UMLALL (multiple and indexed vector)

Multi-vector unsigned integer multiply-add long-long by indexed element

This unsigned integer multiply-add long-long instruction multiplies each unsigned 8-bit or 16-bit element in the one, two, or four first source vectors with each unsigned 8-bit or 16-bit indexed element of second source vector, widens each product to 32-bits or 64-bits and destructively adds these values to the corresponding 32-bit or 64-bit elements of the ZA quad-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 3 to 4 bits depending on the size of the element. The lowest of the four consecutive vector numbers forming the quad-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.

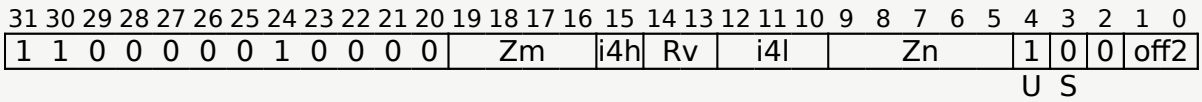
The vector group symbol, VGx2 or VGx4, indicates that the ZA operand consists of two or four ZA quad-vector groups respectively. The vector group symbol is preferred for disassembly, but optional in assembler source code.

This instruction is unpredicated.

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

It has encodings from 6 classes: One ZA quad-vector of 32-bit elements , One ZA quad-vector of 64-bit elements , Two ZA quad-vectors of 32-bit elements , Two ZA quad-vectors of 64-bit elements , Four ZA quad-vectors of 32-bit elements and Four ZA quad-vectors of 64-bit elements

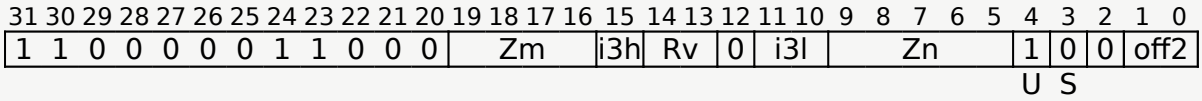
One ZA quad-vector of 32-bit elements
(FEAT_SME2)



UMLALL ZA.S[<Wv>, <offs1>:<offs4>], <Zn>.B, <Zm>.B[<index>]

```
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:'00');
integer index = UInt(i4h:i4l);
constant integer nreg = 1;
```

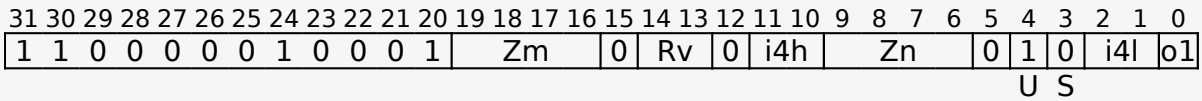
One ZA quad-vector of 64-bit elements
(FEAT_SME_I16I64)



UMLALL ZA.D[<Wv>, <offs1>:<offs4>], <Zn>.H, <Zm>.H[<index>]

```
if !(HaveSME2() && HaveSMEI16I64()) then UNDEFINED;
constant integer esize = 64;
integer v = UInt('010':Rv);
integer n = UInt(Zn);
integer m = UInt('0':Zm);
integer offset = UInt(off2:'00');
integer index = UInt(i3h:i3l);
constant integer nreg = 1;
```

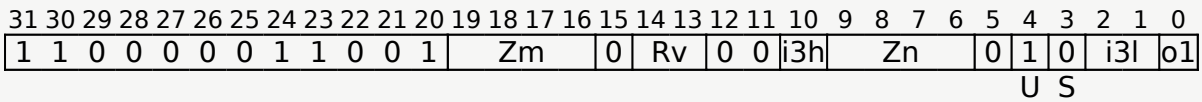
Two ZA quad-vectors of 32-bit elements
(FEAT_SME2)



UMLALL ZA.S[<Wv>, <offs1>:<offs4>{, VGx2}], { <Zn1>.B-<Zn2>.B }, <Zm>

```
if !HaveSME2() then UNDEFINED;
constant integer esize = 32;
integer v = UInt('010':Rv);
integer n = UInt(Zn:'0');
integer m = UInt('0':Zm);
integer offset = UInt(o1:'00');
integer index = UInt(i4h:i4l);
constant integer nreg = 2;
```

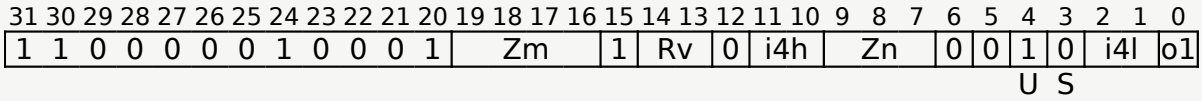
Two ZA quad-vectors of 64-bit elements
(FEAT_SME_I16I64)



UMLALL ZA.D[<Wv>, <offs1>:<offs4>{, VGx2}], { <Zn1>.H-<Zn2>.H }, <Zm>

```
if !(HaveSME2() && HaveSMEI16I64()) then UNDEFINED;
constant integer esize = 64;
integer v = UInt('010':Rv);
integer n = UInt(Zn:'0');
integer m = UInt('0':Zm);
integer offset = UInt(o1:'00');
integer index = UInt(i3h:i3l);
constant integer nreg = 2;
```

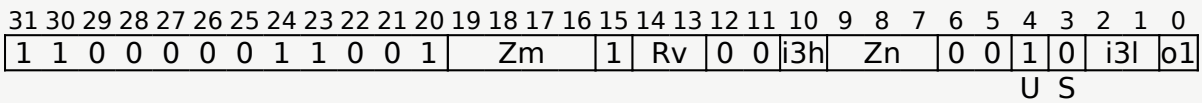
Four ZA quad-vectors of 32-bit elements
(FEAT_SME2)



UMLALL ZA.S[<Wv>, <offs1>:<offs4>{, VGx4}], { <Zn1>.B-<Zn4>.B }, <Zm>

```
if !HaveSME2() then UNDEFINED;
constant integer esize = 32;
integer v = UInt('010':Rv);
integer n = UInt(Zn:'00');
integer m = UInt('0':Zm);
integer offset = UInt(o1:'00');
integer index = UInt(i4h:i4l);
constant integer nreg = 4;
```

Four ZA quad-vectors of 64-bit elements
(FEAT_SME_I16I64)



UMLALL ZA.D[<Wv>, <offs1>:<offs4>{, VGx4}], { <Zn1>.H-<Zn4>.H }, <Zm>

```
if !(HaveSME2() && HaveSMEI16I64()) then UNDEFINED;
constant integer esize = 64;
integer v = UInt('010':Rv);
integer n = UInt(Zn:'00');
integer m = UInt('0':Zm);
integer offset = UInt(o1:'00');
integer index = UInt(i3h:i3l);
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 quad-vector of 32-bit elements and one ZA quad-vector of 64-bit elements variant: is the vector select offset, pointing to first of four consecutive vectors, encoded as "off2" field times 4. For the four ZA quad-vectors of 32-bit elements, four ZA quad-vectors of 64-bit elements, two ZA quad-vectors of 32-bit elements and two ZA quad-vectors of 64-bit elements variant: is the vector select offset, pointing to first of four consecutive vectors, encoded as "o1" field times 4.

<offs4> For the one ZA quad-vector of 32-bit elements and one ZA quad-vector of 64-bit elements variant: is the vector select offset, pointing to last of four consecutive vectors, encoded as "off2" field times 4 plus 3.

For the four ZA quad-vectors of 32-bit elements, four ZA quad-vectors of 64-bit elements, two ZA quad-vectors of 32-bit elements and two ZA quad-vectors of 64-bit elements variant: is the vector select offset, pointing to last of four consecutive vectors, encoded as "o1" field times 4 plus 3.

<Zn> Is the name of the first source scalable vector register, encoded in the "Zn" field.

<Zn1> For the two ZA quad-vectors of 32-bit elements and two ZA quad-vectors of 64-bit 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 quad-vectors of 32-bit elements and four ZA quad-vectors of 64-bit 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 quad-vectors of 32-bit elements, one ZA quad-vector of 32-bit elements and two ZA quad-vectors of 32-bit elements variant: is the element index, in the range 0 to 15, encoded in the "i4h:i4l" fields.

For the four ZA quad-vectors of 64-bit elements, one ZA quad-vector of 64-bit elements and two ZA quad-vectors of 64-bit elements variant: is the element index, in the range 0 to 7, encoded in the "i3h:i3l" fields.

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;
vec = vec - (vec MOD 4);
```

```
for r = 0 to nreg-1
    bits(VL) operand1 = Z[n+r, VL];
    bits(VL) operand2 = Z[m, VL];
    for i = 0 to 3
        bits(VL) operand3 = ZAvector[vec + i, VL];
        for e = 0 to elements-1
            integer segmentbase = e - (e MOD eltspersegment);
            integer s = 4 * segmentbase + index;
            integer element1 = UInt(Elem[operand1, 4 * e + i, esize DIV
            integer element2 = UInt(Elem[operand2, s, esize DIV 4]);
            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