SQSHL (immediate)

Signed saturating Shift Left (immediate). This instruction reads each vector element in the source SIMD&FP register, shifts each result by an immediate value, places the final result in a vector, and writes the vector to the destination SIMD&FP register. The results are truncated. For rounded results, see *UQRSHL*.

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	1	0		!= 0000			immb			0	1	1	1	0	1			Rn					Rd		

U   immh   op

```
        SQSHL <V><d>, <V><n>, #<shift>


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

    if immh == '0000' then UNDEFINED;
    constant integer esize = 8 << HighestSetBit(immh);
    constant integer datasize = esize;
    integer elements = 1;

    integer shift = UInt(immh:immb) - esize;

    boolean src_unsigned;
    boolean dst_unsigned;
    case op:U of
        when '00' UNDEFINED;
        when '01' src_unsigned = FALSE; dst_unsigned = TRUE;
        when '10' src_unsigned = FALSE; dst_unsigned = FALSE;
        when '11' src_unsigned = TRUE; dst_unsigned = TRUE;
```

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	1	0		!= 0000			immb			0	1	1	1	0	1			Rn					Rd		

U   immh   op

```
        SQSHL <Vd>.<T>, <Vn>.<T>, #<shift>


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

```
        if immh == '0000' then SEE(asimdimm);
        if immh<3>:Q == '10' then UNDEFINED;
        constant integer esize = 8 << HighestSetBit(immh);
        constant integer datasize = 64 << UInt(Q);
        integer elements = datasize DIV esize;

        integer shift = UInt(immh:immb) - esize;

        boolean src_unsigned;
        boolean dst_unsigned;
        case op:U of
            when '00' UNDEFINED;
            when '01' src_unsigned = FALSE; dst_unsigned = TRUE;
            when '10' src_unsigned = FALSE; dst_unsigned = FALSE;
            when '11' src_unsigned = TRUE; dst_unsigned = TRUE;
```

Assembler Symbols

<V>

Is a width specifier, encoded in "immh":

immh	<V>
0000	RESERVED
0001	B
001x	H
01xx	S
1xxx	D

<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.

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

<T>

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

immh	Q	<T>
0000	x	SEE Advanced SIMD modified immediate
0001	0	8B
0001	1	16B
001x	0	4H
001x	1	8H
01xx	0	2S
01xx	1	4S
1xxx	0	RESERVED
1xxx	1	2D

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

<shift>

> For the scalar variant: is the left shift amount, in the range 0 to the operand width in bits minus 1, encoded in "immh:immb":

immh	<shift>
0000	RESERVED
0001	(UInt(immh:immb)−8)
001x	(UInt(immh:immb)−16)
01xx	(UInt(immh:immb)−32)
1xxx	(UInt(immh:immb)−64)

> For the vector variant: is the left shift amount, in the range 0 to the element width in bits minus 1, encoded in "immh:immb":

immh	<shift>
0000	SEE Advanced SIMD modified immediate
0001	(UInt(immh:immb)−8)
001x	(UInt(immh:immb)−16)
01xx	(UInt(immh:immb)−32)
1xxx	(UInt(immh:immb)−64)

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand = V[n, datasize];
bits(datasize) result;
integer element;
boolean sat;

for e = 0 to elements-1
    element = Int(Elem[operand, e, esize], src_unsigned) << shift;
    (Elem[result, e, esize], sat) = SatQ(element, esize, dst_unsigned);
    if sat then FPSR.QC = '1';

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