CMP<cc> (vectors)

Compare vectors

Compare active integer elements in the first source vector with corresponding elements in the second source vector, 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
NE	not equal

This instruction is used by the pseudo-instructions CMPLE (vectors), CMPLO (vectors), CMPLS (vectors), and CMPLT (vectors).

It has encodings from 6 classes: Equal , Greater than , Greater than or equal , Higher , Higher 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	0	size	0	Zm	1	0	1	Pg	Zn	0	Pd
																ne	

```
CMPEQ <Pd>.<T>, <Pg>/Z, <Zn>.<T>, <Zm>.<T>
```

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer d = UInt(Pd);
SVECmp op = Cmp_EQ;
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	0	size	0	Zm	1	0	0	Pg	Zn	1	Pd
																ne	

CMPGT <Pd>.<T>, <Pg>/Z, <Zn>.<T>, <Zm>.<T>

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

Greater than or 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	0	size		0	Zm					1	0	0	Pg			Zn					0	Pd			
																											ne				

CMPGE <Pd>.<T>, <Pg>/Z, <Zn>.<T>, <Zm>.<T>

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

Higher

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	0	size		0	Zm					0	0	0	Pg			Zn					1	Pd			
																											ne				

CMPHI <Pd>.<T>, <Pg>/Z, <Zn>.<T>, <Zm>.<T>

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

Higher or same

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	0	size		0	Zm					0	0	0	Pg			Zn					0	Pd			
																											ne				

CMPHS <Pd>.<T>, <Pg>/Z, <Zn>.<T>, <Zm>.<T>

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

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

Not 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	0	size		0	Zm					1	0	1	Pg			Zn					1	Pd			
																											ne				

CMPNE <Pd>.<T>, <Pg>/Z, <Zn>.<T>, <Zm>.<T>

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8 << UInt(size);
integer g = UInt(Pg);
integer n = UInt(Zn);
integer m = UInt(Zm);
integer d = UInt(Pd);
SVECmp op = Cmp_NE;
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 first source scalable vector register, 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[g, PL];
bits(VL) operand1 = if AnyActiveElement(mask, esize) then Z[n, VL] els
```

```
    bits(VL) operand2 = if AnyActiveElement(mask, esize) then Z[m, 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;
            integer element2 = Int(Elem[operand2, e, esize], unsigned);
            case op of
                when Cmp_EQ cond = element1 == element2;
                when Cmp_NE cond = element1 != element2;
                when Cmp_GE cond = element1 >= element2;
                when Cmp_LT cond = element1 <  element2;
                when Cmp_GT cond = element1 >  element2;
                when Cmp_LE cond = element1 <= element2;
            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.