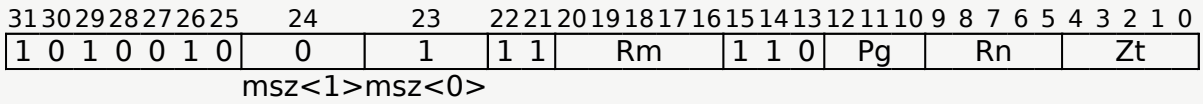
LD4H (scalar plus scalar)

Contiguous load four-halfword structures to four vectors (scalar index)

Contiguous load four-halfword structures, each to the same element number in four vector registers from 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 four. The index register is not updated by the instruction.

Each predicate element applies to the same element number in each of the four vector registers, or equivalently to the four consecutive halfwords in memory which make up each structure. Inactive elements will not cause a read from Device memory or signal a fault, and the corresponding element is set to zero in each of the four destination vector registers.



LD4H { <Zt1>.H, <Zt2>.H, <Zt3>.H, <Zt4>.H }, <Pg>/Z, [<Xn|SP>, <Xm>,

```
if !HaveSVE() && !HaveSME() 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 = 16;
constant integer nreg = 4;
```

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.
<Zt3>	Is the name of the third scalable vector register to be transferred, encoded as "Zt" plus 2 modulo 32.
<Zt4>	Is the name of the fourth scalable vector register to be transferred, encoded as "Zt" plus 3 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..3] of bits(VL) values;
boolean contiguous = TRUE;
boolean nontemporal = FALSE;
boolean tagchecked = TRUE;
AccessDescriptor accdesc = CreateAccDescSVE(MemOp_LOAD, nontemporal, co

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 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;
            Elem[values[r], e, esize] = Mem[addr, mbytes, accdesc];
        else
            Elem[values[r], e, esize] = Zeros(esize);

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

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then 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