The Removal/Demotion of MinNum and MaxNum Operations from IEEE 754™-2018

David H.C. Chen
February 21, 2017

Acknowledgements

In writing this report, I received help from many colleagues. A thank you to all my colleagues from Floating-Point Working Group for providing crucial insights and expertise. I would also like to express special appreciation to Bob Alverson, Steve Canon, Mike Cowlishaw, David Gay, Michel Hack, John Hauser, David Hough, William Huffman, Grant Martin, Terje Mathisen, Jason Riedy, and Lee Winter, whose valuable inputs, suggestions and comments have significantly improved this report. Any errors or omissions are mine and should not reflect on my colleagues.

1 Introduction

IEEE Standard for Floating-Point Arithmetic 754™ 2008 version (*IEEE std 754™-2008*) [6] specifies four minimum and maximum (min-max) operations: **minNum**, **maxNum**, **minNumMag**, and **maxNumMag**, in sub-clause "*5.3.1 General operations*". These four min-max operations are removed from or demoted in *IEEE std 754™-2018* [7], due to their non-associativity. No existing implementations become non-conformant due to this removal/demotion.

This report explains the technical reasons for the removal/demotion of the above four min-max operations, and reviews existing min-max implementations in some programming language standards and commercial hardware. Many implementations support several floating-point formats, but this report will only present one or two selected formats for an implementation when additional formats are supported in the same way.

To be brief, this report focuses on treatments of Not a Number (NaN) by max operations. A given implementation may also provide max magnitude, min, and min magnitude operations treating signaling NaN (sNaN) and quiet NaN (qNaN) in the same way. Please notice that the treatment of NaN by min-max operations does not represent the way NaN is treated by other operations in the same implementation.

This report also suggests a possible future course for the development of *754™-2018* or *754™-2028* min-max operations, by outlining criteria for other min-max variants. This report does not document the intent or application of min-max operations, and does not argue for a particular min-max operation.

2 Non-Associativity of *754™-2008* Min-Max Definitions

754™-2008 [6] is the first *754™* standard to define any min-max operations. In sub-clause "*5.3.1 General operations*", *754™-2008* [6] defines the following four min-max operations:

> **minNum**(*x*, *y*) is the canonicalized number *x* if *x* < *y*, *y* if *y* < *x*, the canonicalized number if one operand is a number and the other a quiet NaN. Otherwise it is either *x* or *y,* canonicalized (this means results might differ among implementations). When either *x* or *y* is a signaling NaN, then the result is according to 6.2.
> **maxNum**(*x*, *y*) is the canonicalized number *y* if *x* < *y*, *x* if *y* < *x*, the canonicalized number if one operand is a number and the other a quiet NaN. Otherwise it is either *x* or *y*, canonicalized (this means results might differ among implementations). When either *x* or *y* is a signaling NaN, then the result is according to 6.2.
> **minNumMag**(*x*, *y*) is the canonicalized number *x* if | *x*| < | *y*|, *y* if | *y*| < | *x*|, otherwise minNum(*x*, *y*).
> **maxNumMag**(*x*, *y*) is the canonicalized number *x* if | *x*| > | *y*|, *y* if | *y*| > | *x*|, otherwise maxNum(*x*, *y*).

In sub-clause "*7.2 Invalid operation*", *754™-2008* [6] also requires the following:

```
For operations producing results in floating-point format, the default result
of an operation that signals the invalid operation exception shall be a quiet
NaN that should provide some diagnostic information (see 6.2).
```

Combining the above requirements from sub-clauses *5.3.1* and *7.2* results in the following non-associative behavior affecting all four min-max operations:

***minNum**(1, **minNum**(1, sNaN)) -> **minNum**(1, qNaN) -> 1* (1)
***minNum**(**minNum**(1, 1), sNaN) -> **minNum**(1, sNaN) -> qNaN* (2)

With this non-associativity, different compilations or runs on parallel processing can return different answers, depending on whether the sNaN is encountered last or not in a sequence of operations.

3 Language Standards and Commercial Hardware Implementations

754™-2008 [6] and *754™-2018* [7] are really meta-standards for programming language standards. The Floating-Point Working Group reviewed the following implementations of language standards and commercial hardware before the removal/demotion of the min-max operations. An underlying issue identified was that sometimes a NaN is the most interesting part of a vector; sometimes the least.

Different language and hardware implementations treat NaN differently. Some of these implementations treat sNaN and qNaN in the same way, while others have different treatments depending on whether the NaN is signaling or quiet. Some implement two sets of min-max operations with different behaviors.

An implement may offer min-max in scalar only, scalar and vector, or scalar, vector and vector reduction forms. The following table lists implementations with their scalar min-max behaviors for quick comparison.

TABLE I. SCALAR MIN-MAX BEHAVIOR COMPARISON

Implementations	sNaN treatment	qNaN treatment	Comments
C11 **fmin** / **fmax**	(1, sNaN) -> undefined	(1, qNaN) -> 1	qNaN as "missing data"
TS18661-1 **fmin** / **fmax**	(1, sNaN) -> qNaN	(1, qNaN) -> 1	qNaN as "missing data" sNaN "propagate"
Java™ Platform SE 8 Strictmath **min** / **max**	(1, sNaN) -> qNaN	(1, qNaN) -> qNaN	NaN "propagate"
Java™ Platform SE 8 Math **min** / **max**	(1, sNaN) -> qNaN	(1, qNaN) -> qNaN	NaN "propagate"
ARM® ARMv8.2 **FMINNM** / **FMAXNM**	(1, sNaN) -> qNaN and signal Invalid	(1, qNaN) -> 1	qNaN as "missing data" sNaN "propagate"
Cadence® Tensilica® Fusion G3 **XT_MINNUM_S** / **XT_MAXNUM_S**	(1, sNaN) -> 1 and signal Invalid	(1, qNaN) -> 1	NaN as "missing data"
Decimal Arithmetic Specification version 1.70 **min** / **max**	(1, sNaN) -> qNaN and signal Invalid	(1, qNaN) -> 1	qNaN as "missing data" sNaN "propagate"
IBM® Power ISA™ Version 3.0 **vminfp** / **vmaxfp** for binary32	(1, sNaN) -> qNaN and signal Invalid	(1, qNaN) -> qNaN	NaN "propagate"
IBM® Power ISA™ Version 3.0 **xsmindp** / **xsmaxdp** for binary64	(1, sNaN) -> qNaN and signal Invalid	(1, qNaN) -> 1	qNaN as "missing data" sNaN "propagate"
Intel® AVX-512 **VRAGESS**	(1, sNaN) -> qNaN and signal Invalid	(1, qNaN) -> 1	qNaN as "missing data" sNaN "propagate"

These implementations behave differently when an operand is a number and the other operand is either sNaN or qNaN. For details and additional information, please look into each section of this chapter.

3.1 C11: qNaN Are Missing Data

ISO/IEC 9899:2011 International Standard for Programming Languages – C (*C11*) [9] defines min-max functions. *C11* sub-clauses *7.12.12.2* and *7.12.12.3* describe **fmax** and **fmix** functions, respectively:

> The **fmax** functions determine the maximum numeric value of their arguments.242)
> The **fmin** functions determine the minimum numeric value of their arguments.243)
>
> 242) NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, then the **fmax** functions choose the numeric value. See F.10.9.2.
> 243) The **fmin** functions are analogous to the **fmax** functions in their treatment of NaNs.

C11 [9] Annex *F.2.1* clarifies the *NaN* denotation of *C11*:

> This specification does not define the behavior of signaling NaNs.359) It generally uses the term NaN to denote quiet NaNs.
>
> 359) Since NaNs created by IEC 60559 operations are always quiet, quiet NaNs (along with infinities) are sufficient for closure of the arithmetic.

The above descriptions specify the following behavior:

***fmax** (1, qNaN) -> 1* (3)
***fmax** (1, sNaN) -> undefined* (4)

3.2 TS18661-1: sNaN Propagate but qNaN Are Missing Data

ISO/IEC TS18661-1 Technical Specification – Information technology – Programming languages, their environments, and system software interfaces – Floating-point extensions for C – Part 1: Binary floating-point arithmetic (*TS18661-1*) [10] binds C operation **fmin** to *ISO/IEC/IEEE 60559:2011* (*IEEE 754™-2008*) [6] operation **minNum**, and **fmax** to **maxNum**. *TS18661-1* [10] changes the *footnotes 242)* and *243)* in the C standard to exclude sNaN:

> Change footnotes 242) and 243) from:
>
> 242) NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, then the **fmax** functions choose the numeric value. See F.10.9.2.
>
> 243) The **fmin** functions are analogous to the **fmax** functions in their treatment of NaNs.
>
> to:
>
> 242) Quiet NaN arguments are treated as missing data: if one argument is a quiet NaN and the other numeric, then the **fmax** functions choose the numeric value. See F.10.9.2.
>
> 243) The **fmin** functions are analogous to the **fmax** functions in their treatment of quiet NaNs.

As a result, fmax will have the following new behavior:

***fmax** (1, qNaN) -> 1* (5)
***fmax** (1, sNaN) -> qNaN* (6)

3.3 Java™ Platform SE 8: All NaN Propagate

According to *Java™ Platform, Standard Edition 8, API Specification*, both *java.lang.Strictmath* [12] and *java.lang.Math* [11] provide min-max methods. *java.lang.Strictmath* and *java.lang.Math* contains the identical method details:

> public static float max(float a, float b)

> Returns the greater of two float values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is NaN, then the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

This thus specifies the following behavior:

max *(1, qNaN) -> qNaN* (7)
max *(1, sNaN) -> qNaN* (8)

3.4 ARM® ARMv8.2: sNaN Propagate but qNaN Are Missing Data

ARM® ARMv8.2 [1] provides scalar, vector, and reduction implementations of min-max instructions. The scalar instructions include **FMINNM** and **FMAXNM**. The vector instructions are **FMINNMP** and **FMAXNMP**. The reduction instructions are **FMINNMV** and **FMAXNMV**. sNaN signals and returns the quiet version of the NaN.

FMAXNM *(1, qNaN) -> 1* (9)
FMAXNM *(1, sNaN) -> qNaN and signal Invalid* (10)

ARMv8 Instruction Set Overview [1] specifies the reduction instructions to reduce pairwise.

> **FMAXNMV** Sd, Vn.4S Floating-point maxNum element to scalar (vector), equivalent to a sequence of pairwise reductions.

The non-associativity of *754* ™*-2008* [6] max-min operations may cause the reduction instructions to generate an improper result when an operand is sNaN:

FMAXNMV *of [1, 2, 3, sNaN] -> 2 and signal Invalid* (11)

3.5 Cadence® Tensilica® Fusion G3: All NaN Are Missing Data

Cadence® Tensilica® Fusion G3 [2] provides scalar, vector, and reduction min-max instructions. The scalar instructions comprise **XT_MAXNUM_S** and **XT_MINNUM_S**. The vector instructions comprise **PDX_MINNUM_MXF32** and **PDX_MAXNUM_MXF32**. The reduction instructions comprise **PDX_RMINNUM_MXF32** and **PDX_RMAXNUM_MXF32**. "*Fusion G3 User's Guide*" details their behavior:

> MINNUM implements "fmin" in C; while MAXNUM implements "fmax" in C. Both MINNUM and MAXNUM return a number whenever at least one operand is a number. Both MINNUM and MAXNUM signal Invalid exception when either operand is a sNaN.

Therefore:

***XT_MAXNUM_S** (1, qNaN) -> 1* (12)
***XT_MAXNUM_S** (1, sNaN) -> 1 and signal Invalid* (13)

Cadence® Tensilica® implement min-max to be associative, such that the reduction is well-defined:

***PDX_RMAXNUM_MXF32** [1, 2, 3, sNaN] -> 3 and signal Invalid* (14)
***PDX_RMINNUM_MXF32** [3, 2, 1, sNaN] -> 1 and signal Invalid* (15)

3.6 Decimal Arithmetic Specification: sNaN Propagate but qNaN Are Missing Data

Decimal Arithmetic Specification version 1.70 [4] specifies **max**, **max-magnitude**, **min**, and **min-magnitude** operations. [4] provides the details on NaN cases.

> If either operand is a NaN then the general rules apply, unless one is a quiet NaN and the other is numeric, in which case the numeric operand is returned.

The following behavior is observed:

***max** (1, qNaN) -> 1* (16)
***max** (1, sNaN) -> qNaN and signal Invalid* (17)

3.7 IBM® Power ISA™ 3.0: Two Different Implementations Coexist

IBM® Power ISA™ Version 3.0 [5] provides two implementations with different behaviors.

The first behavior is "All NaN Propagate", implemented with a pair of vector min-max instructions in binary32. *Power ISA™ 3.0* [5], section "*6.10.2 Vector Floating-Point Maximum and Minimum Instructions*" specifies the **vmaxfp** and **vminfp** instructions.

> The maximum of +0 and -0 is +0. The maximum of any value and a NaN is a QNaN.
> The minimum of +0 and -0 is -0. The minimum of any value and a NaN is a QNaN.

Therefore:

***vmaxfp** (1, qNaN) -> qNaN* (18)
***vmaxfp** (1, sNaN) -> qNaN and signal Invalid* (19)

The second behavior is "sNaN Propagate but qNaN Are Missing Data", implemented with another pair of vector-scalar min-max instructions in binary64. *Power ISA™ 3.0* [5], section "*7.6.3 VSX Instruction Descriptions*" specify the **xsmaxdp** and **xsmindp** instructions.

> The maximum of +0 and –0 is +0. The maximum of a QNaN and any value is that value. The maximum of any value and an SNaN is that SNaN converted to a QNaN.

The minimum of +0 and –0 is –0. The minimum of a QNaN and any value is that value. The minimum of any value and an SNaN is that SNaN converted to a QNaN.

Therefore:

xsmaxdp *(1, qNaN) -> 1* (20)
xsmaxdp *(1, sNaN) -> qNaN and signal Invalid* (21)

3.8 Intel® AVX-512: sNaN Propagate but qNaN Are Missing Data

Intel® Advanced Vector Extensions 512 (*AVX-512*) [8] provides scalar, and vector implementations of min-max instructions. The scalar instructions comprise **VRAGESS**. The vector instructions comprise **VRAGEPS**. An immediate field turns these instructions into **minNum**, **maxNum**, **minNumMag**, and **maxNumMag**. *AVX-512* [8] does not implement a reduction version. [3] provides the following table:

TABLE II. AVX-512 VRAGEPD RESULTS OF ONE OR MORE NAN OPERANDS

Src1	Src2	Result	IE Signaling Due to Comparison	Imm8[3:2] Effect to Range Output
sNaN1	sNaN2	Quiet(sNaN1)	Yes	Ignored
sNaN1	qNaN2	Quiet(sNaN1)	Yes	Ignored
sNaN1	Norm2	Quiet(sNaN1)	Yes	Ignored
qNaN1	sNaN2	Quiet(sNaN2)	Yes	Ignored
qNaN1	qNaN2	qNaN1	No	Applicable
qNaN1	Norm2	Norm2	No	Applicable
Norm1	sNaN2	Quiet(sNaN2)	Yes	Ignored
Norm1	qNaN2	Norm1	No	Applicable

In summary:

VRAGESS *(1, qNaN) -> 1* (22)
VRAGESS *(1, sNaN) -> qNaN and signal Invalid* (23)

4 Criteria for Future Min-Max Operations

754™-2018 or *754™-2028* may add one or more sets of new min-max variants with different names. Criteria for future min-max operations may include primitives, associativity, commutativity, treatment of signed zero, priority of NaN, and special NaN. This analysis is intended to assist in future discussions, but does not suggest particular alternatives.

4.1 Primitives

754™-2018 or *754™-2028* may conclude min-max as high-level routines and exclude min-max from the floating-point arithmetic standard. Such an exclusion will allow language standards define, or not define, their own min-max operations. Language standards might construct min-max routines by choosing *754™* comparison predicates/relations and a selection based on the predicate/relation results.

Otherwise, *754*™*-2018* or *754*™*-2028* may conclude min-max as primitives and include min-max as operations in the standard. This inclusion will require or recommend language standards to have same standardized min-max behaviors.

4.2 Associativity

754™ **addition** and **multiplication** operations [6] are associative when results are exact. When results are inexact, rounding errors may cause **addition** and **multiplication** to be non-associative. Min-max operations require no rounding and always produce exact results. It is thus intuitive to believe min-max is associative. Associativity is especially important in parallel computing. *754*™*-2018* or *754*™*-2028* may consider new min-max operations with the associative property. Otherwise, *754*™*-2018* or *754*™*-2028* may consider expressively informing readers of non-associativity with a NOTE if defining new min-max operations with non-associativity.

4.3 Commutativity

754™ **addition** and **multiplication** operations [6] are commutative unless both operands are NaNs. When both operands are NaNs, *754*™ allows implementations to propagate either NaN operand. Some implementations may propagate NaN depending on the order of operands, as being non-commutative when both operands are NaNs. When at most one operand is NaN, the result may not be exact, due to rounding error or quieting a sNaN; but reversing the operands gives the same result.

It is also intuitive to believe min-max are commutative unless two NaNs are involved. *754*™*-2018* or *754*™*-2028* might consider new min-max operations with the commutative property. Otherwise, *754*™*-2018* or *754*™*-2028* might consider expressively informing readers of non-commutativity with a NOTE if defining new min-max operations with non-commutativity.

4.4 Signed Zero

For comparisons, *754*™ defines +0 to be equal to -0 [6]:

```
Comparisons shall ignore the sign of zero (so +0 = −0).
```

However, min-max operations have to return a number, instead of a relation or a predicate. Min/max(+0,-0) could return either +0 or -0. It is preferable to be commutative and to return the same result, regardless of the order of the operands. Some existing min-max implementations consider -0 to be strictly smaller than +0. In either software or hardware, this could be easily implemented by comparing operands as sign-magnitude integers. *754*™*-2018* or *754*™*-2028* might consider new min-max operations which require or recommend that -0 to be considered strictly smaller than +0.

4.5 NaN Priority

When both operands are NaNs, some existing implementations select sNaN over qNaN. Some other implementations guarantee NaN associativity and commutativity, by having **minNum**(NaN1,NaN2) select a NaN based on **totalOrder**(NaN1, NaN2)?NaN1:NaN2 and having **maxNum**(NaN1,NaN2) select a NaN based on of

totalOrder(NaN2,NaN1)?NaN1:NaN2. *754*™*-2018* or *754*™*-2028* might consider new min-max operations which require or recommend associativity/commutativity when both operands are NaNs. Please refer to [6] for **totalOrder** definition.

4.6 Special NaN

754™*-2018* or *754*™*-2028* might define a special NaN to represent negligible operands having no effects on computations. i.e. an intentional absence of an object, an empty element of a vector, an @NA in spreadsheet vocabulary, or a database NULL. The NaN propagation rules might exempt the special NaN. The availability of such special NaN may affect whether and how *754*™*-2018* or *754*™*-2028* might define new min-max operations.

References

[1] ARM Limited, "ARMv8 Instruction Set Overview," PRD03-GENC-010197 15.0 2011-11-11.
[2] Cadence Design Systems, Inc., "Fusion G3 User's Guide," PD-16-3331-10-00 2016-09.
[3] GitHub, Inc., "x86 Assembly Documentation, AVX-512, updated with Intel®'s documentation of June 2016," https://hjlebbink.github.io/x86doc/html/VRANGEPD.html retrieved 2017-01-17.
[4] IBM Corporation, "Decimal Arithmetic Specification version 1.70, 2009," http://speleotrove.com/decimal/daops.html#refmax retrieved 2017-01-17.
[5] IBM Corporation, "Power ISA™ Version 3.0," 2015-11-30.
[6] Institue of Electrical and Electronics Engineers, Inc., "IEEE standard for floating-point arithmetic, IEEE Std 754™-2008," 2008.
[7] Institue of Electrical and Electronics Engineers, Inc., "IEEE standard for floating-point arithmetic, IEEE Std 754™-2018," 2018.
[8] Intel Corporation, "Intel® 64 and IA-32 Architectures Software Developer's Manual," 325383-060US 2016-09.
[9] ISO/IEC, "ISO/IEC 9899:2011 International Standard for Programming Languages – C," ISO/IEC 9899:2011, 2011.
[10] ISO/IEC, "Technical Specification – Information technology – Programming languages, their environments, and system software interfaces – Floating-point extensions for C – Part 1: Binary floating-point arithmetic," ISO/IEC TS18661-1, 2014-07-15.
[11] Oracle Corporation, "Java™ Platform, Standard Edition 8, API Specification, Class Math," https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html retrieved 2017-01-17.
[12] Oracle Corporation, "Java™ Platform, Standard Edition 8, API Specification, Class StrictMath," https://docs.oracle.com/javase/8/docs/api/java/lang/StrictMath.html retrieved 2017-01-17.