SPLICE

Splice two vectors under predicate control

Select a region from the first source vector and copy it to the lowest-numbered elements of the result. Then set any remaining elements of the result to a copy of the lowest-numbered elements from the second source vector. The region is selected using the first and last true elements in the vector select predicate register. The result is placed destructively in the destination and first source vector, or constructively in the destination vector.

The Destructive encoding of this instruction might be immediately preceded in program order by a MOVPRFX instruction. The MOVPRFX instruction must conform to all of the following requirements, otherwise the behavior of the MOVPRFX and this instruction is UNPREDICTABLE: The MOVPRFX instruction must be unpredicated. The MOVPRFX instruction must specify the same destination register as this instruction. The destination register must not refer to architectural register state referenced by any other source operand register of this instruction.

It has encodings from 2 classes: Constructive and Destructive

Constructive

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	0	0	0	0	1	0	1	size		1	0	1	1	0	1	1	0	0	Pv			Zn					Zd				

SPLICE <Zd>.<T>, <Pv>, { <Zn1>.<T>, <Zn2>.<T> }

```
if !HaveSVE2() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer v = UInt(Pv);
integer dst = UInt(Zd);
integer s1 = UInt(Zn);
integer s2 = (s1 + 1) MOD 32;
```

Destructive

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	0	0	0	0	1	0	1	size		1	0	1	1	0	0	1	0	0	Pv			Zm					Zdn				

SPLICE <Zdn>.<T>, <Pv>, <Zdn>.<T>, <Zm>.<T>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer v = UInt(Pv);
integer dst = UInt(Zdn);
integer s1 = dst;
integer s2 = UInt(Zm);
```

Assembler Symbols

<Zd>	Is the name of the destination scalable vector register, encoded in the "Zd" field.
<Zdn>	Is the name of the first source and destination scalable vector register, encoded in the "Zdn" field.
<T>	Is the size specifier, encoded in “size”:

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

<Pv>	Is the name of the vector select predicate register P0-P7, encoded in the "Pv" field.
<Zn1>	Is the name of the first scalable vector register of a multi-vector sequence, encoded in the "Zn" field.
<Zn2>	Is the name of the second scalable vector register of a multi-vector sequence, encoded in the "Zn" field.
<Zm>	Is the name of the second source scalable vector register, encoded in the "Zm" field.

Operation

```
CheckSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;
bits(PL) mask = P[v, PL];
bits(VL) operand1 = if AnyActiveElement(mask, esize) then Z[s1, VL] els
bits(VL) operand2 = Z[s2, VL];
bits(VL) result;
integer x = 0;
boolean active = FALSE;
constant integer lastnum = LastActiveElement(mask, esize);

if lastnum >= 0 then
    for e = 0 to lastnum
        active = active || ActivePredicateElement(mask, e, esize);
        if active then
            Elem[result, x, esize] = Elem[operand1, e, esize];
            x = x + 1;

constant integer nelements = (elements - x) - 1;
for e = 0 to nelements
    Elem[result, x, esize] = Elem[operand2, e, esize];
    x = x + 1;

Z[dst, VL] = result;
```