Xsfvfexpa Extension Specification: Exponential Approximation Instruction

Version 0.2

Copyright © by SiFive, Inc. All rights reserved.

Xsfvfexpa Extension Specification: Exponential Approximation Instruction

Proprietary Notice

Copyright © 2025 by SiFive, Inc. All rights reserved.

Information in this document is provided “as is,” with all faults.

SiFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose and non-infringement.

SiFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

SiFive reserves the right to make changes without further notice to any products herein.

Revision History

Version	Date	Changes
0.2	September 11, 2025	• First public release

Contents

Copyright © by SiFive, Inc. All rights reserved.

List of Figures

Copyright © by SiFive, Inc. All rights reserved.

Xsfvfexpa Extension Specification: Exponential Approximation Instruction

1.1 Xsfvfexpa Extension, Version 0.2

Synopsis

Vector Floating-Point Exponential Approximation Instruction

Mnemonic

sf.vfexpa.v vd, vs2, vm

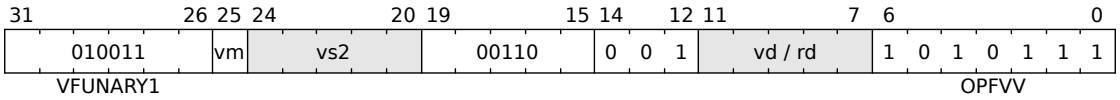

Figure 1: *Encoding (Single-Width Vector)*

Reserved Encodings

Arguments

Register	Direction	Definition
`vs2`	input	vector of SEW-bit encoded fixed-point
`vd`	output	vector of SEW-bit floating-point exponential approximations

Description

`sf.vfexpa.v` approximates two raised to a fractional power.

Let `W` be the width in bits of the exponent field of the floating-point type, and `D` be the width of an integer index: 5 for SEW=16 and 6 for SEW=32.

The `sf.vfexpa.v` instruction only supports SEW of 32 and, if `Zvfh` is present, 16; use at any other element width is reserved. `Xsfvfexpa` depends on `Zve32f`.

`sf.vfexpa.v` uses the low `D` bits of the input to index into a table of precomputed values and copies that value to the mantissa of the floating-point result. It then copies the next `W` bits of the input into the exponent field of the result. Finally, it sets the sign bit to zero. Other bits from the input are ignored.

Copyright © by SiFive, Inc. All rights reserved.

The `sf.vfexpa.v` instruction does not set any floating-point exception flags and does not modify the `fcsr` register.

```
#                         SEW     SEW
vfexpa.v  vd, vs2, vm  # vd[i] = vs2[i]<D+W-1:D> :: LUT[vs2[i]<D-1:0>]

cond SEW in
  16 : LUT[32] = {
         0x000, 0x016, 0x02d, 0x045, 0x05d, 0x075, 0x08e, 0x0a8,
         0x0c2, 0x0dc, 0x0f8, 0x114, 0x130, 0x14d, 0x16b, 0x189,
         0x1a8, 0x1c8, 0x1e8, 0x209, 0x22b, 0x24e, 0x271, 0x295,
         0x2ba, 0x2e0, 0x306, 0x32e, 0x356, 0x37f, 0x3a9, 0x3d4,
       }
  32 : LUT[64] = {
         0x000000, 0x0164d2, 0x02cd87, 0x043a29,
         0x05aac3, 0x071f62, 0x08980f, 0x0a14d5,
         0x0b95c2, 0x0d1adf, 0x0ea43a, 0x1031dc,
         0x11c3d3, 0x135a2b, 0x14f4f0, 0x16942d,
         0x1837f0, 0x19e046, 0x1b8d3a, 0x1d3eda,
         0x1ef532, 0x20b051, 0x227043, 0x243516,
         0x25fed7, 0x27cd94, 0x29a15b, 0x2b7a3a,
         0x2d583f, 0x2f3b79, 0x3123f6, 0x3311c4,
         0x3504f3, 0x36fd92, 0x38fbaf, 0x3aff5b,
         0x3d08a4, 0x3f179a, 0x412c4d, 0x4346cd,
         0x45672a, 0x478d75, 0x49b9be, 0x4bec15,
         0x4e248c, 0x506334, 0x52a81e, 0x54f35b,
         0x5744fd, 0x599d16, 0x5bfbb8, 0x5e60f5,
         0x60ccdf, 0x633f89, 0x65b907, 0x68396a,
         0x6ac0c7, 0x6d4f30, 0x6fe4ba, 0x728177,
         0x75257d, 0x77d0df, 0x7a83b3, 0x7d3e0c,
       }
end
```

Explanation (Non-Normative)

In a mathematical sense, the fractional power is encoded as the unsigned numerator `P` of a biased fixed-point number in the lower `W+D` bits of the input floating-point's mantissa, with the implicit denominator equal to `2^D`. For all SEW, the bias `B` is equal to the exponent bias of the corresponding floating-point type: `2^(W-1) - 1`.

The "intended value" is `2^(P / 2^D) / 2^B` rounded to the nearest normal floating-point value.

The "alternate value" is `(2^(P / 2^D + 1) - 2) / 2^B` rounded to the nearest subnormal floating-point value.

When `P` is `2^D` or greater the actual result returned by `sf.vfexpa.v` is the intended value.

When `P` is between 1 and `2^D`, inclusive, the actual result is the alternate value.

When `P` is equal to 0, the result is equal to floating-point zero.

For `P` greater than `2^(W-1)` the result is undefined.

Copyright © by SiFive, Inc. All rights reserved.

NOTE

`sf.vfexpa.v` requires no normalization or subnormalization of results. It is impossible to encode in the input an exponent less than `-B`

NOTE

When the biased encoded exponent is equal to `0`, corresponding to an unbiased exponent of `-B`, it is possible to recover a faithfully-rounded indended value from an alternate value by dividing by two and adding `2^-B`. It is unclear whether this would ever be useful in practice.

1.2 Xsfvfexpa64e Extension, Version 0.2

Synopsis

Vector Floating-Point Exponential Approximation Instruction with Double-Precision Support

Description

`Xsfvfexpa64e` augments the `Xsfvfexpa` extension to support SEW=64; it implies the `Xsfvfexpa` extension and it depends on `Zve64d`. It supplies the same `sf.vfexpa.v` instruction, using the same encoding, which approximates two raised to a fractional power.

The operation of `sf.vfexpa.v` is the same as in `Xsfvfexpa` except that behavior for 64-bit elements is now defined by the following additions to the lookup table specified above, where `D`=6 and `W`=11 for SEW=64:

```
#                        SEW=64     SEW=64
vfexpa.v  vd, vs2, vm  # vd[i] = vs2[i]<D+W-1:D> :: LUT[vs2[i]<D-1:0>]
SEW=64 : LUT[64] = {
       0x0000000000000, 0x02c9a3e778061,
       0x059b0d3158574, 0x0874518759bc8,
       0x0b5586cf9890f, 0x0e3ec32d3d1a2,
       0x11301d0125b51, 0x1429aaea92de0,
       0x172b83c7d517b, 0x1a35beb6fcb75,
       0x1d4873168b9aa, 0x2063b88628cd6,
       0x2387a6e756238, 0x26b4565e27cdd,
       0x29e9df51fdee1, 0x2d285a6e4030b,
       0x306fe0a31b715, 0x33c08b26416ff,
       0x371a7373aa9cb, 0x3a7db34e59ff7,
       0x3dea64c123422, 0x4160a21f72e2a,
       0x44e086061892d, 0x486a2b5c13cd0,
       0x4bfdad5362a27, 0x4f9b2769d2ca7,
       0x5342b569d4f82, 0x56f4736b527da,
       0x5ab07dd485429, 0x5e76f15ad2148,
       0x6247eb03a5585, 0x6623882552225,
       0x6a09e667f3bcd, 0x6dfb23c651a2f,
       0x71f75e8ec5f74, 0x75feb564267c9,
       0x7a11473eb0187, 0x7e2f336cf4e62,
       0x82589994cce13, 0x868d99b4492ed,
       0x8ace5422aa0db, 0x8f1ae99157736,
       0x93737b0cdc5e5, 0x97d829fde4e50,
       0x9c49182a3f090, 0xa0c667b5de565,
       0xa5503b23e255d, 0xa9e6b5579fdbf,
```

Copyright © by SiFive, Inc. All rights reserved.

```
    0xae89f995ad3ad, 0xb33a2b84f15fb,
    0xb7f76f2fb5e47, 0xbcc1e904bc1d2,
    0xc199bdd85529c, 0xc67f12e57d14b,
    0xcb720dcef9069, 0xd072d4a07897c,
    0xd5818dcfba487, 0xda9e603db3285,
    0xdfc97337b9b5f, 0xe502ee78b3ff6,
    0xea4afa2a490da, 0xefa1bee615a27,
    0xf50765b6e4540, 0xfa7c1819e90d8,
  }
```

1.3 Intrinsics

1.3.1 Xsfvfexpa

Non-policy non-overloaded

```
vfloat16mf4_t __riscv_sf_vfexpa_v_f16mf4(vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_v_f16mf2(vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexpa_v_f16m1(vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_v_f16m2(vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_v_f16m4(vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_v_f16m8(vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_v_f32mf2(vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_v_f32m1(vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_v_f32m2(vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_v_f32m4(vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_v_f32m8(vfloat32m8_t vs2, size_t vl);

// masked functions
vfloat16mf4_t __riscv_sf_vfexpa_v_f16mf4_m(vbool64_t vm, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_v_f16mf2_m(vbool32_t vm, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexpa_v_f16m1_m(vbool16_t vm, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_v_f16m2_m(vbool8_t vm, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_v_f16m4_m(vbool4_t vm, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_v_f16m8_m(vbool2_t vm, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_v_f32mf2_m(vbool64_t vm, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_v_f32m1_m(vbool32_t vm, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_v_f32m2_m(vbool16_t vm, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_v_f32m4_m(vbool8_t vm, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_v_f32m8_m(vbool4_t vm, vfloat32m8_t vs2, size_t vl);
```

Non-policy overloaded

```
vfloat16mf4_t __riscv_sf_vfexpa(vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa(vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexpa(vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa(vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa(vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa(vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa(vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa(vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa(vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa(vfloat32m4_t vs2, size_t vl);
```

Copyright © by SiFive, Inc. All rights reserved.

```
vfloat32m8_t __riscv_sf_vfexpa(vfloat32m8_t vs2, size_t vl);

// masked functions
vfloat16mf4_t __riscv_sf_vfexpa(vbool64_t vm, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa(vbool32_t vm, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexpa(vbool16_t vm, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa(vbool8_t vm, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa(vbool4_t vm, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa(vbool2_t vm, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa(vbool64_t vm, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa(vbool32_t vm, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa(vbool16_t vm, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa(vbool8_t vm, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa(vbool4_t vm, vfloat32m8_t vs2, size_t vl);
```

Policy non-overloaded

```
vfloat16mf4_t __riscv_sf_vfexpa_v_f16mf4_tu(vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_v_f16mf2_tu(vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexpa_v_f16m1_tu(vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_v_f16m2_tu(vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_v_f16m4_tu(vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_v_f16m8_tu(vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_v_f32mf2_tu(vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_v_f32m1_tu(vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_v_f32m2_tu(vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_v_f32m4_tu(vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_v_f32m8_tu(vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexpa_v_f16mf4_tum(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2,
size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_v_f16mf2_tum(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2,
size_t vl);
vfloat16m1_t __riscv_sf_vfexpa_v_f16m1_tum(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2,
size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_v_f16m2_tum(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2,
size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_v_f16m4_tum(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2,
size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_v_f16m8_tum(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2,
size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_v_f32mf2_tum(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2,
size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_v_f32m1_tum(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2,
size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_v_f32m2_tum(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2,
size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_v_f32m4_tum(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2,
size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_v_f32m8_tum(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2,
size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexpa_v_f16mf4_tumu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t
vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_v_f16mf2_tumu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t
vs2, size_t vl);
```

Copyright © by SiFive, Inc. All rights reserved.

```
vfloat16m1_t __riscv_sf_vfexpa_v_f16m1_tumu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2,
size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_v_f16m2_tumu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2,
size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_v_f16m4_tumu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2,
size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_v_f16m8_tumu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2,
size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_v_f32mf2_tumu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t
vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_v_f32m1_tumu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2,
size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_v_f32m2_tumu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2,
size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_v_f32m4_tumu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2,
size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_v_f32m8_tumu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2,
size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexpa_v_f16mf4_mu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2,
size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_v_f16mf2_mu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2,
size_t vl);
vfloat16m1_t __riscv_sf_vfexpa_v_f16m1_mu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2,
size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_v_f16m2_mu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2,
size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_v_f16m4_mu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2,
size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_v_f16m8_mu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2,
size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_v_f32mf2_mu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2,
size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_v_f32m1_mu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2,
size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_v_f32m2_mu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2,
size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_v_f32m4_mu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2,
size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_v_f32m8_mu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2,
size_t vl);
```

Policy overloaded

```
vfloat16mf4_t __riscv_sf_vfexpa_tu(vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexpa_tu(vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexpa_tu(vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_tu(vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_tu(vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_tu(vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_tu(vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexpa_tu(vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_tu(vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_tu(vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_tu(vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
```

Copyright © by SiFive, Inc. All rights reserved.

```
vfloat16mf4_t __riscv_sf_vfexpa_tum(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t
vl);
vfloat16mf2_t __riscv_sf_vfexpa_tum(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t
vl);
vfloat16m1_t __riscv_sf_vfexpa_tum(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_tum(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_tum(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_tum(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_tum(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t
vl);
vfloat32m1_t __riscv_sf_vfexpa_tum(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_tum(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_tum(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_tum(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexpa_tumu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t
vl);
vfloat16mf2_t __riscv_sf_vfexpa_tumu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t
vl);
vfloat16m1_t __riscv_sf_vfexpa_tumu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_tumu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_tumu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_tumu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_tumu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t
vl);
vfloat32m1_t __riscv_sf_vfexpa_tumu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_tumu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_tumu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_tumu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexpa_mu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t
vl);
vfloat16mf2_t __riscv_sf_vfexpa_mu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t
vl);
vfloat16m1_t __riscv_sf_vfexpa_mu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexpa_mu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexpa_mu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexpa_mu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
vfloat32mf2_t __riscv_sf_vfexpa_mu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t
vl);
vfloat32m1_t __riscv_sf_vfexpa_mu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexpa_mu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexpa_mu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexpa_mu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
```

1.3.2 Xsfvfexpa64e

Non-policy non-overloaded

```
vfloat64m1_t __riscv_sf_vfexpa_v_f64m1(vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_v_f64m2(vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_v_f64m4(vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_v_f64m8(vfloat64m8_t vs2, size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_v_f64m1_m(vbool64_t vm, vfloat64m1_t vs2, size_t vl);
```

Copyright © by SiFive, Inc. All rights reserved.

```
vfloat64m2_t __riscv_sf_vfexpa_v_f64m2_m(vbool32_t vm, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_v_f64m4_m(vbool16_t vm, vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_v_f64m8_m(vbool8_t vm, vfloat64m8_t vs2, size_t vl);
```

Non-policy overloaded

```
vfloat64m1_t __riscv_sf_vfexpa(vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa(vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa(vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa(vfloat64m8_t vs2, size_t vl);

// masked functions
vfloat64m1_t __riscv_sf_vfexpa(vbool64_t vm, vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa(vbool32_t vm, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa(vbool16_t vm, vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa(vbool8_t vm, vfloat64m8_t vs2, size_t vl);
```

Policy non-overloaded

```
vfloat64m1_t __riscv_sf_vfexpa_v_f64m1_tu(vfloat64m1_t vd, vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_v_f64m2_tu(vfloat64m2_t vd, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_v_f64m4_tu(vfloat64m4_t vd, vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_v_f64m8_tu(vfloat64m8_t vd, vfloat64m8_t vs2, size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_v_f64m1_tum(vbool64_t vm, vfloat64m1_t vd, vfloat64m1_t vs2,
size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_v_f64m2_tum(vbool32_t vm, vfloat64m2_t vd, vfloat64m2_t vs2,
size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_v_f64m4_tum(vbool16_t vm, vfloat64m4_t vd, vfloat64m4_t vs2,
size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_v_f64m8_tum(vbool8_t vm, vfloat64m8_t vd, vfloat64m8_t vs2,
size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_v_f64m1_tumu(vbool64_t vm, vfloat64m1_t vd, vfloat64m1_t vs2,
size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_v_f64m2_tumu(vbool32_t vm, vfloat64m2_t vd, vfloat64m2_t vs2,
size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_v_f64m4_tumu(vbool16_t vm, vfloat64m4_t vd, vfloat64m4_t vs2,
size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_v_f64m8_tumu(vbool8_t vm, vfloat64m8_t vd, vfloat64m8_t vs2,
size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_v_f64m1_mu(vbool64_t vm, vfloat64m1_t vd, vfloat64m1_t vs2,
size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_v_f64m2_mu(vbool32_t vm, vfloat64m2_t vd, vfloat64m2_t vs2,
size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_v_f64m4_mu(vbool16_t vm, vfloat64m4_t vd, vfloat64m4_t vs2,
size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_v_f64m8_mu(vbool8_t vm, vfloat64m8_t vd, vfloat64m8_t vs2,
size_t vl);
```

Policy overloaded

```
vfloat64m1_t __riscv_sf_vfexpa_tu(vfloat64m1_t vd, vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_tu(vfloat64m2_t vd, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_tu(vfloat64m4_t vd, vfloat64m4_t vs2, size_t vl);
```

Copyright © by SiFive, Inc. All rights reserved.

```
vfloat64m8_t __riscv_sf_vfexpa_tu(vfloat64m8_t vd, vfloat64m8_t vs2, size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_tum(vbool64_t vm, vfloat64m1_t vd, vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_tum(vbool32_t vm, vfloat64m2_t vd, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_tum(vbool16_t vm, vfloat64m4_t vd, vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_tum(vbool8_t vm, vfloat64m8_t vd, vfloat64m8_t vs2, size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_tumu(vbool64_t vm, vfloat64m1_t vd, vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_tumu(vbool32_t vm, vfloat64m2_t vd, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_tumu(vbool16_t vm, vfloat64m4_t vd, vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_tumu(vbool8_t vm, vfloat64m8_t vd, vfloat64m8_t vs2, size_t vl);
// masked functions
vfloat64m1_t __riscv_sf_vfexpa_mu(vbool64_t vm, vfloat64m1_t vd, vfloat64m1_t vs2, size_t vl);
vfloat64m2_t __riscv_sf_vfexpa_mu(vbool32_t vm, vfloat64m2_t vd, vfloat64m2_t vs2, size_t vl);
vfloat64m4_t __riscv_sf_vfexpa_mu(vbool16_t vm, vfloat64m4_t vd, vfloat64m4_t vs2, size_t vl);
vfloat64m8_t __riscv_sf_vfexpa_mu(vbool8_t vm, vfloat64m8_t vd, vfloat64m8_t vs2, size_t vl);
```

Copyright © by SiFive, Inc. All rights reserved.