LDNT1D (scalar plus immediate, strided registers)

Contiguous load non-temporal of doublewords to multiple strided vectors (immediate index)

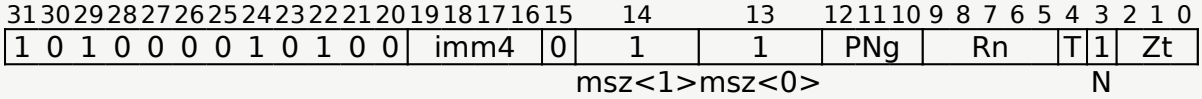
Contiguous load non-temporal of doublewords to elements of two or four strided vector registers from the memory address generated by a 64-bit scalar base and immediate index which is multiplied by the vector's in-memory size, irrespective of predication, and added to the base address.

Inactive elements will not cause a read from Device memory or signal a fault, and are set to zero in the destination vector.

A non-temporal load is a hint to the system that this data is unlikely to be referenced again soon.

It has encodings from 2 classes: Two registers and Four registers

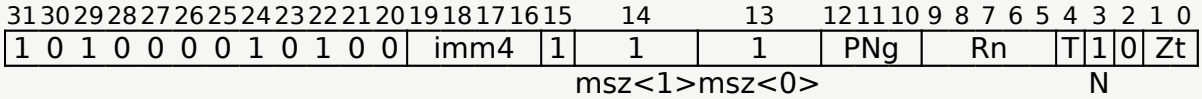
Two registers
(FEAT_SME2)



LDNT1D { <Zt1>.D, <Zt2>.D }, <PNg>/Z, [<Xn|SP>{, #<imm>, MUL VL}]

```
if !HaveSME2() then UNDEFINED;
integer n = UInt(Rn);
integer g = UInt('1':PNg);
constant integer nreg = 2;
integer tstride = 8;
integer t = UInt(T:'0':Zt);
constant integer esize = 64;
integer offset = SInt(imm4);
```

Four registers
(FEAT_SME2)



LDNT1D { <Zt1>.D, <Zt2>.D, <Zt3>.D, <Zt4>.D }, <PNg>/Z, [<Xn|SP>{,

```
if !HaveSME2() then UNDEFINED;
integer n = UInt(Rn);
integer g = UInt('1':PNg);
constant integer nreg = 4;
integer tstride = 4;
integer t = UInt(T:'00':Zt);
constant integer esize = 64;
integer offset = SInt(imm4);
```

Assembler Symbols

<Zt1>	For the two registers variant: is the name of the first scalable vector register Z0-Z7 or Z16-Z23 to be transferred, encoded as "T:'0':Zt". For the four registers variant: is the name of the first scalable vector register Z0-Z3 or Z16-Z19 to be transferred, encoded as "T:'00':Zt".
<Zt2>	For the two registers variant: is the name of the second scalable vector register Z8-Z15 or Z24-Z31 to be transferred, encoded as "T:'1':Zt". For the four registers variant: is the name of the second scalable vector register Z4-Z7 or Z20-Z23 to be transferred, encoded as "T:'01':Zt".
<Zt3>	Is the name of the third scalable vector register Z8-Z11 or Z24-Z27 to be transferred, encoded as "T:'10':Zt".
<Zt4>	Is the name of the fourth scalable vector register Z12-Z15 or Z28-Z31 to be transferred, encoded as "T:'11':Zt".
<PNg>	Is the name of the governing scalable predicate register PN8-PN15, with predicate-as-counter encoding, encoded in the "PNg" field.
<Xn|SP>	Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.
<imm>	For the two registers variant: is the optional signed immediate vector offset, a multiple of 2 in the range -16 to 14, defaulting to 0, encoded in the "imm4" field. For the four registers variant: is the optional signed immediate vector offset, a multiple of 4 in the range -32 to 28, defaulting to 0, encoded in the "imm4" field.

Operation

```
CheckStreamingSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;
constant integer mbytes = esize DIV 8;
bits(64) base;
bits(PL) pred = P[g, PL];
bits(PL * nreg) mask = CounterToPredicate(pred<15:0>, PL * nreg);
array [0..3] of bits(VL) values;
boolean contiguous = TRUE;
boolean nontemporal = TRUE;
boolean tagchecked = n != 31;
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];

for r = 0 to nreg-1
    for e = 0 to elements-1
        if ActivePredicateElement(mask, r * elements + e, esize) then
            bits(64) addr = base + (offset * nreg * elements + r * eleme
            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, VL] = values[r];
    t = t + tstride;
```

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