Document Number:	P3844R4
Date:	2026-02-13
Reply-to:	Matthias Kretz <m.kretz@gsi.de>
Audience:	LWG
Target:	C++26

Reword [simd.math] for consteval conversions

ABSTRACT

If a conversion to `basic_vec` is marked `consteval` then [simd.math] functions fail to work equivalent to `<cmath>` functions, which perform conversions on the caller side. Performing conversions after the function call leads to immediate escalation, which requires all arguments to the [simd.math] function to be constant expressions. This does not match the design intent for [simd.math] and can be fixed by respecifying the math overloads.

CONTENTS

1 CHANGELOG

1.1 CHANGES FROM REVISION 0

Previous revision: P3844R0

- Constrain the `consteval` ctor to arithmetic types that naturally convert to the `basic_vec`'s value-type (checked via `common_type`).
- Discuss consequence on `common_type` (besides `convertible_to`).
- Discuss consequence on [simd.math].
- Discuss `consteval` ctor as immediate-escalating expression.
- Discuss potential consequences for [simd.math].

1.2 CHANGES FROM REVISION 1

Previous revision: P3844R1

- Remove proposed polls.
- Update wording to remove unnecessary *`simd-broadcast-arg`*.
- Add [simd.math] wording changes to match original design intent (as if it were an explicit overload set).
- Drive-by wording fixes/improvements to [simd.math]:
  - *`math-common-simd-t<V0, V1>`* did not work for non-default ABI tag combinations with scalars.
  - Spell out precondition on integral `abs`.

1.3 CHANGES FROM REVISION 2

Previous revision: P3844R2

- Restore incorrectly removed *`math-floating-point`* wording.
- Fix incorrect constraint in `hypot` expample.
- Split wording section into the `consteval` ctor part and the [simd.math] part.

1.4 Changes from revision 3

Previous revision: P3844R3

- Move `consteval` constructor discussion and wording into a separate paper [P4012R0].
- In [simd.math] wording, move all functions with the same name together.
- Fix return type of additional overloads of `isgreater`, `islessequal`, `islessgreater`, and `isunordered`.
- Add missing overloads for `isgreaterequal`, `isless`.
- Simplify placeholder functions to `static_cast` to prvalue rather than const-ref. Since `basic_vec` types are trivially copyable a copy can be elided by the implementation as-if.
- Say "parameters" instead of "arguments" when referencing the math function parameters.
- Add a missing space.
- Work around a LaTeX rendering issue.

2 Straw Polls

2.1 LEWG @ Kona 2025

Poll: We would like to pursue fixing the issue brought up in "DE-286 29.10.7.2p1–4 [simd.ctor] Add consteval broadcast constructor from constant integer (P3844)" for C++26.

SF	F	N	A	SA
13	5	1	0	0

Poll: Resolve "DE-286 29.10.7.2p1–4 [simd.ctor] Add consteval broadcast constructor from constant integer (P3844)" by adding a consteval broadcast overload for value-preserving conversions, and re-specify in [simd.math] and send to LWG.

SF	F	N	A	SA
3	15	1	0	0

3 Motivation

Refresher (`https://compiler-explorer.com/z/qvdoxavMK`):

```
struct A
{ consteval A(int) {} };

constexpr A f(int, auto y) // f gets promoted to an immediate function
{ return y; } // immediate-escalating expression 'A(y)'

A test(int x)
{ return f(x, 1); } // Error: 'x' is not a constant expression
```

If we remove `constexpr` from `f`, the underlying issue becomes apparent: `f` calls a `consteval` constructor that uses `y` as it's argument. And that can't work, because `y` is not a constant expression. The magic of promotion to immediate function simply makes the compiler try harder to make it compile anyway.

3.1 Escalating [simd.math] functions

Any `simd::vec` broadcast expression that promotes the surrounding function to an immediate function becomes "interesting" if not surprising or problematic. The [simd.math] functions as discussed above are affected. While `simd::hypot(vec<float>(), 1)` is fine (because the first argument is a constant expression),

```
auto f(simd::vec<float> x) { return simd::hypot(x, 1); }
```

is not, even though `1` can convert to `float` without loss of value. That's because the `hypot` implementation needs to call a `consteval` function, which then promotes `hypot` itself to an immediate function, which in turn requires all arguments to `hypot` to be constant expressions. If immediate-escalation were not applied, then the conversion from `int` to `vec<float>` inside of `hypot` would be ill-formed. Either way, `simd::hypot(x, 1)` with not-constant `x` cannot be valid.

3.2 Move conversions before math calls

We can respecify [simd.math] in such a way that conversions happen before the function is called, mirroring the actual behavior of `<cmath>` overloads. For 2-arg math functions we would have to change from one function template to three function templates:

```
template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const V&, const V&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const V&, const deduced-vec-t<V>&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const deduced-vec-t<V>&, const V&);
```

However, for 3-arg math functions we would have to change to seven function templates. I can report that this works for all my test cases. I believe I tested a representative set of argument types and permutations.[1]

```
template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const V&, const V&, const V&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const V&, const deduced-vec-t<V>&, const deduced-vec-t<V>&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const deduced-vec-t<V>&, const V&, const deduced-vec-t<V>&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
```

1 I tested arguments of type `reference_wrapper<vec<float>>`, `reference_wrapper<float>`, `reference_wrapper<short>`, `vec<float>`, `float`, `short`, and immediate arguments with value `1` (consteval broadcast from `int`). I tested all permutations where at least one argument is either `vec<float>` or `reference_wrapper<vec<float>>`.

```
  hypot(const deduced-vec-t<V>&, const deduced-vec-t<V>&, const V&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const V&, const V&, const deduced-vec-t<V>&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const V&, const deduced-vec-t<V>&, const V&);

template<math-floating-point V>
  constexpr deduced-vec-t<V>
  hypot(const deduced-vec-t<V>&, const V&, const V&);
```

3.3 alternative via "expression alias" P2826

P2826, which is awaiting a revision for consideration for C++29, could solve this more elegantly. However, we don't have the feature available yet. Thus we would need to ship C++26 with a [simd.math] specification that is forward-compatible

3.4 the importantance of conversions

Consider the `pow` function. It is fairly common to call `pow` with an integral exponent, e.g.

```
std::pow(x, 3); // x³
```

This is well-formed if `x` is of floating-point type. It would be unfortunate if the same expression would not work for `x` of type `vec<floating-point-type>`.

4 Implementation experience

A representative set of [simd.math] is implemented.

5 WORDING FOR [SIMD.MATH]

5.1 MODIFY [SIMD.EXPOS]

In [simd.expos], change:

[simd.expos]

```
~~template<class V, class T> using make-compatible-simd-t = see below; // exposition only~~

template<class V>
  concept simd-vec-type =                                          // exposition only
    same_as<V, basic_vec<typename V::value_type, typename V::abi_type>> &&
    is_default_constructible_v<V>;

template<class V>
  concept simd-mask-type =                                         // exposition only
    same_as<V, basic_mask<mask-element-size<V>, typename V::abi_type>> &&
    is_default_constructible_v<V>;

template<class V>
  concept simd-floating-point =                                    // exposition only
    simd-vec-type<V> && floating_point<typename V::value_type>;

template<class V>
  concept simd-integral =                                          // exposition only
    simd-vec-type<V> && integral<typename V::value_type>;

template<class V>
  using simd-complex-value-type = V::value_type::value_type; // exposition only

template<class V>
  concept simd-complex =                                           // exposition only
    simd-vec-type<V> && same_as<typename V::value_type, complex<simd-complex-value-type<V>>>;

template<class~~... Ts~~ T>
  concept math-floating-point =                                    // exposition only
    ~~(~~simd-floating-point<deduced-vec-t<T~~s~~>>~~ || ...)~~;

~~template<class... Ts>~~
  ~~requires math-floating-point<Ts...>~~
    ~~using math-common-simd-t = see below;~~                      ~~// exposition only~~

template<class BinaryOperation, class T>
  concept reduction-binary-operation = see below;                  // exposition only
```

5.2 modify [simd.expos.defn]

In [simd.expos.defn], remove:

[simd.expos.defn]

```
template<class T> using deduced-vec-t = see below;
```

1 Let x denote an lvalue of type const T.

2 *deduced-vec-t*<T> is an alias for

- ~~decltype(x + x), if the type of x + x is an enabled specialization of basic_vec; otherwise~~
- ~~void~~.

~~template<class V, class T> using make-compatible-simd-t = see below;~~

3 ~~Let x denote an lvalue of type const T.~~

4 ~~*make-compatible-simd-t*<V, T> is an alias for~~

- ~~*deduced-vec-t*<T>, if that type is not void, otherwise~~
- ~~vec<decltype(x + x), V::size()>.~~

~~template<class... Ts>~~
~~requires *math-floating-point*<Ts...>~~
~~using *math-common-simd-t* = see below;~~

5 ~~Let T0 denote Ts...[0]. Let T1 denote Ts...[1]. Let TRest denote a pack such that T0, T1, TRest... is equivalent to Ts....~~

6 ~~Let *math-common-simd-t*<Ts...> be an alias for~~

- ~~*deduced-vec-t*<T0>, if sizeof...(Ts) equals 1; otherwise~~
- ~~common_type_t<*deduced-vec-t*<T0>, *deduced-vec-t*<T1>, if sizeof...(Ts) equals 2 and *math-floating-point*<T0> && *math-floating-point*<T1> is true; otherwise~~
- ~~common_type_t<*deduced-vec-t*<T0>, T1>, if sizeof...(Ts) equals 2 and *math-floating-point*<T0 is true; otherwise~~
- ~~common_type_t<T0, *deduced-vec-t*<T1>, if sizeof...(Ts) equals 2; otherwise~~
- ~~common_type_t<*math-common-simd-t*<T0, T1>, TRest...>, if *math-common-simd-t*<T0, T1> is valid and denotes a type; otherwise~~
- ~~common_type_t<*math-common-simd-t*<TRest...>, T0, T1>.~~

5.3 modify [simd.syn]

In [simd.syn], change:

[simd.syn]

```
template<size_t Bytes, class Abi, class T, class U>
  constexpr auto select(const basic_mask<Bytes, Abi>& c, const T& a, const U& b)
  noexcept -> decltype(simd-select-impl(c, a, b));

// ([simd.math]), mathematical functions
template<math-floating-point V> constexpr deduced-vec-t<V> acos(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> asin(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> atan(const V& x);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> atan2(const V~~0~~& y, const V~~1~~& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> atan2(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> atan2(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V> constexpr deduced-vec-t<V> cos(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> sin(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> tan(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> acosh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> asinh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> atanh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> cosh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> sinh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> tanh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> exp(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> exp2(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> expm1(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    frexp(const V& value, rebind_t<int, deduced-vec-t<V>>* exp);
template<math-floating-point V>
  constexpr rebind_t<int, deduced-vec-t<V>> ilogb(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> ldexp(const V& x, const rebind_t<int, deduced-vec-t<V>>& exp);
template<math-floating-point V> constexpr deduced-vec-t<V> log(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log10(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log1p(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log2(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> logb(const V& x);
template<class T, class Abi>
  constexpr basic_vec<T, Abi>
    modf(const type_identity_t<basic_vec<T, Abi>>& value, basic_vec<T, Abi>* iptr);
template<math-floating-point V>
  constexpr deduced-vec-t<V> scalbn(const V& x, const rebind_t<int, deduced-vec-t<V>>& n);
template<math-floating-point V>
  constexpr deduced-vec-t<V> scalbln(
    const V& x, const rebind_t<long int, deduced-vec-t<V>>& n);
template<math-floating-point V> constexpr deduced-vec-t<V> cbrt(const V& x);
template<signed_integral T, class Abi>
  constexpr basic_vec<T, Abi> abs(const basic_vec<T, Abi>& j);
template<math-floating-point V> constexpr deduced-vec-t<V> abs(const V& j);
```

```
template<math-floating-point V> constexpr deduced-vec-t<V> fabs(const V& x);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> hypot(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1, class V2math-floating-point V>
  constexpr math-common-simd-t<V0, V1, V2>deduced-vec-t<V> hypot(const V0& x, const V1& y, const V2& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y
                                   const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const V& y
                                   const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const deduced-vec-t<V>& y
                                   const deduced-vec-t<V>& z);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> pow(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> pow(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> pow(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V> constexpr deduced-vec-t<V> sqrt(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> erf(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> erfc(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> lgamma(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> tgamma(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> ceil(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> floor(const V& x);
template<math-floating-point V> deduced-vec-t<V> nearbyint(const V& x);
template<math-floating-point V> deduced-vec-t<V> rint(const V& x);
template<math-floating-point V>
  rebind_t<long int, deduced-vec-t<V>> lrint(const V& x);
template<math-floating-point V>
  rebind_t<long long int, V> llrint(const deduced-vec-t<V>& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> round(const V& x);
template<math-floating-point V>
  constexpr rebind_t<long int, deduced-vec-t<V>> lround(const V& x);
template<math-floating-point V>
  constexpr rebind_t<long long int, deduced-vec-t<V>> llround(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> trunc(const V& x);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> fmod(const V0& x, const V1& y);
template<math-floating-point V>
```

```
  constexpr deduced-vec-t<V> fmod(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmod(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> remainder(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> remainder(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> remainder(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>
    remquo(const V~~0~~& x, const V~~1~~& y, rebind_t<int, ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>>* quo);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    remquo(const deduced-vec-t<V>& x, const V& y, rebind_t<int, deduced-vec-t<V>> quo);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    remquo(const V& x, const deduced-vec-t<V>& y, rebind_t<int, deduced-vec-t<V>> quo);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> copysign(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> copysign(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> copysign(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> nextafter(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> nextafter(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> nextafter(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> fdim(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fdim(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fdim(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> fmax(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmax(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmax(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> fmin(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmin(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmin(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1, class V2~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1, V2>~~deduced-vec-t<V> fma(const V~~0~~& x, const V~~1~~& y, const V~~2~~& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const V& x, const deduced-vec-t<V>& y, const V& z);
```

```
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y
                                 const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const deduced-vec-t<V>& x, const V& y
                                 const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const V& x, const deduced-vec-t<V>& y
                                 const deduced-vec-t<V>& z);
template<~~class V0, class V1, class V2~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1, V2>~~deduced-vec-t<V>
    lerp(const V~~0~~& a, const V~~1~~& b, const V~~2~~& t) noexcept;
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y
                                  const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const deduced-vec-t<V>& x, const V& y
                                  const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const V& x, const deduced-vec-t<V>& y
                                  const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr rebind_t<int, deduced-vec-t<V>> fpclassify(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isfinite(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isinf(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isnan(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isnormal(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type signbit(const V& x);
template<~~class V0, class V1~~math-floating-point V>
  constexpr typename ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>::mask_type
    isgreater(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isgreater(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isgreater(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr typename ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>::mask_type
    isgreaterequal(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    isgreaterequal(const deduced-vec-t<V>& x, const V& y);
```

```
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    isgreaterequal(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr typename ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>::mask_type
    isless(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isless(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isless(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr typename ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>::mask_type
    islessequal(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    islessequal(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    islessequal(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr typename ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>::mask_type
    islessgreater(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    islessgreater(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    islessgreater(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1~~math-floating-point V>
  constexpr typename ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V>::mask_type
    isunordered(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    isunordered(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    isunordered(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V>
  deduced-vec-t<V> assoc_laguerre(const rebind_t<unsigned, deduced-vec-t<V>>& n,
                                  const rebind_t<unsigned, deduced-vec-t<V>>& m, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> assoc_legendre(const rebind_t<unsigned, deduced-vec-t<V>>& l,
                                  const rebind_t<unsigned, deduced-vec-t<V>>& m, const V& x);
template<~~class V0, class V1~~math-floating-point V>
  ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> beta(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  deduced-vec-t<V> beta(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> beta(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V> deduced-vec-t<V> comp_ellint_1(const V& k);
template<math-floating-point V> deduced-vec-t<V> comp_ellint_2(const V& k);
template<~~class V0, class V1~~math-floating-point V>
  ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> comp_ellint_3(const V~~0~~& k, const V~~1~~& nu);
template<math-floating-point V>
```

```
  deduced-vec-t<V> comp_ellint_3(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> comp_ellint_3(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_bessel_i(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_i(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_i(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_bessel_j(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_j(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_j(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_bessel_k(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_k(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_k(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_neumann(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_neumann(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_neumann(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> ellint_1(const V0& k, const V1& phi);
template<math-floating-point V>
  deduced-vec-t<V> ellint_1(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> ellint_1(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> ellint_2(const V0& k, const V1& phi);
template<math-floating-point V>
  deduced-vec-t<V> ellint_2(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> ellint_2(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1, class V2math-floating-point V>
  math-common-simd-t<V0, V1, V2>deduced-vec-t<V> ellint_3(const V0& k, const V1& nu, const V2& phi);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const deduced-vec-t<V>& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const V& x, const deduced-vec-t<V>& y, const deduced-vec-t<V>& z);
template<math-floating-point V> deduced-vec-t<V> expint(const V& x);
```

```
    template<math-floating-point V>
      deduced-vec-t<V> hermite(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
    template<math-floating-point V>
      deduced-vec-t<V> laguerre(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
    template<math-floating-point V>
      deduced-vec-t<V> legendre(const rebind_t<unsigned, deduced-vec-t<V>>& l, const V& x);
    template<math-floating-point V>
      deduced-vec-t<V> riemann_zeta(const V& x);
    template<math-floating-point V>
      deduced-vec-t<V> sph_bessel(
        const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
    template<math-floating-point V>
      deduced-vec-t<V> sph_legendre(const rebind_t<unsigned, deduced-vec-t<V>>& l,
        const rebind_t<unsigned, deduced-vec-t<V>>& m, const V& theta);
    template<math-floating-point V>
      deduced-vec-t<V>
        sph_neumann(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);

    // ([simd.bit]), bit manipulation
    template<simd-vec-type V> constexpr V byteswap(const V& v) noexcept;
```

5.4 MODIFY [SIMD.MATH]

In [simd.math], change:

[simd.math]

```
template<math-floating-point V>
  constexpr rebind_t<int, deduced-vec-t<V>> ilogb(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> ldexp(const V& x, const rebind_t<int, deduced-vec-t<V>>& exp);
template<math-floating-point V>
  constexpr deduced-vec-t<V> scalbn(const V& x, const rebind_t<int, deduced-vec-t<V>>& n);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    scalbln(const V& x, const rebind_t<long int, deduced-vec-t<V>>& n);
template<signed_integral T, class Abi>
  constexpr basic_vec<T, Abi> abs(const basic_vec<T, Abi>& j);
template<math-floating-point V>
  constexpr deduced-vec-t<V> abs(const V& j);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fabs(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> ceil(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> floor(const V& x);
template<math-floating-point V>
  deduced-vec-t<V> nearbyint(const V& x);
template<math-floating-point V>
  deduced-vec-t<V> rint(const V& x);
template<math-floating-point V>
  rebind_t<long int, deduced-vec-t<V>> lrint(const V& x);
template<math-floating-point V>
```

```
  rebind_t<long long int, deduced-vec-t<V>> llrint(const V& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> round(const V& x);
template<math-floating-point V>
  constexpr rebind_t<long int, deduced-vec-t<V>> lround(const V& x);
template<math-floating-point V>
  constexpr rebind_t<long long int, deduced-vec-t<V>> llround(const V& x);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> fmod(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmod(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmod(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> trunc(const V& x);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> remainder(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> remainder(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> remainder(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> copysign(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> copysign(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> copysign(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> nextafter(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> nextafter(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> nextafter(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> fdim(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fdim(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fdim(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> fmax(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmax(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmax(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> fmin(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmin(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fmin(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1, class V2math-floating-point V>
  constexpr math-common-simd-t<V0, V1, V2>deduced-vec-t<V> fma(const V0& x, const V1& y, const V2& z);
template<math-floating-point V>
```

```
  constexpr deduced-vec-t<V> fma(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const deduced-vec-t<V>& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> fma(const V& x, const deduced-vec-t<V>& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr rebind_t<int, deduced-vec-t<V>> fpclassify(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isfinite(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isinf(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isnan(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isnormal(const V& x);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type signbit(const V& x);
template<class V0, class V1math-floating-point V>
  constexpr typename math-common-simd-t<V0, V1>deduced-vec-t<V>::mask_type isgreater(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isgreater(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isgreater(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr typename math-common-simd-t<V0, V1>deduced-vec-t<V>::mask_type
    isgreaterequal(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    isgreaterequal(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type
    isgreaterequal(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr typename math-common-simd-t<V0, V1>deduced-vec-t<V>::mask_type isless(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isless(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isless(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr typename math-common-simd-t<V0, V1>deduced-vec-t<V>::mask_type islessequal(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type islessequal(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type islessequal(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr typename math-common-simd-t<V0, V1>deduced-vec-t<V>::mask_type islessgreater(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type islessgreater(const deduced-vec-t<V>& x, const V& y);
```

```
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type islessgreater(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  constexpr typename math-common-simd-t<V0, V1>deduced-vec-t<V>::mask_type isunordered(const V0& x, const V1& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isunordered(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr typename deduced-vec-t<V>::mask_type isunordered(const V& x, const deduced-vec-t<V>& y);
```

7 Let `Ret` denote the return type of the specialization of a function template with the name *math-func*. Let *math-func-vec* denote:

```
template<class... Args>
Ret math-func-vec(const Args&... args) {
  return Ret([&](simd-size-type i) {
    return math-func(make-compatible-simd-t<Ret, Args>deduced-vec-t<V>(args)[i]...);
  });
}
```

8 *Returns:* A value `ret` of type `Ret`, that is element-wise equal to the result of calling *math-func-vec* with the ~~arguments~~parameters of the above functions. If in an invocation of a scalar overload of *math-func* for index `i` in *math-func-vec* a domain, pole, or range error would occur, the value of `ret[i]` is unspecified.

9 *Remarks:* It is unspecified whether `errno` ([errno]) is accessed.

```
template<math-floating-point V>
  constexpr deduced-vec-t<V> ldexp(const V& x, const rebind_t<int, deduced-vec-t<V>>& exp);
template<math-floating-point V>
  constexpr deduced-vec-t<V> scalbn(const V& x, const rebind_t<int, deduced-vec-t<V>>& n);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    scalbln(const V& x, const rebind_t<long int, deduced-vec-t<V>>& n);
```

-?- Let `Ret` be *deduced-vec-t*`<V>`. Let *math-func* denote the name of the function template. Let *math-func-vec* denote:

```
Ret math-func-vec(const deduced-vec-t<V>& a, const auto& b) {
  return Ret([&](simd-size-type i) {
    return math-func(a[i], b[i]);
  });
}
```

-?- *Returns:* A value `ret` of type `Ret`, that is element-wise equal to the result of calling *math-func-vec* with the parameters of the above functions. If in an invocation of a scalar overload of *math-func* for index `i` in *math-func-vec* a domain, pole, or range error would occur, the value of `ret[i]` is unspecified.

-?- *Remarks:* It is unspecified whether `errno` ([errno]) is accessed.

```
template<signed_integral T, class Abi>
  constexpr basic_vec<T, Abi> abs(const basic_vec<T, Abi>& j);
```

-?- *Preconditions*: `all_of(j >= -numeric_limits<T>::max())` is `true`.

-?- *Returns:* An object where the i^{th} element is initialized to the result of `std::abs(j[`i`])` for all i in the range $[0, \texttt{j.size()})$.

```
template<math-floating-point V> constexpr deduced-vec-t<V> acos(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> asin(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> atan(const V& x);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> atan2(const V~~0~~& y, const V~~1~~& x);
template<math-floating-point V>
  constexpr deduced-vec-t<V> atan2(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> atan2(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V> constexpr deduced-vec-t<V> cos(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> sin(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> tan(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> acosh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> asinh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> atanh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> cosh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> sinh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> tanh(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> exp(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> exp2(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> expm1(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log10(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log1p(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> log2(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> logb(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> cbrt(const V& x);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> hypot(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const deduced-vec-t<V>& y);
template<~~class V0, class V1, class V2~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1, V2>~~deduced-vec-t<V> hypot(const V~~0~~& x, const V~~1~~& y, const V~~2~~& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const deduced-vec-t<V>& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> hypot(const V& x, const deduced-vec-t<V>& y, const deduced-vec-t<V>& z);
template<~~class V0, class V1~~math-floating-point V>
  constexpr ~~math-common-simd-t<V0, V1>~~deduced-vec-t<V> pow(const V~~0~~& x, const V~~1~~& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> pow(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  constexpr deduced-vec-t<V> pow(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V> constexpr deduced-vec-t<V> sqrt(const V& x);
```

```
template<math-floating-point V> constexpr deduced-vec-t<V> erf(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> erfc(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> lgamma(const V& x);
template<math-floating-point V> constexpr deduced-vec-t<V> tgamma(const V& x);
template<class V0, class V1, class V2math-floating-point V>
  constexpr math-common-simd-t<V0, V1, V2>deduced-vec-t<V> lerp(const V0& a, const V1& b, const V2& t) noexcept;
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const deduced-vec-t<V>& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  constexpr deduced-vec-t<V> lerp(const V& x, const deduced-vec-t<V>& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  deduced-vec-t<V> assoc_laguerre(const rebind_t<unsigned, deduced-vec-t<V>> n, const
    rebind_t<unsigned, deduced-vec-t<V>> m, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> assoc_legendre(const rebind_t<unsigned, deduced-vec-t<V>> l, const
    rebind_t<unsigned, deduced-vec-t<V>> m, const V& x);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> beta(const V0& x, const V1& y);
template<math-floating-point V>
  deduced-vec-t<V> beta(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> beta(const V& x, const deduced-vec-t<V>& y);
template<math-floating-point V> deduced-vec-t<V> comp_ellint_1(const V& k);
template<math-floating-point V> deduced-vec-t<V> comp_ellint_2(const V& k);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> comp_ellint_3(const V0& k, const V1& nu);
template<math-floating-point V>
  deduced-vec-t<V> comp_ellint_3(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> comp_ellint_3(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_bessel_i(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_i(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_i(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_bessel_j(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_j(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_j(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_bessel_k(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_k(const deduced-vec-t<V>& x, const V& y);
```

```
template<math-floating-point V>
  deduced-vec-t<V> cyl_bessel_k(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> cyl_neumann(const V0& nu, const V1& x);
template<math-floating-point V>
  deduced-vec-t<V> cyl_neumann(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> cyl_neumann(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> ellint_1(const V0& k, const V1& phi);
template<math-floating-point V>
  deduced-vec-t<V> ellint_1(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> ellint_1(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1math-floating-point V>
  math-common-simd-t<V0, V1>deduced-vec-t<V> ellint_2(const V0& k, const V1& phi);
template<math-floating-point V>
  deduced-vec-t<V> ellint_2(const deduced-vec-t<V>& x, const V& y);
template<math-floating-point V>
  deduced-vec-t<V> ellint_2(const V& x, const deduced-vec-t<V>& y);
template<class V0, class V1, class V2math-floating-point V>
  math-common-simd-t<V0, V1, V2>deduced-vec-t<V> ellint_3(const V0& k, const V1& nu, const V2& phi);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const deduced-vec-t<V>& x, const V& y, const V& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const V& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const V& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const deduced-vec-t<V>& x, const deduced-vec-t<V>& y, const V& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const deduced-vec-t<V>& x, const V& y, const deduced-vec-t<V>& z);
template<math-floating-point V>
  deduced-vec-t<V> ellint_3(const V& x, const deduced-vec-t<V>& y, const deduced-vec-t<V>& z);
template<math-floating-point V> deduced-vec-t<V> expint(const V& x);
template<math-floating-point V> deduced-vec-t<V> hermite(const rebind_t<unsigned,
deduced-vec-t<V>> n, const V& x);
template<math-floating-point V> deduced-vec-t<V> laguerre(const rebind_t<unsigned,
deduced-vec-t<V>> n, const V& x);
template<math-floating-point V> deduced-vec-t<V> legendre(const rebind_t<unsigned,
deduced-vec-t<V>> l, const V& x);
template<math-floating-point V> deduced-vec-t<V> riemann_zeta(const V& x);
template<math-floating-point V> deduced-vec-t<V> sph_bessel(const rebind_t<unsigned,
deduced-vec-t<V>> n, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> sph_legendre(const rebind_t<unsigned, deduced-vec-t<V>> l,
                                const rebind_t<unsigned, deduced-vec-t<V>> m,
                                const V& theta);
template<math-floating-point V> deduced-vec-t<V> sph_neumann(const rebind_t<unsigned,
deduced-vec-t<V>> n, const V& x);
```

10 Let `Ret` denote the return type of the specialization of a function template with the name *math-func*. Let *math-func-vec* denote:

```
template<class... Args>
Ret math-func-vec(const Args&... args) {
```

```
    return Ret([&](simd-size-type i) {
      return math-func(~~make-compatible-simd-t<Ret, Args>~~deduced-vec-t<V>(args)[i]...);
    });
  }
```

11 *Returns:* A value `ret` of type `Ret`, that is element-wise approximately equal to the result of calling *math-func-vec* with the ~~arguments~~parameters of the above functions. If in an invocation of a scalar overload of *math-func* for index `i` in *math-func-vec* a domain, pole, or range error would occur, the value of `ret[i]` is unspecified.

12 *Remarks:* It is unspecified whether `errno` ([errno]) is accessed.

```
template<math-floating-point V>
  deduced-vec-t<V> assoc_laguerre(const rebind_t<unsigned, deduced-vec-t<V>>& n,
                                  const rebind_t<unsigned, deduced-vec-t<V>>& m, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> assoc_legendre(const rebind_t<unsigned, deduced-vec-t<V>>& l,
                                  const rebind_t<unsigned, deduced-vec-t<V>>& m, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> sph_legendre(const rebind_t<unsigned, deduced-vec-t<V>>& l,
                                const rebind_t<unsigned, deduced-vec-t<V>>& m, const V& theta);
```

-?- Let *math-func* denote the name of the function template. Let *math-func-vec* denote:

```
auto math-func-vec(const auto& a, const auto& b, const deduced-vec-t<V>& c) {
  return deduced-vec-t<V>([&](simd-size-type i) {
    return std::math-func(a[i], b[i], c[i]);
  });
}
```

-?- *Returns:* An object that is element-wise approximately equal to the result of calling *math-func-vec* with the parameters of the above functions.

```
template<math-floating-point V>
  deduced-vec-t<V> hermite(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> laguerre(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> legendre(const rebind_t<unsigned, deduced-vec-t<V>>& l, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> sph_bessel(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
template<math-floating-point V>
  deduced-vec-t<V> sph_neumann(const rebind_t<unsigned, deduced-vec-t<V>>& n, const V& x);
```

-?- Let *math-func* denote the name of the function template. Let *math-func-vec* denote:

```
auto math-func-vec(const auto& a, const deduced-vec-t<V>& b) {
  return deduced-vec-t<V>([&](simd-size-type i) { return std::math-func(a[i], b[i]); });
}
```

-?- *Returns:* An object that is element-wise approximately equal to the result of calling *math-func-vec* with the parameters of the above functions.

```
template<math-floating-point V>
  constexpr deduced-vec-t<V> frexp(const V& value, rebind_t<int, deduced-vec-t<V>>* exp);
```

13 Let `Ret` be *deduced-vec-t*`<V>`. Let *frexp-vec* denote:

```
template<class V>
pair<Ret, rebind_t<int, Ret>> frexp-vec(const deduced-vec-t<V>& x) {
  int r1[Ret::size()];
  Ret r0([&](simd-size-type i) {
    return frexp(make-compatible-simd-t<Ret, V>(x)x[i], &r1[i]);
  });
  return {r0, rebind_t<int, Ret>(r1)};
}
```

Let `ret` be a value of type `pair<Ret, rebind_t<int, Ret>>` that is the same value as the result of calling *frexp-vec*(x).

14 *Effects*: Sets `*exp` to `ret.second`.

15 *Returns:* `ret.first`.

```
template<class V0, class V1math-floating-point V>
  constexpr math-common-simd-t<V0, V1>deduced-vec-t<V> remquo(const V0& x, const V1& y,
                          rebind_t<int, math-common-simd-t<V0, V1>deduced-vec-t<V>>* quo);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    remquo(const deduced-vec-t<V>& x, const V& y, rebind_t<int, deduced-vec-t<V>> quo);
template<math-floating-point V>
  constexpr deduced-vec-t<V>
    remquo(const V& x, const deduced-vec-t<V>& y, rebind_t<int, deduced-vec-t<V>> quo);
```

16 Let ~~Ret be *math-common-simd-t*<V0, V1>~~V0 be *deduced-vec-t*<V>. Let *remquo-vec* denote:

```
template<class V0, class V1>
pair<RetV0, rebind_t<int, RetV0>> remquo-vec(const V0& x, const V1V0& y) {
  int r1[RetV0::size()];
  V0 r0([&](simd-size-type i) {
      return remquo(make-compatible-simd-t<Ret, V0>(x)x[i],
      make-compatible-simd-t<Ret, V1>(y)y[i], &r1[i]);
  });
  return {r0, rebind_t<int, RetV0>(r1)};
}
```

Let `ret` be a value of type ~~`pair<Ret, rebind_t<int, Ret>>`~~`pair<V0, rebind_t<int, V0>>` that is the same value as the result of calling *remquo-vec*(x, y). If in an invocation of a scalar overload of `remquo` for index `i` in *remquo-vec* a domain, pole, or range error would occur, the value of `ret[i]` is unspecified.

17 *Effects*: Sets `*quo` to `ret.second`.

18 *Returns:* `ret.first`.

19 *Remarks:* It is unspecified whether `errno` ([errno]) is accessed.

```
template<class T, class Abi>
  constexpr basic_vec<T, Abi> modf(const type_identity_t<basic_vec<T, Abi>>& value,
                                   basic_vec<T, Abi>* iptr);
```

20 Let `V` be `basic_vec<T, Abi>`. Let *modf-vec* denote:

```
pair<V, V> modf-vec(const V& x) {
  T r1[Ret::size()];
  V r0([&](simd-size-type i) {
    return modf(V(x)[i], &r1[i]);
  });
  return {r0, V(r1)};
}
```

Let `ret` be a value of type `pair<V, V>` that is the same value as the result of calling *`modf-vec`*`(value)`.

21 *Effects*: Sets `*iptr` to `ret.second`.

22 *Returns:* `ret.first`.

A BIBLIOGRAPHY

[P4012R0] Matthias Kretz. *value-preserving consteval broadcast to `simd::basic_vec`*. ISO/IEC C++ Standards Committee Paper. 2026. URL: `https://wg21.link/p4012r0`.