15 = 0_0010_101 = +0b1.101×2^-6 = 0.025390625
0x16 = 0_0010_110 = +0b1.110×2^-6 = 0.02734375
0x17 = 0_0010_111 = +0b1.111×2^-6 = 0.029296875
0x18 = 0_0011_000 = +0b1.000×2^-5 = 0.03125
0x19 = 0_0011_001 = +0b1.001×2^-5 = 0.03515625
0x1a = 0_0011_010 = +0b1.010×2^-5 = 0.0390625
0x1b = 0_0011_011 = +0b1.011×2^-5 = 0.04296875
0x1c = 0_0011_100 = +0b1.100×2^-5 = 0.046875
0x1d = 0_0011_101 = +0b1.101×2^-5 = 0.05078125
0x1e = 0_0011_110 = +0b1.110×2^-5 = 0.0546875
0x1f = 0_0011_111 = +0b1.111×2^-5 = 0.05859375
0x20 = 0_0100_000 = +0b1.000×2^-4 = 0.0625
0x21 = 0_0100_001 = +0b1.001×2^-4 = 0.0703125
0x22 = 0_0100_010 = +0b1.010×2^-4 = 0.078125
0x23 = 0_0100_011 = +0b1.011×2^-4 = 0.0859375
0x24 = 0_0100_100 = +0b1.100×2^-4 = 0.09375
0x25 = 0_0100_101 = +0b1.101×2^-4 = 0.1015625
0x26 = 0_0100_110 = +0b1.110×2^-4 = 0.109375
0x27 = 0_0100_111 = +0b1.111×2^-4 = 0.1171875
0x28 = 0_0101_000 = +0b1.000×2^-3 = 0.125
0x29 = 0_0101_001 = +0b1.001×2^-3 = 0.140625
0x2a = 0_0101_010 = +0b1.010×2^-3 = 0.15625
0x2b = 0_0101_011 = +0b1.011×2^-3 = 0.171875
0x2c = 0_0101_100 = +0b1.100×2^-3 = 0.1875
0x2d = 0_0101_101 = +0b1.101×2^-3 = 0.203125
0x2e = 0_0101_110 = +0b1.110×2^-3 = 0.21875
0x2f = 0_0101_111 = +0b1.111×2^-3 = 0.234375
0x30 = 0_0110_000 = +0b1.000×2^-2 = 0.25
0x31 = 0_0110_001 = +0b1.001×2^-2 = 0.28125
0x32 = 0_0110_010 = +0b1.010×2^-2 = 0.3125
0x33 = 0_0110_011 = +0b1.011×2^-2 = 0.34375
0x34 = 0_0110_100 = +0b1.100×2^-2 = 0.375
0x35 = 0_0110_101 = +0b1.101×2^-2 = 0.40625
0x36 = 0_0110_110 = +0b1.110×2^-2 = 0.4375
0x37 = 0_0110_111 = +0b1.111×2^-2 = 0.46875
0x38 = 0_0111_000 = +0b1.000×2^-1 = 0.5
0x39 = 0_0111_001 = +0b1.001×2^-1 = 0.5625
0x3a = 0_0111_010 = +0b1.010×2^-1 = 0.625
0x3b = 0_0111_011 = +0b1.011×2^-1 = 0.6875
0x3c = 0_0111_100 = +0b1.100×2^-1 = 0.75
0x3d = 0_0111_101 = +0b1.101×2^-1 = 0.8125
0x3e = 0_0111_110 = +0b1.110×2^-1 = 0.875
0x3f = 0_0111_111 = +0b1.111×2^-1 = 0.9375
0x40 = 0_1000_000 = +0b1.000×2^0 = 1.0
0x41 = 0_1000_001 = +0b1.001×2^0 = 1.125
0x42 = 0_1000_010 = +0b1.010×2^0 = 1.25
0x43 = 0_1000_011 = +0b1.011×2^0 = 1.375
0x44 = 0_1000_100 = +0b1.100×2^0 = 1.5
0x45 = 0_1000_101 = +0b1.101×2^0 = 1.625
0x46 = 0_1000_110 = +0b1.110×2^0 = 1.75
0x47 = 0_1000_111 = +0b1.111×2^0 = 1.875
0x48 = 0_1001_000 = +0b1.000×2^1 = 2.0
0x49 = 0_1001_001 = +0b1.001×2^1 = 2.25
0x4a = 0_1001_010 = +0b1.010×2^1 = 2.5
0x4b = 0_1001_011 = +0b1.011×2^1 = 2.75
0x4c = 0_1001_100 = +0b1.100×2^1 = 3.0
0x4d = 0_1001_101 = +0b1.101×2^1 = 3.25
0x4e = 0_1001_110 = +0b1.110×2^1 = 3.5
0x4f = 0_1001_111 = +0b1.111×2^1 = 3.75
0x50 = 0_1010_000 = +0b1.000×2^2 = 4.0
0x51 = 0_1010_001 = +0b1.001×2^2 = 4.5
0x52 = 0_1010_010 = +0b1.010×2^2 = 5.0
0x53 = 0_1010_011 = +0b1.011×2^2 = 5.5
0x54 = 0_1010_100 = +0b1.100×2^2 = 6.0
0x55 = 0_1010_101 = +0b1.101×2^2 = 6.5
0x56 = 0_1010_110 = +0b1.110×2^2 = 7.0
0x57 = 0_1010_111 = +0b1.111×2^2 = 7.5
0x58 = 0_1011_000 = +0b1.000×2^3 = 8.0
0x59 = 0_1011_001 = +0b1.001×2^3 = 9.0
0x5a = 0_1011_010 = +0b1.010×2^3 = 10.0
0x5b = 0_1011_011 = +0b1.011×2^3 = 11.0
0x5c = 0_1011_100 = +0b1.100×2^3 = 12.0
0x5d = 0_1011_101 = +0b1.101×2^3 = 13.0
0x5e = 0_1011_110 = +0b1.110×2^3 = 14.0
0x5f = 0_1011_111 = +0b1.111×2^3 = 15.0
0x60 = 0_1100_000 = +0b1.000×2^4 = 16.0
0x61 = 0_1100_001 = +0b1.001×2^4 = 18.0
0x62 = 0_1100_010 = +0b1.010×2^4 = 20.0
0x63 = 0_1100_011 = +0b1.011×2^4 = 22.0
0x64 = 0_1100_100 = +0b1.100×2^4 = 24.0
0x65 = 0_1100_101 = +0b1.101×2^4 = 26.0
0x66 = 0_1100_110 = +0b1.110×2^4 = 28.0
0x67 = 0_1100_111 = +0b1.111×2^4 = 30.0
0x68 = 0_1101_000 = +0b1.000×2^5 = 32.0
0x69 = 0_1101_001 = +0b1.001×2^5 = 36.0
0x6a = 0_1101_010 = +0b1.010×2^5 = 40.0
0x6b = 0_1101_011 = +0b1.011×2^5 = 44.0
0x6c = 0_1101_100 = +0b1.100×2^5 = 48.0
0x6d = 0_1101_101 = +0b1.101×2^5 = 52.0
0x6e = 0_1101_110 = +0b1.110×2^5 = 56.0
0x6f = 0_1101_111 = +0b1.111×2^5 = 60.0
0x70 = 0_1110_000 = +0b1.000×2^6 = 64.0
0x71 = 0_1110_001 = +0b1.001×2^6 = 72.0
0x72 = 0_1110_010 = +0b1.010×2^6 = 80.0
0x73 = 0_1110_011 = +0b1.011×2^6 = 88.0
0x74 = 0_1110_100 = +0b1.100×2^6 = 96.0
0x75 = 0_1110_101 = +0b1.101×2^6 = 104.0
0x76 = 0_1110_110 = +0b1.110×2^6 = 112.0
0x77 = 0_1110_111 = +0b1.111×2^6 = 120.0
0x78 = 0_1111_000 = +0b1.000×2^7 = 128.0
0x79 = 0_1111_001 = +0b1.001×2^7 = 144.0
0x7a = 0_1111_010 = +0b1.010×2^7 = 160.0
0x7b = 0_1111_011 = +0b1.011×2^7 = 176.0
0x7c = 0_1111_100 = +0b1.100×2^7 = 192.0
0x7d = 0_1111_101 = +0b1.101×2^7 = 208.0
0x7e = 0_1111_110 = +0b1.110×2^7 = 224.0
0x7f = 0_1111_111 = +Inf
0x80 = 1_0000_000 = NaN
0x81 = 1_0000_001 = −0b0.001×2^-7 = -0.0009765625
0x82 = 1_0000_010 = −0b0.010×2^-7 = -0.001953125
0x83 = 1_0000_011 = −0b0.011×2^-7 = -0.0029296875
0x84 = 1_0000_100 = −0b0.100×2^-7 = -0.00390625
0x85 = 1_0000_101 = −0b0.101×2^-7 = -0.0048828125
0x86 = 1_0000_110 = −0b0.110×2^-7 = -0.005859375
0x87 = 1_0000_111 = −0b0.111×2^-7 = -0.0068359375
0x88 = 1_0001_000 = −0b1.000×2^-7 = -0.0078125
0x89 = 1_0001_001 = −0b1.001×2^-7 = -0.0087890625
0x8a = 1_0001_010 = −0b1.010×2^-7 = -0.009765625
0x8b = 1_0001_011 = −0b1.011×2^-7 = -0.0107421875
0x8c = 1_0001_100 = −0b1.100×2^-7 = -0.01171875
0x8d = 1_0001_101 = −0b1.101×2^-7 = -0.0126953125
0x8e = 1_0001_110 = −0b1.110×2^-7 = -0.013671875
0x8f = 1_0001_111 = −0b1.111×2^-7 = -0.0146484375
0x90 = 1_0010_000 = −0b1.000×2^-6 = -0.015625
0x91 = 1_0010_001 = −0b1.001×2^-6 = -0.017578125
0x92 = 1_0010_010 = −0b1.010×2^-6 = -0.01953125
0x93 = 1_0010_011 = −0b1.011×2^-6 = -0.021484375
0x94 = 1_0010_100 = −0b1.100×2^-6 = -0.0234375
0x95 = 1_0010_101 = −0b1.101×2^-6 = -0.025390625
0x96 = 1_0010_110 = −0b1.110×2^-6 = -0.02734375
0x97 = 1_0010_111 = −0b1.111×2^-6 = -0.029296875
0x98 = 1_0011_000 = −0b1.000×2^-5 = -0.03125
0x99 = 1_0011_001 = −0b1.001×2^-5 = -0.03515625
0x9a = 1_0011_010 = −0b1.010×2^-5 = -0.0390625
0x9b = 1_0011_011 = −0b1.011×2^-5 = -0.04296875
0x9c = 1_0011_100 = −0b1.100×2^-5 = -0.046875
0x9d = 1_0011_101 = −0b1.101×2^-5 = -0.05078125
0x9e = 1_0011_110 = −0b1.110×2^-5 = -0.0546875
0x9f = 1_0011_111 = −0b1.111×2^-5 = -0.05859375
0xa0 = 1_0100_000 = −0b1.000×2^-4 = -0.0625
0xa1 = 1_0100_001 = −0b1.001×2^-4 = -0.0703125
0xa2 = 1_0100_010 = −0b1.010×2^-4 = -0.078125
0xa3 = 1_0100_011 = −0b1.011×2^-4 = -0.0859375
0xa4 = 1_0100_100 = −0b1.100×2^-4 = -0.09375
0xa5 = 1_0100_101 = −0b1.101×2^-4 = -0.1015625
0xa6 = 1_0100_110 = −0b1.110×2^-4 = -0.109375
0xa7 = 1_0100_111 = −0b1.111×2^-4 = -0.1171875
0xa8 = 1_0101_000 = −0b1.000×2^-3 = -0.125
0xa9 = 1_0101_001 = −0b1.001×2^-3 = -0.140625
0xaa = 1_0101_010 = −0b1.010×2^-3 = -0.15625
0xab = 1_0101_011 = −0b1.011×2^-3 = -0.171875
0xac = 1_0101_100 = −0b1.100×2^-3 = -0.1875
0xad = 1_0101_101 = −0b1.101×2^-3 = -0.203125
0xae = 1_0101_110 = −0b1.110×2^-3 = -0.21875
0xaf = 1_0101_111 = −0b1.111×2^-3 = -0.234375
0xb0 = 1_0110_000 = −0b1.000×2^-2 = -0.25
0xb1 = 1_0110_001 = −0b1.001×2^-2 = -0.28125
0xb2 = 1_0110_010 = −0b1.010×2^-2 = -0.3125
0xb3 = 1_0110_011 = −0b1.011×2^-2 = -0.34375
0xb4 = 1_0110_100 = −0b1.100×2^-2 = -0.375
0xb5 = 1_0110_101 = −0b1.101×2^-2 = -0.40625
0xb6 = 1_0110_110 = −0b1.110×2^-2 = -0.4375
0xb7 = 1_0110_111 = −0b1.111×2^-2 = -0.46875
0xb8 = 1_0111_000 = −0b1.000×2^-1 = -0.5
0xb9 = 1_0111_001 = −0b1.001×2^-1 = -0.5625
0xba = 1_0111_010 = −0b1.010×2^-1 = -0.625
0xbb = 1_0111_011 = −0b1.011×2^-1 = -0.6875
0xbc = 1_0111_100 = −0b1.100×2^-1 = -0.75
0xbd = 1_0111_101 = −0b1.101×2^-1 = -0.8125
0xbe = 1_0111_110 = −0b1.110×2^-1 = -0.875
0xbf = 1_0111_111 = −0b1.111×2^-1 = -0.9375
0xc0 = 1_1000_000 = −0b1.000×2^0 = -1.0
0xc1 = 1_1000_001 = −0b1.001×2^0 = -1.125
0xc2 = 1_1000_010 = −0b1.010×2^0 = -1.25
0xc3 = 1_1000_011 = −0b1.011×2^0 = -1.375
0xc4 = 1_1000_100 = −0b1.100×2^0 = -1.5
0xc5 = 1_1000_101 = −0b1.101×2^0 = -1.625
0xc6 = 1_1000_110 = −0b1.110×2^0 = -1.75
0xc7 = 1_1000_111 = −0b1.111×2^0 = -1.875
0xc8 = 1_1001_000 = −0b1.000×2^1 = -2.0
0xc9 = 1_1001_001 = −0b1.001×2^1 = -2.25
0xca = 1_1001_010 = −0b1.010×2^1 = -2.5
0xcb = 1_1001_011 = −0b1.011×2^1 = -2.75
0xcc = 1_1001_100 = −0b1.100×2^1 = -3.0
0xcd = 1_1001_101 = −0b1.101×2^1 = -3.25
0xce = 1_1001_110 = −0b1.110×2^1 = -3.5
0xcf = 1_1001_111 = −0b1.111×2^1 = -3.75
0xd0 = 1_1010_000 = −0b1.000×2^2 = -4.0
0xd1 = 1_1010_001 = −0b1.001×2^2 = -4.5
0xd2 = 1_1010_010 = −0b1.010×2^2 = -5.0
0xd3 = 1_1010_011 = −0b1.011×2^2 = -5.5
0xd4 = 1_1010_100 = −0b1.100×2^2 = -6.0
0xd5 = 1_1010_101 = −0b1.101×2^2 = -6.5
0xd6 = 1_1010_110 = −0b1.110×2^2 = -7.0
0xd7 = 1_1010_111 = −0b1.111×2^2 = -7.5
0xd8 = 1_1011_000 = −0b1.000×2^3 = -8.0
0xd9 = 1_1011_001 = −0b1.001×2^3 = -9.0
0xda = 1_1011_010 = −0b1.010×2^3 = -10.0
0xdb = 1_1011_011 = −0b1.011×2^3 = -11.0
0xdc = 1_1011_100 = −0b1.100×2^3 = -12.0
0xdd = 1_1011_101 = −0b1.101×2^3 = -13.0
0xde = 1_1011_110 = −0b1.110×2^3 = -14.0
0xdf = 1_1011_111 = −0b1.111×2^3 = -15.0
0xe0 = 1_1100_000 = −0b1.000×2^4 = -16.0
0xe1 = 1_1100_001 = −0b1.001×2^4 = -18.0
0xe2 = 1_1100_010 = −0b1.010×2^4 = -20.0
0xe3 = 1_1100_011 = −0b1.011×2^4 = -22.0
0xe4 = 1_1100_100 = −0b1.100×2^4 = -24.0
0xe5 = 1_1100_101 = −0b1.101×2^4 = -26.0
0xe6 = 1_1100_110 = −0b1.110×2^4 = -28.0
0xe7 = 1_1100_111 = −0b1.111×2^4 = -30.0
0xe8 = 1_1101_000 = −0b1.000×2^5 = -32.0
0xe9 = 1_1101_001 = −0b1.001×2^5 = -36.0
0xea = 1_1101_010 = −0b1.010×2^5 = -40.0
0xeb = 1_1101_011 = −0b1.011×2^5 = -44.0
0xec = 1_1101_100 = −0b1.100×2^5 = -48.0
0xed = 1_1101_101 = −0b1.101×2^5 = -52.0
0xee = 1_1101_110 = −0b1.110×2^5 = -56.0
0xef = 1_1101_111 = −0b1.111×2^5 = -60.0
0xf0 = 1_1110_000 = −0b1.000×2^6 = -64.0
0xf1 = 1_1110_001 = −0b1.001×2^6 = -72.0
0xf2 = 1_1110_010 = −0b1.010×2^6 = -80.0
0xf3 = 1_1110_011 = −0b1.011×2^6 = -88.0
0xf4 = 1_1110_100 = −0b1.100×2^6 = -96.0
0xf5 = 1_1110_101 = −0b1.101×2^6 = -104.0
0xf6 = 1_1110_110 = −0b1.110×2^6 = -112.0
0xf7 = 1_1110_111 = −0b1.111×2^6 = -120.0
0xf8 = 1_1111_000 = −0b1.000×2^7 = -128.0
0xf9 = 1_1111_001 = −0b1.001×2^7 = -144.0
0xfa = 1_1111_010 = −0b1.010×2^7 = -160.0
0xfb = 1_1111_011 = −0b1.011×2^7 = -176.0
0xfc = 1_1111_100 = −0b1.100×2^7 = -192.0
0xfd = 1_1111_101 = −0b1.101×2^7 = -208.0
0xfe = 1_1111_110 = −0b1.110×2^7 = -224.0
0xff = 1_1111_111 = −Inf

References

[1] 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.

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

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

[4] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "Adaptive loss scaling for mixed precision training," tech. rep., arXiv cs.LG, 2019.
https://arxiv.org/abs/1910.12385.

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

[6] 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.

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

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

[9] 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.