FMOPA (non-widening)

Floating-point outer product and accumulate

The half-precision variant works with a 16-bit element ZA tile.
The single-precision variant works with a 32-bit element ZA tile.
The double-precision variant works with a 64-bit element ZA tile.
These instructions generate an outer product of the first source vector and the second source vector. In case of the half-precision variant, the first source is SVL_HÃ—1 vector and the second source is 1Ã—SVL_H vector. In case of the single-precision variant, the first source is SVL_SÃ—1 vector and the second source is 1Ã—SVL_S vector. In case of the double-precision variant, the first source is SVL_DÃ—1 vector and the second source is 1Ã—SVL_D vector.

Each source vector is independently predicated by a corresponding governing predicate. When either source vector element is Inactive the corresponding destination tile element remains unmodified.
The resulting outer product, SVL_HÃ—SVL_H in case of half-precision variant, SVL_SÃ—SVL_S in case of single-precision variant or SVL_DÃ—SVL_D in case of double-precision variant, is then destructively added to the destination tile. This is equivalent to performing a single multiply-accumulate to each of the destination tile elements.
This instruction follows SME ZA-targeting floating-point behaviors.
ID_AA64SMFR0_EL1.F64F64 indicates whether the double-precision variant is implemented, and ID_AA64SMFR0_EL1.F16F16 indicates whether the half-precision variant is implemented.
It has encodings from 3 classes: Half-precision , Single-precision and Double-precision

Half-precision
(FEAT_SME_F16F16)

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

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

```
if !HaveSME2() || !IsFeatureImplemented(FEAT_SME_F16F16) then UNDEFINED
constant integer esize = 16;
integer a = UInt(Pn);
integer b = UInt(Pm);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer da = UInt(ZAda);
boolean sub_op = FALSE;
```

Single-precision
(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	0	0	0	0	0	0	1	0	0	Zm					Pm			Pn			Zn					0	0	0	ZAda	
																											S				

```
FMOPA <ZAda>.S, <Pn>/M, <Pm>/M, <Zn>.S, <Zm>.S
```

```
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;
```

Double-precision
(FEAT_SME_F64F64)

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

```
FMOPA <ZAda>.D, <Pn>/M, <Pm>/M, <Zn>.D, <Zm>.D
```

```
if !HaveSMEF64F64() 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;
```

Assembler Symbols

<ZAda>	For the half-precision variant: is the name of the ZA tile ZA0-ZA1, encoded in the "ZAda" field. For the single-precision variant: is the name of the ZA tile ZA0-ZA3, encoded in the "ZAda" field. For the double-precision 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;

for row = 0 to dim-1
    for col = 0 to dim-1
        bits(esize) element1 = Elem[operand1, row, esize];
        bits(esize) element2 = Elem[operand2, col, esize];
        bits(esize) element3 = Elem[operand3, row*dim+col, esize];

        if (ActivePredicateElement(mask1, row, esize) &&
              ActivePredicateElement(mask2, col, esize)) then
            if sub_op then element1 = FPNeg(element1);
            Elem[result, row*dim+col, esize] = FPMulAdd_ZA(element3, el
        else
            Elem[result, row*dim+col, esize] = element3;

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