FMINQV

Floating-point minimum recursive reduction of quadword vector segments

Floating-point minimum of the same element numbers from each 128-bit source vector segment using a recursive pairwise reduction, placing each result into the corresponding element number of the 128-bit SIMD&FP destination register. Inactive elements in the source vector are treated as +Infinity.
When FPCR.AH is 0, the behavior is as follows:

- Negative zero compares less than positive zero.
- When FPCR.DN is 0, if either value is a NaN, the result is a quiet NaN.
- When FPCR.DN is 1, if either value is a NaN, the result is Default NaN.

When FPCR.AH is 1, the behavior is as follows:

- If both values are zeros, regardless of the sign of either zero, the result is the second value.
- If either value is a NaN, regardless of the value of FPCR.DN, the result is the second value.

SVE2
(FEAT_SVE2p1)

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	0	0	1	0	0	size		0	1	0	1	1	1	1	0	1	Pg			Zn					Vd				

FMINQV <Vd>.<T>, <Pg>, <Zn>.<Tb>

```
if !HaveSVE2p1() && !HaveSME2p1() then UNDEFINED;
if size == '00' then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Vd);
```

Assembler Symbols

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

<T>

Is an arrangement specifier, encoded in “size”:

size	<T>
00	RESERVED
01	8H
10	4S
11	2D

<Pg>

Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Zn>

Is the name of the source scalable vector register, encoded in the "Zn" field.

<Tb>

Is the size specifier, encoded in “size”:

size	<Tb>
00	RESERVED
01	H
10	S
11	D

Operation

```
CheckSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer segments = VL DIV 128;
constant integer elempersegment = 128 DIV esize;
bits(PL) mask = P[g, PL];
bits(VL) operand = if AnyActiveElement(mask, esize) then Z[n, VL] else
bits(esize) identity = FPInfinity('0', esize);
bits(128) result = Zeros(128);

constant integer p2bits = CeilPow2(segments*esize);
constant integer p2elems = p2bits DIV esize;

for e = 0 to elempersegment-1
    bits(p2bits) stmp;
    bits(esize) dtmp;
    for s = 0 to p2elems-1
        if s < segments && ActivePredicateElement(mask, s * elempersegm
            Elem[stmp, s, esize] = Elem[operand, s * elempersegment + e
        else
            Elem[stmp, s, esize] = identity;
    dtmp = Reduce(ReduceOp_FMIN, stmp, esize);
    Elem[result, e, esize] = dtmp;
V[d, 128] = 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