UMOPA (4-way)

Unsigned integer sum of outer products and accumulate

The 8-bit integer variant works with a 32-bit element ZA tile.

The 16-bit integer variant works with a 64-bit element ZA tile.

The unsigned integer sum of outer products and accumulate instructions multiply the sub-matrix in the first source vector by the sub-matrix in the second source vector. In case of the 8-bit integer variant, the first source holds SVL_SÃ—4 sub-matrix of unsigned 8-bit integer values, and the second source holds 4Ã—SVL_S sub-matrix of unsigned 8-bit integer values. In case of the 16-bit integer variant, the first source holds SVL_DÃ—4 sub-matrix of unsigned 16-bit integer values, and the second source holds 4Ã—SVL_D sub-matrix of unsigned 16-bit integer values.

Each source vector is independently predicated by a corresponding governing predicate. When an 8-bit source element in case of 8-bit integer variant or a 16-bit source element in case of 16-bit integer variant is Inactive, it is treated as having the value 0.

The resulting SVL_SÃ—SVL_S widened 32-bit integer or SVL_DÃ—SVL_D widened 64-bit integer sum of outer products is then destructively added to the 32-bit integer or 64-bit integer destination tile, respectively for 8-bit integer and 16-bit integer instruction variants. This is equivalent to performing a 4-way dot product and accumulate to each of the destination tile elements.

In case of the 8-bit integer variant, each 32-bit container of the first source vector holds 4 consecutive column elements of each row of a SVL_SÃ—4 sub-matrix, and each 32-bit container of the second source vector holds 4 consecutive row elements of each column of a 4Ã—SVL_S sub-matrix. In case of the 16-bit integer variant, each 64-bit container of the first source vector holds 4 consecutive column elements of each row of a SVL_DÃ—4 sub-matrix, and each 64-bit container of the second source vector holds 4 consecutive row elements of each column of a 4Ã—SVL_D sub-matrix.

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

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

32-bit
(FEAT_SME)

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	0	1	0	0	0	0	1	1	0	1	Zm	Pm	Pn	Zn	0	0	0	ZAda
							u0			u1					S			

UMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.B, <Zm>.B

```
if !HaveSME() then UNDEFINED;
constant integer esize = 32;
integer a = UInt(Pn);
integer b = UInt(Pm);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(ZAda);
boolean sub_op = FALSE;
boolean op1_unsigned = TRUE;
boolean op2_unsigned = TRUE;
```

64-bit
(FEAT_SME_I16I64)

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	0	1	0	0	0	0	1	1	1	1	Zm	Pm	Pn	Zn	0	0	ZAda
							u0			u1					S		

UMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.H, <Zm>.H

```
if !HaveSMEI16I64() then UNDEFINED;
constant integer esize = 64;
integer a = UInt(Pn);
integer b = UInt(Pm);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(ZAda);
boolean sub_op = FALSE;
boolean op1_unsigned = TRUE;
boolean op2_unsigned = TRUE;
```

Assembler Symbols

<ZAda>	For the 32-bit variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field. For the 64-bit variant: is the name of the ZA tile ZA0-ZA7, encoded in the "ZAda" field.
<Pn>	Is the name of the first governing scalable predicate register P0-P7, encoded in the "Pn" field.
<Pm>	Is the name of the second governing scalable predicate register P0-P7, encoded in the "Pm" field.
<Zn>	Is the name of the first source scalable vector register, encoded in the "Zn" field.
<Zm>	Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```
CheckStreamingSVEAndZAEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer dim = VL DIV esize;
bits(PL) mask1 = P[a, PL];
bits(PL) mask2 = P[b, PL];
bits(VL) operand1 = Z[n, VL];
bits(VL) operand2 = Z[m, VL];
bits(dim*dim*esize) operand3 = ZAtile[da, esize, dim*dim*esize];
bits(dim*dim*esize) result;
integer  prod;

for row = 0 to dim-1
    for col = 0 to dim-1
        bits(esize) sum = Elem[operand3, row*dim+col, esize];
        for k = 0 to 3
            if ActivePredicateElement(mask1, 4*row + k, esize DIV 4) &&
                    ActivePredicateElement(mask2, 4*col + k, esize DIV
                prod = (Int(Elem[operand1, 4*row + k, esize DIV 4], op1
                        Int(Elem[operand2, 4*col + k, esize DIV 4], op2
                if sub_op then prod = -prod;
                sum = sum + prod;

        Elem[result, row*dim+col, esize] = sum;

ZAtile[da, esize, dim*dim*esize] = result;
```

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 operand registers when its governing predicate registers contain the same value for each execution.
  - 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 operand registers when its governing predicate registers contain the same value for each execution.
  - 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