Posit Standard Documentation
Release 3.2-draft

Posit Working Group

Jun 23, 2018

Contents

Standard for Posit Arithmetic

Sponsor

Agency for Science, Technology and Research (A*STAR)

Abstract

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

Keywords

Arithmetic, binary, exponent, format, fraction, NaR, number rounding, quire, regime

Participants

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

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

1 Overview

1.1 Scope

This standard specifies the storage format and mathematical behavior of posit numbers, and the set of functions a posit arithmetic system must support, including basic arithmetic operations. It includes a description of how results are to be rounded, what situations generate an exception, and whether those exceptions are handled by the implementation or by the user.

1.2 Purpose

This standard provides a system for computing 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 of values. All features are accessible by programming languages; the source program and input data are sufficient to specify the output exactly on any computer system, similar to the way 2's complement integer arithmetic produces bitwise-identical results.

1.3 Inclusions

This standard specifies:

- Formats for binary data, for computation and data interchange
- Addition, subtraction, multiplication, division, dot product, compare, and other operations
- Mathematical functions such as logarithm, exponential, trigonometric, and hyperbolic functions
- Conversions of other number representations to posit format
- Conversions between different posit formats
- Exception handling when a result is not a real number (NaR).

1.4 Requirements vs. Recommendations

All descriptions herein are requirements of the behavior of the system. The decision of how to satisfy the requirements (using any combination of hardware and software) 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.

1.5 Programming Environment

A programming environment may claim to be compliant with this standard if it supports at least one of the four precisions (8, 16, 32, 64) completely. If it includes more than one precision, then it must also provide the ability to convert between those precisions.

2 Definitions, abbreviations, and acronyms

2.1 Definitions

correct rounding This standard's method of converting an infinitely precise value to a posit. A posit so obtained is said to be correctly rounded.

es exponent size. The maximum number of bits 0, 1, 2, 3, ... that are available for expressing the exponent.

exponent The part of the power-of-two scaling determined by the exponent bits.

exponent bits A field of bits within a posit that, in combination with the regime bits, determines the power-of-two scaling of the fraction.

exponent size es

format A set of bits and the definition of their meaning.

fraction The component of a posit containing its significant binary digits after the binary point; $0 \leq fraction < 1$.

hidden bit An assumed 1 bit before the MSB of the fraction.

LSB least significant bit

lg logarithm base 2

MSB most significant bit

maxpos The largest real value expressible as a posit.

minpos The smallest nonzero value expressible as a posit.

NaR Not a real. A value that has infinite magnitude, is indeterminate, is multi-valued, or requires an imaginary component to express (like $\sqrt{-1}$) is represented as `NaR` .

nbits number of bits. The precision of a posit format, the total number of bits (8, 16, 32, or 64).

not a real number `NaR`

number of bits `nbits`

pintmax posit integer maximum. The largest consecutive integer expressible as a posit.

posit A real number that is exactly representable using a fixed number of bits in the format described in this standard, or a `NaR` .

precision The number of bits available for expressing a quantity.

quire A fixed-point format capable of storing sums of products of posits without rounding, up to some large number of such products.

regime A subfield of a posit consisting of some number of identical bits terminated by the opposite bit or the end of the number, that contributes to the specification of the power-of-two scaling of the fraction.

sign The value +1 for positive numbers, -1 for negative numbers. Exception values 0 and `NaR` have no sign.

sign bit The MSB of a posit or quire, 0 or 1.

significand The implicit 1 bit followed by the fraction bits; $1 \leq significand < 2$.

universal number unum

unum universal numbers express real numbers (posits) and ranges of real numbers (valids).

unum seed useed

useed unum seed. A value obtained by starting with 2 and squaring repeatedly es times: 2, 4, 16, 256, ... influencing the way the projective real circle of unums gets populated.

property	**posit8**	**posit16**	**posit32**	**posit64**
Max significand bits	6	13	28	59
Max exponent bits, `es`	0	1	2	3
`minpos`	$2^{-6} \approx 1.5 \times 10^{-2}$	$2^{-28} \approx 3.7 \times 10^{-9}$	$2^{-120} \approx 7.5 \times 10^{-37}$	$2^{-496} \approx 4.9 \times 10^{-150}$
`maxpos`	$2^{6} \approx 6.4 \times 10^{1}$	$2^{28} \approx 2.7 \times 10^{8}$	$2^{120} \approx 1.3 \times 10^{36}$	$2^{496} \approx 2.0 \times 10^{149}$
`pintmax`	8	256	2^{22}	2^{52}
quire bits	32	128	512	2048
Exact sum quire limit	32767	$2^{43}-1$	$2^{151}-1$	$2^{559}-1$
Exact dot product quire limit	127	32767	$2^{31}-1$	$2^{63}-1$

Table 1: Properties of posit formats

3 Posit and quire formats

3.1 Overview

3.1.1 Formats

This clause defines posit formats, which are used to represent a finite set of real numbers. Posit formats are specified by their precision, *nbits*. For each posit format, there is also a format of size $nbits^2/2$ that is used to contain exact sums of products of posits. All properties such as dynamic range, accuracy, quire size and format, are determined solely by the precision.

There are four precisions described in this standard: 8, 16, 32, and 64. We sometimes refer to the four corresponding formats as posit8, posit16, posit32, and posit64.

3.1.2 Compliance

An implementation is compliant with this standard if it supports full functionality of at least one precision (8, 16, 32, or 64). If the implementation supports more than one precision, then it must support conversions between the precisions that it supports.

Note: If hardware supports posit multiplication, addition, subtraction, and the quire, all remaining functionality can be supported with software.

3.1.3 Represented data

Within each format, a posit represents either NaR, or a number of the form $m \times 2^n$, where m and n are integers limited to a range symmetrical about and including zero. The maximum m range is $-2^p < m < 2^p$ where $p = nbits - lg(nbits) + 1$ is the maximum number of significant digits (bits).

The smallest positive posit, *minpos*, is $2^{\frac{1}{8}\text{nbits}(2-nbits)}$ and the largest positive posit, *maxpos*, is the reciprocal of *minpos*, or $2^{\frac{1}{8}\text{nbits}(nbits-2)}$. Every posit is an integer multiple of *minpos*. Every real number maps to a unique posit representation; there are no redundant representations.

The quire represents either NaR or an integer multiple of $minpos^2$, represented as a 2's complement binary number with $2^{\text{nbits}^2/2}$ bits. This enables it to add a list of posits or a list of exact products of posits without rounding error and thereby satisfy the associative and distributive laws of algebra up to some minimum length. Sums of lists longer than that minimum are capable of integer overflow.

Posits can exactly express all integers i in a range $-pintmax \leq i \leq pintmax$; outside that range, integers exist that cannot be expressed as a posit without rounding to a different integer.

The values for the posit formats are summarized in properties-table.

The exact sum quire limit and exact dot product quire limit are the number of additions or multiplication-additions up to which the quire cannot overflow. Up to these limits, the quire obeys the associative law of addition and the distributive law.

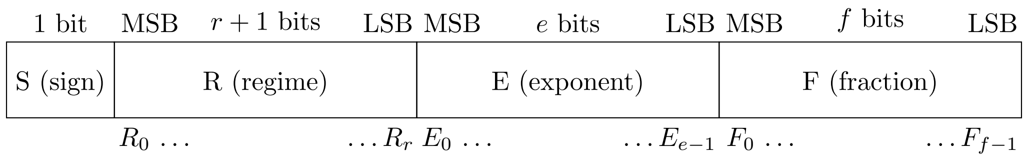

Figure 1: General binary posit format

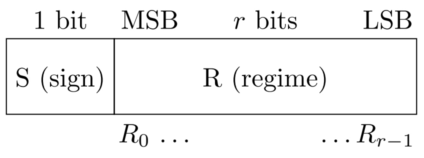

Figure 2: Binary posit format with zero-length exponent and fraction

3.2 Binary interchange format encoding

3.2.1 Posit format encoding

All posits have just one encoding in a binary interchange format shown in posit_format_encoding-general and posit_format_encoding-signedregime. The four fields are:

1. Sign bit S
2. Regime R consisting of r bits identical to R_0, terminated by $1 - R_0$ ($r+1$ bits total length) or the end of the posit (r bits total length).
3. Exponent E represented by e exponent bits, terminated by a maximum of es or the end of the posit
4. Fraction F represented by f fraction bits, terminated by the end of the posit

The meaning of each field is as follows:

1. S is its literal value, 0 or 1.
2. R is $-r$ if R_0 is 0, and $r-1$ if R_0 is 1.
3. E is an es-bit unsigned integer, with 0 bit padding in the least significant bits if the exponent field has fewer than es bits because of the regime length.
4. F represents an unsigned integer divided by 2^f.

Note

The exponent field size e and fraction field size f can each be 0, in which case they represent 0; $0 \leq e \leq es$ and $0 \leq f \leq nbits - lg(nbits)$. The hidden bit is 1 even if f is 0.

The representation (S, R, E, F) of the posit and value v of the datum represented are inferred from the fields as follows:

1. If $S = 0$ and all other fields contain only 0 bits, then $v = 0$.
2. If $S = 1$ and all other fields contain only 0 bits, then v is NaR and undefined.
3. If any bits in the (R, E, F) are 1, then $(1 - 3S + F) \times 2^{(-1)^S (R \times 2^{es} + E + S)}$.

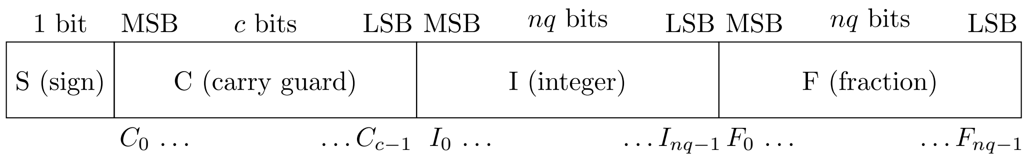

Figure 3: Binary quire format

3.2.2 Quire format encoding

The quire is a fixed-point 2's complement value of length $nbits^2/2$ which is 32, 128, 512, or 2048 bits for the posit sizes 8, 16, 32, and 64 respectively.

The number of bits for the fraction is $nq = 1/4nbits^2 - 1/2nbits$. The integer part also has nq bits. The carry guard has $c = nbits - 1$ bits to guarantee that sums of products cannot overflow, up to $2^{nbits-1} - 1$ products.

The representation (S, C, I, F) of the quire and value v of the datum represented are inferred from the fields as follows:

1. If $S = 1$ and all other fields contain only 0 bits, then v is NaR and undefined.
2. For all other cases, the value v is the 2's complement signed integer represented by all bits, divided by 2^{nq}.

4 Rounding

4.1 Definition and Method

Rounding is the substitution of an expressible posit for any exact real number that is not expressible as a posit. The results of all operations are regarded as mathematically exact prior to rounding.

The method for rounding a real value x is as follows:

1. If x is exactly expressible as a posit, it is unchanged.
2. If $|x| > maxpos$, x is rounded to `sign(x) * maxpos`.
3. If $0 < |x| < minpos$, x is rounded to `sign(x) * minpos`.
4. For all other values, the value is rounded to the nearest binary value if the posit were encoded to infinite precision beyond the *nbits* length; if two posits are equally near, the one with binary encoding ending in 0 is selected.

Note: Rule (4) has the effect of rounding to the posit with the nearest logarithm when the dropped bit is an exponent bit, and to the nearest posit by absolute difference in other cases.

4.2 Fused Operations

A fused operation is an expression with two or more operations that is not rounded until the entire expression is evaluated exactly. Fused operations are distinct from non-fused operations and must be explicitly requested in a posit-compliant programming environment. Fused operations are those expressible as sums and differences of the exact product of two posits; no other fused operations are allowed.

All fused operations can be performed as accumulations in a . A particular posit environment may perform fused operations without using a quire, but may not fuse any operations that cannot be performed as exact dot products of vectors with posit components. Exact sums are dot products where one vector consists of all 1 values. The fused multiply-add operation $ab + c$ is a dot product of vectors $(a, 1)$ and (b, c). A complex product $(a + bi) \times (c + di)$ can be performed as two fused operations, $ac - bd$ and $ad + bc$. An expression such as abc or ab/d is not in the form of the sum or difference of products and may not be fused.

5 Operations

5.1 Guiding principles

If `NaR` is the input to an operation, the result is also `NaR` . For a function $f(p)$ of a posit p, the mathematical value y is the limit $f(x)$ as x approaches p within the domain of the function, from any direction. If that limit is a determinate real number, the operation should return the closest posit to y using the rounding rules of clause 4. Similarly, for functions of more than one argument; the limits for each argument are taken without correlation to one another. If the limit is not a real number, the result is `NaR` .

Many functions are identical to standard 2's complement integer operations. The internal processor flags for those posit operations behave identically.

5.2 Mathematical functions

The following functions shall be supported, with correct rounding for all input arguments per clause 4. If cases can produce `NaR` from non-NaR inputs, the function description notes those cases.

5.2.1 Elementary functions of one argument

negate(*posit*) is identical to 2's complement integer negation.

abs(*posit*) is identical to 2's complement integer absolute value.

round(*posit*) converts *posit* to the nearest posit with integer value, and the nearest even integer if two integers are equally far from *posit*.

sign(*posit*) returns 1 if the value of *posit* is positive, and -1 if the value of *posit* is 1. If *posit* is zero or NaR, sign(*posit*) returns 0.

5.2.2 Elementary functions of two arguments

addition(posit1, posit2) returns `posit1 + posit2`, rounded

subtraction(posit1, posit2) returns `posit1 - posit2`, rounded

multiplication(posit1, posit2) returns `posit1 * posit2`, rounded

division(posit1, posit2) returns `NaR` if `posit2` is 0, else `posit1 / posit2`, rounded

5.2.3 Comparison functions of two arguments

All comparison functions are identical to the comparisons of the posit bit strings regarded as 2's complement integers, so there is no need for separate machine-level instructions. The value `NaR` has the bit string of the most negative integer, so `NaR < posit` returns True if `posit` is not `NaR` . The posit environment shall support:

```
boolean compareEqual(posit1, posit2)
boolean compareNotEqual(posit1, posit2)
boolean compareGreater(posit1, posit2)
boolean compareGreaterEqual(posit1, posit2)
boolean compareLess(posit1, posit2)
boolean compareLessEqual(posit1, posit2)
```

Note: Testing if a posit is `NaR` is not an exception: **compareEqual**(NaR, *posit*).

5.2.4 Functions of one argument

sqrt(*posit*) returns `NaR` if $posit < 0$, else the square root of *posit*, rounded.
rSqrt(*posit*) returns `NaR` if $posit < 0$, else 1/sqrt(*posit*), rounded.
exp(*posit*) returns e^{posit}, rounded.
expm1(*posit*) returns $e^{posit} - 1$, rounded.

exp2(*posit*) returns 2^{posit}, rounded.
exp2m1(*posit*) returns $2^{posit} - 1$, rounded.
exp10(*posit*) returns 10^{posit}, rounded.
exp10m1(*posit*) returns $10^{posit} - 1$, rounded.
log(*posit*) returns $log_e(posit)$, rounded. Returns NaR if $posit \leq 0$.
logp1(*posit*) returns $log_e(posit + 1)$, rounded. Returns NaR if $posit \leq -1$.
log2(*posit*) returns $log_2(posit)$, rounded. Returns NaR if $posit \leq 0$.
log2p1(*posit*) returns $log_2(posit + 1)$, rounded. Returns `NaR` if $posit \leq -1$.
log10(*posit*) returns $log_{10}(posit)$, rounded. Returns NaR if $posit \leq 0$.
log10p1(*posit*) returns $log_{10}(posit + 1)$, rounded. Returns `NaR` if $posit \leq -1$.
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.
asin(*posit*) returns `NaR` if $|posit| > 1$, else arcsin(*posit*), rounded.
asinPi(*posit*) returns `NaR` if $|posit| > 1$, else arcsin(*posit*) / π, rounded.
acos(*posit*) returns `NaR` if $|posit| > 1$, else arccos(*posit*), rounded.
acosPi(*posit*) returns `NaR` if $|posit| > 1$, else arccos(*posit*) / π, rounded.
atan(*posit*) returns arctan(*posit*), rounded.
atanPi(*posit*) returns arctan(*posit*) / π, rounded.
sinh(*posit*) returns sin(*posit*), rounded.
cosh(*posit*) returns cos(*posit*), rounded.
tanh(*posit*) returns tan(*posit*), rounded.
asinh(*posit*) returns `NaR` if $|posit| > 1$, else arcsin(*posit*), rounded.
acosh(*posit*) returns `NaR` if $|posit| > 1$, else arccos(*posit*), rounded.
atanh(*posit*) returns arctan(*posit*), rounded.

5.2.5 Functions of two posit arguments

hypot(*posit*1, *posit*2) returns the square root of $posit1^2 + posit2^2$, rounded.
pow(*posit*1, *posit*2) returns $posit1^{posit2}$, rounded. (exceptions…)
atan2(*posit*1, *posit*2) atan2Pi**(*posit*1, *posit*2)

5.2.6 Functions of a posit argument and an integer argument

compound(*posit*, *n*) returns `NaR` if $x \leq -1$, else $(1 + x)^n$, rounded
pown(*posit*, *n*) returns `NaR` if …,
rootn(*posit*, *n*)

6 Conversion to and from character format

6.1 Guiding principles

Complete section on conversion to and from character format

7 Language support

7.1 Guiding principles

Complete section on language support