FMLS (by element)

Floating-point fused Multiply-Subtract from accumulator (by element). This instruction multiplies the vector elements in the first source SIMD&FP register by the specified value in the second source SIMD&FP register, and subtracts the results from the vector elements of the destination SIMD&FP register. All the values in this instruction are floating-point values.

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	1	0	0	L	M	Rm				0	1	0	1	H	0	Rn					Rd				
																	o2														

```
FMLS <Hd>, <Hn>, <Vm>.H[<index>]
```

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

constant integer idxdsize = 64 << UInt(H);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer d = UInt(Rd);
integer index = UInt(H:L:M);

constant integer esize = 16;
constant integer datasize = esize;
integer elements = 1;
boolean sub_op = (o2 == '1');
```

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	1	1	sz	L	M	Rm				0	1	0	1	H	0	Rn					Rd				
																	o2														

```
FMLS <V><d>, <V><n>, <Vm>.<Ts>[<index>]
```

```
constant integer idxdsize = 64 << UInt(H);
integer index;
```

```
bit Rmhi = M;
case sz:L of
    when '0x' index = UInt(H:L);
    when '10' index = UInt(H);
    when '11' UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rmhi:Rm);

constant integer esize = 32 << UInt(sz);
constant integer datasize = esize;
integer elements = 1;
boolean sub_op = (o2 == '1');
```

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	1	0	0	L	M	Rm				0	1	0	1	H	0	Rn					Rd				
																	o2														

FMLS <Vd>.<T>, <Vn>.<T>, <Vm>.H[<index>]

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

constant integer idxdsize = 64 << UInt(H);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer d = UInt(Rd);
integer index = UInt(H:L:M);

constant integer esize = 16;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
boolean sub_op = (o2 == '1');
```

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	1	1	sz	L	M	Rm				0	1	0	1	H	0	Rn					Rd				
																	o2														

FMLS <Vd>.<T>, <Vn>.<T>, <Vm>.<Ts>[<index>]

```
constant integer idxdsize = 64 << UInt(H);
integer index;
bit Rmhi = M;
case sz:L of
    when '0x' index = UInt(H:L);
    when '10' index = UInt(H);
    when '11' UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rmhi: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 sub_op = (o2 == '1');
```

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 first 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 first 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 “Q:sz”:

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

<Vn>	Is the name of the first SIMD&FP source register, encoded in the "Rn" field.
<Vm>	For the half-precision variant: is the name of the second SIMD&FP source register, in the range V0 to V15, encoded in the "Rm" field.

For the single-precision and double-precision variant: is the name of the second SIMD&FP source register, encoded in the "M:Rm" fields.

<Ts>

Is an element size specifier, encoded in “sz”:

sz	<Ts>
0	S
1	D

<index>

For the half-precision variant: is the element index, in the range 0 to 7, encoded in the "H:L:M" fields.

For the single-precision and double-precision variant: is the element index, encoded in “sz:L:H”:

sz	L	<index>
0	x	H:L
1	0	H
1	1	RESERVED

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = V[n, datasize];
bits(idxdsize) operand2 = V[m, idxdsize];
bits(datasize) operand3 = V[d, datasize];
bits(esize) element1;
bits(esize) element2 = Elem[operand2, index, esize];
FPCRType fpcr = FPCR[];
boolean merge = elements == 1 && IsMerging(fpcr);
bits(128) result = if merge then V[d, 128] else Zeros(128);

for e = 0 to elements-1
    element1 = Elem[operand1, e, esize];
    if sub_op then element1 = FPNeg(element1);
    Elem[result, e, esize] = FPMulAdd(Elem[operand3, e, esize], element

V[d, 128] = result;
```