

## SMLAL (multiple and single vector)

Multi-vector signed integer multiply-add long by vector

This signed integer multiply-add long instruction multiplies each signed 16-bit element in the one, two, or four first source vectors with each signed 16-bit element in the second source vector, widens each product to 32-bits and destructively adds these values to the corresponding 32-bit elements of the ZA double-vector groups. The lowest of the two consecutive vector numbers forming the double-vector group within all of, each half of, or each quarter of the ZA array are selected by the sum of the vector select register and immediate offset, modulo all, 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 double-vector groups respectively. The vector group symbol is preferred for disassembly, but optional in assembler source code.

This instruction is unpredicated.

It has encodings from 3 classes: [One ZA double-vector](#) , [Two ZA double-vectors](#) and [Four ZA double-vectors](#)

### One ZA double-vector (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	0	1	1	0	Zm		0		Rv		0		1		1		Zn			0		0		off3	
																								U				S				

**SMLAL** ZA.S[<Wv>, <offs1>:<offs2>], <Zn>.H, <Zm>.H

```
if !HaveSME2() then UNDEFINED;
constant integer esize = 32;
integer v = UInt('010':Rv);
integer n = UInt(Zn);
integer m = UInt('0':Zm);
integer offset = UInt(off3:'0');
constant integer nreg = 1;
```

### Two ZA double-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	0	1	1	0	Zm		0	Rv	0	1	0	Zn			0	0	0	off2						
																								U				S			

**SMLAL** ZA.S[<Wv>, <offs1>:<offs2>{, VGx2}], { <Zn1>.H-<Zn2>.H }, <Zm>.H

```
if !HaveSME2() then UNDEFINED;
constant integer esize = 32;
integer v = UInt('010':Rv);
```

```
integer n = UInt(Zn);
integer m = UInt('0':Zm);
integer offset = UInt(off2:'0');
constant integer nreg = 2;
```

### Four ZA double-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	0	1	1	1	Zm			0	Rv	0	1	0	Zn			0	0	0	off2									
																										U				S					

```
SMLAL ZA.S[<Wv>, <offs1>:<offs2>{, VGx4}], { <Zn1>.H-<Zn4>.H }, <Zm>.I
```

```
if !HaveSME2() then UNDEFINED;
constant integer esize = 32;
integer v = UInt('010':Rv);
integer n = UInt(Zn);
integer m = UInt('0':Zm);
integer offset = UInt(off2:'0');
constant integer nreg = 4;
```

## Assembler Symbols

- |         |  |
|---------|--|
| <Wv>    | Is the 32-bit name of the vector select register W8-W11, encoded in the "Rv" field.  |
| <offs1> | For the one ZA double-vector variant: is the vector select offset, pointing to first of two consecutive vectors, encoded as "off3" field times 2.<br><br>For the four ZA double-vectors and two ZA double-vectors variant: is the vector select offset, pointing to first of two consecutive vectors, encoded as "off2" field times 2.             |
| <offs2> | For the one ZA double-vector variant: is the vector select offset, pointing to last of two consecutive vectors, encoded as "off3" field times 2 plus 1.<br><br>For the four ZA double-vectors and two ZA double-vectors variant: is the vector select offset, pointing to last of two consecutive vectors, encoded as "off2" field times 2 plus 1. |
| <Zn>    | Is the name of the first source scalable vector register, encoded in the "Zn" field.   |
| <Zn1>   | Is the name of the first scalable vector register of a multi-vector sequence, encoded as "Zn".   |
| <Zn4>   | Is the name of the fourth scalable vector register of a multi-vector sequence, encoded as "Zn" plus 3 modulo 32.   |
| <Zn2>   | Is the name of the second scalable vector register of a multi-vector sequence, encoded as "Zn" plus 1 modulo 32.   |
| <Zm>    | Is the name of the second source scalable vector register Z0-Z15, encoded in the "Zm" field.   |

## Operation

```
CheckStreamingSVEAndZAEEnabled();
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 2);

for r = 0 to nreg-1
  bits(VL) operand1 = Z[(n+r) MOD 32, VL];
  bits(VL) operand2 = Z[m, VL];
  for i = 0 to 1
    bits(VL) operand3 = ZAvector[vec + i, VL];
    for e = 0 to elements-1
      integer element1 = SInt(Elem[operand1, 2 * e + i, esize DIV 2]);
      integer element2 = SInt(Elem[operand2, 2 * e + i, esize DIV 2]);
      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.

[Base  
Instructions](#)

[SIMD&FP  
Instructions](#)

[SVE  
Instructions](#)

[SME  
Instructions](#)

[Index by  
Encoding](#)

[Sh  
Pseud](#)

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.