SQRDMLAH (vector)

Signed Saturating Rounding Doubling Multiply Accumulate returning High Half (vector). This instruction multiplies the vector elements of the first source SIMD&FP register with the corresponding vector elements of the second source SIMD&FP register without saturating the multiply results, doubles the results, and accumulates the most significant half of the final results with the vector elements of the destination SIMD&FP register. The results are rounded.

If any of the results overflow, they are saturated. The cumulative saturation bit, *FPSR*.QC, is set if saturation occurs.

Depending on the settings in the *CPACR_EL1*, *CPTR_EL2*, and *CPTR_EL3* registers, and the current Security state and Exception level, an attempt to execute the instruction might be trapped.

It has encodings from 2 classes: Scalar and Vector

Scalar

(FEAT_RDM)

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	1	1	1	1	1	0	size		0	Rm					1	0	0	0	0	1	Rn					Rd				
																				S											

SQRDMLAH <V><d>, <V><n>, <V><m>

```
if !IsFeatureImplemented(FEAT_RDM) then UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rm);
if size == '11' || size == '00' then UNDEFINED;
constant integer esize = 8 << UInt(size);
constant integer datasize = esize;
integer elements = 1;
boolean rounding = TRUE;
boolean sub_op = (S == '1');
```

Vector

(FEAT_RDM)

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	Q	1	0	1	1	1	0	size		0	Rm					1	0	0	0	0	1	Rn					Rd				
																				S											

SQRDMLAH <Vd>.<T>, <Vn>.<T>, <Vm>.<T>

```
if !IsFeatureImplemented(FEAT_RDM) then UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
```

```
integer m = UInt(Rm);
if size == '11' || size == '00' then UNDEFINED;
constant integer esize = 8 << UInt(size);
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
boolean rounding = TRUE;
boolean sub_op = (S == '1');
```

Assembler Symbols

<V>

Is a width specifier, encoded in “size”:

size	<V>
00	RESERVED
01	H
10	S
11	RESERVED

<d> Is the number of the SIMD&FP destination register, in the "Rd" field.

<n> Is the number of the first SIMD&FP source register, encoded in the "Rn" field.

<m> Is the number of the second SIMD&FP source register, encoded in the "Rm" field.

<Vd> Is the name of the SIMD&FP destination register, encoded in the "Rd" field.

<T>

Is an arrangement specifier, encoded in “size:Q”:

size	Q	<T>
00	x	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	x	RESERVED

<Vn> Is the name of the first SIMD&FP source register, encoded in the "Rn" field.

<Vm> Is the name of the second SIMD&FP source register, encoded in the "Rm" field.

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = V[n, datasize];
bits(datasize) operand2 = V[m, datasize];
bits(datasize) operand3 = V[d, datasize];
bits(datasize) result;
```

```
    integer element1;
    integer element2;
    integer element3;
    integer accum;
    boolean sat;

    for e = 0 to elements-1
        element1 = SInt(Elem[operand1, e, esize]);
        element2 = SInt(Elem[operand2, e, esize]);
        element3 = SInt(Elem[operand3, e, esize]);
        if sub_op then
            accum = (element3 << esize) - 2 * (element1 * element2);
        else
            accum = (element3 << esize) + 2 * (element1 * element2);
        accum = RShr(accum, esize, rounding);
        (Elem[result, e, esize], sat) = SignedSatQ(accum, esize);
        if sat then FPSR.QC = '1';

    V[d, datasize] = 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 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