FMLS (indexed)

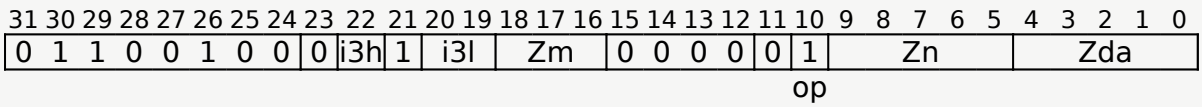
Floating-point fused multiply-subtract by indexed elements (Zda = Zda + -Zn * Zm[indexed])

Multiply all floating-point elements within each 128-bit segment of the first source vector by the specified element in the corresponding second source vector segment. The products are then destructively subtracted without intermediate rounding from the corresponding elements of the addend and destination vector.

The elements within the second source vector are specified using an immediate index which selects the same element position within each 128-bit vector segment. The index range is from 0 to one less than the number of elements per 128-bit segment, encoded in 1 to 3 bits depending on the size of the element. This instruction is unpredicated.

It has encodings from 3 classes: Half-precision , Single-precision and Double-precision

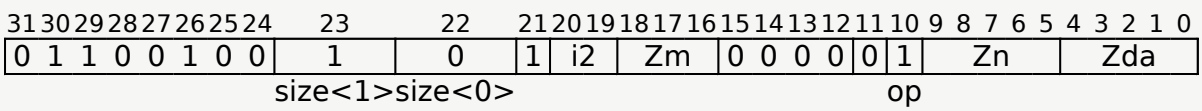
Half-precision



FMLS <Zda>.H, <Zn>.H, <Zm>.H[<imm>]

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 16;
integer index = UInt(i3h:i3l);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(Zda);
boolean op1_neg = TRUE;
boolean op3_neg = FALSE;
```

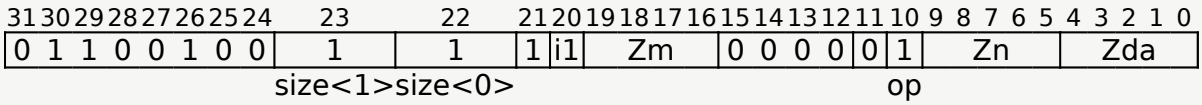
Single-precision



FMLS <Zda>.S, <Zn>.S, <Zm>.S[<imm>]

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 32;
integer index = UInt(i2);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(Zda);
boolean op1_neg = TRUE;
boolean op3_neg = FALSE;
```

Double-precision



FMLS <Zda>.D, <Zn>.D, <Zm>.D[<imm>]

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 64;
integer index = UInt(i1);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(Zda);
boolean op1_neg = TRUE;
boolean op3_neg = FALSE;
```

Assembler Symbols

<Zda>	Is the name of the third source and destination scalable vector register, encoded in the "Zda" field.
<Zn>	Is the name of the first source scalable vector register, encoded in the "Zn" field.
<Zm>	For the half-precision and single-precision variant: is the name of the second source scalable vector register Z0-Z7, encoded in the "Zm" field. For the double-precision variant: is the name of the second source scalable vector register Z0-Z15, encoded in the "Zm" field.
<imm>	For the half-precision variant: is the immediate index, in the range 0 to 7, encoded in the "i3h:i3l" fields. For the single-precision variant: is the immediate index, in the range 0 to 3, encoded in the "i2" field. For the double-precision variant: is the immediate index, in the range 0 to 1, encoded in the "i1" field.

Operation

```
CheckSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;
constant integer eltspersegment = 128 DIV esize;
bits(VL) operand1 = Z[n, VL];
bits(VL) operand2 = Z[m, VL];
bits(VL) result = Z[da, VL];

for e = 0 to elements-1
    integer segmentbase = e - (e MOD eltspersegment);
    integer s = segmentbase + index;
    bits(esize) element1 = Elem[operand1, e, esize];
    bits(esize) element2 = Elem[operand2, s, esize];
    bits(esize) element3 = Elem[result, e, esize];
    if op1_neg then element1 = FPNeg(element1);
    if op3_neg then element3 = FPNeg(element3);
    Elem[result, e, esize] = FPMulAdd(element3, element1, element2, FPC

Z[da, VL] = result;
```

Operational information

This instruction might be immediately preceded in program order by a MOVPRFX instruction. The MOVPRFX instruction must conform to all of the following requirements, otherwise the behavior of the MOVPRFX and this instruction is unpredictable:

- The MOVPRFX instruction must be unpredicated.
- The MOVPRFX instruction must specify the same destination register as this instruction.
- The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

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