SQDMLALT (indexed)

> Signed saturating doubling multiply-add long to accumulator (top, indexed)

Multiply then double the odd-numbered signed elements within each 128-bit segment of the first source vector and the specified signed element in the corresponding second source vector segment. Each intermediate value is saturated to the double-width N-bit value's signed integer range $-2^{(N-1)}$ to $(2^{(N-1)})-1$. Then destructively add to the overlapping double-width elements of the addend and destination vector. Each destination element is saturated to the double-width N-bit element's signed integer range $-2^{(N-1)}$ to $(2^{(N-1)})-1$.

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 2 or 3 bits depending on the size of the element.

It has encodings from 2 classes: 32-bit and 64-bit

32-bit

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	0	0	1	0	0	1	0	1	i3h	Zm	0	0	1	0	i3l	1	Zn	Zda
								size<1>	size<0>							S		T		

SQDMLALT <Zda>.S, <Zn>.H, <Zm>.H[<imm>]

```
if !HaveSVE2() && !HaveSME() then UNDEFINED;
constant integer esize = 16;
integer index = UInt(i3h:i3l);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(Zda);
integer sel = 1;
```

64-bit

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	0	0	1	0	0	1	1	1	i2h	Zm	0	0	1	0	i2l	1	Zn	Zda
								size<1>	size<0>							S		T		

SQDMLALT <Zda>.D, <Zn>.S, <Zm>.S[<imm>]

```
if !HaveSVE2() && !HaveSME() then UNDEFINED;
constant integer esize = 32;
integer index = UInt(i2h:i2l);
integer n = UInt(Zn);
integer m = UInt(Zm);
```

```
integer da = UInt(Zda);
integer sel = 1;
```

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 32-bit variant: is the name of the second source scalable vector register Z0-Z7, encoded in the "Zm" field. For the 64-bit variant: is the name of the second source scalable vector register Z0-Z15, encoded in the "Zm" field.
<imm>	For the 32-bit variant: is the element index, in the range 0 to 7, encoded in the "i3h:i3l" fields. For the 64-bit variant: is the element index, in the range 0 to 3, encoded in the "i2h:i2l" fields.

Operation

```
CheckSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV (2 * esize);
constant integer eltspersegment = 128 DIV (2 * 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 s = e - (e MOD eltspersegment);
    integer element1 = SInt(Elem[operand1, 2 * e + sel,   esize]);
    integer element2 = SInt(Elem[operand2, 2 * s + index, esize]);
    integer element3 = SInt(Elem[result, e, 2*esize]);
    integer product = SInt(SignedSat(2 * element1 * element2, 2*esize))
    integer res = element3 + product;
    Elem[result, e, 2*esize] = SignedSat(res, 2*esize);

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