FCMLT (zero)

Floating-point Compare Less than zero (vector). This instruction reads each floating-point value in the source SIMD&FP register and if the value is less than zero sets every bit of the corresponding vector element in the destination SIMD&FP register to one, otherwise sets every bit of the corresponding vector element in the destination SIMD&FP register to zero.

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 4 classes: Scalar half precision , Scalar single-precision and double-precision , Vector half precision and Vector single-precision and double-precision

Scalar 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	1	0	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	0	1	0	Rn					Rd				

FCMLT <Hd>, <Hn>, #0.0

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

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

constant integer esize = 16;
constant integer datasize = esize;
integer elements = 1;

CompareOp comparison = CompareOp_LT;
```

Scalar 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	1	0	1	1	1	1	0	1	sz	1	0	0	0	0	0	1	1	1	0	1	0	Rn					Rd				

FCMLT <V><d>, <V><n>, #0.0

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

constant integer esize = 32 << UInt(sz);
constant integer datasize = esize;
integer elements = 1;
```

```
CompareOp comparison = CompareOp_LT;
```

Vector 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	0	1	1	1	0	1	0	Rn					Rd				

FCMLT <Vd>.<T>, <Vn>.<T>, #0.0

```
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;

CompareOp comparison = CompareOp_LT;
```

Vector 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	0	1	1	1	0	1	0	Rn					Rd				

FCMLT <Vd>.<T>, <Vn>.<T>, #0.0

```
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;

CompareOp comparison = CompareOp_LT;
```

Assembler Symbols

<Hd>	Is the 16-bit name of the SIMD&FP destination register, encoded in the "Rd" field.
<Hn>	Is the 16-bit name of the SIMD&FP source register, encoded in the "Rn" field.
<V>	Is a width specifier, encoded in “sz”:

sz	<V>
0	S
1	D

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

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

<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) zero = FPZero('0', esize);
bits(esize) element;
boolean test_passed;

for e = 0 to elements-1
    element = Elem[operand, e, esize];
    case comparison of
        when CompareOp_GT test_passed = FPCompareGT(element, zero, FPCR
        when CompareOp_GE test_passed = FPCompareGE(element, zero, FPCR
        when CompareOp_EQ test_passed = FPCompareEQ(element, zero, FPCR
        when CompareOp_LE test_passed = FPCompareGE(zero, element, FPCR
        when CompareOp_LT test_passed = FPCompareGT(zero, element, FPCR
    Elem[result, e, esize] = if test_passed then Ones(esize) else Zeros

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