CMP<cc> (immediate)

Compare vector to immediate

Compare active integer elements in the source vector with an immediate, and place the boolean results of the specified comparison in the corresponding elements of the destination predicate. Inactive elements in the destination predicate register are set to zero. Sets the first (N), none (Z), !last (C) condition flags based on the predicate result, and the V flag to zero.

<cc>	Comparison
EQ	equal
GE	signed greater than or equal
GT	signed greater than
HI	unsigned higher than
HS	unsigned higher than or same
LE	signed less than or equal
LO	unsigned lower than
LS	unsigned lower than or same
LT	signed less than
NE	not equal

It has encodings from 10 classes: Equal , Greater than , Greater than or equal , Higher , Higher or same , Less than , Less than or equal , Lower , Lower or same and Not equal

Equal

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	1	0	0	1	0	1	size		0	imm5					1	0	0	Pg			Zn					0	Pd			
																											ne				

CMPEQ <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_EQ;
integer imm = SInt(imm5);
boolean unsigned = FALSE;
```

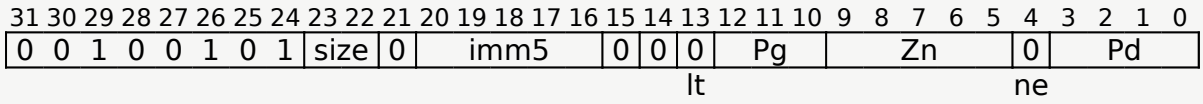
Greater than

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	1	0	0	1	0	1	size		0	imm5					0	0	0	Pg			Zn					1	Pd			
																		lt									ne				

CMPGT <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_GT;
integer imm = SInt(imm5);
boolean unsigned = FALSE;
```

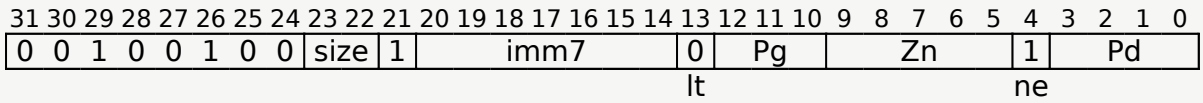
Greater than or equal



CMPGE <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_GE;
integer imm = SInt(imm5);
boolean unsigned = FALSE;
```

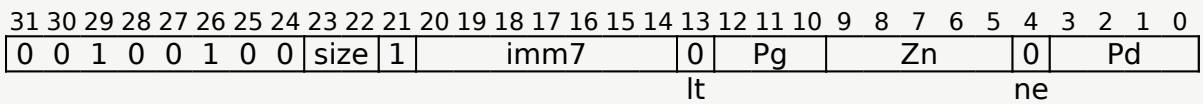
Higher



CMPHI <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_GT;
integer imm = UInt(imm7);
boolean unsigned = TRUE;
```

Higher or same

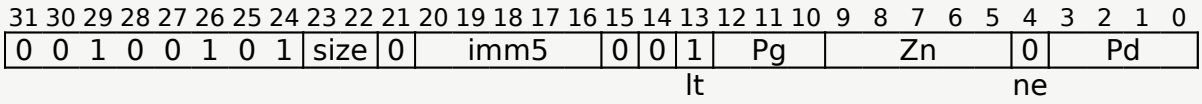


CMPHS <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
```

```
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_GE;
integer imm = UInt(imm7);
boolean unsigned = TRUE;
```

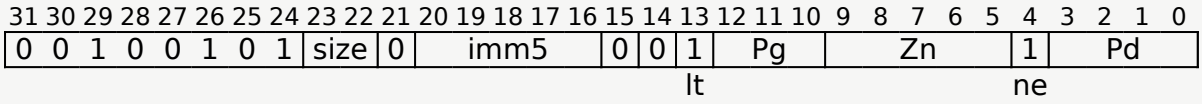
Less than



CMPLT <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_LT;
integer imm = SInt(imm5);
boolean unsigned = FALSE;
```

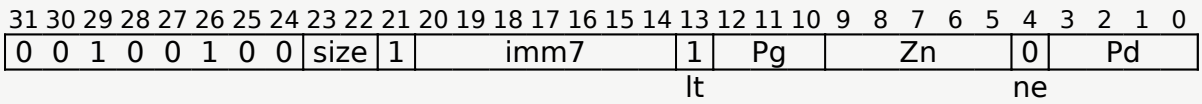
Less than or equal



CMPLE <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_LE;
integer imm = SInt(imm5);
boolean unsigned = FALSE;
```

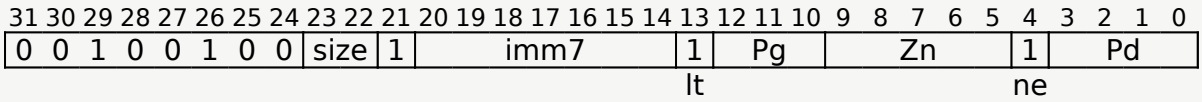
Lower



CMPLO <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_LT;
integer imm = UInt(imm7);
boolean unsigned = TRUE;
```

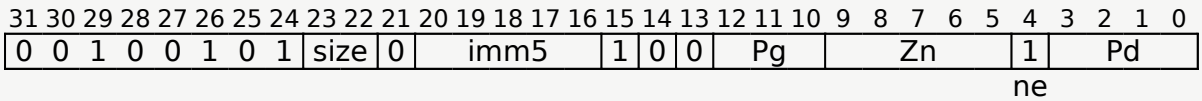
Lower or same



CMPLS <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_LE;
integer imm = UInt(imm7);
boolean unsigned = TRUE;
```

Not equal



CMPNE <Pd>.<T>, <Pg>/Z, <Zn>.<T>, #<imm>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Pd);
SVECmp op = Cmp_NE;
integer imm = SInt(imm5);
boolean unsigned = FALSE;
```

Assembler Symbols

<Pd>	Is the name of the destination scalable predicate register, encoded in the "Pd" field.

<T>

Is the size specifier, encoded in “size”:

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

<Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.

<Zn> Is the name of the source scalable vector register, encoded in the "Zn" field.

<imm> For the equal, greater than, greater than or equal, less than, less than or equal and not equal variant: is the signed immediate operand, in the range -16 to 15, encoded in the "imm5" field.

For the higher, higher or same, lower and lower or same variant: is the unsigned immediate operand, in the range 0 to 127, encoded in the "imm7" field.

Operation

```
CheckSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;
bits(PL) mask = P[g, PL];
bits(VL) operand1 = if AnyActiveElement(mask, esize) then Z[n, VL] else
bits(PL) result;
constant integer psize = esize DIV 8;

for e = 0 to elements-1
    integer element1 = Int(Elem[operand1, e, esize], unsigned);
    if ActivePredicateElement(mask, e, esize) then
        boolean cond;
        case op of
            when Cmp_EQ cond = element1 == imm;
            when Cmp_NE cond = element1 != imm;
            when Cmp_GE cond = element1 >= imm;
            when Cmp_LT cond = element1 <  imm;
            when Cmp_GT cond = element1 >  imm;
            when Cmp_LE cond = element1 <= imm;
        bit pbit = if cond then '1' else '0';
        Elem[result, e, psize] = ZeroExtend(pbit, psize);
    else
        Elem[result, e, psize] = ZeroExtend('0', psize);

PSTATE.<N,Z,C,V> = PredTest(mask, result, esize);
P[d, PL] = result;
```

Operational information

If FEAT_SVE2 is implemented or FEAT_SME is implemented, then if PSTATE.DIT is 1:

- The execution time of this instruction is independent of:
  - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
  - The values of the NZCV flags.
- The response of this instruction to asynchronous exceptions does not vary based on:
  - The values of the data supplied in any of its operand registers when its governing predicate register contains the same value for each execution.
  - The values of the NZCV flags.

If FEAT_SME is implemented and the PE is in Streaming SVE mode, then any subsequent instruction which is dependent on the predicate register or NZCV condition flags written by this instruction might be significantly delayed.

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