FRINTZ (vector)

Floating-point Round to Integral, toward Zero (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 Zero 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	1	0	0	1	1	0	0	1	1	0			Rn					Rd		
	U							o2											o1												

FRINTZ **\<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	1	1	0			Rn					Rd		
	U							o2											o1												

```
        FRINTZ <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(esize) element;

    for e = 0 to elements-1
        element = Elem[operand, e, esize];
```

```
        Elem[result, e, esize] = FPRoundInt(element, FPCR[], rounding, exac

    V[d, datasize] = result;
```

```
        Elem[result, e, esize] = FPRoundInt(element, FPCR[], rounding, exac

    V[d, datasize] = result;
```