

## FRINTP (vector)

Floating-point Round to Integral, toward Plus infinity (vector). This instruction rounds a vector of floating-point values in the SIMD&FP source register to integral floating-point values of the same size using the Round towards Plus Infinity rounding mode, and writes the result to the SIMD&FP destination register.

A zero input gives a zero result with the same sign, an infinite input gives an infinite result with the same sign, and a NaN is propagated as for normal arithmetic.

A floating-point exception can be generated by this instruction. Depending on the settings in [FPCR](#), the exception results in either a flag being set in [FPSR](#), or a synchronous exception being generated. For more information, see [Floating-point exception traps](#).

Depending on the settings in the [CPACR\\_EL1](#), [CPTR\\_EL2](#), and [CPTR\\_EL3](#) registers, and the current Security state and Exception level, an attempt to execute the instruction might be trapped.

It has encodings from 2 classes: [Half-precision](#) and [Single-precision and double-precision](#)

### Half-precision (FEAT\_FP16)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Q	0	0	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	1	0	Rn				Rd						
U				o2				o1																							

**FRINTP** <Vd>.<T>, <Vn>.<T>

```

if !IsFeatureImplemented(FEAT_FP16) then UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);

constant integer esize = 16;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;

boolean exact = FALSE;
FPRounding rounding;
case U:o1:o2 of
    when '0xx' rounding = FPDecodeRounding(o1:o2);
    when '100' rounding = FPRounding\_TIEAWAY;
    when '101' UNDEFINED;
    when '110' rounding = FPRoundingMode(FPCR[]); exact = TRUE;
    when '111' rounding = FPRoundingMode(FPCR[]);

```

### Single-precision and double-precision

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Q	0	0	1	1	1	0	1	sz	1	0	0	0	0	1	1	0	0	0	1	0	Rn				Rd					

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U										o2										o1											

**FRINTP** <Vd>.<T>, <Vn>.<T>

```
integer d = UInt(Rd);
integer n = UInt(Rn);

if sz:Q == '10' then UNDEFINED;
constant integer esize = 32 << UInt(sz);
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;

boolean exact = FALSE;
FPRounding rounding;
case U:o1:o2 of
  when '0xx' rounding = FPDecodeRounding(o1:o2);
  when '100' rounding = FPRounding_TIEAWAY;
  when '101' UNDEFINED;
  when '110' rounding = FPRoundingMode(FPCR[]); exact = TRUE;
  when '111' rounding = FPRoundingMode(FPCR[]);
```

## Assembler Symbols

<Vd> Is the name of the SIMD&FP destination register, encoded in the "Rd" field.

<T> For the half-precision variant: is an arrangement specifier, encoded in "Q":

Q	<T>
0	4H
1	8H

For the single-precision and double-precision variant: is an arrangement specifier, encoded in "sz:Q":

sz	Q	<T>
0	0	2S
0	1	4S
1	0	RESERVED
1	1	2D

<Vn> Is the name of the SIMD&FP source register, encoded in the "Rn" field.

## Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand = V[n, datasize];
bits(datasize) result;
bits(esome) element;
```

```
for e = 0 to elements-1
    element = Elem[operand, e, esize];
    Elem[result, e, esize] = FPRoundInt(element, FPCR[], rounding, exact)
V[d, datasize] = result;
```

---

[Base  
Instructions](#)

[SIMD&FP  
Instructions](#)

[SVE  
Instructions](#)

[SME  
Instructions](#)

[Index by  
Encoding](#)

[Sh  
Pseudocode](#)

Internal version only: isa v33.64, AdvSIMD v29.12, pseudocode  
no\_diffs\_2023\_09\_RC2, sve v2023-06\_rel ; Build timestamp: 2023-09-18T17:56

Copyright Â© 2010-2023 Arm Limited or its affiliates. All rights reserved. This  
document is Non-Confidential.