UQRSHRN, UQRSHRN2

Unsigned saturating Rounded Shift Right Narrow (immediate). This instruction reads each vector element in the source SIMD&FP register, right shifts each result by an immediate value, puts the final result into a vector, and writes the vector to the lower or upper half of the destination SIMD&FP register. All the values in this instruction are unsigned integer values. The results are rounded. For truncated results, see *UQSHRN*.

The `UQRSHRN` instruction writes the vector to the lower half of the destination register and clears the upper half, while the `UQRSHRN2` instruction writes the vector to the upper half of the destination register without affecting the other bits of the register.

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	1	1	1	1	1	1	0	!= 0000	immb	1	0	0	1	1	1	Rn	Rd
		U							immh						op			

UQRSHRN <Vb><d>, <Va><n>, #<shift>

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

if immh == '0000' then UNDEFINED;
if immh<3> == '1' then UNDEFINED;
constant integer esize = 8 << HighestSetBit(immh);
constant integer datasize = esize;
integer elements = 1;
integer part = 0;

integer shift = (2 * esize) - UInt(immh:immb);
boolean round = (op == '1');
boolean unsigned = (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	1	0	1	1	1	1	0	!= 0000	immb	1	0	0	1	1	1	Rn	Rd
		U							immh						op			

```
UQRSHRN{2} <Vd>.<Tb>, <Vn>.<Ta>, #<shift>
```

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

if immh == '0000' then SEE(asimdimm);
if immh<3> == '1' then UNDEFINED;
constant integer esize = 8 << HighestSetBit(immh);
constant integer datasize = 64;
integer part = UInt(Q);
integer elements = datasize DIV esize;

integer shift = (2 * esize) - UInt(immh:immb);
boolean round = (op == '1');
boolean unsigned = (U == '1');
```

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.

<Tb>

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

immh	Q	<Tb>
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	x	RESERVED

<Vn>

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

<Ta>

Is an arrangement specifier, encoded in “immh”:

immh	<Ta>
0000	SEE Advanced SIMD modified immediate
0001	8H
001x	4S
01xx	2D
1xxx	RESERVED

<Vb>

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

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

<d>

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

<Va>

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

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

<n>

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

<shift>

For the scalar variant: is the right shift amount, in the range 1 to the destination operand width in bits, encoded in “immh:immb”:

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

For the vector variant: is the right shift amount, in the range 1 to the destination element width in bits, encoded in “immh:immb”:

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

Operation

```
CheckFPAdvSIMDEnabled64();
bits(datasize*2) operand = V[n, datasize*2];
bits(datasize) result;
integer element;
boolean sat;

for e = 0 to elements-1
    element = RShr(Int(Elem[operand, e, 2*esize], unsigned), shift, rou
    (Elem[result, e, esize], sat) = SatQ(element, esize, unsigned);
    if sat then FPSR.QC = '1';

Vpart[d, part, 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