Index by

Sh

Pseu

SIMD&FP Base **Instructions Instructions**

SVE Instructions

SME Instructions

Encoding

SQRDMLAH (by element)

Signed Saturating Rounding Doubling Multiply Accumulate returning High Half (by element). This instruction multiplies the vector elements of the first source SIMD&FP register with the value of a vector element 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 1 size L M
                                       1 1 0 1 H 0
                                                                       Rd
                                Rm
                                                           Rn
```

SQRDMLAH <V><d>, <V><n>, <Vm>.<Ts>[<index>]

```
if !IsFeatureImplemented(FEAT_RDM) then UNDEFINED;
constant integer idxdsize = 64 << UInt(H);</pre>
integer index;
bit Rmhi;
case size of
    when '01' index = \underline{UInt} (H:L:M); Rmhi = '0';
    when '10' index = \overline{UInt}(H:L); Rmhi = M;
    otherwise UNDEFINED;
integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = <u>UInt</u>(Rmhi:Rm);
constant integer esize = 8 << UInt(size);</pre>
constant integer datasize = esize;
integer elements = 1;
boolean rounding = TRUE;
boolean sub_op = (S == '1');
```

Vector (FEAT_RDM)

0 Q 1 0 1 1 1 1 size L M Rm 1 1 0 1 H 0 Rn Rd	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	1	size	L	М	Rm	1	1	0	1	Н	0			Rn					Rd		

SQRDMLAH <Vd>.<T>, <Vn>.<T>, <Vm>.<Ts>[<index>]

```
if !IsFeatureImplemented(FEAT_RDM) then UNDEFINED;
constant integer idxdsize = 64 << UInt(H);</pre>
integer index;
bit Rmhi;
case size of
    when '01' index = UInt(H:L:M); Rmhi = '0';
    when '10' index = UInt(H:L); Rmhi = M;
    otherwise UNDEFINED;
integer d = <u>UInt</u>(Rd);
integer n = UInt(Rn);
integer m = <u>UInt</u>(Rmhi:Rm);
constant integer esize = 8 << UInt(size);</pre>
constant integer datasize = 64 << UInt(Q);</pre>
integer elements = datasize DIV esize;
boolean rounding = TRUE;
boolean sub_op = (S == '1');
```

Assembler Symbols

<V>

<Vd>

<T>

Is a width specifier, encoded in "size":

size	<v></v>
0.0	RESERVED
01	Н
10	S
11	RESERVED

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

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

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

Is an arrangement specifier, encoded in "size:Q":

size	Q	<t></t>
00	Х	RESERVED
01	0	4H
01	1	8H
10	0	2S
10	1	4S
11	Х	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 "size:M:Rm":

size	<vm></vm>
0.0	RESERVED
01	0:Rm
10	M:Rm
11	RESERVED

Restricted to V0-V15 when element size <Ts> is H.

<Ts>

Is an element size specifier, encoded in "size":

size	<ts></ts>
0.0	RESERVED
01	Н
10	S
11	RESERVED

<index>

Is the element index, encoded in "size:L:H:M":

size	<index></index>
00	RESERVED
01	H:L:M
10	H:L
11	RESERVED

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = \underline{V}[n, datasize];
bits(idxdsize) operand2 = V[m, idxdsize];
bits(datasize) operand3 = V[d, datasize];
bits(datasize) result;
integer element1;
integer element2;
integer element3;
integer accum;
boolean sat;
element2 = SInt(Elem[operand2, index, esize]);
for e = 0 to elements-1
    element1 = <u>SInt(Elem[operand1, e, esize]);</u>
    element3 = SInt(Elem[operand3, e, esize]);
    if sub_op then
        accum = (element3 << esize) - 2 * (element1 * element2);</pre>
        accum = (element3 << esize) + 2 * (element1 * element2);</pre>
```

```
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

Copyright © 2010-2023 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

Sh Pseu