

FMINNM (vector)

Floating-point Minimum Number (vector). This instruction compares corresponding vector elements in the two source SIMD&FP registers, writes the smaller of the two floating-point values into a vector, and writes the vector to the destination SIMD&FP register.

Regardless of the value of [FPCR.AH](#), the behavior is as follows:

- Negative zero compares less than positive zero.
- If one element is numeric and the other is a quiet NaN, the result is the numeric value.
- When [FPCR.DN](#) is 0, if either element is a signaling NaN or if both elements are NaNs, the result is a quiet NaN.
- When [FPCR.DN](#) is 1, if either element is a signaling NaN or if both elements are NaNs, the result is Default NaN.

This instruction can generate a floating-point exception. 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	0	Rm			0	0	0	0	0	1	Rn			Rd								
U								a																							

FMINNM [<Vd>.<T>](#), [<Vn>.<T>](#), [<Vm>.<T>](#)

```
if !IsFeatureImplemented(FEAT_FP16) then UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
constant integer esize = 16;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;

boolean pair = (U == '1');
boolean minimum = (a == '1');
```

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	0	0	0	1	1	1	0	1	sz	1				Rm		1	1	0	0	0	1				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																o1															

FMINNM <Vd>.<T>, <Vn>.<T>, <Vm>.<T>

```
integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
if sz:Q == '10' then UNDEFINED;
constant integer esize = 32 << UInt(sz);
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;

boolean pair = (U == '1');
boolean minimum = (o1 == '1');
```

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 first SIMD&FP source register, encoded in the "Rn" field.

<Vm> Is the name of the second SIMD&FP source register, encoded in the "Rm" field.

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = V[n, datasize];
bits(datasize) operand2 = V[m, datasize];
bits(datasize) result;
bits(2*datasize) concat = operand2:operand1;
bits(esize) element1;
bits(esize) element2;
```

```

for e = 0 to elements-1
  if pair then
    element1 = Elem[concat, 2*e, esize];
    element2 = Elem[concat, (2*e)+1, esize];
  else
    element1 = Elem[operand1, e, esize];
    element2 = Elem[operand2, e, esize];

  if minimum then
    Elem[result, e, esize] = FPMinNum(element1, element2, FPCR[]);
  else
    Elem[result, e, esize] = FPMaxNum(element1, element2, FPCR[]);

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.