ST1H (scalar plus scalar, single register)

Contiguous store halfwords from vector (scalar index)

Contiguous store of halfwords from elements of a vector register to the memory address generated by a 64-bit scalar base and scalar index which is multiplied by 2 and added to the base address. After each element access the index value is incremented, but the index register is not updated. Inactive elements are not written to memory.

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
1 1 1 0 0 1 0	0 1	size Rm	0 1 0	Pg	Rn	Zt

ST1H { **<Zt>.<T>** }, **<Pg>**, [**<Xn|SP>**, **<Xm>**, LSL #1]

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
if size == '00' 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 = 8 << UInt(size);
constant integer msize = 16;
```

Assembler Symbols

<Zt>   Is the name of the scalable vector register to be transferred, encoded in the "Zt" field.

<T>

Is the size specifier, encoded in "size":

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

<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;
      bits(VL) src;
      constant integer mbytes = msize DIV 8;
      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];
          src = Z[t, VL];

      for e = 0 to elements-1
          if ActivePredicateElement(mask, e, esize) then
              bits(64) addr = base + (UInt(offset) + e) * mbytes;
              Mem[addr, mbytes, accdesc] = Elem[src, e, esize]<msize-1:0>;
```

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.

| Base Instructions | SIMD&FP Instructions | SVE Instructions | SME Instructions | Index by Encoding | Sh Pseu |