SQDMULL, SQDMULL2 (by element)

Signed saturating Doubling Multiply Long (by element). This instruction multiplies each vector element in the lower or upper half of the first source SIMD&FP register by the specified vector element of the second source SIMD&FP register, doubles the results, places the final results in a vector, and writes the vector to the destination SIMD&FP register. All the values in this instruction are signed integer values.

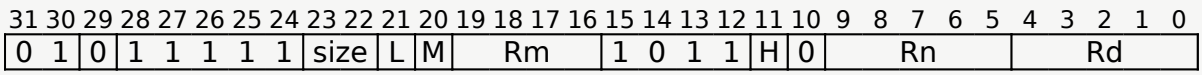
If overflow occurs with any of the results, those results are saturated. If saturation occurs, the cumulative saturation bit *FPSR*.QC is set.

The SQDMULL instruction extracts the first source vector from the lower half of the first source register. The SQDMULL2 instruction extracts the first source vector from the upper half of the first source register.

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



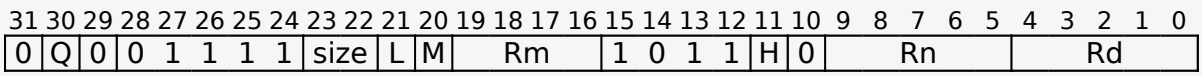
SQDMULL <Va><d>, <Vb><n>, <Vm>.<Ts>[<index>]

```
constant integer idxdsize = 64 << UInt(H);
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 = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rmhi:Rm);

constant integer esize = 8 << UInt(size);
constant integer datasize = esize;
integer elements = 1;
integer part = 0;
```

Vector



SQDMULL{2} <Vd>.<Ta>, <Vn>.<Tb>, <Vm>.<Ts>[<index>]

```
constant integer idxdsize = 64 << UInt(H);
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 = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rmhi:Rm);

constant integer esize = 8 << UInt(size);
constant integer datasize = 64;
integer part = UInt(Q);
integer elements = datasize DIV esize;
```

Assembler Symbols

2

Is the second and upper half specifier. If present it causes the operation to be performed on the upper 64 bits of the registers holding the narrower elements, and is encoded in “Q”:

Q	2
0	[absent]
1	[present]

<Vd>

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

<Ta>

Is an arrangement specifier, encoded in “size”:

size	<Ta>
00	RESERVED
01	4S
10	2D
11	RESERVED

<Vn>

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

<Tb>

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

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

<Va>

Is the destination width specifier, encoded in “size”:

size	<Va>
00	RESERVED
01	S
10	D
11	RESERVED

<d>

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

<Vb>

Is the source width specifier, encoded in “size”:

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

<n>

Is the number 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>
00	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>
00	RESERVED
01	H
10	S
11	RESERVED

<index>

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

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

Operation

```
CheckFPAdvSIMDEnabled64();

bits(datasize) operand1 = Vpart[n, part, datasize];
bits(idxdsize) operand2 = V[m, idxdsize];
bits(2*datasize) result;
integer element1;
integer element2;
bits(2*esize) product;
boolean sat;

element2 = SInt(Elem[operand2, index, esize]);
for e = 0 to elements-1
    element1 = SInt(Elem[operand1, e, esize]);
    (product, sat) = SignedSatQ(2 * element1 * element2, 2 * esize);
    Elem[result, e, 2*esize] = product;
    if sat then FPSR.QC = '1';

V[d, 2*datasize] = result;
```


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