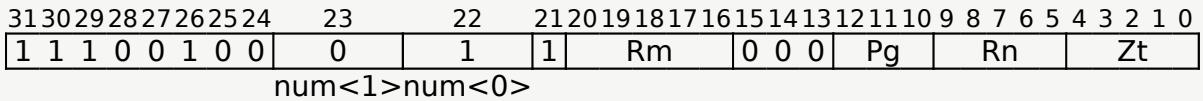
ST2Q (scalar plus scalar)

Contiguous store two-quadword structures from two vectors (scalar index)

Contiguous store two-quadword structures, each from the same element number in two vector registers to the memory address generated by a 64-bit scalar base and a 64-bit scalar index register scaled by the element size (LSL option) and added to the base address. After each structure access the index value is incremented by two. The index register is not updated by the instruction.
Each predicate element applies to the same element number in each of the two vector registers, or equivalently to the two consecutive quadwords in memory which make up each structure. Inactive structures are not written to memory.

SVE
(FEAT_SVE2p1)



ST2Q { <Zt1>.Q, <Zt2>.Q }, <Pg>, [<Xn|SP>, <Xm>, LSL #4]

```
if !HaveSVE2p1() && !HaveSME2p1() then UNDEFINED;
if Rm == '11111' then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer g = UInt(Pg);
constant integer esize = 128;
constant integer nreg = 2;
```

Assembler Symbols

<Zt1>	Is the name of the first scalable vector register to be transferred, encoded in the "Zt" field.
<Zt2>	Is the name of the second scalable vector register to be transferred, encoded as "Zt" plus 1 modulo 32.
<Pg>	Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
<Xn|SP>	Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.
<Xm>	Is the 64-bit name of the general-purpose offset register, encoded in the "Rm" field.

Operation

```
CheckSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;
bits(64) base;
bits(PL) mask = P[g, PL];
bits(64) offset;
constant integer mbytes = esize DIV 8;
array [0..1] of bits(VL) values;
boolean contiguous = TRUE;
boolean nontemporal = FALSE;
boolean tagchecked = TRUE;
AccessDescriptor accdesc = CreateAccDescSVE(MemOp_STORE, nontemporal, c

if !AnyActiveElement(mask, esize) then
    if n == 31 && ConstrainUnpredictableBool(Unpredictable_CHECKSPNONEA
        CheckSPAlignment();
else
    if n == 31 then CheckSPAlignment();
    base = if n == 31 then SP[] else X[n, 64];
    offset = X[m, 64];

for r = 0 to nreg-1
    values[r] = Z[(t+r) MOD 32, VL];

for e = 0 to elements-1
    for r = 0 to nreg-1
        if ActivePredicateElement(mask, e, esize) then
            integer eoff = UInt(offset) + (e * nreg) + r;
            bits(64) addr = base + eoff * mbytes;
            Mem[addr, mbytes, accdesc] = Elem[values[r], e, esize];
```

Operational information

If PSTATE.DIT is 1, the timing of this instruction is insensitive to the value of the data being loaded or stored when its governing predicate register contains the same value for each execution.

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