

FMLSL, FMLSL2 (vector)

Floating-point fused Multiply-Subtract Long from accumulator (vector). This instruction negates the values in the vector of one SIMD&FP register, multiplies these with the corresponding values in another vector, and accumulates the product to the corresponding vector element of the destination SIMD&FP register. The instruction does not round the result of the multiply before the accumulation.

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.

In Armv8.2 and Armv8.3, this is an optional instruction. From Armv8.4 it is mandatory for all implementations to support it.

Note

ID_AA64ISAR0_EL1.FHM indicates whether this instruction is supported.

It has encodings from 2 classes: [FMLSL](#) and [FMLSL2](#)

FMLSL (FEAT_FHM)

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	0	1	Rm				1	1	1	0	1	1	Rn				Rd						
S sz																															

FMLSL <Vd>.<Ta>, <Vn>.<Tb>, <Vm>.<Tb>

```
if !IsFeatureImplemented(FEAT_FHM) then UNDEFINED;
integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
if sz == '1' then UNDEFINED;
constant integer esize = 32;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
boolean sub_op = (S == '1');
integer part = 0;
```

FMLSL2 (FEAT_FHM)

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	1	0	1	1	1	0	1	0	1	Rm				1	1	0	0	1	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
S sz

FMLSL2 <Vd> .<Ta>, <Vn>.<Tb>, <Vm>.<Tb>

```
if !IsFeatureImplemented(FEAT_FHM) then UNDEFINED;
integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
if sz == '1' then UNDEFINED;
constant integer esize = 32;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
boolean sub_op = (S == '1');
integer part = 1;
```

Assembler Symbols

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

<Ta> Is an arrangement specifier, encoded in "Q":

Q	<Ta>
0	2S
1	4S

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

<Tb> Is an arrangement specifier, encoded in "Q":

Q	<Tb>
0	2H
1	4H

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

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize DIV 2) operand1 = Vpart[n, part, datasize DIV 2];
bits(datasize DIV 2) operand2 = Vpart[m, part, datasize DIV 2];
bits(datasize) operand3 = V[d, datasize];
bits(datasize) result;
bits(esome DIV 2) element1;
bits(esome DIV 2) element2;

for e = 0 to elements-1
    element1 = Elem[operand1, e, esize DIV 2];
    element2 = Elem[operand2, e, esize DIV 2];
```

```
if sub_op then element1 = FPNeg(element1);  
  Elem[result, e, esize] = FPMulAddH(Elem[operand3, e, esize], element1,  
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.