## SQDMULH (vector)

Signed saturating Doubling Multiply returning High half. This instruction multiplies the values of corresponding elements of the two source SIMD&FP registers, doubles the results, places the most significant half of the final results into a vector, and writes the vector to the destination SIMD&FP register.

The results are truncated. For rounded results, see *SQRDMULH*.

If overflow occurs with any of the results, those results are saturated. If saturation occurs, the cumulative saturation bit *FPSR*.QC is set.

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

| 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 | 0 | 1 | 1 | 1 | 1 | 0 | size | 1 | | Rm | 1 | 0 | 1 | 1 | 0 | 1 | Rn | Rd |

U

        SQDMULH **<V><d>**, **<V><n>**, **<V><m>**

```
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 = (U == '1');
```

### Vector

| 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 | 0 | 0 | 1 | 1 | 1 | 0 | size | 1 | | Rm | 1 | 0 | 1 | 1 | 0 | 1 | Rn | Rd |

U

        SQDMULH **<Vd>.<T>**, **<Vn>.<T>**, **<Vm>.<T>**

```
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 = (U == '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) result;
integer element1;
integer element2;
integer product;
boolean sat;

for e = 0 to elements-1
    element1 = SInt(Elem[operand1, e, esize]);
    element2 = SInt(Elem[operand2, e, esize]);
    product = 2 * element1 * element2;
```

```
        product = RShr(product, esize, rounding);
        (Elem[result, e, esize], sat) = SignedSatQ(product, 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.

---