Xsfvfexp32e, Xsfvfexp16e, Xsfvfbfexp16e Extensions Specification

Version 0.5

Xsfvfexp32e, Xsfvfexp16e, Xsfvfbfexp16e Extensions Specification

Proprietary Notice

Revision History

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

Contents

List of Figures

Xsfvfexp32e, Xsfvfexp16e, Xsfvfbfexp16e Extensions Specification

1.1 Introduction

The exponential function, exp(x), is ubiquitous in machine learning where it appears in non-linear layers including the widespread softmax. Efficient software implementation exist, however their throughput is limited by the vector machine as they often require multiple tens of vector instructions. The extensions specified in this document offer the possibility for implementer to provide a fast approximation of the exponential function with good accuracy (relative error around 4 units in the last place) suitable for a fully pipelined hardware implementation. They cover a wide range of small precisions (16-bit and 32-bit) to address multiple needs.

1.2 Extensions Overview

The extensions listed in this specification make use of the `vtype.altfmt` bit defined in the proposed **Zvfbfa** extension.

This specification defines 3 new extensions:

- `Xsfvfexp32e`: single-width vector exponential function with single precision (binary32) input and output
- `Xsfvfexp16e`: single-width vector exponential function with half precision (binary16) input and output
- `Xsfvfbfexp16e`: single-width vector exponential function with BFloat16 precision input and output

Each of those extensions defines the same instruction: `sf.vfexp.v` under different conditions.

`Xsfvfexp32e` depends on `Zve32f`. `Xsfvfexp16e` depends on `Zvfh`. `Xsfvfbfexp16e` depends on `Zve32f` and either `Zvfbfmin` or `Zvfbfa`.

1.3 Instruction(s)

1.3.1 sf.vfexp.v

Synopsis

Vector Floating-Point Exponential Function Instruction

Mnemonic

sf.vfexp.v vd, vs2, vm

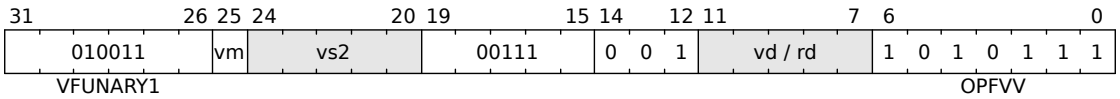

Figure 1: *Encoding (Single-Width Vector)*

The `sf.vfexp.v` instruction is only defined for `SEW=32` if `Xsfvfexp32e` is supported, and `SEW=16` if either `Xsfvfexp16e` or `Xsfvfbfexp16e` is supported.

Reserved Encodings

- `SEW == 32` except when `Xsfvfexp32e` is supported
- `SEW == 16` and `vtype.altfmt == 0` except when `Xsfvfexp16e` is supported
- `SEW == 16` and `vtype.altfmt == 1` except when `Xsfvfbfexp16e` is supported
- any other `SEW` and `vtype.altfmt` combinations

Arguments

Register	Direction	Definition
`vs2`	input	vector of SEW-bit floating-point values
`vd`	output	vector of SEW-bit floating-point exponential function evaluations

Description

`sf.vfexp.v` approximates `e` raised to the input value (a.k.a. exponential function).

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

```
#                         SEW     SEW
sf.vfexp.v  vd, vs2, vm   # vd[i] = exp(vs2[i])
```

Accuracy

`sf.vfexp.v` provides a reduced accuracy approximation with slightly less than 22 bits of accuracy in its valid range for single-precision (`SEW=32`) and slightly more than 9 bits of accuracy for half-precision (`SEW=16` and `vtype.altfmt=0`) and slightly more than 7 bits of accuracy for BFloat16 precision (`SEW=16` and `vtype.altfmt=1`).

Boundary condition

The validity range is reduced to:

- `(-64, 64)` for single and BFloat16 precisions
- `(-8, 8)` for half precision

Any numerical input outside of those range will return `+infinity` if positive and `+0` if negative.

Similarly, a positive infinity input will return `+infinity` and a negative infinity input will return `+0`.

Note

Those values were selected to facilitate input comparison. To allow more valid values, the validity range could be extended to `(-87, 87)` (single and BFloat16 precisions) and `(-9.5,9.5)` (half precision). Those symmetric ranges only contain values for which the exponential result is normal.

The canonical NaN of the output format is returned for any NaN input.

Approximation

Note

The approximation scheme has been designed specifically for the approximation of the exponential function. It exploits the exponential function morphism property: `exp(a + b) = exp(a) * exp(b)`.

The floating-point input `X` is converted to a signed fixed-point representation `fx` with 24-bit fractional part and 9-bit integer part. An approximation to `exp(fx)` is evaluated. `fx` is sliced into 4 segments such that `fx = U.2^-1 + M8 . 2^-8 + M16 . 2^-16 + M24 . 2^-24`.

- `M16` and `M24` are 8-bit unsigned integers
- `M8` is a 7-bit unsigned integer
- `U` is a 9-bit signed integer

Note

the conversion from a sign-magnitude significand to a signed 2's complement value allows to share most of the tables between positive and negative inputs.

The approximation is built using 5 mantissa tables:

- UTP_M: 256-entry table storing 24-bit mantissas
- UTN_M: 256-entry table storing 24-bit mantissas
- TEM1_M8: 128-entry table storing 25-bit mantissas
- TEM1_M16: 256-entry table storing 24-bit mantissas
- TEM1_M24: 256-entry table storing 24-bit mantissas

and 2 exponent tables:

- UTP_E: 256-entry table storing 8-bit exponents

- UTN_E: 256-entry table storing 8-bit exponents

For the most significant input slice, `U`, 2 tables are used, one for positive values of `U` and one for negative values.

The contents of the tables are listed in Section 1.4.

Note
the U tables could be merged into a single table by extracting the sign bit. This would require an extra multiplier to multiply by `exp(2^msb)` or `exp(2^-msb)`.

Note
F[U/S]<x>.<y> is a fractional number format with `x` integer bits (including a sign bit for `FS`) and `y` fractional bits. Either `x` or `y` can be negative to indicate a zero-width integer (resp. fractional) part and a reduced fractional (resp. integer) part. Some examples: `FU32.0` is a 32-bit unsigned integer format, `FS2.8` is a 10-bit signed format with a 8-bit fractional part, `FU-8.16` is a 8-bit format offset by `2^-16` (8-bit fractional part shifted right by 8 bits, divided by 2^-8).

In 16-bit mode (`SEW=16`, regardless of `vtype.altfmt`), the table for the least significant term, `TEM1_M24`, is unused and returns zero when addressed.

Note
Not using the table `TEM1_M24` can also be implemented by forcing the index used to address it to zero.

Note
defining a single approximation scheme and conditioning some part of it on the format size allows to implement a single datapath supporting multiple precisions while still being able to optimize this datapath when only smaller formats are supported.

Note
Table storage optimization: this specification does not cover the many possible optimizations that could be applied to reduce the storage size of the tables (e.g. storing biased exponents). As long as results are numerically identical to the specification, any optimization is allowed.

Note
the fixed-point approximation `HI` can be greater than 2.0 due to the multiplication of multiple terms in [1, 2). This overflow is not possible in the first three steps (involving M24, M16, M8) due to the value of the maximum stored in each table, but can occur in the last fixed-point multiplication. The value can never be greater than 4.0 (or even equal to it).

the following is a pseudo-SAIL description

The approximation is evaluated as:

```
function clause execute (VFEXP(vs2, vd)) = {
  foreach (i from vstart to vl - 1) {
```

```
let X : bits (SEW) = get_velem(vs2, i);
// truncate(EXPR, F) converts EXPR to a fixed-point number with F fractional bits
// (with truncation)
// FIXED(EXPR, I, F) truncates EXPR to I integer bits and F fractional bits
// FIXED_I1F26(EXPR) = FIXED(EXPR, 1, 26)
// isNaN(x, SEW, altfmt) returns true if x is a NaN in the format described by SEW
// and altfmt
// exp(x, SEW, alftfmt) returns the unbiased exponent of x (assuming the format is
// described by SEW and altfmt)
// sign(x, SEW, alftfmt) returns the sign bit of x (assuming the format is described
// by SEW and altfmt)

// returning the format canonical NaN if the input is any NaN
if (isNaN(x, SEW, vtype.altfmt)) {
  if (SEW == 32) set_velem(vd, i, 0x7fc00000);
  else if (SEW == 16 and vtype.altfmt == 0) set_velem(vd, i, 0x7e00);
  else if (SEW == 16 and vtype.altfmt == 1) set_velem(vd, i, 0x7fc0);
  else RESERVED;
  continue;
}
let exp : bits(8) = exp(X, SEW, vtype.altfmt)
let sign : bits(1) = sign(X, SEW, vtype.altfmt)

// Managing overflow and underflow cases
// For infinite inputs, it is assumed that exp(x, SEW, altfmt) returns
// an exponent always larger than the validity bound
if (SEW == 32 && exp >= 6) {
  if (sign) set_velem(vd, i, 0x0);
  else set_velem(vd, i, 0x7f800000);
  continue;
} else if (SEW == 16 && vtype.altfmt == 0 && exp >= 3) {
  if (sign) set_velem(vd, i, 0x0);
  else set_velem(vd, i, 0x7c00);
  continue;
} else if (SEW == 16 && vtype.altfmt == 1 && exp >= 6) {
  if (sign) set_velem(vd, i, 0x0);
  else set_velem(vd, i, 0x7f80);
  continue;
}

ONE = 1.0

// fixed-point conversion and slicing
let fx : bits(32) = truncate(X, 24)
// M24, M16 and M8 are unsigned (respectively 8, 8 and 7-bit) integers
let M24 : bits(8) = if (SEW == 32) then (fx & 0xff) else 0;
let M16 : bits(8) = (fx >> 8) & 0xff
let M8 : bits(7) = (fx >> 16) & 0x7f
// U is a signed 9-bit integer (right shift must be arithmetic here)
let U : bits(9) = (fx >> 24)

let EM1_M24 : bits(24) = TEM1_M24[M24]; // F-16.40
let EM1_M16 : bits(24) = TEM1_M16[M16]; // F-8.32
let EM1_M8  : bits(25) = TEM1_M8[M8];   // F0.25

let LO : bits(27) = FIXED_I1F26(EM1_M24 * EM1_M16 + EM1_M24 + EM1_M16 + ONE) // truncated
```

```
to F1.26
    let ME : bits(27) = FIXED_I1F26((ONE + EM1_M8) * LO) // truncated to F1.26

    let HI : bits(27);
    let exp : bits(8);
    if U < 0 then {
      let E_HI : bits(24) = UTN_M[U & 0xff]; // F1.23 (the table entry LSB is always 0
                                             // and can be discarded)
      HI = FIXED_I1F26(E_HI * ME); // truncated to F1.26
      exp = UTN_E[U & 0xff];
    }
    else {
      let E_HI : bits(24) = UTP_M[U & 0xff]; // F1.23 (the table entry LSB is always 0
                                             // and can be discarded)
      HI = FIXED_I1F26(E_HI * ME); // truncated to F1.26
      exp = UTP_E[U & 0xff];
    }
    // post correction: when the fixed-point value HI exceeds 2
    // it must be divided by 2 and the exponent incremented
    if (HI >= 2) then {
      HI = HI / 2;
      exp = exp + 1;
    }

    if (SEW == 32) then {
      let mant : bits = truncate(HI, 23) & 0x7fffff   // truncate HI to a 24-bit significand
                                                     // and mask the implicit one
      let res : bits (SEW) = (exp + 127) << 23 | mant   // reconstruction: exponent biasing and
                                                       // alignment (no sign needed)
      set_velem(vd, i, res);
    } else if (SEW == 16 and vtype.altfmt == 0) then {
      let mant : bits = truncate(HI, 10) & 0x3ff       // truncate HI to a 11-bit significand
                                                       // and mask the implicit one
      let res : bits (SEW) = (exp + 15) << 10 | mant   // reconstruction: exponent biasing and
                                                       // alignment (no sign needed)
      set_velem(vd, i, res);
    } else if (SEW == 16 and vtype.altfmt == 1) then {
      let mant : bits = truncate(HI, 7) & 0x7f         // truncate HI to a 8-bit significand and
                                                       // mask the implicit one
      let res : bits (SEW) = (exp + 127) << 7 | mant   // reconstruction: exponent biasing and
                                                       // alignment (no sign needed)
      set_velem(vd, i, res);
    } else {
      RESERVED
    }
  }
}
```

1.4 Approximation Tables

The tables' content is listed below as:

- `"h<mant_hex_value>" // [index] = <numerical value>` for `TEM1_M24`, `TEM1_M16`, `TEM1_M8`

- `(exp, "h<mant_hex_value>" // [index] = <numerical value>` for `UTP` and `UTN`

1.4.1 `TEM1_M24` Table

This table stores `exp(index * 2^-24) - 1` for integer `index` in [0, 255].

Note

this table could be further optimized by storing `exp(index * 2^-24) - 1 - index`. This has no impact on the functionality.

```
// {bits=8, offset=5.960464477539063e-08 storeOffset=-1, negInput=False
// # min=0, max=0x1.fe00fep-17, max bit index: -16
"h0", // [0]=0x1p0
"h10000", // [1]=0x1.000001p0
"h20000", // [2]=0x1.000002p0
"h30000", // [3]=0x1.000003p0
"h40000", // [4]=0x1.000004p0
"h50000", // [5]=0x1.000005p0
"h60000", // [6]=0x1.000006p0
"h70000", // [7]=0x1.000007p0
"h80000", // [8]=0x1.000008p0
"h90000", // [9]=0x1.000009p0
"ha0000", // [a]=0x1.00000ap0
"hb0000", // [b]=0x1.00000bp0
"hc0000", // [c]=0x1.00000cp0
"hd0000", // [d]=0x1.00000dp0
"he0000", // [e]=0x1.00000ep0
"hf0000", // [f]=0x1.00000fp0
"h100000", // [10]=0x1.00001p0
"h110001", // [11]=0x1.0000110001p0
"h120001", // [12]=0x1.0000120001p0
"h130001", // [13]=0x1.0000130001p0
"h140001", // [14]=0x1.0000140001p0
"h150001", // [15]=0x1.0000150001p0
"h160001", // [16]=0x1.0000160001p0
"h170001", // [17]=0x1.0000170001p0
"h180001", // [18]=0x1.0000180001p0
"h190001", // [19]=0x1.0000190001p0
"h1a0001", // [1a]=0x1.00001a0001p0
"h1b0001", // [1b]=0x1.00001b0001p0
"h1c0002", // [1c]=0x1.00001c0002p0
"h1d0002", // [1d]=0x1.00001d0002p0
"h1e0002", // [1e]=0x1.00001e0002p0
"h1f0002", // [1f]=0x1.00001f0002p0
"h200002", // [20]=0x1.0000200002p0
"h210002", // [21]=0x1.0000210002p0
"h220002", // [22]=0x1.0000220002p0
"h230002", // [23]=0x1.0000230002p0
"h240002", // [24]=0x1.0000240002p0
"h250003", // [25]=0x1.0000250003p0
"h260003", // [26]=0x1.0000260003p0
"h270003", // [27]=0x1.0000270003p0
"h280003", // [28]=0x1.0000280003p0
"h290003", // [29]=0x1.0000290003p0
```

```
"h2a0004", // [2a]=0x1.00002a0004p0
"h2b0004", // [2b]=0x1.00002b0004p0
"h2c0004", // [2c]=0x1.00002c0004p0
"h2d0004", // [2d]=0x1.00002d0004p0
"h2e0004", // [2e]=0x1.00002e0004p0
"h2f0004", // [2f]=0x1.00002f0004p0
"h300004", // [30]=0x1.0000300004p0
"h310005", // [31]=0x1.0000310005p0
"h320005", // [32]=0x1.0000320005p0
"h330005", // [33]=0x1.0000330005p0
"h340005", // [34]=0x1.0000340005p0
"h350006", // [35]=0x1.0000350006p0
"h360006", // [36]=0x1.0000360006p0
"h370006", // [37]=0x1.0000370006p0
"h380006", // [38]=0x1.0000380006p0
"h390006", // [39]=0x1.0000390006p0
"h3a0006", // [3a]=0x1.00003a0006p0
"h3b0007", // [3b]=0x1.00003b0007p0
"h3c0007", // [3c]=0x1.00003c0007p0
"h3d0007", // [3d]=0x1.00003d0007p0
"h3e0008", // [3e]=0x1.00003e0008p0
"h3f0008", // [3f]=0x1.00003f0008p0
"h400008", // [40]=0x1.0000400008p0
"h410008", // [41]=0x1.0000410008p0
"h420008", // [42]=0x1.0000420008p0
"h430009", // [43]=0x1.0000430009p0
"h440009", // [44]=0x1.0000440009p0
"h45000a", // [45]=0x1.000045000ap0
"h46000a", // [46]=0x1.000046000ap0
"h47000a", // [47]=0x1.000047000ap0
"h48000a", // [48]=0x1.000048000ap0
"h49000a", // [49]=0x1.000049000ap0
"h4a000a", // [4a]=0x1.00004a000ap0
"h4b000b", // [4b]=0x1.00004b000bp0
"h4c000c", // [4c]=0x1.00004c000cp0
"h4d000c", // [4d]=0x1.00004d000cp0
"h4e000c", // [4e]=0x1.00004e000cp0
"h4f000c", // [4f]=0x1.00004f000cp0
"h50000c", // [50]=0x1.000050000cp0
"h51000d", // [51]=0x1.000051000dp0
"h52000d", // [52]=0x1.000052000dp0
"h53000e", // [53]=0x1.000053000ep0
"h54000e", // [54]=0x1.000054000ep0
"h55000e", // [55]=0x1.000055000ep0
"h56000e", // [56]=0x1.000056000ep0
"h57000f", // [57]=0x1.000057000fp0
"h58000f", // [58]=0x1.000058000fp0
"h590010", // [59]=0x1.000059001p0
"h5a0010", // [5a]=0x1.00005a001p0
"h5b0010", // [5b]=0x1.00005b001p0
"h5c0010", // [5c]=0x1.00005c001p0
"h5d0011", // [5d]=0x1.00005d0011p0
"h5e0012", // [5e]=0x1.00005e0012p0
"h5f0012", // [5f]=0x1.00005f0012p0
"h600012", // [60]=0x1.0000600012p0
"h610012", // [61]=0x1.0000610012p0
```

```
"h620013", // [62]=0x1.0000620013p0
"h630013", // [63]=0x1.0000630013p0
"h640014", // [64]=0x1.0000640014p0
"h650014", // [65]=0x1.0000650014p0
"h660014", // [66]=0x1.0000660014p0
"h670014", // [67]=0x1.0000670014p0
"h680015", // [68]=0x1.0000680015p0
"h690016", // [69]=0x1.0000690016p0
"h6a0016", // [6a]=0x1.00006a0016p0
"h6b0016", // [6b]=0x1.00006b0016p0
"h6c0017", // [6c]=0x1.00006c0017p0
"h6d0017", // [6d]=0x1.00006d0017p0
"h6e0018", // [6e]=0x1.00006e0018p0
"h6f0018", // [6f]=0x1.00006f0018p0
"h700018", // [70]=0x1.0000700018p0
"h710019", // [71]=0x1.0000710019p0
"h72001a", // [72]=0x1.000072001ap0
"h73001a", // [73]=0x1.000073001ap0
"h74001a", // [74]=0x1.000074001ap0
"h75001a", // [75]=0x1.000075001ap0
"h76001b", // [76]=0x1.000076001bp0
"h77001c", // [77]=0x1.000077001cp0
"h78001c", // [78]=0x1.000078001cp0
"h79001c", // [79]=0x1.000079001cp0
"h7a001d", // [7a]=0x1.00007a001dp0
"h7b001e", // [7b]=0x1.00007b001ep0
"h7c001e", // [7c]=0x1.00007c001ep0
"h7d001e", // [7d]=0x1.00007d001ep0
"h7e001f", // [7e]=0x1.00007e001fp0
"h7f0020", // [7f]=0x1.00007f002p0
"h800020", // [80]=0x1.000080002p0
"h810021", // [81]=0x1.0000810021p0
"h820021", // [82]=0x1.0000820021p0
"h830022", // [83]=0x1.0000830022p0
"h840022", // [84]=0x1.0000840022p0
"h850023", // [85]=0x1.0000850023p0
"h860023", // [86]=0x1.0000860023p0
"h870024", // [87]=0x1.0000870024p0
"h880024", // [88]=0x1.0000880024p0
"h890025", // [89]=0x1.0000890025p0
"h8a0025", // [8a]=0x1.00008a0025p0
"h8b0026", // [8b]=0x1.00008b0026p0
"h8c0026", // [8c]=0x1.00008c0026p0
"h8d0027", // [8d]=0x1.00008d0027p0
"h8e0027", // [8e]=0x1.00008e0027p0
"h8f0028", // [8f]=0x1.00008f0028p0
"h900029", // [90]=0x1.0000900029p0
"h910029", // [91]=0x1.0000910029p0
"h92002a", // [92]=0x1.000092002ap0
"h93002a", // [93]=0x1.000093002ap0
"h94002b", // [94]=0x1.000094002bp0
"h95002b", // [95]=0x1.000095002bp0
"h96002c", // [96]=0x1.000096002cp0
"h97002d", // [97]=0x1.000097002dp0
"h98002d", // [98]=0x1.000098002dp0
"h99002e", // [99]=0x1.000099002ep0
```

```
"h9a002e", // [9a]=0x1.00009a002ep0
"h9b002f", // [9b]=0x1.00009b002fp0
"h9c0030", // [9c]=0x1.00009c003p0
"h9d0030", // [9d]=0x1.00009d003p0
"h9e0031", // [9e]=0x1.00009e0031p0
"h9f0031", // [9f]=0x1.00009f0031p0
"ha00032", // [a0]=0x1.0000a00032p0
"ha10033", // [a1]=0x1.0000a10033p0
"ha20033", // [a2]=0x1.0000a20033p0
"ha30034", // [a3]=0x1.0000a30034p0
"ha40035", // [a4]=0x1.0000a40035p0
"ha50035", // [a5]=0x1.0000a50035p0
"ha60036", // [a6]=0x1.0000a60036p0
"ha70036", // [a7]=0x1.0000a70036p0
"ha80037", // [a8]=0x1.0000a80037p0
"ha90038", // [a9]=0x1.0000a90038p0
"haa0038", // [aa]=0x1.0000aa0038p0
"hab0039", // [ab]=0x1.0000ab0039p0
"hac003a", // [ac]=0x1.0000ac003ap0
"had003a", // [ad]=0x1.0000ad003ap0
"hae003b", // [ae]=0x1.0000ae003bp0
"haf003c", // [af]=0x1.0000af003cp0
"hb0003d", // [b0]=0x1.0000b0003dp0
"hb1003d", // [b1]=0x1.0000b1003dp0
"hb2003e", // [b2]=0x1.0000b2003ep0
"hb3003f", // [b3]=0x1.0000b3003fp0
"hb4003f", // [b4]=0x1.0000b4003fp0
"hb50040", // [b5]=0x1.0000b5004p0
"hb60041", // [b6]=0x1.0000b60041p0
"hb70041", // [b7]=0x1.0000b70041p0
"hb80042", // [b8]=0x1.0000b80042p0
"hb90043", // [b9]=0x1.0000b90043p0
"hba0044", // [ba]=0x1.0000ba0044p0
"hbb0044", // [bb]=0x1.0000bb0044p0
"hbc0045", // [bc]=0x1.0000bc0045p0
"hbd0046", // [bd]=0x1.0000bd0046p0
"hbe0047", // [be]=0x1.0000be0047p0
"hbf0047", // [bf]=0x1.0000bf0047p0
"hc00048", // [c0]=0x1.0000c00048p0
"hc10049", // [c1]=0x1.0000c10049p0
"hc2004a", // [c2]=0x1.0000c2004ap0
"hc3004a", // [c3]=0x1.0000c3004ap0
"hc4004b", // [c4]=0x1.0000c4004bp0
"hc5004c", // [c5]=0x1.0000c5004cp0
"hc6004d", // [c6]=0x1.0000c6004dp0
"hc7004d", // [c7]=0x1.0000c7004dp0
"hc8004e", // [c8]=0x1.0000c8004ep0
"hc9004f", // [c9]=0x1.0000c9004fp0
"hca0050", // [ca]=0x1.0000ca005p0
"hcb0050", // [cb]=0x1.0000cb005p0
"hcc0051", // [cc]=0x1.0000cc0051p0
"hcd0052", // [cd]=0x1.0000cd0052p0
"hce0053", // [ce]=0x1.0000ce0053p0
"hcf0054", // [cf]=0x1.0000cf0054p0
"hd00055", // [d0]=0x1.0000d00055p0
"hd10055", // [d1]=0x1.0000d10055p0
```

```
"hd20056", // [d2]=0x1.0000d20056p0
"hd30057", // [d3]=0x1.0000d30057p0
"hd40058", // [d4]=0x1.0000d40058p0
"hd50059", // [d5]=0x1.0000d50059p0
"hd60059", // [d6]=0x1.0000d60059p0
"hd7005a", // [d7]=0x1.0000d7005ap0
"hd8005b", // [d8]=0x1.0000d8005bp0
"hd9005c", // [d9]=0x1.0000d9005cp0
"hda005d", // [da]=0x1.0000da005dp0
"hdb005e", // [db]=0x1.0000db005ep0
"hdc005f", // [dc]=0x1.0000dc005fp0
"hdd005f", // [dd]=0x1.0000dd005fp0
"hde0060", // [de]=0x1.0000de006p0
"hdf0061", // [df]=0x1.0000df0061p0
"he00062", // [e0]=0x1.0000e00062p0
"he10063", // [e1]=0x1.0000e10063p0
"he20064", // [e2]=0x1.0000e20064p0
"he30065", // [e3]=0x1.0000e30065p0
"he40066", // [e4]=0x1.0000e40066p0
"he50066", // [e5]=0x1.0000e50066p0
"he60067", // [e6]=0x1.0000e60067p0
"he70068", // [e7]=0x1.0000e70068p0
"he80069", // [e8]=0x1.0000e80069p0
"he9006a", // [e9]=0x1.0000e9006ap0
"hea006b", // [ea]=0x1.0000ea006bp0
"heb006c", // [eb]=0x1.0000eb006cp0
"hec006d", // [ec]=0x1.0000ec006dp0
"hed006e", // [ed]=0x1.0000ed006ep0
"hee006f", // [ee]=0x1.0000ee006fp0
"hef0070", // [ef]=0x1.0000ef007p0
"hf00071", // [f0]=0x1.0000f00071p0
"hf10071", // [f1]=0x1.0000f10071p0
"hf20072", // [f2]=0x1.0000f20072p0
"hf30073", // [f3]=0x1.0000f30073p0
"hf40074", // [f4]=0x1.0000f40074p0
"hf50075", // [f5]=0x1.0000f50075p0
"hf60076", // [f6]=0x1.0000f60076p0
"hf70077", // [f7]=0x1.0000f70077p0
"hf80078", // [f8]=0x1.0000f80078p0
"hf90079", // [f9]=0x1.0000f90079p0
"hfa007a", // [fa]=0x1.0000fa007ap0
"hfb007b", // [fb]=0x1.0000fb007bp0
"hfc007c", // [fc]=0x1.0000fc007cp0
"hfd007d", // [fd]=0x1.0000fd007dp0
"hfe007e", // [fe]=0x1.0000fe007ep0
"hff007f", // [ff]=0x1.0000ff007fp0
```

1.4.2 `TEM1_M16` Table

This table stores `exp(index * 2^-16) - 1` for integer `index` in [0, 255].

```
# table bits=8, offset=1.52587890625e-05 storeOffset=-1, negInput=False
# min=0, max=0x1.fefe56p-9, max bit index: -8
"h0", // [0]=0x1p0
```

```
"h10000", // [1]=0x1.0001p0
"h20002", // [2]=0x1.00020002p0
"h30004", // [3]=0x1.00030004p0
"h40008", // [4]=0x1.00040008p0
"h5000c", // [5]=0x1.0005000cp0
"h60012", // [6]=0x1.00060012p0
"h70018", // [7]=0x1.00070018p0
"h80020", // [8]=0x1.0008002p0
"h90028", // [9]=0x1.00090028p0
"ha0032", // [a]=0x1.000a0032p0
"hb003c", // [b]=0x1.000b003cp0
"hc0048", // [c]=0x1.000c0048p0
"hd0054", // [d]=0x1.000d0054p0
"he0062", // [e]=0x1.000e0062p0
"hf0070", // [f]=0x1.000f007p0
"h100080", // [10]=0x1.0010008p0
"h110090", // [11]=0x1.0011009p0
"h1200a2", // [12]=0x1.001200a2p0
"h1300b4", // [13]=0x1.001300b4p0
"h1400c8", // [14]=0x1.001400c8p0
"h1500dc", // [15]=0x1.001500dcp0
"h1600f2", // [16]=0x1.001600f2p0
"h170108", // [17]=0x1.00170108p0
"h180120", // [18]=0x1.0018012p0
"h190138", // [19]=0x1.00190138p0
"h1a0152", // [1a]=0x1.001a0152p0
"h1b016c", // [1b]=0x1.001b016cp0
"h1c0188", // [1c]=0x1.001c0188p0
"h1d01a4", // [1d]=0x1.001d01a4p0
"h1e01c2", // [1e]=0x1.001e01c2p0
"h1f01e1", // [1f]=0x1.001f01e1p0
"h200200", // [20]=0x1.002002p0
"h210220", // [21]=0x1.0021022p0
"h220242", // [22]=0x1.00220242p0
"h230264", // [23]=0x1.00230264p0
"h240288", // [24]=0x1.00240288p0
"h2502ad", // [25]=0x1.002502adp0
"h2602d2", // [26]=0x1.002602d2p0
"h2702f9", // [27]=0x1.002702f9p0
"h280320", // [28]=0x1.0028032p0
"h290349", // [29]=0x1.00290349p0
"h2a0372", // [2a]=0x1.002a0372p0
"h2b039d", // [2b]=0x1.002b039dp0
"h2c03c8", // [2c]=0x1.002c03c8p0
"h2d03f5", // [2d]=0x1.002d03f5p0
"h2e0422", // [2e]=0x1.002e0422p0
"h2f0451", // [2f]=0x1.002f0451p0
"h300480", // [30]=0x1.0030048p0
"h3104b1", // [31]=0x1.003104b1p0
"h3204e2", // [32]=0x1.003204e2p0
"h330515", // [33]=0x1.00330515p0
"h340548", // [34]=0x1.00340548p0
"h35057d", // [35]=0x1.0035057dp0
"h3605b2", // [36]=0x1.003605b2p0
"h3705e9", // [37]=0x1.003705e9p0
"h380620", // [38]=0x1.0038062p0
```

```
"h390659", // [39]=0x1.00390659p0
"h3a0692", // [3a]=0x1.003a0692p0
"h3b06cd", // [3b]=0x1.003b06cdp0
"h3c0708", // [3c]=0x1.003c0708p0
"h3d0745", // [3d]=0x1.003d0745p0
"h3e0782", // [3e]=0x1.003e0782p0
"h3f07c1", // [3f]=0x1.003f07c1p0
"h400800", // [40]=0x1.004008p0
"h410841", // [41]=0x1.00410841p0
"h420882", // [42]=0x1.00420882p0
"h4308c6", // [43]=0x1.004308c6p0
"h440909", // [44]=0x1.00440909p0
"h45094e", // [45]=0x1.0045094ep0
"h460993", // [46]=0x1.00460993p0
"h4709da", // [47]=0x1.004709dap0
"h480a21", // [48]=0x1.00480a21p0
"h490a6a", // [49]=0x1.00490a6ap0
"h4a0ab3", // [4a]=0x1.004a0ab3p0
"h4b0afe", // [4b]=0x1.004b0afep0
"h4c0b49", // [4c]=0x1.004c0b49p0
"h4d0b96", // [4d]=0x1.004d0b96p0
"h4e0be3", // [4e]=0x1.004e0be3p0
"h4f0c32", // [4f]=0x1.004f0c32p0
"h500c82", // [50]=0x1.00500c82p0
"h510cd2", // [51]=0x1.00510cd2p0
"h520d24", // [52]=0x1.00520d24p0
"h530d76", // [53]=0x1.00530d76p0
"h540dca", // [54]=0x1.00540dcap0
"h550e1e", // [55]=0x1.00550e1ep0
"h560e74", // [56]=0x1.00560e74p0
"h570eca", // [57]=0x1.00570ecap0
"h580f22", // [58]=0x1.00580f22p0
"h590f7a", // [59]=0x1.00590f7ap0
"h5a0fd4", // [5a]=0x1.005a0fd4p0
"h5b102e", // [5b]=0x1.005b102ep0
"h5c108a", // [5c]=0x1.005c108ap0
"h5d10e6", // [5d]=0x1.005d10e6p0
"h5e1144", // [5e]=0x1.005e1144p0
"h5f11a2", // [5f]=0x1.005f11a2p0
"h601202", // [60]=0x1.00601202p0
"h611263", // [61]=0x1.00611263p0
"h6212c4", // [62]=0x1.006212c4p0
"h631327", // [63]=0x1.00631327p0
"h64138a", // [64]=0x1.0064138ap0
"h6513ef", // [65]=0x1.006513efp0
"h661454", // [66]=0x1.00661454p0
"h6714bc", // [67]=0x1.006714bcp0
"h681523", // [68]=0x1.00681523p0
"h69158c", // [69]=0x1.0069158cp0
"h6a15f5", // [6a]=0x1.006a15f5p0
"h6b1660", // [6b]=0x1.006b166p0
"h6c16cb", // [6c]=0x1.006c16cbp0
"h6d1738", // [6d]=0x1.006d1738p0
"h6e17a6", // [6e]=0x1.006e17a6p0
"h6f1814", // [6f]=0x1.006f1814p0
"h701884", // [70]=0x1.00701884p0
```

```
"h7118f4", // [71]=0x1.007118f4p0
"h721966", // [72]=0x1.00721966p0
"h7319d8", // [73]=0x1.007319d8p0
"h741a4c", // [74]=0x1.00741a4cp0
"h751ac0", // [75]=0x1.00751acp0
"h761b36", // [76]=0x1.00761b36p0
"h771bad", // [77]=0x1.00771badp0
"h781c24", // [78]=0x1.00781c24p0
"h791c9d", // [79]=0x1.00791c9dp0
"h7a1d16", // [7a]=0x1.007a1d16p0
"h7b1d91", // [7b]=0x1.007b1d91p0
"h7c1e0d", // [7c]=0x1.007c1e0dp0
"h7d1e8a", // [7d]=0x1.007d1e8ap0
"h7e1f07", // [7e]=0x1.007e1f07p0
"h7f1f86", // [7f]=0x1.007f1f86p0
"h802005", // [80]=0x1.00802005p0
"h812086", // [81]=0x1.00812086p0
"h822108", // [82]=0x1.00822108p0
"h83218a", // [83]=0x1.0083218ap0
"h84220e", // [84]=0x1.0084220ep0
"h852292", // [85]=0x1.00852292p0
"h862318", // [86]=0x1.00862318p0
"h87239f", // [87]=0x1.0087239fp0
"h882426", // [88]=0x1.00882426p0
"h8924af", // [89]=0x1.008924afp0
"h8a2539", // [8a]=0x1.008a2539p0
"h8b25c3", // [8b]=0x1.008b25c3p0
"h8c264f", // [8c]=0x1.008c264fp0
"h8d26dc", // [8d]=0x1.008d26dcp0
"h8e2769", // [8e]=0x1.008e2769p0
"h8f27f8", // [8f]=0x1.008f27f8p0
"h902888", // [90]=0x1.00902888p0
"h912918", // [91]=0x1.00912918p0
"h9229aa", // [92]=0x1.009229aap0
"h932a3d", // [93]=0x1.00932a3dp0
"h942ad0", // [94]=0x1.00942adp0
"h952b65", // [95]=0x1.00952b65p0
"h962bfb", // [96]=0x1.00962bfbp0
"h972c91", // [97]=0x1.00972c91p0
"h982d29", // [98]=0x1.00982d29p0
"h992dc2", // [99]=0x1.00992dc2p0
"h9a2e5b", // [9a]=0x1.009a2e5bp0
"h9b2ef6", // [9b]=0x1.009b2ef6p0
"h9c2f92", // [9c]=0x1.009c2f92p0
"h9d302e", // [9d]=0x1.009d302ep0
"h9e30cc", // [9e]=0x1.009e30ccp0
"h9f316b", // [9f]=0x1.009f316bp0
"ha0320a", // [a0]=0x1.00a0320ap0
"ha132ab", // [a1]=0x1.00a132abp0
"ha2334d", // [a2]=0x1.00a2334dp0
"ha333f0", // [a3]=0x1.00a333fp0
"ha43493", // [a4]=0x1.00a43493p0
"ha53538", // [a5]=0x1.00a53538p0
"ha635de", // [a6]=0x1.00a635dep0
"ha73684", // [a7]=0x1.00a73684p0
"ha8372c", // [a8]=0x1.00a8372cp0
```

```
"ha937d5", // [a9]=0x1.00a937d5p0
"haa387f", // [aa]=0x1.00aa387fp0
"hab3929", // [ab]=0x1.00ab3929p0
"hac39d5", // [ac]=0x1.00ac39d5p0
"had3a82", // [ad]=0x1.00ad3a82p0
"hae3b2f", // [ae]=0x1.00ae3b2fp0
"haf3bde", // [af]=0x1.00af3bdep0
"hb03c8e", // [b0]=0x1.00b03c8ep0
"hb13d3f", // [b1]=0x1.00b13d3fp0
"hb23df0", // [b2]=0x1.00b23dfp0
"hb33ea3", // [b3]=0x1.00b33ea3p0
"hb43f57", // [b4]=0x1.00b43f57p0
"hb5400c", // [b5]=0x1.00b5400cp0
"hb640c1", // [b6]=0x1.00b640c1p0
"hb74178", // [b7]=0x1.00b74178p0
"hb84230", // [b8]=0x1.00b8423p0
"hb942e9", // [b9]=0x1.00b942e9p0
"hba43a2", // [ba]=0x1.00ba43a2p0
"hbb445d", // [bb]=0x1.00bb445dp0
"hbc4519", // [bc]=0x1.00bc4519p0
"hbd45d6", // [bd]=0x1.00bd45d6p0
"hbe4693", // [be]=0x1.00be4693p0
"hbf4752", // [bf]=0x1.00bf4752p0
"hc04812", // [c0]=0x1.00c04812p0
"hc148d3", // [c1]=0x1.00c148d3p0
"hc24995", // [c2]=0x1.00c24995p0
"hc34a57", // [c3]=0x1.00c34a57p0
"hc44b1b", // [c4]=0x1.00c44b1bp0
"hc54be0", // [c5]=0x1.00c54bep0
"hc64ca6", // [c6]=0x1.00c64ca6p0
"hc74d6d", // [c7]=0x1.00c74d6dp0
"hc84e34", // [c8]=0x1.00c84e34p0
"hc94efd", // [c9]=0x1.00c94efdp0
"hca4fc7", // [ca]=0x1.00ca4fc7p0
"hcb5092", // [cb]=0x1.00cb5092p0
"hcc515e", // [cc]=0x1.00cc515ep0
"hcd522a", // [cd]=0x1.00cd522ap0
"hce52f8", // [ce]=0x1.00ce52f8p0
"hcf53c7", // [cf]=0x1.00cf53c7p0
"hd05497", // [d0]=0x1.00d05497p0
"hd15568", // [d1]=0x1.00d15568p0
"hd2563a", // [d2]=0x1.00d2563ap0
"hd3570c", // [d3]=0x1.00d3570cp0
"hd457e0", // [d4]=0x1.00d457ep0
"hd558b5", // [d5]=0x1.00d558b5p0
"hd6598b", // [d6]=0x1.00d6598bp0
"hd75a62", // [d7]=0x1.00d75a62p0
"hd85b3a", // [d8]=0x1.00d85b3ap0
"hd95c13", // [d9]=0x1.00d95c13p0
"hda5cec", // [da]=0x1.00da5cecp0
"hdb5dc7", // [db]=0x1.00db5dc7p0
"hdc5ea3", // [dc]=0x1.00dc5ea3p0
"hdd5f80", // [dd]=0x1.00dd5f8p0
"hde605e", // [de]=0x1.00de605ep0
"hdf613d", // [df]=0x1.00df613dp0
"he0621d", // [e0]=0x1.00e0621dp0
```

```
"he162fd", // [e1]=0x1.00e162fdp0
"he263df", // [e2]=0x1.00e263dfp0
"he364c2", // [e3]=0x1.00e364c2p0
"he465a6", // [e4]=0x1.00e465a6p0
"he5668b", // [e5]=0x1.00e5668bp0
"he66771", // [e6]=0x1.00e66771p0
"he76858", // [e7]=0x1.00e76858p0
"he86940", // [e8]=0x1.00e8694p0
"he96a29", // [e9]=0x1.00e96a29p0
"hea6b13", // [ea]=0x1.00ea6b13p0
"heb6bfe", // [eb]=0x1.00eb6bfep0
"hec6ce9", // [ec]=0x1.00ec6ce9p0
"hed6dd6", // [ed]=0x1.00ed6dd6p0
"hee6ec4", // [ee]=0x1.00ee6ec4p0
"hef6fb3", // [ef]=0x1.00ef6fb3p0
"hf070a3", // [f0]=0x1.00f070a3p0
"hf17194", // [f1]=0x1.00f17194p0
"hf27286", // [f2]=0x1.00f27286p0
"hf37379", // [f3]=0x1.00f37379p0
"hf4746d", // [f4]=0x1.00f4746dp0
"hf57562", // [f5]=0x1.00f57562p0
"hf67658", // [f6]=0x1.00f67658p0
"hf7774f", // [f7]=0x1.00f7774fp0
"hf87847", // [f8]=0x1.00f87847p0
"hf97940", // [f9]=0x1.00f9794p0
"hfa7a3a", // [fa]=0x1.00fa7a3ap0
"hfb7b35", // [fb]=0x1.00fb7b35p0
"hfc7c31", // [fc]=0x1.00fc7c31p0
"hfd7d2e", // [fd]=0x1.00fd7d2ep0
"hfe7e2c", // [fe]=0x1.00fe7e2cp0
"hff7f2b", // [ff]=0x1.00ff7f2bp0
```

1.4.3 `TEM1_M8` Table

This table stores `exp(index * 2^-8) - 1` for integer `index` in [0, 127].

```
# table bits=7, offset=0.00390625 storeOffset=-1, negInput=False
# min=0, max=0x1.48dab2p-1, max bit index: -1
"h0", // [0]=0x1p0
"h20100", // [1]=0x1.01008p0
"h40403", // [2]=0x1.0202018p0
"h60909", // [3]=0x1.0304848p0
"h81015", // [4]=0x1.04080a8p0
"ha192a", // [5]=0x1.050c95p0
"hc2448", // [6]=0x1.061224p0
"he3173", // [7]=0x1.0718b98p0
"h1040ac", // [8]=0x1.082056p0
"h1251f5", // [9]=0x1.0928fa8p0
"h146551", // [a]=0x1.0a32a88p0
"h167ac0", // [b]=0x1.0b3d6p0
"h189247", // [c]=0x1.0c49238p0
"h1aabe6", // [d]=0x1.0d55f3p0
"h1cc79f", // [e]=0x1.0e63cf8p0
"h1ee576", // [f]=0x1.0f72bbp0
```

```
"h21056b", // [10]=0x1.1082b58p0
"h232781", // [11]=0x1.1193c08p0
"h254bbb", // [12]=0x1.12a5dd8p0
"h27721a", // [13]=0x1.13b90dp0
"h299aa0", // [14]=0x1.14cd5p0
"h2bc54f", // [15]=0x1.15e2a78p0
"h2df22b", // [16]=0x1.16f9158p0
"h302134", // [17]=0x1.18109ap0
"h32526e", // [18]=0x1.192937p0
"h3485da", // [19]=0x1.1a42edp0
"h36bb7a", // [1a]=0x1.1b5dbdp0
"h38f352", // [1b]=0x1.1c79a9p0
"h3b2d62", // [1c]=0x1.1d96b1p0
"h3d69ad", // [1d]=0x1.1eb4d68p0
"h3fa836", // [1e]=0x1.1fd41bp0
"h41e8fe", // [1f]=0x1.20f47fp0
"h442c08", // [20]=0x1.221604p0
"h467157", // [21]=0x1.2338ab8p0
"h48b8ec", // [22]=0x1.245c76p0
"h4b02ca", // [23]=0x1.258165p0
"h4d4ef2", // [24]=0x1.26a779p0
"h4f9d68", // [25]=0x1.27ceb4p0
"h51ee2e", // [26]=0x1.28f717p0
"h544146", // [27]=0x1.2a20a3p0
"h5696b2", // [28]=0x1.2b4b59p0
"h58ee74", // [29]=0x1.2c773ap0
"h5b488f", // [2a]=0x1.2da4478p0
"h5da506", // [2b]=0x1.2ed283p0
"h6003da", // [2c]=0x1.3001edp0
"h62650e", // [2d]=0x1.313287p0
"h64c8a5", // [2e]=0x1.3264528p0
"h672ea0", // [2f]=0x1.33975p0
"h699703", // [30]=0x1.34cb818p0
"h6c01cf", // [31]=0x1.3600e78p0
"h6e6f08", // [32]=0x1.373784p0
"h70deae", // [33]=0x1.386f57p0
"h7350c6", // [34]=0x1.39a863p0
"h75c550", // [35]=0x1.3ae2a8p0
"h783c51", // [36]=0x1.3c1e288p0
"h7ab5ca", // [37]=0x1.3d5ae5p0
"h7d31be", // [38]=0x1.3e98dfp0
"h7fb02e", // [39]=0x1.3fd817p0
"h82311f", // [3a]=0x1.41188f8p0
"h84b491", // [3b]=0x1.425a488p0
"h873a88", // [3c]=0x1.439d44p0
"h89c307", // [3d]=0x1.44e1838p0
"h8c4e0f", // [3e]=0x1.4627078p0
"h8edba4", // [3f]=0x1.476dd2p0
"h916bc8", // [40]=0x1.48b5e4p0
"h93fe7c", // [41]=0x1.49ff3ep0
"h9693c5", // [42]=0x1.4b49e28p0
"h992ba5", // [43]=0x1.4c95d28p0
"h9bc61e", // [44]=0x1.4de30fp0
"h9e6332", // [45]=0x1.4f3199p0
"ha102e5", // [46]=0x1.5081728p0
"ha3a539", // [47]=0x1.51d29c8p0
```

```
"ha64a30", // [48]=0x1.532518p0
"ha8f1ce", // [49]=0x1.5478e7p0
"hab9c15", // [4a]=0x1.55ce0a8p0
"hae4907", // [4b]=0x1.5724838p0
"hb0f8a8", // [4c]=0x1.587c54p0
"hb3aaf9", // [4d]=0x1.59d57c8p0
"hb65ffe", // [4e]=0x1.5b2fffp0
"hb917ba", // [4f]=0x1.5c8bddp0
"hbbd22f", // [50]=0x1.5de9178p0
"hbe8f5f", // [51]=0x1.5f47af8p0
"hc14f4e", // [52]=0x1.60a7a7p0
"hc411ff", // [53]=0x1.6208ff8p0
"hc6d773", // [54]=0x1.636bb98p0
"hc99faf", // [55]=0x1.64cfd78p0
"hcc6ab4", // [56]=0x1.66355ap0
"hcf3885", // [57]=0x1.679c428p0
"hd20926", // [58]=0x1.690493p0
"hd4dc98", // [59]=0x1.6a6e4cp0
"hd7b2e0", // [5a]=0x1.6bd97p0
"hda8bff", // [5b]=0x1.6d45ff8p0
"hdd67f9", // [5c]=0x1.6eb3fc8p0
"he046d0", // [5d]=0x1.702368p0
"he32887", // [5e]=0x1.7194438p0
"he60d22", // [5f]=0x1.730691p0
"he8f4a2", // [60]=0x1.747a51p0
"hebdf0c", // [61]=0x1.75ef86p0
"heecc62", // [62]=0x1.776631p0
"hf1bca6", // [63]=0x1.78de53p0
"hf4afdc", // [64]=0x1.7a57eep0
"hf7a607", // [65]=0x1.7bd3038p0
"hfa9f29", // [66]=0x1.7d4f948p0
"hfd9b46", // [67]=0x1.7ecda3p0
"h1009a60", // [68]=0x1.804d3p0
"h1039c7c", // [69]=0x1.81ce3ep0
"h106a19a", // [6a]=0x1.8350cdp0
"h109a9c0", // [6b]=0x1.84d4ep0
"h10cb4ee", // [6c]=0x1.865a77p0
"h10fc32a", // [6d]=0x1.87e195p0
"h112d476", // [6e]=0x1.896a3bp0
"h115e8d4", // [6f]=0x1.8af46ap0
"h1190048", // [70]=0x1.8c8024p0
"h11c1ad6", // [71]=0x1.8e0d6bp0
"h11f3880", // [72]=0x1.8f9c4p0
"h1225948", // [73]=0x1.912ca4p0
"h1257d34", // [74]=0x1.92be9ap0
"h128a444", // [75]=0x1.945222p0
"h12bce7c", // [76]=0x1.95e73ep0
"h12efbe2", // [77]=0x1.977df1p0
"h1322c76", // [78]=0x1.99163bp0
"h135603c", // [79]=0x1.9ab01ep0
"h1389738", // [7a]=0x1.9c4b9cp0
"h13bd16c", // [7b]=0x1.9de8b6p0
"h13f0edc", // [7c]=0x1.9f876ep0
"h1424f8a", // [7d]=0x1.a127c5p0
"h145937c", // [7e]=0x1.a2c9bep0
"h148dab2", // [7f]=0x1.a46d59p0
```

1.4.4 (`UTP_E`, `UTP_M`) Tables

This table stores `exp(index * 2^-1)` for integer `index` in [0, 255]; Each entry is stored as a pair (real exponent, significand). The significand's integer part is always 1.0 and can be omitted in storage.

```
# table bits=8, offset=0.5 storeOffset=0, negInput=False
# min=0, max=0x1.99b988p127, max bit index: 128
(0x0, "h1000000"), // [0]=0x1p0
(0x0, "h1a61298"), // [1]=0x1.a61298p0
(0x1, "h15bf0a8"), // [2]=0x1.5bf0a8p1
(0x2, "h11ed3fe"), // [3]=0x1.1ed3fep2
(0x2, "h1d8e64c"), // [4]=0x1.d8e64cp2
(0x3, "h185d6fe"), // [5]=0x1.85d6fep3
(0x4, "h1415e5c"), // [6]=0x1.415e5cp4
(0x5, "h108ec72"), // [7]=0x1.08ec72p5
(0x5, "h1b4c902"), // [8]=0x1.b4c902p5
(0x6, "h168118a"), // [9]=0x1.68118ap6
(0x7, "h128d38a"), // [a]=0x1.28d38ap7
(0x7, "h1e96244"), // [b]=0x1.e96244p7
(0x8, "h1936dc6"), // [c]=0x1.936dc6p8
(0x9, "h14c9222"), // [d]=0x1.4c9222p9
(0xa, "h1122886"), // [e]=0x1.122886p10
(0xa, "h1c402b6"), // [f]=0x1.c402b6p10
(0xb, "h1749ea8"), // [10]=0x1.749ea8p11
(0xc, "h1332c4e"), // [11]=0x1.332c4ep12
(0xc, "h1fa7158"), // [12]=0x1.fa7158p12
(0xd, "h1a17dd0"), // [13]=0x1.a17ddp13
(0xe, "h15829dc"), // [14]=0x1.5829dcp14
(0xf, "h11bb702"), // [15]=0x1.1bb702p15
(0xf, "h1d3c448"), // [16]=0x1.d3c448p15
(0x10, "h1819bc6"), // [17]=0x1.819bc6p16
(0x11, "h13de166"), // [18]=0x1.3de166p17
(0x12, "h1060c52"), // [19]=0x1.060c52p18
(0x12, "h1b00b5a"), // [1a]=0x1.b00b5ap18
(0x13, "h164290c"), // [1b]=0x1.64290cp19
(0x14, "h1259ac4"), // [1c]=0x1.259ac4p20
(0x14, "h1e41274"), // [1d]=0x1.e41274p20
(0x15, "h18f0cca"), // [1e]=0x1.8f0ccap21
(0x16, "h148f60a"), // [1f]=0x1.48f60ap22
(0x17, "h10f2ebe"), // [20]=0x1.0f2ebep23
(0x17, "h1bf1abe"), // [21]=0x1.bf1abep23
(0x18, "h1709348"), // [22]=0x1.709348p24
(0x19, "h12fd6c8"), // [23]=0x1.2fd6c8p25
(0x19, "h1f4f220"), // [24]=0x1.f4f22p25
(0x1a, "h19cf5c2"), // [25]=0x1.9cf5c2p26
(0x1b, "h1546d90"), // [26]=0x1.546d9p27
(0x1c, "h118a2aa"), // [27]=0x1.18a2aap28
(0x1c, "h1ceb088"), // [28]=0x1.ceb088p28
(0x1d, "h17d6c50"), // [29]=0x1.7d6c5p29
(0x1e, "h13a6e20"), // [2a]=0x1.3a6e2p30
(0x1f, "h1033430"), // [2b]=0x1.03343p31
(0x1f, "h1ab5adc"), // [2c]=0x1.ab5adcp31
(0x20, "h1604b68"), // [2d]=0x1.604b68p32
(0x21, "h1226af4"), // [2e]=0x1.226af4p33
```

```
(0x21, "h1ded166"), // [2f]=0x1.ded166p33
(0x22, "h18ab7fc"), // [30]=0x1.8ab7fcp34
(0x23, "h14563fa"), // [31]=0x1.4563fap35
(0x24, "h10c3d3a"), // [32]=0x1.0c3d3ap36
(0x24, "h1ba4068"), // [33]=0x1.ba4068p36
(0x25, "h16c9326"), // [34]=0x1.6c9326p37
(0x26, "h12c8a86"), // [35]=0x1.2c8a86p38
(0x26, "h1ef8230"), // [36]=0x1.ef823p38
(0x27, "h1987a4c"), // [37]=0x1.987a4cp39
(0x28, "h150bba4"), // [38]=0x1.50bba4p40
(0x29, "h11596e2"), // [39]=0x1.1596e2p41
(0x29, "h1c9aae4"), // [3a]=0x1.c9aae4p41
(0x2a, "h179487a"), // [3b]=0x1.79487ap42
(0x2b, "h1370470"), // [3c]=0x1.37047p43
(0x2c, "h10063f4"), // [3d]=0x1.0063f4p44
(0x2c, "h1a6b766"), // [3e]=0x1.a6b766p44
(0x2d, "h15c7884"), // [3f]=0x1.5c7884p45
(0x2e, "h11f43fc"), // [40]=0x1.1f43fcp46
(0x2e, "h1d99ef2"), // [41]=0x1.d99ef2p46
(0x2f, "h1866f34"), // [42]=0x1.866f34p47
(0x30, "h141dbd6"), // [43]=0x1.41dbd6p48
(0x31, "h10953e2"), // [44]=0x1.0953e2p49
(0x31, "h1b5738e"), // [45]=0x1.b5738ep49
(0x32, "h1689e22"), // [46]=0x1.689e22p50
(0x33, "h1294770"), // [47]=0x1.29477p51
(0x33, "h1ea215a"), // [48]=0x1.ea215ap51
(0x34, "h1940b4a"), // [49]=0x1.940b4ap52
(0x35, "h14d13fc"), // [4a]=0x1.4d13fcp53
(0x36, "h1129392"), // [4b]=0x1.129392p54
(0x36, "h1c4b334"), // [4c]=0x1.c4b334p54
(0x37, "h1753026"), // [4d]=0x1.753026p55
(0x38, "h133a43e"), // [4e]=0x1.33a43ep56
(0x38, "h1fb3716"), // [4f]=0x1.fb3716p56
(0x39, "h1a220d4"), // [50]=0x1.a220d4p57
(0x3a, "h158b03e"), // [51]=0x1.58b03ep58
(0x3b, "h11c25c8"), // [52]=0x1.1c25c8p59
(0x3b, "h1d47aec"), // [53]=0x1.d47aecp59
(0x3c, "h1823256"), // [54]=0x1.823256p60
(0x3d, "h13e5d84"), // [55]=0x1.3e5d84p61
(0x3e, "h10672a4"), // [56]=0x1.0672a4p62
(0x3e, "h1b0b40a"), // [57]=0x1.b0b40ap62
(0x3f, "h164b41c"), // [58]=0x1.64b41cp63
(0x40, "h1260d68"), // [59]=0x1.260d68p64
(0x40, "h1e4cf76"), // [5a]=0x1.e4cf76p64
(0x41, "h18fa89a"), // [5b]=0x1.8fa89ap65
(0x42, "h149767c"), // [5c]=0x1.49767cp66
(0x43, "h10f98a0"), // [5d]=0x1.0f98ap67
(0x43, "h1bfc952"), // [5e]=0x1.bfc952p67
(0x44, "h1712332"), // [5f]=0x1.712332p68
(0x45, "h1304d6a"), // [60]=0x1.304d6ap69
(0x45, "h1f5b5ba"), // [61]=0x1.f5b5bap69
(0x46, "h19d9702"), // [62]=0x1.9d9702p70
(0x47, "h154f27c"), // [63]=0x1.54f27cp71
(0x48, "h119103e"), // [64]=0x1.19103ep72
(0x48, "h1cf6532"), // [65]=0x1.cf6532p72
(0x49, "h17e013c"), // [66]=0x1.7e013cp73
```

```
(0x4a, "h13ae8e6"), // [67]=0x1.3ae8e6p74
(0x4b, "h1039966"), // [68]=0x1.039966p75
(0x4b, "h1ac01b8"), // [69]=0x1.ac01b8p75
(0x4c, "h160d4f6"), // [6a]=0x1.60d4f6p76
(0x4d, "h122dc58"), // [6b]=0x1.22dc58p77
(0x4d, "h1df8c5a"), // [6c]=0x1.df8c5ap77
(0x4e, "h18b521a"), // [6d]=0x1.8b521ap78
(0x4f, "h145e308"), // [6e]=0x1.45e308p79
(0x50, "h10ca5f6"), // [6f]=0x1.0ca5f6p80
(0x50, "h1baed16"), // [70]=0x1.baed16p80
(0x51, "h16d2180"), // [71]=0x1.6d218p81
(0x52, "h12cffe0"), // [72]=0x1.2cffep82
(0x52, "h1f043a8"), // [73]=0x1.f043a8p82
(0x53, "h19919ca"), // [74]=0x1.9919cap83
(0x54, "h1513f1e"), // [75]=0x1.513f1ep84
(0x55, "h1160346"), // [76]=0x1.160346p85
(0x55, "h1ca5d98"), // [77]=0x1.ca5d98p85
(0x56, "h179dbca"), // [78]=0x1.79dbcap86
(0x57, "h1377de0"), // [79]=0x1.377dep87
(0x58, "h100c810"), // [7a]=0x1.00c81p88
(0x58, "h1a75c74"), // [7b]=0x1.a75c74p88
(0x59, "h15d0094"), // [7c]=0x1.5d0094p89
(0x5a, "h11fb426"), // [7d]=0x1.1fb426p90
(0x5a, "h1da57de"), // [7e]=0x1.da57dep90
(0x5b, "h18707a8"), // [7f]=0x1.8707a8p91
(0x5c, "h1425982"), // [80]=0x1.425982p92
(0x5d, "h109bb7c"), // [81]=0x1.09bb7cp93
(0x5d, "h1b61e5c"), // [82]=0x1.b61e5cp93
(0x5e, "h1692af0"), // [83]=0x1.692afp94
(0x5f, "h129bb82"), // [84]=0x1.29bb82p95
(0x5f, "h1eae0ba"), // [85]=0x1.eae0bap95
(0x60, "h194a90e"), // [86]=0x1.94a90ep96
(0x61, "h14d960a"), // [87]=0x1.4d960ap97
(0x62, "h112fec8"), // [88]=0x1.12fec8p98
(0x62, "h1c563f6"), // [89]=0x1.c563f6p98
(0x63, "h175c1dc"), // [8a]=0x1.75c1dcp99
(0x64, "h1341c5c"), // [8b]=0x1.341c5cp100
(0x64, "h1fbfd22"), // [8c]=0x1.fbfd22p100
(0x65, "h1a2c416"), // [8d]=0x1.a2c416p101
(0x66, "h15936d4"), // [8e]=0x1.5936d4p102
(0x67, "h11c94ba"), // [8f]=0x1.1c94bap103
(0x67, "h1d531d8"), // [90]=0x1.d531d8p103
(0x68, "h182c920"), // [91]=0x1.82c92p104
(0x69, "h13ed9d2"), // [92]=0x1.3ed9d2p105
(0x6a, "h106d91e"), // [93]=0x1.06d91ep106
(0x6a, "h1b15cfe"), // [94]=0x1.b15cfep106
(0x6b, "h1653f64"), // [95]=0x1.653f64p107
(0x6c, "h1268038"), // [96]=0x1.268038p108
(0x6c, "h1e58cc2"), // [97]=0x1.e58cc2p108
(0x6d, "h19044a8"), // [98]=0x1.9044a8p109
(0x6e, "h149f720"), // [99]=0x1.49f72p110
(0x6f, "h11002ac"), // [9a]=0x1.1002acp111
(0x6f, "h1c0782a"), // [9b]=0x1.c0782ap111
(0x70, "h171b354"), // [9c]=0x1.71b354p112
(0x71, "h130c43c"), // [9d]=0x1.30c43cp113
(0x71, "h1f6799e"), // [9e]=0x1.f6799ep113
```

```
(0x72, "h19e387e"), // [9f]=0x1.9e387ep114
(0x73, "h155779c"), // [a0]=0x1.55779cp115
(0x74, "h1197dfc"), // [a1]=0x1.197dfcp116
(0x74, "h1d01a22"), // [a2]=0x1.d01a22p116
(0x75, "h17e9664"), // [a3]=0x1.7e9664p117
(0x76, "h13b63da"), // [a4]=0x1.3b63dap118
(0x77, "h103fec2"), // [a5]=0x1.03fec2p119
(0x77, "h1aca8d6"), // [a6]=0x1.aca8d6p119
(0x78, "h1615eba"), // [a7]=0x1.615ebap120
(0x79, "h1234dea"), // [a8]=0x1.234deap121
(0x79, "h1e04798"), // [a9]=0x1.e04798p121
(0x7a, "h18bec76"), // [aa]=0x1.8bec76p122
(0x7b, "h1466246"), // [ab]=0x1.466246p123
(0x7c, "h10d0eda"), // [ac]=0x1.0d0edap124
(0x7c, "h1bb9a08"), // [ad]=0x1.bb9a08p124
(0x7d, "h16db012"), // [ae]=0x1.6db012p125
(0x7e, "h12d7566"), // [af]=0x1.2d7566p126
(0x7e, "h1f1056e"), // [b0]=0x1.f1056ep126
(0x7f, "h199b988"), // [b1]=0x1.99b988p127
(0x0, "h0"), // [b2]=0
(0x0, "h0"), // [b3]=0
(0x0, "h0"), // [b4]=0
(0x0, "h0"), // [b5]=0
(0x0, "h0"), // [b6]=0
(0x0, "h0"), // [b7]=0
(0x0, "h0"), // [b8]=0
(0x0, "h0"), // [b9]=0
(0x0, "h0"), // [ba]=0
(0x0, "h0"), // [bb]=0
(0x0, "h0"), // [bc]=0
(0x0, "h0"), // [bd]=0
(0x0, "h0"), // [be]=0
(0x0, "h0"), // [bf]=0
(0x0, "h0"), // [c0]=0
(0x0, "h0"), // [c1]=0
(0x0, "h0"), // [c2]=0
(0x0, "h0"), // [c3]=0
(0x0, "h0"), // [c4]=0
(0x0, "h0"), // [c5]=0
(0x0, "h0"), // [c6]=0
(0x0, "h0"), // [c7]=0
(0x0, "h0"), // [c8]=0
(0x0, "h0"), // [c9]=0
(0x0, "h0"), // [ca]=0
(0x0, "h0"), // [cb]=0
(0x0, "h0"), // [cc]=0
(0x0, "h0"), // [cd]=0
(0x0, "h0"), // [ce]=0
(0x0, "h0"), // [cf]=0
(0x0, "h0"), // [d0]=0
(0x0, "h0"), // [d1]=0
(0x0, "h0"), // [d2]=0
(0x0, "h0"), // [d3]=0
(0x0, "h0"), // [d4]=0
(0x0, "h0"), // [d5]=0
(0x0, "h0"), // [d6]=0
```

```
(0x0, "h0"), // [d7]=0
(0x0, "h0"), // [d8]=0
(0x0, "h0"), // [d9]=0
(0x0, "h0"), // [da]=0
(0x0, "h0"), // [db]=0
(0x0, "h0"), // [dc]=0
(0x0, "h0"), // [dd]=0
(0x0, "h0"), // [de]=0
(0x0, "h0"), // [df]=0
(0x0, "h0"), // [e0]=0
(0x0, "h0"), // [e1]=0
(0x0, "h0"), // [e2]=0
(0x0, "h0"), // [e3]=0
(0x0, "h0"), // [e4]=0
(0x0, "h0"), // [e5]=0
(0x0, "h0"), // [e6]=0
(0x0, "h0"), // [e7]=0
(0x0, "h0"), // [e8]=0
(0x0, "h0"), // [e9]=0
(0x0, "h0"), // [ea]=0
(0x0, "h0"), // [eb]=0
(0x0, "h0"), // [ec]=0
(0x0, "h0"), // [ed]=0
(0x0, "h0"), // [ee]=0
(0x0, "h0"), // [ef]=0
(0x0, "h0"), // [f0]=0
(0x0, "h0"), // [f1]=0
(0x0, "h0"), // [f2]=0
(0x0, "h0"), // [f3]=0
(0x0, "h0"), // [f4]=0
(0x0, "h0"), // [f5]=0
(0x0, "h0"), // [f6]=0
(0x0, "h0"), // [f7]=0
(0x0, "h0"), // [f8]=0
(0x0, "h0"), // [f9]=0
(0x0, "h0"), // [fa]=0
(0x0, "h0"), // [fb]=0
(0x0, "h0"), // [fc]=0
(0x0, "h0"), // [fd]=0
(0x0, "h0"), // [fe]=0
(0x0, "h0"), // [ff]=0
```

1.4.5 (`UTN_E`, `UTN_M`) Tables

This table stores `exp((-2^8 + index) * 2^-1)` for integer `index` in [0, 255]; Each entry is stored as a pair (real exponent, significand). The significand's integer part is always 1.0 and can be omitted in storage.

```
# table bits=8, offset=0.5 storeOffset=0, negInput=True
# min=0, max=0x1.368b3p-1, max bit index: -1
(0x0, "h0"), // [0]=0
(0x0, "h0"), // [1]=0
(0x0, "h0"), // [2]=0
(0x0, "h0"), // [3]=0
```

```
(0x0, "h0"), // [4]=0
(0x0, "h0"), // [5]=0
(0x0, "h0"), // [6]=0
(0x0, "h0"), // [7]=0
(0x0, "h0"), // [8]=0
(0x0, "h0"), // [9]=0
(0x0, "h0"), // [a]=0
(0x0, "h0"), // [b]=0
(0x0, "h0"), // [c]=0
(0x0, "h0"), // [d]=0
(0x0, "h0"), // [e]=0
(0x0, "h0"), // [f]=0
(0x0, "h0"), // [10]=0
(0x0, "h0"), // [11]=0
(0x0, "h0"), // [12]=0
(0x0, "h0"), // [13]=0
(0x0, "h0"), // [14]=0
(0x0, "h0"), // [15]=0
(0x0, "h0"), // [16]=0
(0x0, "h0"), // [17]=0
(0x0, "h0"), // [18]=0
(0x0, "h0"), // [19]=0
(0x0, "h0"), // [1a]=0
(0x0, "h0"), // [1b]=0
(0x0, "h0"), // [1c]=0
(0x0, "h0"), // [1d]=0
(0x0, "h0"), // [1e]=0
(0x0, "h0"), // [1f]=0
(0x0, "h0"), // [20]=0
(0x0, "h0"), // [21]=0
(0x0, "h0"), // [22]=0
(0x0, "h0"), // [23]=0
(0x0, "h0"), // [24]=0
(0x0, "h0"), // [25]=0
(0x0, "h0"), // [26]=0
(0x0, "h0"), // [27]=0
(0x0, "h0"), // [28]=0
(0x0, "h0"), // [29]=0
(0x0, "h0"), // [2a]=0
(0x0, "h0"), // [2b]=0
(0x0, "h0"), // [2c]=0
(0x0, "h0"), // [2d]=0
(0x0, "h0"), // [2e]=0
(0x0, "h0"), // [2f]=0
(0x0, "h0"), // [30]=0
(-0x95, "h1000000"), // [31]=0x1p-149
(-0x95, "h1000000"), // [32]=0x1p-149
(-0x94, "h1000000"), // [33]=0x1p-148
(-0x93, "h1000000"), // [34]=0x1p-147
(-0x93, "h1800000"), // [35]=0x1.8p-147
(-0x92, "h1400000"), // [36]=0x1.4p-146
(-0x91, "h1000000"), // [37]=0x1p-145
(-0x91, "h1b00000"), // [38]=0x1.bp-145
(-0x90, "h1600000"), // [39]=0x1.6p-144
(-0x8f, "h1200000"), // [3a]=0x1.2p-143
(-0x8f, "h1dc0000"), // [3b]=0x1.dcp-143
```

```
(-0x8e, "h1880000"), // [3c]=0x1.88p-142
(-0x8d, "h1430000"), // [3d]=0x1.43p-141
(-0x8c, "h10a8000"), // [3e]=0x1.0a8p-140
(-0x8c, "h1b78000"), // [3f]=0x1.b78p-140
(-0x8b, "h16a4000"), // [40]=0x1.6a4p-139
(-0x8a, "h12ac000"), // [41]=0x1.2acp-138
(-0x8a, "h1ec8000"), // [42]=0x1.ec8p-138
(-0x89, "h1960000"), // [43]=0x1.96p-137
(-0x88, "h14eb000"), // [44]=0x1.4ebp-136
(-0x87, "h113e800"), // [45]=0x1.13e8p-135
(-0x87, "h1c6e400"), // [46]=0x1.c6e4p-135
(-0x86, "h176fe00"), // [47]=0x1.76fep-134
(-0x85, "h1352100"), // [48]=0x1.3521p-133
(-0x85, "h1fdaa00"), // [49]=0x1.fdaap-133
(-0x84, "h1a42580"), // [4a]=0x1.a4258p-132
(-0x83, "h15a5a40"), // [4b]=0x1.5a5a4p-131
(-0x82, "h11d8500"), // [4c]=0x1.1d85p-130
(-0x82, "h1d6be00"), // [4d]=0x1.d6bep-130
(-0x81, "h1840fc0"), // [4e]=0x1.840fcp-129
(-0x80, "h13fe710"), // [4f]=0x1.3fe71p-128
(-0x7f, "h107b710"), // [50]=0x1.07b71p-127
(-0x7f, "h1b2caf0"), // [51]=0x1.b2cafp-127
(-0x7e, "h1666d0e"), // [52]=0x1.666d0ep-126
(-0x7d, "h12778e8"), // [53]=0x1.2778e8p-125
(-0x7d, "h1e726c4"), // [54]=0x1.e726c4p-125
(-0x7c, "h19196a6"), // [55]=0x1.9196a6p-124
(-0x7b, "h14b0dc0"), // [56]=0x1.4b0dcp-123
(-0x7a, "h110e85c"), // [57]=0x1.10e85cp-122
(-0x7a, "h1c1f2da"), // [58]=0x1.c1f2dap-122
(-0x79, "h172eb82"), // [59]=0x1.72eb82p-121
(-0x78, "h131c596"), // [5a]=0x1.31c596p-120
(-0x78, "h1f821ea"), // [5b]=0x1.f821eap-120
(-0x77, "h19f9644"), // [5c]=0x1.9f9644p-119
(-0x76, "h15697f2"), // [5d]=0x1.5697f2p-118
(-0x75, "h11a6bae"), // [5e]=0x1.1a6baep-117
(-0x75, "h1d1a206"), // [5f]=0x1.d1a206p-117
(-0x74, "h17fd974"), // [60]=0x1.7fd974p-116
(-0x73, "h13c6e2c"), // [61]=0x1.3c6e2cp-115
(-0x72, "h104da4e"), // [62]=0x1.04da4ep-114
(-0x72, "h1ae12ce"), // [63]=0x1.ae12cep-114
(-0x71, "h1628920"), // [64]=0x1.62892p-113
(-0x70, "h12443e6"), // [65]=0x1.2443e6p-112
(-0x70, "h1e1dd28"), // [66]=0x1.e1dd28p-112
(-0x6f, "h18d3ac8"), // [67]=0x1.8d3ac8p-111
(-0x6e, "h14775e0"), // [68]=0x1.4775ep-110
(-0x6d, "h10df20c"), // [69]=0x1.0df20cp-109
(-0x6d, "h1bd109e"), // [6a]=0x1.bd109ep-109
(-0x6c, "h16ee4dc"), // [6b]=0x1.6ee4dcp-108
(-0x6b, "h12e73f6"), // [6c]=0x1.2e73f6p-107
(-0x6b, "h1f2a920"), // [6d]=0x1.f2a92p-107
(-0x6a, "h19b1382"), // [6e]=0x1.9b1382p-106
(-0x69, "h152e002"), // [6f]=0x1.52e002p-105
(-0x68, "h1175af0"), // [70]=0x1.175afp-104
(-0x68, "h1cc9434"), // [71]=0x1.cc9434p-104
(-0x67, "h17baee2"), // [72]=0x1.7baee2p-103
(-0x66, "h138feee"), // [73]=0x1.38feeep-102
```

```
(-0x65, "h102057e"), // [74]=0x1.02057ep-101
(-0x65, "h1a967ca"), // [75]=0x1.a967cap-101
(-0x64, "h15eb000"), // [76]=0x1.5ebp-100
(-0x63, "h12117cc"), // [77]=0x1.2117ccp-99
(-0x63, "h1dca23c"), // [78]=0x1.dca23cp-99
(-0x62, "h188eb08"), // [79]=0x1.88eb08p-98
(-0x61, "h143e7fc"), // [7a]=0x1.43e7fcp-97
(-0x60, "h10b03fa"), // [7b]=0x1.0b03fap-96
(-0x60, "h1b83bf2"), // [7c]=0x1.b83bf2p-96
(-0x5f, "h16ae966"), // [7d]=0x1.6ae966p-95
(-0x5e, "h12b2b8e"), // [7e]=0x1.2b2b8ep-94
(-0x5e, "h1ed3f88"), // [7f]=0x1.ed3f88p-94
(-0x5d, "h1969d48"), // [80]=0x1.969d48p-93
(-0x5c, "h14f3266"), // [81]=0x1.4f3266p-92
(-0x5b, "h11452b8"), // [82]=0x1.1452b8p-91
(-0x5b, "h1c7946e"), // [83]=0x1.c7946ep-91
(-0x5a, "h1778fe2"), // [84]=0x1.778fe2p-90
(-0x59, "h135993c"), // [85]=0x1.35993cp-89
(-0x59, "h1fe7116"), // [86]=0x1.fe7116p-89
(-0x58, "h1a4c9c0"), // [87]=0x1.a4c9cp-88
(-0x57, "h15ae192"), // [88]=0x1.5ae192p-87
(-0x56, "h11df484"), // [89]=0x1.1df484p-86
(-0x56, "h1d775d8"), // [8a]=0x1.d775d8p-86
(-0x55, "h184a742"), // [8b]=0x1.84a742p-85
(-0x54, "h14063f8"), // [8c]=0x1.4063f8p-84
(-0x53, "h1081e0a"), // [8d]=0x1.081e0ap-83
(-0x53, "h1b374b4"), // [8e]=0x1.b374b4p-83
(-0x52, "h166f900"), // [8f]=0x1.66f9p-82
(-0x51, "h127ec46"), // [90]=0x1.27ec46p-81
(-0x51, "h1e7e4fa"), // [91]=0x1.e7e4fap-81
(-0x50, "h1923372"), // [92]=0x1.923372p-80
(-0x4f, "h14b8f04"), // [93]=0x1.4b8f04p-79
(-0x4e, "h11152ea"), // [94]=0x1.1152eap-78
(-0x4e, "h1c2a28a"), // [95]=0x1.c2a28ap-78
(-0x4d, "h1737c56"), // [96]=0x1.737c56p-77
(-0x4c, "h1323cfa"), // [97]=0x1.323cfap-76
(-0x4c, "h1f8e6c2"), // [98]=0x1.f8e6c2p-76
(-0x4b, "h1a0388a"), // [99]=0x1.a0388ap-75
(-0x4a, "h1571db8"), // [9a]=0x1.571db8p-74
(-0x49, "h11ad9f4"), // [9b]=0x1.1ad9f4p-73
(-0x49, "h1d257d6"), // [9c]=0x1.d257d6p-73
(-0x48, "h1806f56"), // [9d]=0x1.806f56p-72
(-0x47, "h13ce9ba"), // [9e]=0x1.3ce9bap-71
(-0x46, "h1054028"), // [9f]=0x1.054028p-70
(-0x46, "h1aebaba"), // [a0]=0x1.aebabap-70
(-0x45, "h163138e"), // [a1]=0x1.63138ep-69
(-0x44, "h124b604"), // [a2]=0x1.24b604p-68
(-0x44, "h1e2994c"), // [a3]=0x1.e2994cp-68
(-0x43, "h18dd5e2"), // [a4]=0x1.8dd5e2p-67
(-0x42, "h147f5bc"), // [a5]=0x1.47f5bcp-66
(-0x41, "h10e5b74"), // [a6]=0x1.0e5b74p-65
(-0x41, "h1bdbe64"), // [a7]=0x1.bdbe64p-65
(-0x40, "h16f741e"), // [a8]=0x1.6f741ep-64
(-0x3f, "h12eea0e"), // [a9]=0x1.2eea0ep-63
(-0x3f, "h1f36bd4"), // [aa]=0x1.f36bd4p-63
(-0x3e, "h19bb404"), // [ab]=0x1.9bb404p-62
```

```
(-0x3d, "h1536452"), // [ac]=0x1.536452p-61
(-0x3c, "h117c804"), // [ad]=0x1.17c804p-60
(-0x3c, "h1cd480a"), // [ae]=0x1.cd480ap-60
(-0x3b, "h17c4322"), // [af]=0x1.7c4322p-59
(-0x3a, "h1397924"), // [b0]=0x1.397924p-58
(-0x39, "h1026a3c"), // [b1]=0x1.026a3cp-57
(-0x39, "h1aa0de4"), // [b2]=0x1.aa0de4p-57
(-0x38, "h15f38ee"), // [b3]=0x1.5f38eep-56
(-0x37, "h12188ae"), // [b4]=0x1.2188aep-55
(-0x37, "h1dd5c56"), // [b5]=0x1.dd5c56p-55
(-0x36, "h1898472"), // [b6]=0x1.898472p-54
(-0x35, "h1446676"), // [b7]=0x1.446676p-53
(-0x34, "h10b6c3a"), // [b8]=0x1.0b6c3ap-52
(-0x34, "h1b8e7d6"), // [b9]=0x1.b8e7d6p-52
(-0x33, "h16b771a"), // [ba]=0x1.6b771ap-51
(-0x32, "h12ba05e"), // [bb]=0x1.2ba05ep-50
(-0x32, "h1ee001e"), // [bc]=0x1.ee001ep-50
(-0x31, "h1973c0c"), // [bd]=0x1.973c0cp-49
(-0x30, "h14fb548"), // [be]=0x1.4fb548p-48
(-0x2f, "h114be9c"), // [bf]=0x1.14be9cp-47
(-0x2f, "h1c84650"), // [c0]=0x1.c8465p-47
(-0x2e, "h1782286"), // [c1]=0x1.782286p-46
(-0x2d, "h136121e"), // [c2]=0x1.36121ep-45
(-0x2d, "h1ff3864"), // [c3]=0x1.ff3864p-45
(-0x2c, "h1a56e0c"), // [c4]=0x1.a56e0cp-44
(-0x2b, "h15b6902"), // [c5]=0x1.5b6902p-43
(-0x2a, "h11e642c"), // [c6]=0x1.1e642cp-42
(-0x2a, "h1d82dee"), // [c7]=0x1.d82deep-42
(-0x29, "h1853f02"), // [c8]=0x1.853f02p-41
(-0x28, "h140e112"), // [c9]=0x1.40e112p-40
(-0x27, "h108852a"), // [ca]=0x1.08852ap-39
(-0x27, "h1b41eba"), // [cb]=0x1.b41ebap-39
(-0x26, "h167852a"), // [cc]=0x1.67852ap-38
(-0x25, "h1285fd2"), // [cd]=0x1.285fd2p-37
(-0x25, "h1e8a37a"), // [ce]=0x1.e8a37ap-37
(-0x24, "h192d07e"), // [cf]=0x1.92d07ep-36
(-0x23, "h14c1078"), // [d0]=0x1.4c1078p-35
(-0x22, "h111bda4"), // [d1]=0x1.11bda4p-34
(-0x22, "h1c3527e"), // [d2]=0x1.c3527ep-34
(-0x21, "h1740d62"), // [d3]=0x1.740d62p-33
(-0x20, "h132b48c"), // [d4]=0x1.32b48cp-32
(-0x20, "h1f9abe6"), // [d5]=0x1.f9abe6p-32
(-0x1f, "h1a0db0e"), // [d6]=0x1.a0db0ep-31
(-0x1e, "h157a3b0"), // [d7]=0x1.57a3bp-30
(-0x1d, "h11b4866"), // [d8]=0x1.1b4866p-29
(-0x1d, "h1d30dec"), // [d9]=0x1.d30decp-29
(-0x1c, "h1810570"), // [da]=0x1.81057p-28
(-0x1b, "h13d6578"), // [db]=0x1.3d6578p-27
(-0x1a, "h105a628"), // [dc]=0x1.05a628p-26
(-0x1a, "h1af62ea"), // [dd]=0x1.af62eap-26
(-0x19, "h1639e32"), // [de]=0x1.639e32p-25
(-0x18, "h125284e"), // [df]=0x1.25284ep-24
(-0x18, "h1e355bc"), // [e0]=0x1.e355bcp-24
(-0x17, "h18e7138"), // [e1]=0x1.8e7138p-23
(-0x16, "h14875ca"), // [e2]=0x1.4875cap-22
(-0x15, "h10ec504"), // [e3]=0x1.0ec504p-21
```

```
(-0x15, "h1be6c70"), // [e4]=0x1.be6c7p-21
(-0x14, "h1700398"), // [e5]=0x1.700398p-20
(-0x13, "h12f6054"), // [e6]=0x1.2f6054p-19
(-0x13, "h1f42ed4"), // [e7]=0x1.f42ed4p-19
(-0x12, "h19c54c4"), // [e8]=0x1.9c54c4p-18
(-0x11, "h153e8d8"), // [e9]=0x1.53e8d8p-17
(-0x10, "h1183542"), // [ea]=0x1.183542p-16
(-0x10, "h1cdfc26"), // [eb]=0x1.cdfc26p-16
(-0xf, "h17cd79c"), // [ec]=0x1.7cd79cp-15
(-0xe, "h139f38a"), // [ed]=0x1.39f38ap-14
(-0xd, "h102cf22"), // [ee]=0x1.02cf22p-13
(-0xd, "h1aab440"), // [ef]=0x1.aab44p-13
(-0xc, "h15fc210"), // [f0]=0x1.5fc21p-12
(-0xb, "h121f9ba"), // [f1]=0x1.21f9bap-11
(-0xb, "h1de16ba"), // [f2]=0x1.de16bap-11
(-0xa, "h18a1e18"), // [f3]=0x1.8a1e18p-10
(-0x9, "h144e520"), // [f4]=0x1.44e52p-9
(-0x8, "h10bd4a6"), // [f5]=0x1.0bd4a6p-8
(-0x8, "h1b993fe"), // [f6]=0x1.b993fep-8
(-0x7, "h16c0504"), // [f7]=0x1.6c0504p-7
(-0x6, "h12c155c"), // [f8]=0x1.2c155cp-6
(-0x6, "h1eec102"), // [f9]=0x1.eec102p-6
(-0x5, "h197db0c"), // [fa]=0x1.97db0cp-5
(-0x4, "h150385c"), // [fb]=0x1.50385cp-4
(-0x3, "h1152aaa"), // [fc]=0x1.152aaap-3
(-0x3, "h1c8f878"), // [fd]=0x1.c8f878p-3
(-0x2, "h178b564"), // [fe]=0x1.78b564p-2
(-0x1, "h1368b30"), // [ff]=0x1.368b3p-1
```

1.5 Intrinsics

1.5.1 Xsfvfexp32e

Non-policy non-overloaded

```
vfloat32mf2_t __riscv_sf_vfexp_v_f32mf2(vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp_v_f32m1(vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_v_f32m2(vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp_v_f32m4(vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_v_f32m8(vfloat32m8_t vs2, size_t vl);

// masked functions
vfloat32mf2_t __riscv_sf_vfexp_v_f32mf2_m(vbool64_t vm, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp_v_f32m1_m(vbool32_t vm, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_v_f32m2_m(vbool16_t vm, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp_v_f32m4_m(vbool8_t vm, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_v_f32m8_m(vbool4_t vm, vfloat32m8_t vs2, size_t vl);
```

Non-policy overloaded

```
vfloat32mf2_t __riscv_sf_vfexp(vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp(vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp(vfloat32m2_t vs2, size_t vl);
```

```
vfloat32m4_t __riscv_sf_vfexp(vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp(vfloat32m8_t vs2, size_t vl);

// masked functions
vfloat32mf2_t __riscv_sf_vfexp(vbool64_t vm, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp(vbool32_t vm, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp(vbool16_t vm, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp(vbool8_t vm, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp(vbool4_t vm, vfloat32m8_t vs2, size_t vl);
```

Policy non-overloaded

```
vfloat32mf2_t __riscv_sf_vfexp_v_f32mf2_tu(vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp_v_f32m1_tu(vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_v_f32m2_tu(vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp_v_f32m4_tu(vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_v_f32m8_tu(vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat32mf2_t __riscv_sf_vfexp_v_f32mf2_tum(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2,
size_t vl);
vfloat32m1_t __riscv_sf_vfexp_v_f32m1_tum(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2,
size_t vl);
vfloat32m2_t __riscv_sf_vfexp_v_f32m2_tum(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2,
size_t vl);
vfloat32m4_t __riscv_sf_vfexp_v_f32m4_tum(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2,
size_t vl);
vfloat32m8_t __riscv_sf_vfexp_v_f32m8_tum(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2,
size_t vl);
// masked functions
vfloat32mf2_t __riscv_sf_vfexp_v_f32mf2_tumu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2,
size_t vl);
vfloat32m1_t __riscv_sf_vfexp_v_f32m1_tumu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2,
size_t vl);
vfloat32m2_t __riscv_sf_vfexp_v_f32m2_tumu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2,
size_t vl);
vfloat32m4_t __riscv_sf_vfexp_v_f32m4_tumu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2,
size_t vl);
vfloat32m8_t __riscv_sf_vfexp_v_f32m8_tumu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2,
size_t vl);
// masked functions
vfloat32mf2_t __riscv_sf_vfexp_v_f32mf2_mu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2,
size_t vl);
vfloat32m1_t __riscv_sf_vfexp_v_f32m1_mu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2,
size_t vl);
vfloat32m2_t __riscv_sf_vfexp_v_f32m2_mu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2,
size_t vl);
vfloat32m4_t __riscv_sf_vfexp_v_f32m4_mu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t
vl);
vfloat32m8_t __riscv_sf_vfexp_v_f32m8_mu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t
vl);
```

Policy overloaded

```
vfloat32mf2_t __riscv_sf_vfexp_tu(vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp_tu(vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_tu(vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
```

```
vfloat32m4_t __riscv_sf_vfexp_tu(vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_tu(vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat32mf2_t __riscv_sf_vfexp_tum(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t
vl);
vfloat32m1_t __riscv_sf_vfexp_tum(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_tum(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp_tum(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_tum(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat32mf2_t __riscv_sf_vfexp_tumu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t
vl);
vfloat32m1_t __riscv_sf_vfexp_tumu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_tumu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp_tumu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_tumu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
// masked functions
vfloat32mf2_t __riscv_sf_vfexp_mu(vbool64_t vm, vfloat32mf2_t vd, vfloat32mf2_t vs2, size_t vl);
vfloat32m1_t __riscv_sf_vfexp_mu(vbool32_t vm, vfloat32m1_t vd, vfloat32m1_t vs2, size_t vl);
vfloat32m2_t __riscv_sf_vfexp_mu(vbool16_t vm, vfloat32m2_t vd, vfloat32m2_t vs2, size_t vl);
vfloat32m4_t __riscv_sf_vfexp_mu(vbool8_t vm, vfloat32m4_t vd, vfloat32m4_t vs2, size_t vl);
vfloat32m8_t __riscv_sf_vfexp_mu(vbool4_t vm, vfloat32m8_t vd, vfloat32m8_t vs2, size_t vl);
```

1.5.2 Xsfvfexp16e

Non-policy non-overloaded

```
vfloat16mf4_t __riscv_sf_vfexp_v_f16mf4(vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_v_f16mf2(vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp_v_f16m1(vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_v_f16m2(vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp_v_f16m4(vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_v_f16m8(vfloat16m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_v_f16mf4_m(vbool64_t vm, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_v_f16mf2_m(vbool32_t vm, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp_v_f16m1_m(vbool16_t vm, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_v_f16m2_m(vbool8_t vm, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp_v_f16m4_m(vbool4_t vm, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_v_f16m8_m(vbool2_t vm, vfloat16m8_t vs2, size_t vl);
```

Non-policy overloaded

```
vfloat16mf4_t __riscv_sf_vfexp(vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp(vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp(vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp(vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp(vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp(vfloat16m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp(vbool64_t vm, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp(vbool32_t vm, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp(vbool16_t vm, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp(vbool8_t vm, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp(vbool4_t vm, vfloat16m4_t vs2, size_t vl);
```

```
vfloat16m8_t __riscv_sf_vfexp(vbool2_t vm, vfloat16m8_t vs2, size_t vl);
```

Policy non-overloaded

```
vfloat16mf4_t __riscv_sf_vfexp_v_f16mf4_tu(vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_v_f16mf2_tu(vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp_v_f16m1_tu(vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_v_f16m2_tu(vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp_v_f16m4_tu(vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_v_f16m8_tu(vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_v_f16mf4_tum(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2,
size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_v_f16mf2_tum(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2,
size_t vl);
vfloat16m1_t __riscv_sf_vfexp_v_f16m1_tum(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2,
size_t vl);
vfloat16m2_t __riscv_sf_vfexp_v_f16m2_tum(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2,
size_t vl);
vfloat16m4_t __riscv_sf_vfexp_v_f16m4_tum(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2,
size_t vl);
vfloat16m8_t __riscv_sf_vfexp_v_f16m8_tum(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2,
size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_v_f16mf4_tumu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2,
size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_v_f16mf2_tumu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2,
size_t vl);
vfloat16m1_t __riscv_sf_vfexp_v_f16m1_tumu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2,
size_t vl);
vfloat16m2_t __riscv_sf_vfexp_v_f16m2_tumu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2,
size_t vl);
vfloat16m4_t __riscv_sf_vfexp_v_f16m4_tumu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2,
size_t vl);
vfloat16m8_t __riscv_sf_vfexp_v_f16m8_tumu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2,
size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_v_f16mf4_mu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2,
size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_v_f16mf2_mu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2,
size_t vl);
vfloat16m1_t __riscv_sf_vfexp_v_f16m1_mu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2,
size_t vl);
vfloat16m2_t __riscv_sf_vfexp_v_f16m2_mu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t
vl);
vfloat16m4_t __riscv_sf_vfexp_v_f16m4_mu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t
vl);
vfloat16m8_t __riscv_sf_vfexp_v_f16m8_mu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t
vl);
```

Policy overloaded

```
vfloat16mf4_t __riscv_sf_vfexp_tu(vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_tu(vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp_tu(vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_tu(vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
```

```
vfloat16m4_t __riscv_sf_vfexp_tu(vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_tu(vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_tum(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t
vl);
vfloat16mf2_t __riscv_sf_vfexp_tum(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t
vl);
vfloat16m1_t __riscv_sf_vfexp_tum(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_tum(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp_tum(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_tum(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_tumu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t
vl);
vfloat16mf2_t __riscv_sf_vfexp_tumu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t
vl);
vfloat16m1_t __riscv_sf_vfexp_tumu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_tumu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp_tumu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_tumu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
// masked functions
vfloat16mf4_t __riscv_sf_vfexp_mu(vbool64_t vm, vfloat16mf4_t vd, vfloat16mf4_t vs2, size_t vl);
vfloat16mf2_t __riscv_sf_vfexp_mu(vbool32_t vm, vfloat16mf2_t vd, vfloat16mf2_t vs2, size_t vl);
vfloat16m1_t __riscv_sf_vfexp_mu(vbool16_t vm, vfloat16m1_t vd, vfloat16m1_t vs2, size_t vl);
vfloat16m2_t __riscv_sf_vfexp_mu(vbool8_t vm, vfloat16m2_t vd, vfloat16m2_t vs2, size_t vl);
vfloat16m4_t __riscv_sf_vfexp_mu(vbool4_t vm, vfloat16m4_t vd, vfloat16m4_t vs2, size_t vl);
vfloat16m8_t __riscv_sf_vfexp_mu(vbool2_t vm, vfloat16m8_t vd, vfloat16m8_t vs2, size_t vl);
```

1.5.3 Xsfvfbfexp16e

Non-policy non-overloaded

```
vbfloat16mf4_t __riscv_sf_vfexp_v_bf16mf4(vbfloat16mf4_t vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_v_bf16mf2(vbfloat16mf2_t vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_v_bf16m1(vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_v_bf16m2(vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_v_bf16m4(vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_v_bf16m8(vbfloat16m8_t vs2, size_t vl);

// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_v_bf16mf4_m(vbool64_t vm, vbfloat16mf4_t vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_v_bf16mf2_m(vbool32_t vm, vbfloat16mf2_t vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_v_bf16m1_m(vbool16_t vm, vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_v_bf16m2_m(vbool8_t vm, vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_v_bf16m4_m(vbool4_t vm, vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_v_bf16m8_m(vbool2_t vm, vbfloat16m8_t vs2, size_t vl);
```

Non-policy overloaded

```
vbfloat16mf4_t __riscv_sf_vfexp(vbfloat16mf4_t vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp(vbfloat16mf2_t vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp(vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp(vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp(vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp(vbfloat16m8_t vs2, size_t vl);
```

```
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp(vbool64_t vm, vbfloat16mf4_t vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp(vbool32_t vm, vbfloat16mf2_t vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp(vbool16_t vm, vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp(vbool8_t vm, vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp(vbool4_t vm, vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp(vbool2_t vm, vbfloat16m8_t vs2, size_t vl);
```

Policy non-overloaded

```
vbfloat16mf4_t __riscv_sf_vfexp_v_bf16mf4_tu(vbfloat16mf4_t vd, vbfloat16mf4_t vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_v_bf16mf2_tu(vbfloat16mf2_t vd, vbfloat16mf2_t vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_v_bf16m1_tu(vbfloat16m1_t vd, vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_v_bf16m2_tu(vbfloat16m2_t vd, vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_v_bf16m4_tu(vbfloat16m4_t vd, vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_v_bf16m8_tu(vbfloat16m8_t vd, vbfloat16m8_t vs2, size_t vl);
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_v_bf16mf4_tum(vbool64_t vm, vbfloat16mf4_t vd, vbfloat16mf4_t
vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_v_bf16mf2_tum(vbool32_t vm, vbfloat16mf2_t vd, vbfloat16mf2_t
vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_v_bf16m1_tum(vbool16_t vm, vbfloat16m1_t vd, vbfloat16m1_t vs2,
size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_v_bf16m2_tum(vbool8_t vm, vbfloat16m2_t vd, vbfloat16m2_t vs2,
size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_v_bf16m4_tum(vbool4_t vm, vbfloat16m4_t vd, vbfloat16m4_t vs2,
size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_v_bf16m8_tum(vbool2_t vm, vbfloat16m8_t vd, vbfloat16m8_t vs2,
size_t vl);
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_v_bf16mf4_tumu(vbool64_t vm, vbfloat16mf4_t vd, vbfloat16mf4_t
vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_v_bf16mf2_tumu(vbool32_t vm, vbfloat16mf2_t vd, vbfloat16mf2_t
vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_v_bf16m1_tumu(vbool16_t vm, vbfloat16m1_t vd, vbfloat16m1_t vs2,
size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_v_bf16m2_tumu(vbool8_t vm, vbfloat16m2_t vd, vbfloat16m2_t vs2,
size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_v_bf16m4_tumu(vbool4_t vm, vbfloat16m4_t vd, vbfloat16m4_t vs2,
size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_v_bf16m8_tumu(vbool2_t vm, vbfloat16m8_t vd, vbfloat16m8_t vs2,
size_t vl);
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_v_bf16mf4_mu(vbool64_t vm, vbfloat16mf4_t vd, vbfloat16mf4_t
vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_v_bf16mf2_mu(vbool32_t vm, vbfloat16mf2_t vd, vbfloat16mf2_t
vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_v_bf16m1_mu(vbool16_t vm, vbfloat16m1_t vd, vbfloat16m1_t vs2,
size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_v_bf16m2_mu(vbool8_t vm, vbfloat16m2_t vd, vbfloat16m2_t vs2,
size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_v_bf16m4_mu(vbool4_t vm, vbfloat16m4_t vd, vbfloat16m4_t vs2,
size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_v_bf16m8_mu(vbool2_t vm, vbfloat16m8_t vd, vbfloat16m8_t vs2,
size_t vl);
```

Policy overloaded

```
vbfloat16mf4_t __riscv_sf_vfexp_tu(vbfloat16mf4_t vd, vbfloat16mf4_t vs2, size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_tu(vbfloat16mf2_t vd, vbfloat16mf2_t vs2, size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_tu(vbfloat16m1_t vd, vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_tu(vbfloat16m2_t vd, vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_tu(vbfloat16m4_t vd, vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_tu(vbfloat16m8_t vd, vbfloat16m8_t vs2, size_t vl);
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_tum(vbool64_t vm, vbfloat16mf4_t vd, vbfloat16mf4_t vs2, size_t
vl);
vbfloat16mf2_t __riscv_sf_vfexp_tum(vbool32_t vm, vbfloat16mf2_t vd, vbfloat16mf2_t vs2, size_t
vl);
vbfloat16m1_t __riscv_sf_vfexp_tum(vbool16_t vm, vbfloat16m1_t vd, vbfloat16m1_t vs2, size_t
vl);
vbfloat16m2_t __riscv_sf_vfexp_tum(vbool8_t vm, vbfloat16m2_t vd, vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_tum(vbool4_t vm, vbfloat16m4_t vd, vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_tum(vbool2_t vm, vbfloat16m8_t vd, vbfloat16m8_t vs2, size_t vl);
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_tumu(vbool64_t vm, vbfloat16mf4_t vd, vbfloat16mf4_t vs2,
size_t vl);
vbfloat16mf2_t __riscv_sf_vfexp_tumu(vbool32_t vm, vbfloat16mf2_t vd, vbfloat16mf2_t vs2,
size_t vl);
vbfloat16m1_t __riscv_sf_vfexp_tumu(vbool16_t vm, vbfloat16m1_t vd, vbfloat16m1_t vs2, size_t
vl);
vbfloat16m2_t __riscv_sf_vfexp_tumu(vbool8_t vm, vbfloat16m2_t vd, vbfloat16m2_t vs2, size_t
vl);
vbfloat16m4_t __riscv_sf_vfexp_tumu(vbool4_t vm, vbfloat16m4_t vd, vbfloat16m4_t vs2, size_t
vl);
vbfloat16m8_t __riscv_sf_vfexp_tumu(vbool2_t vm, vbfloat16m8_t vd, vbfloat16m8_t vs2, size_t
vl);
// masked functions
vbfloat16mf4_t __riscv_sf_vfexp_mu(vbool64_t vm, vbfloat16mf4_t vd, vbfloat16mf4_t vs2, size_t
vl);
vbfloat16mf2_t __riscv_sf_vfexp_mu(vbool32_t vm, vbfloat16mf2_t vd, vbfloat16mf2_t vs2, size_t
vl);
vbfloat16m1_t __riscv_sf_vfexp_mu(vbool16_t vm, vbfloat16m1_t vd, vbfloat16m1_t vs2, size_t vl);
vbfloat16m2_t __riscv_sf_vfexp_mu(vbool8_t vm, vbfloat16m2_t vd, vbfloat16m2_t vs2, size_t vl);
vbfloat16m4_t __riscv_sf_vfexp_mu(vbool4_t vm, vbfloat16m4_t vd, vbfloat16m4_t vs2, size_t vl);
vbfloat16m8_t __riscv_sf_vfexp_mu(vbool2_t vm, vbfloat16m8_t vd, vbfloat16m8_t vs2, size_t vl);
```