FMLS (multiple and indexed vector)

Multi-vector floating-point fused multiply-subtract by indexed element

Multiply the indexed element of the second source vector by the corresponding floating-point elements of the two or four first source vectors and destructively subtract without intermediate rounding from the corresponding elements of the ZA single-vector groups.

The elements within the second source vector are specified using an immediate element 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 2 bits depending on the size of the element. The vector numbers forming the single-vector group within each half of or each quarter of the ZA array are selected by the sum of the vector select register and immediate offset, modulo half or quarter the number of ZA array vectors.

The vector group symbol, VGx2 or VGx4, indicates that the ZA operand consists of two or four ZA single-vector groups respectively. The vector group symbol is preferred for disassembly, but optional in assembler source code.

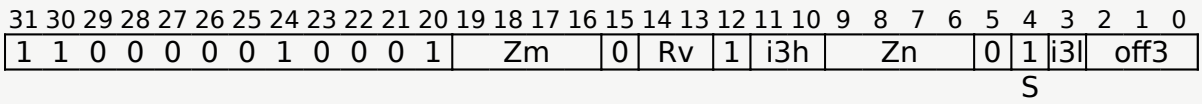
This instruction follows SME ZA-targeting floating-point behaviors.

This instruction is unpredicated.

ID_AA64SMFR0_EL1.F64F64 indicates whether the double-precision variant is implemented, and ID_AA64SMFR0_EL1.F16F16 indicates whether the half-precision variant is implemented.

It has encodings from 6 classes: Two ZA single-vectors of half precision elements , Two ZA single-vectors of single precision elements , Two ZA single-vectors of double precision elements , Four ZA single-vectors of half precision elements , Four ZA single-vectors of single precision elements and Four ZA single-vectors of double precision elements

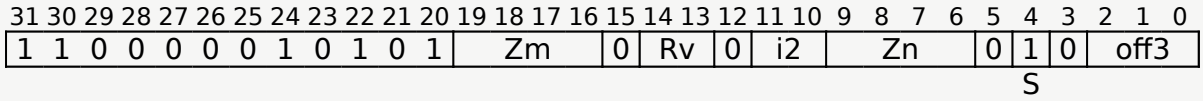
Two ZA single-vectors of half precision elements
(FEAT_SME_F16F16)




```
FMLS ZA.H[<Wv>, <offs>{, VGx2}], { <Zn1>.H-<Zn2>.H }, <Zm>.H[<index>]
```

```
if !HaveSME2() || !IsFeatureImplemented(FEAT_SME_F16F16) then UNDEFINED
integer v = UInt('010':Rv);
constant integer esize = 16;
integer n = UInt(Zn:'0');
integer m = UInt('0':Zm);
integer offset = UInt(off3);
integer index = UInt(i3h:i3l);
boolean sub_op = TRUE;
constant integer nreg = 2;
```

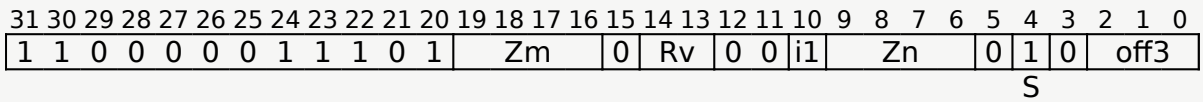
Two ZA single-vectors of single precision elements
(FEAT_SME2)



FMLS ZA.S[<Wv>, <offs>{, VGx2}], { <Zn1>.S-<Zn2>.S }, <Zm>.S[<index>]

```
if !HaveSME2() then UNDEFINED;
integer v = UInt('010':Rv);
constant integer esize = 32;
integer n = UInt(Zn:'0');
integer m = UInt('0':Zm);
integer offset = UInt(off3);
integer index = UInt(i2);
boolean sub_op = TRUE;
constant integer nreg = 2;
```

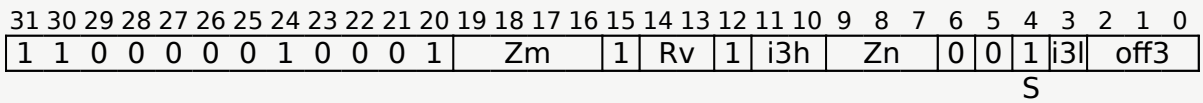
Two ZA single-vectors of double precision elements
(FEAT_SME_F64F64)



FMLS ZA.D[<Wv>, <offs>{, VGx2}], { <Zn1>.D-<Zn2>.D }, <Zm>.D[<index>]

```
if !(HaveSME2() && HaveSMEF64F64()) then UNDEFINED;
integer v = UInt('010':Rv);
constant integer esize = 64;
integer n = UInt(Zn:'0');
integer m = UInt('0':Zm);
integer offset = UInt(off3);
integer index = UInt(i1);
boolean sub_op = TRUE;
constant integer nreg = 2;
```

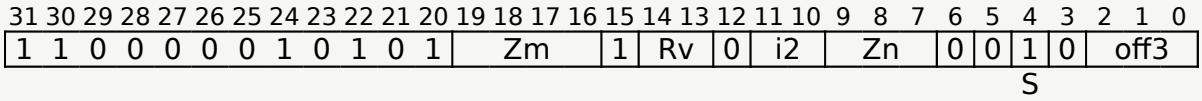
Four ZA single-vectors of half precision elements
(FEAT_SME_F16F16)



FMLS ZA.H[<Wv>, <offs>{, VGx4}], { <Zn1>.H-<Zn4>.H }, <Zm>.H[<index>]

```
if !HaveSME2() || !IsFeatureImplemented(FEAT_SME_F16F16) then UNDEFINED
integer v = UInt('010':Rv);
constant integer esize = 16;
integer n = UInt(Zn:'00');
integer m = UInt('0':Zm);
integer offset = UInt(off3);
integer index = UInt(i3h:i3l);
boolean sub_op = TRUE;
constant integer nreg = 4;
```

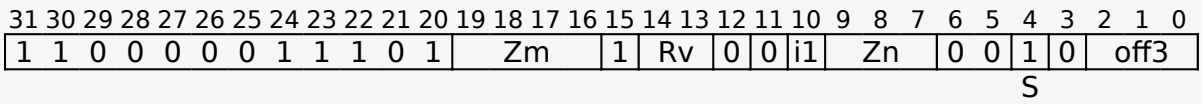
Four ZA single-vectors of single precision elements
(FEAT_SME2)



FMLS ZA.S[<Wv>, <offs>{, VGx4}], { <Zn1>.S-<Zn4>.S }, <Zm>.S[<index>]

```
if !HaveSME2() then UNDEFINED;
integer v = UInt('010':Rv);
constant integer esize = 32;
integer n = UInt(Zn:'00');
integer m = UInt('0':Zm);
integer offset = UInt(off3);
integer index = UInt(i2);
boolean sub_op = TRUE;
constant integer nreg = 4;
```

Four ZA single-vectors of double precision elements
(FEAT_SME_F64F64)



FMLS ZA.D[<Wv>, <offs>{, VGx4}], { <Zn1>.D-<Zn4>.D }, <Zm>.D[<index>]

```
if !(HaveSME2() && HaveSMEF64F64()) then UNDEFINED;
integer v = UInt('010':Rv);
constant integer esize = 64;
integer n = UInt(Zn:'00');
integer m = UInt('0':Zm);
integer offset = UInt(off3);
integer index = UInt(i1);
boolean sub_op = TRUE;
constant integer nreg = 4;
```

Assembler Symbols

<Wv>	Is the 32-bit name of the vector select register W8-W11, encoded in the "Rv" field.
<offs>	Is the vector select offset, in the range 0 to 7, encoded in the "off3" field.
<Zn1>	For the two ZA single-vectors of double precision elements, two ZA single-vectors of half precision elements and two ZA single-vectors of single precision elements variant: is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zn" times 2.

	For the four ZA single-vectors of double precision elements, four ZA single-vectors of half precision elements and four ZA single-vectors of single precision elements variant: is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zn" times 4.
<Zn4>	Is the name of the fourth scalable vector register of a multi-vector sequence, encoded as "Zn" times 4 plus 3.
<Zn2>	Is the name of the second scalable vector register of a multi-vector sequence, encoded as "Zn" times 2 plus 1.
<Zm>	Is the name of the second source scalable vector register Z0-Z15, encoded in the "Zm" field.
<index>	For the four ZA single-vectors of half precision elements and two ZA single-vectors of half precision elements variant: is the element index, in the range 0 to 7, encoded in the "i3h:i3l" fields. For the four ZA single-vectors of single precision elements and two ZA single-vectors of single precision elements variant: is the element index, in the range 0 to 3, encoded in the "i2" field. For the four ZA single-vectors of double precision elements and two ZA single-vectors of double precision elements variant: is the element index, in the range 0 to 1, encoded in the "i1" field.

Operation

```
CheckStreamingSVEAndZAEnabled();
constant integer VL = CurrentVL;
constant integer elements = VL DIV esize;
integer vectors = VL DIV 8;
integer vstride = vectors DIV nreg;
integer eltspersegment = 128 DIV esize;
bits(32) vbase = X[v, 32];
integer vec = (UInt(vbase) + offset) MOD vstride;
bits(VL) result;

for r = 0 to nreg-1
    bits(VL) operand1 = Z[n+r, VL];
    bits(VL) operand2 = Z[m, VL];
    bits(VL) operand3 = ZAvector[vec, VL];
    for e = 0 to elements-1
        bits(esize) element1 = Elem[operand1, e, esize];
        integer segmentbase = e - (e MOD eltspersegment);
        integer s = segmentbase + index;
        bits(esize) element2 = Elem[operand2, s, esize];
        bits(esize) element3 = Elem[operand3, e, esize];
        if sub_op then element1 = FPNeg(element1);
        Elem[result, e, esize] = FPMulAdd_ZA(element3, element1, elemen
    ZAvector[vec, VL] = result;
    vec = vec + vstride;
```

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