SMLSLL (multiple vectors)

Multi-vector signed integer multiply-subtract long-long

This signed integer multiply-subtract long-long instruction multiplies each signed 8-bit or 16-bit element in the two or four first source vectors with each signed 8-bit or 16-bit element in the one, two, or four second source vectors, widens each product to 32-bits or 64-bits and destructively subtracts these values from the corresponding 32-bit or 64-bit elements of the ZA quad-vector groups. The lowest of the four consecutive vector numbers forming the quad-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 quad-vector groups respectively. The vector group symbol is preferred for disassembly, but optional in assembler source code.

This instruction is unpredicated.

ID_AA64SMFR0_EL1.I16I64 indicates whether the 16-bit integer variant is implemented.

It has encodings from 2 classes: Two ZA quad-vectors and Four ZA quad-vectors

Two ZA quad-vectors
(FEAT_SME2)

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
1	1	0	0	0	0	0	1	1	sz	1	Zm				0	0	Rv			0	0	0	Zn				0	0	1	0	0	o1
																											U	S				

```
SMLSLL ZA.<T>[<Wv>, <offs1>:<offs4>{, VGx2}], { <Zn1>.<Tb>-<Zn2>.<Tb>
```

```
if !HaveSME2() then UNDEFINED;
if sz == '1' && !HaveSMEI16I64() then UNDEFINED;
constant integer esize = 32 << UInt(sz);
integer v = UInt('010':Rv);
integer n = UInt(Zn:'0');
integer m = UInt(Zm:'0');
integer offset = UInt(o1:'00');
constant integer nreg = 2;
```

Four ZA quad-vectors
(FEAT_SME2)

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
1	1	0	0	0	0	0	1	1	sz	1	Zm				0	1	0	Rv			0	0	0	Zn			0	0	0	1	0	0	o1
																											U	S				

SMLSLL ZA.<T>[<Wv>, <offs1>:<offs4>{, VGx4}], { <Zn1>.<Tb>-<Zn4>.<Tb>

```
if !HaveSME2() then UNDEFINED;
if sz == '1' && !HaveSMEI16I64() then UNDEFINED;
constant integer esize = 32 << UInt(sz);
integer v = UInt('010':Rv);
integer n = UInt(Zn:'00');
integer m = UInt(Zm:'00');
integer offset = UInt(o1:'00');
constant integer nreg = 4;
```

Assembler Symbols

<T>

Is the size specifier, encoded in “sz”:

sz	<T>
0	S
1	D

<Wv> Is the 32-bit name of the vector select register W8-W11, encoded in the "Rv" field.

<offs1> Is the vector select offset, pointing to first of four consecutive vectors, encoded as "o1" field times 4.

<offs4> Is the vector select offset, pointing to last of four consecutive vectors, encoded as "o1" field times 4 plus 3.

<Zn1> For the two ZA quad-vectors variant: is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zn" times 2.

For the four ZA quad-vectors variant: is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zn" times 4.

<Tb>

Is the size specifier, encoded in “sz”:

sz	<Tb>
0	B
1	H

<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.

<Zm1> For the two ZA quad-vectors variant: is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zm" times 2.

	For the four ZA quad-vectors variant: is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zm" times 4.
<Zm4>	Is the name of the fourth scalable vector register of a multi-vector sequence, encoded as "Zm" times 4 plus 3.
<Zm2>	Is the name of the second scalable vector register of a multi-vector sequence, encoded as "Zm" times 2 plus 1.

Operation

```
CheckStreamingSVEAndZAEnabled();
constant integer VL = CurrentVL;
constant integer elements = VL DIV esize;
integer vectors = VL DIV 8;
integer vstride = vectors DIV nreg;
bits(32) vbase = X[v, 32];
integer vec = (UInt(vbase) + offset) MOD vstride;
bits(VL) result;
vec = vec - (vec MOD 4);

for r = 0 to nreg-1
    bits(VL) operand1 = Z[n+r, VL];
    bits(VL) operand2 = Z[m+r, VL];
    for i = 0 to 3
        bits(VL) operand3 = ZAvector[vec + i, VL];
        for e = 0 to elements-1
            integer element1 = SInt(Elem[operand1, 4 * e + i, esize DIV
            integer element2 = SInt(Elem[operand2, 4 * e + i, esize DIV
            bits(esize) product = (element1 * element2)<esize-1:0>;
            Elem[result, e, esize] = Elem[operand3, e, esize] - product
        ZAvector[vec + i, VL] = result;
    vec = vec + vstride;
```

Operational information

If PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
  - The values of the data supplied in any of its registers.
  - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
  - The values of the data supplied in any of its registers.
  - The values of the NZCV flags.

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