

## CNTB, CNTD, CNTH, CNTW

Set scalar to multiple of predicate constraint element count

Determines the number of active elements implied by the named predicate constraint, multiplies that by an immediate in the range 1 to 16 inclusive, and then places the result in the scalar destination.

The named predicate constraint limits the number of active elements in a single predicate to:

- A fixed number (VL1 to VL256)
- The largest power of two (POW2)
- The largest multiple of three or four (MUL3 or MUL4)
- All available, implicitly a multiple of two (ALL).

Unspecified or out of range constraint encodings generate an empty predicate or zero element count rather than Undefined Instruction exception.

It has encodings from 4 classes: [Byte](#) , [Doubleword](#) , [Halfword](#) and [Word](#)

### Byte

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	0	0	0	1	0	imm4				1	1	1	0	0	0	pattern				Rd					
size<1>								size<0>																							

**CNTB** <Xd>{, <pattern>{, MUL #<imm>}}

```

if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 8;
integer d = UInt(Rd);
bits(5) pat = pattern;
integer imm = UInt(imm4) + 1;

```

### Doubleword

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	0	1	1	1	0	imm4				1	1	1	0	0	0	pattern				Rd					
size<1>								size<0>																							

**CNTD** <Xd>{, <pattern>{, MUL #<imm>}}

```

if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 64;
integer d = UInt(Rd);
bits(5) pat = pattern;
integer imm = UInt(imm4) + 1;

```

## Halfword

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	0	0	1	1	0	0	imm4	1	1	1	0	0	0	0	pattern							Rd			
								size<1>		size<0>																					

**CNTH** <Xd>{, <pattern>{, MUL #<imm>}}

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 16;
integer d = UInt(Rd);
bits(5) pat = pattern;
integer imm = UInt(imm4) + 1;
```

## Word

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	0	1	0	imm4	1	1	1	0	0	0	pattern	Rd																
								size<1>		size<0>																								

**CNTW** <Xd>{, <pattern>{, MUL #<imm>}}

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 32;
integer d = UInt(Rd);
bits(5) pat = pattern;
integer imm = UInt(imm4) + 1;
```

## Assembler Symbols

<Xd> Is the 64-bit name of the destination general-purpose register, encoded in the "Rd" field.

<pattern>

Is the optional pattern specifier, defaulting to ALL, encoded in "pattern":

pattern	<pattern>
00000	POW2
00001	VL1
00010	VL2
00011	VL3
00100	VL4
00101	VL5
00110	VL6
00111	VL7
01000	VL8
01001	VL16
01010	VL32
01011	VL64
01100	VL128
01101	VL256
0111x	#uimm5
101x1	#uimm5
10110	#uimm5
1x0x1	#uimm5
1x010	#uimm5
1xx00	#uimm5
11101	MUL4
11110	MUL3
11111	ALL

<imm>

Is the immediate multiplier, in the range 1 to 16, defaulting to 1, encoded in the "imm4" field.

## Operation

```
CheckSVEEnabled();  
integer count = DecodePredCount(pat, esize);  
X[d, 64] = (count * imm)<63:0>;
```

## 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 registers.
  - 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 registers.
  - The values of the NZCV flags.

[Base  
Instructions](#)

[SIMD&FP  
Instructions](#)

[SVE  
Instructions](#)

[SME  
Instructions](#)

[Index by  
Encoding](#)

[Sh  
Pseu](#)

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

Copyright Â© 2010-2023 Arm Limited or its affiliates. All rights reserved. This  
document is Non-Confidential.