Standard for Posit™ Arithmetic (2022)[*]

Posit Working Group

March 2, 2022

[*]The initial development of posit arithmetic was supported by Singapore's Agency for Science, Technology and Research (A*STAR) and by the USA's DARPA TRADES Program, Contract #HR0011-17-9-0007.

Standard for Posit™ Arithmetic

Sponsor

National Supercomputing Centre (NSCC) Singapore

Abstract

This standard specifies the storage format, operation behavior, and required mathematical functions for posit arithmetic. It describes the binary storage used by the computer and the human-readable character input and output for posit representation. A system that meets this standard is said to be *posit compliant* and will produce results that are identical to those produced by any other posit compliant system. A posit compliant system may be realized using software or hardware or any combination.

Key search phrases

posit arithmetic, reproducible computer arithmetic, efficient binary number format, Not a Real, “regime exponent fraction”, binary rounding rules, quire arithmetic, fused expressions

Participants

The following people in the Posit Working Group contributed to the development of this standard:

John Gustafson, *Chair*
Gerd Bohlender
Shin Yee Chung
Vassil Dimitrov
Geoff Jones
Siew Hoon Leong (Cerlane)
Peter Lindstrom
Theodore Omtzigt
Hauke Rehr
Andrew Shewmaker
Isaac Yonemoto

Contents

1 Overview

1.1 Scope

This standard specifies the storage formats and mathematical behavior of posit™ numbers, including basic arithmetic operations and the set of functions a posit system must support. It describes how results are to be rounded to a real posit or determined to be a non-real exception.

1.2 Purpose

This standard provides a system for computations with real numbers represented in a computer using fixed-size binary values. Deviations from mathematical behavior (including loss of accuracy) are kept to a minimum while preserving the ability to represent a wide dynamic range. All features are accessible by programming languages; the source program and input data suffice to specify the output exactly on any computer system.

1.3 Inclusions and exclusions

This standard specifies:

- Binary formats for posits, for computation and data interchange
- Addition, subtraction, multiplication, division, dot product, comparison, and other operations
- Fused expressions that are computed exactly, then rounded to posit format
- Mathematical elementary functions such as logarithm, exponential, and trigonometric functions
- Conversions of other number representations to and from posit formats
- Conversions between posit formats with different precisions
- Function behavior when an input or output value is not a real number (NaR)

Excluded from the standard are the specific names of the values and operations described here. The lower camelCase naming style is used here, but naming style is excluded from this standard. Implementations may use alternative names and symbols for values and operations that match the behavior described here.[1]

Also excluded are rules for how an implementation should handle and report errors. If a program attempts a computation on posit values outside the domain that produces a real-valued output, or compares the NaR value with a real number, behavior beyond the arithmetic result specified here (such as issuance of warnings) is up to the implementation designers.

This is a numerical format standard, not a language standard. This standard *enables* a language to provide deterministic rounding as a posit compliant mode.

1.4 Requirements vs. recommendations, and posit-compliance

All descriptions herein are requirements of system behavior, not recommendations. The decision of how to satisfy the requirements and which precisions to support is up to the implementer of this standard, but all functionality must be provided and behave as described for a system to be posit compliant. An implementation is compliant with this standard if it supports full functionality of at least one precision. If the implementation supports more than one precision, then it must support conversions between them and every precision supported must be posit compliant.

[1]For example, the arc hyperbolic cosine is here shown as **arcCosH**, but it may be called `acosh` in the math library for `C` so long as it meets this standard's requirement of correct rounding for all inputs. Similarly, a language may express a sum of two posits a and b as $a + b$, though that function is here called **addition**(a, b). Rounding behavior must follow the rules in this document for any implementation to be considered posit compliant.

2 Definitions, abbreviations, and acronyms

bit field A contiguous set of bits in a format with a defined meaning. A bit field may extend beyond explicit bits $b_{n-1} \cdots b_0$; bits beyond the format's explicit bits are considered 0 bits.

exception A special case in the interpretation of representations in posit format: 0 or NaR.[2]

exponent The power-of-two scaling determined by the exponent bits, in the set $\{0, 1, 2, 3\}$.

exponent bits A two-bit unsigned integer bit field that determines the exponent.

format A set of bit fields and the definition of their meaning.

fraction The value represented by the fraction bits; $0 \leq$ fraction < 1.

fraction bits The bit field following the exponent bits.

fused rounded only after an exact evaluation of an expression involving more than one operation.

implicit value A value added to the fraction based on the sign: -2 for negative posits, 1 for positive posits. Zero and NaR do not have an implicit value.

LSB The least significant bit of a format or a bit field within a format.

maxPos The largest positive posit value. It is a function of n.

minPos The smallest positive posit value. It is a function of n.

MSB The most significant bit of a format or a bit field within a format.

NaR Not a real. Umbrella value for anything not mathematically definable as a unique real number.

n The number of bits in a posit format. It can be any integer greater than 1.

pIntMax The largest consecutive integer-valued posit value. It is a function of n.

posit value A real number representable using a posit format described in this standard, or NaR.

precision The total storage size for expressing any number format, in bits. For a posit, precision is n bits.

quire value A real number representable using a quire format described in this standard, or NaR.

quire sum limit The minimum number of additions of posit values that can overflow the quire format.

regime The power-of-16 scaling determined by the regime bits. It is a signed integer.

regime bits A posit bit field following the MSB that uses a form of signed unary encoding (as opposed to positional notation) to represent the regime. There is always at least one regime bit R_0. For $n > 2$, there are always at least two regime bits.

rounded Converted from a real number to a posit value, according to the rules of this standard.

sign The value 1 for positive numbers, -1 for negative numbers, and 0 for 0. The NaR value has no sign.

sign bit The MSB of a posit or quire format.

significand The implicit value plus the fraction; $-2 \leq$ significand < -1 for negative posit values, and $1 \leq$ significand < 2 for positive posit values.

[2]Posit representation exceptions do not imply a need for status flags or heavyweight operating system or language runtime handling.

3 Posit and quire formats

3.1 Formats

This section defines posit and quire formats and their representation as a finite set of real numbers or the exception value NaR. Formats are specified by their precision, n. There is a quire format of precision $16n$ that is used to contain exact sums of products of posits of precision n. Dynamic range and accuracy are determined solely by n. This standard describes example choices for n like 8, 16, and 32. The posit format's type label is "posit" with the decimal string for n appended. The corresponding quire format's type label is "quire" with the decimal string for n appended, even though quire format has $16n$ bits.

3.2 Represented data

A posit value is either the exception value NaR or a real number x of the form $K \times 2^M$, where K and M are integers limited to a range symmetric about and including zero. The smallest positive posit value, *minPos*, is 2^{-4n+8} and the largest positive posit value, *maxPos*, is $1/minPos$, or 2^{4n-8}. Every posit value is an integer multiple of *minPos*. Every real number maps to a unique posit representation; there are no redundant representations. The posit values are a superset of all integers i in a range $-pIntMax \le i \le pIntMax$. Outside that range, integers exist that cannot be expressed as a posit value without rounding to a different integer; *pIntMax* is $\lceil 2^{\lfloor 4(n-3)/5 \rfloor} \rceil$.

A quire value is either NaR or an integer multiple of the square of *minPos*, represented as a 2's complement binary number with $16n$ bits. Quire format can represent the exact dot product of two posit vectors having at most 2^{31} (approximately two billion) terms without the possibility of rounding or overflow.[3]

The properties of example and general posit format precisions are summarized in Table 1:

Property	**posit8**	**posit16**	**posit32**	**posit$n$**
fraction length	0 to 3 bits	0 to 11 bits	0 to 27 bits	0 to $\max(0, n-5)$ bits
minPos	$2^{-24} \approx 6.0 \times 10^{-8}$	$2^{-56} \approx 1.4 \times 10^{-17}$	$2^{-120} \approx 7.5 \times 10^{-37}$	2^{-4n+8}
maxPos	$2^{24} \approx 1.7 \times 10^{7}$	$2^{56} \approx 7.2 \times 10^{16}$	$2^{120} \approx 1.3 \times 10^{36}$	2^{4n-8}
pIntMax	$2^4 = 16$	$2^{10} = 1024$	$2^{23} = 8388608$	$\lceil 2^{\lfloor 4(n-3)/5 \rfloor} \rceil$
quire format precision	128 bits	256 bits	512 bits	$16n$ bits
quire sum limit	$2^{55} \approx 3.6 \times 10^{16}$	$2^{87} \approx 1.5 \times 10^{26}$	$2^{151} \approx 2.9 \times 10^{45}$	2^{23+4n}

Table 1: Properties of posit formats

3.3 Posit format encoding

Figure 1 defines the general format for posit encoding. The regime is a variable-length field. All of its bits but the last are identical. The longer the regime, the more bits of fields to its right are not represented. These truncated bits extending beyond the LSB are treated as `0` bits. Figure 2 shows how part of the exponent field and all of the fraction field can be truncated. Figure 3 shows the extreme case where the regime extends to the LSB. The four constituting bit fields in order of decreasing significant bits are:

1. Sign bit S. S represents an integer s, its literal value, 0 or 1. The *implicit value* is $(1-3s)$.
2. Regime bit field R consisting of k bits identical to R_0, terminated by $\overline{R_0} = 1 - R_0$ as shown in Figures 1 and 2, or just after the LSB as shown in Figure 3. R represents $r = -k$ if R_0 is `0`, or $r = k - 1$ if R_0 is `1`.
3. The exponent bit field E has length 2 bits, but one or both bits may be beyond the LSB and thus have value `0`. E represents an integer e, its bits treated as a 2-bit unsigned integer. $0 \le e \le 3$.
4. The fraction bit field F has length $\max(0, n-5)$ bits, but any number of those bits may be beyond the LSB of the posit representation and thus are taken to be `0` bits. The number of explicit bits is m. F represents the fraction f, an m-bit unsigned integer divided by 2^m. $0 \le f < 1$.

[3]The product of two posit values in a format of precision n is always exactly expressible in a posit format of precision $2n$, but the quire format obviates such temporary doubling of precision when computing sums and differences of products. Sums of posit values using the quire are guaranteed exact up to 2^{23+4n} terms, per the *quire sum limit*.

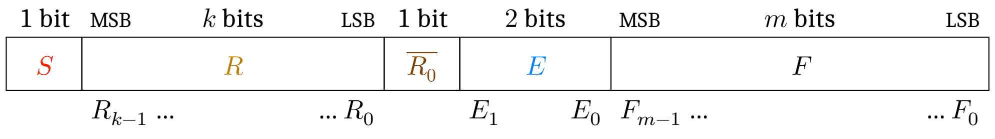

Figure 1: General binary posit representation with all fields explicit

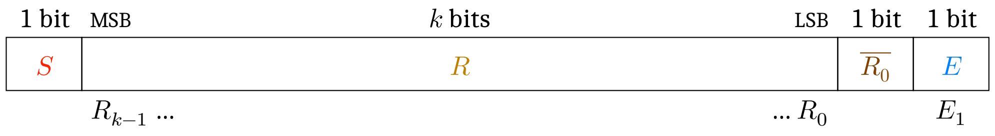

Figure 2: Example with all F fraction bits and the E_0 bit truncated

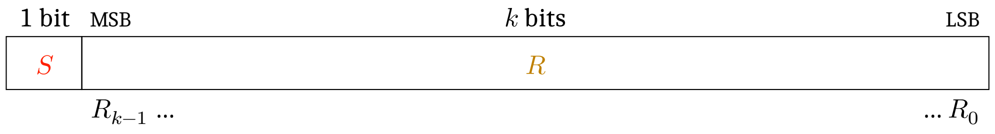

Figure 3: Example with the F fraction and E exponent bit fields, and the regime termination bit $\overline{R_0}$ all truncated

The posit value p is inferred from the bit fields S, R, E, and F as follows:

1. Check if p represents an exception: if all bits except S are 0 bits (cf. figure 3),
   - if $S = 0$, then $p = 0$.
   - if $S = 1$, then p is NaR.
2. Otherwise, let $f := 2^{-m} \sum_{\ell=0}^{m-1} f_\ell 2^\ell$,
   - if $R_0 = 0$, then $r = -k$.
   - if $R_0 = 1$, then $r = k - 1$.

   And $p = ((1 - 3s) + f) \times 2^{(1-2s)\times(4r+e+s)}$.

In Figure 3, if R_0 is 1, then it represents *maxPos* ($S = 0$) or $-$*minPos* ($S = 1$).

3.4 Quire format encoding

Quire format is a fixed-point 2's complement format of precision $16n$, with fields as follows:

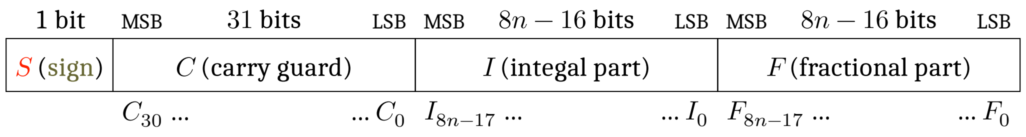

Figure 4: Binary quire format

The quire value q is inferred from the bit fields S, C, I, and F as follows:

- If S is 1 and all other fields contain only 0 bits, then q is NaR.
- Otherwise q is 2^{16-8n} times the 2's complement signed integer represented by all bits concatenated.

4 Rounding

4.1 Definition and method

Rounding is the substitution of a posit value for any real number. Operation results are regarded as exact prior to rounding. The method for rounding a real value x is described by the following algorithm:

```
Data: $x$, a real number
Result: Rounded $x$, a posit value
if $x$ is exactly expressible in the posit format in question then
    return $x$
if $|x| > maxPos$ then
    return $sign(x) \times maxPos$
if $|x| < minPos$ then
    return $sign(x) \times minPos$
Let $u$ and $w$ be $n$-bit posit values such that the open interval $(u, w)$ contains $x$ but no $n$-bit posit value.
Let $U$ be the $n$-bit representation of $u$.
Let $v$ be the $(n + 1)$-bit posit value associated with the $(n + 1)$-bit representation $U1$.
if $u < x < v$ or $(x = v$ and LSB of $U$ is 0) then
    return $u$
else
    return $w$
end
```

4.2 Fused expressions

A *fused expression* is an expression with two or more operations that is evaluated exactly before rounding to a posit value. Expressions that can be written in the form of a dot product of vectors of length less than 2^{31} can be evaluated exactly using quire representations and then rounded to posit format to create a fused expression, if so defined by the rules of the language. If a fused expression is computed in parallel, sufficient intermediate result information must be communicated that the result is identical to the single-processor result.[4] Fused expressions (such as fused multiply-add and fused multiply-subtract) need not be performed using quire representations to be posit compliant.

4.3 Program execution restrictions

For any language where the order of operations is well-defined, the execution order of operations in posit compliant mode cannot be changed from that expressed in the source code if it affects rounding. This includes any use of precisions or operation fusing not expressed in the source code. [5] If a language permits mixed data types in expressions, including quire and posit format types, or posit and other formats for representing real numbers, the language must specify how such expressions are evaluated in order to be posit compliant.

[4]Note that functions in Section 5 which are rounded and have two or more operations in their mathematical definition are fused expressions, such as **rSqrt**, **expMinus1**, **fMM**, and **hypot**.

[5]Languages that offer optimization modes that covertly change rounding do so at the cost of bitwise-reproducible results and are non-compliant when used in those modes.

5 Functions

5.1 Guiding principles for the NaR exceptional value

If an operation usually produces real-valued output, any NaR input produces NaR output, with the exception of **next** and **prior**. NaR is output when the function's value is not arbitrarily close to a unique real number for open neighborhoods of complex values sufficiently close to the input values,[6] except for discontinuous functions in Section 5.2. Functions with multiple branches such as roots and inverse trig functions apply this criterion to a single branch. A test of equality between NaR values returns True. The NaR value has no sign, so **sign**(NaR) returns NaR.

The following functions shall be supported, with rounding per Section 4.1. Functions that take more than one posit value for input must have the same precision for all posit value inputs, and any posit value in the output must have the same precision as the input posits. Conversion routines may be used to make mixed-precision posit value inputs the same precision, per Section 6.1. Conversions may be explicit in source code or implicit by language rules.

5.2 Basic functions of one posit value argument

negate(*posit*)	returns $-$*posit*.[7]
abs(*posit*)	returns **negate**(*posit*) if *posit* < 0, else *posit*.
sign(*posit*)	returns a posit value: 1 if *posit* > 0, -1 if *posit* < 0, or 0 if *posit* $= 0$.
nearestInt(*posit*)	returns the integer-valued posit value nearest to *posit*, and returns the nearest even integer-valued posit value if two integers are equally near.
ceil(*posit*)	returns the smallest integer-valued posit value greater than or equal to *posit*.
floor(*posit*)	returns the largest integer-valued posit value less than or equal to *posit*.
next(*posit*)	returns the posit value of the lexicographic successor of *posit*'s representation.[8]
prior(*posit*)	returns the posit value of the lexicographic predecessor of *posit*'s representation.[9]

5.3 Comparison functions of two posit value arguments

All comparison functions return Boolean values identical to comparisons of the posits' representations regarded as 2's complement integers, so there is no need for separate machine-level instructions. The representation of NaR coincides with the 2's complement bit string of the most negative integer, so if *posit* is real, **compareLess**(NaR, *posit*) returns True, etc.

compareEqual(*posit1*, *posit2*)	returns True if *posit1* $=$ *posit2*, else False.
compareNotEqual(*posit1*, *posit2*)	returns True if *posit1* $\neq$ *posit2*, else False.
compareGreater(*posit1*, *posit2*)	returns True if *posit1* $>$ *posit2*, else False.
compareGreaterEqual(*posit1*, *posit2*)	returns True if *posit1* $\geq$ *posit2*, else False.
compareLess(*posit1*, *posit2*)	returns True if *posit1* $<$ *posit2*, else False.
compareLessEqual(*posit1*, *posit2*)	returns True if *posit1* $\leq$ *posit2*, else False.

5.4 Arithmetic functions of two posit value arguments

addition(*posit1*, *posit2*)	returns *posit1* $+$ *posit2*, rounded.
subtraction(*posit1*, *posit2*)	returns *posit1* $-$ *posit2*, rounded.
multiplication(*posit1*, *posit2*)	returns *posit1* $\times$ *posit2*, rounded.
division(*posit1*, *posit2*)	returns *posit1* $\div$ *posit2*, rounded.

[6] The function may be complex-valued in the neighborhood of the inputs, but still have a real-valued limit. For example, **pow**$(-1, -3) = (-1)^{-3}$ is -1 even though the function is complex-valued in any neighborhood of the second argument. Similarly, **sqrt**$(0) = 0$.

[7] This is the 2's complement of the posit representation. 2's complement affects neither 0 nor NaR, since they are unsigned.

[8] wrapping around, if necessary

[9] wrapping around, if necessary

5.5 Elementary functions of one posit value argument

sqrt(*posit*)	returns $\sqrt{posit}$, rounded.	
rSqrt(*posit*)	returns $1/\sqrt{posit}$, rounded.	
exp(*posit*)	returns e^{posit}, rounded.	
expMinus1(*posit*)	returns $e^{posit} - 1$, rounded.	
exp2(*posit*)	returns 2^{posit}, rounded.	
exp2Minus1(*posit*)	returns $2^{posit} - 1$, rounded.	
exp10(*posit*)	returns 10^{posit}, rounded.	
exp10Minus1(*posit*)	returns $10^{posit} - 1$, rounded.	
log(*posit*)	returns $\log_e(posit)$, rounded.	
logPlus1(*posit*)	returns $\log_e(posit + 1)$, rounded.	
log2(*posit*)	returns $\log_2(posit)$, rounded.	
log2Plus1(*posit*)	returns $\log_2(posit + 1)$, rounded.	
log10(*posit*)	returns $\log_{10}(posit)$, rounded.	
log10Plus1(*posit*)	returns $\log_{10}(posit + 1)$, rounded.	
sin(*posit*)	returns $\sin(posit)$, rounded.	
sinPi(*posit*)	returns $\sin(\pi \times posit)$, rounded.	
cos(*posit*)	returns $\cos(posit)$, rounded.	
cosPi(*posit*)	returns $\cos(\pi \times posit)$, rounded.	
tan(*posit*)	returns $\tan(posit)$, rounded.	
tanPi(*posit*)	returns $\tan(\pi \times posit)$, rounded.	
arcSin(*posit*)	returns $\arcsin(posit)$, rounded.	**abs**(**arcSin**) $\leq$ ($\pi/2$, rounded).
arcSinPi(*posit*)	returns $\arcsin(posit)\ /\ \pi$, rounded.	**abs**(**arcSinPi**) $\leq 1/2$.
arcCos(*posit*)	returns $\arccos(posit)$, rounded.	$0 \leq$ **arcCos** $\leq$ (π, rounded).
arcCosPi(*posit*)	returns $\arccos(posit)\ /\ \pi$, rounded.	$0 \leq$ **arcCosPi** ≤ 1.
arcTan(*posit*)	returns $\arctan(posit)$, rounded.	**abs**(**arcTan**) $\leq$ ($\pi/2$, rounded).
arcTanPi(*posit*)	returns $\arctan(posit)\ /\ \pi$, rounded.	**abs**(**arcTanPi**) $\leq 1/2$.
sinH(*posit*)	returns $\sinh(posit)$, rounded.	
cosH(*posit*)	returns $\cosh(posit)$, rounded.	
tanH(*posit*)	returns $\tanh(posit)$, rounded.	
arcSinH(*posit*)	returns $\operatorname{arcsinh}(posit)$, rounded.	
arcCosH(*posit*)	returns $\operatorname{arccosh}(posit)$, rounded.	$0 \leq$ **arcCosH**.
arcTanH(*posit*)	returns $\operatorname{arctanh}(posit)$, rounded.	

5.6 Elementary functions of two posit value arguments

hypot(*posit1*, *posit2*)	returns $\sqrt{posit1^2 + posit2^2}$, rounded.
pow(*posit1*, *posit2*)	returns $posit1^{posit2}$, rounded.[10]
arcTan2(*posit1*, *posit2*)	returns the argument t of $posit1 + i posit2$, $-\pi < t \leq \pi$, rounded.[11]
arcTan2Pi(*posit1*, *posit2*)	returns **arcTan2**(*posit1*, *posit2*)$/\pi$, rounded.

5.7 Functions of three posit value arguments

fMM(*posit1*, *posit2*, *posit3*) returns $posit1 \times posit2 \times posit3$, rounded.[12]

[10] See Section 5.1 for situations that generate NaR. For example, x^y is not arbitrarily close to a single real number for any complex-valued neighborhoods of $x = y = 0$, so **pow**$(0, 0)$ returns NaR.

[11] The apparent discontinuity in **arcTan2**(x, y) for $x \leq 0, y = 0$ is spurious since it results from jumping between branches of a multi-valued function. It should return π, rounded, if $x < 0, y = 0$. **arcTan2**(0,0) must return NaR since the function is not arbitrarily close to a unique real value for any complex-valued open neighborhoods of the inputs.

[12] Because multiplication is commutative and associative, any permutation of the inputs will return the same rounded result.

5.8 Functions of one posit value argument and one integer argument

compound(*posit*, *integer*)	returns $(1 + \mathit{posit})^{\mathit{integer}}$, rounded.
rootN(*posit*, *integer*)	returns $\mathit{posit}^{1/\mathit{integer}}$, rounded. If *integer* is even, **rootN** ≥ 0.

5.9 Functions that do not round correctly for all arguments

Computing environments that support versions of functions in any of the above subsections that do not round correctly for all inputs must supply the source code for such functions, and use a notation for them that is distinct from the notation for the corresponding function that rounds correctly for all inputs.

5.10 Functions not yet required for compliance

Special functions such as error functions, Bessel functions, gamma and digamma functions, beta and zeta functions, etc. are not presently required for a system to be posit compliant. They may be required in a future revision of this standard.

5.11 Functions involving quire value arguments

With the exception of **qToP** which returns a posit value result, these functions return a quire value result.[13] If any operation on quire values overflows the carry bits of a quire's representation, the result is NaR in quire format. Where any posit values are involved, their precisions n must agree, and the quire must be of the corresponding precision $16n$.

pToQ(*posit*)	returns *posit* converted to quire format.
qNegate(*quire*)	returns $-\mathit{quire}$.
qAbs(*quire*)	returns **qNegate**(*quire*) if $\mathit{quire} < 0$, else *quire*.
qAddP(*quire*, *posit*)	returns $\mathit{quire} + \mathit{posit}$.
qSubP(*quire*, *posit*)	returns $\mathit{quire} - \mathit{posit}$.
qAddQ(*quire1*, *quire2*)	returns $\mathit{quire1} + \mathit{quire2}$.
qSubQ(*quire1*, *quire2*)	returns $\mathit{quire1} - \mathit{quire2}$.
qMulAdd(*quire*, *posit1*, *posit2*)	returns $\mathit{quire} + (\mathit{posit1} \times \mathit{posit2})$.
qMulSub(*quire*, *posit1*, *posit2*)	returns $\mathit{quire} - (\mathit{posit1} \times \mathit{posit2})$.
qToP(*quire*)	returns *quire* rounded to posit format per Section 4.1.

Other functions of quire values may be provided, but are not required for compliance. They may be required in a future revision of this standard.

[13] These functions can be used to compute the real and imaginary parts of complex number products, exact sums up to length 2^{23+4n}, exact dot products and scaled sums of vectors up to length $2^{31} - 1$, exact determinants of 2-by-2 matrices, exact discriminants of quadratic equations, exact residuals of solutions to systems of linear equations, and higher-precision arithmetic for addition, subtraction, multiplication, division, and square root of values expressed as unevaluated sums of posit value lists. A quire value's rounded posit value can repeatedly be subtracted from it, and those rounded values gathered for representing a quire value as an unevaluated sum of posit values. For example, if quire8 contained a π approximation $\frac{3217}{1024} = 3.1416015625$, repeated rounding to posit8 and subtracting would produce terms of an unevaluated sum $\{\frac{13}{4}, \frac{-7}{64}, \frac{1}{1024}\}$. An unevaluated sum can be multiplied by another evaluated sum as an exact dot product, using the quire.

6 Conversion operations for posit format

6.1 Conversions between different precisions

Converting a posit value to higher precision is exact, by appending 0 bits to its representation. Conversion to a lower precision is rounded, per Section 4.1.[14] In the function notation used here,

> **p**m**To**n(*posit*) returns the n-bit posit representation of an m-bit posit value *posit* by these conversion rules.

6.2 Conversions involving quire values

A posit compliant system only needs to support rounding from quire to posit values and conversion of posit to s in the matching precision, per Section 5.11.

6.3 Conversions between posit format and decimal character strings

Table 2 shows examples of the minimum number of significant decimals needed to express a posit value such that the real number represented by the decimal form will round to the same posit value.

precision	**posit8**	**posit16**	**posit32**	**posit64**
Decimals	2	5	10	21

Table 2: Examples of minimum decimals in a base-ten significand to preserve any posit value

6.4 Conversions between posit format and integer format

Supported posit formats must provide conversion to and from all integer formats supported in a computing environment. In converting a posit value to an integer value, if the posit value is out of integer range after rounding or is NaR, the integer value is returned the representation of which has its MSB $= 1$ and all other bits 0. In converting an integer value to a posit value, the integer representation with its MSB $= 1$ and all other bits 0 converts to NaR; otherwise, the integer value is rounded, per Section 4.1.

6.5 Conversions between posit format and IEEE Std 754™ float format

Supported posit formats must provide conversions to and from the IEEE Std 754 float formats supported in the computing environment, if any. In converting a posit value to an IEEE Std 754 float value of any type, the posit value zero converts to the "positive zero" float value, and NaR converts to quiet NaN. Otherwise, the posit value is converted to a float value per the float rounding mode in use. In converting a float value to a posit value, all forms of infinity and NaN convert to NaR. Otherwise, the float value is rounded, per Section 4.1. "Negative zero" and "positive zero" float values convert to the posit value zero. □

[14]Note that precision conversion does not require decoding a posit representation into its bit fields.