FCVTZS

Floating-point convert to signed integer, rounding toward zero (predicated)

Convert to the signed integer nearer to zero from each active floating-point element of the source vector, and place the results in the corresponding elements of the destination vector. Inactive elements in the destination vector register remain unmodified.

If the input and result types have a different size the smaller type is held unpacked in the least significant bits of elements of the larger size. When the input is the smaller type the upper bits of each source element are ignored. When the result is the smaller type the results are sign-extended to fill each destination element.

It has encodings from 7 classes: Half-precision to 16-bit , Half-precision to 32-bit , Half-precision to 64-bit , Single-precision to 32-bit , Single-precision to 64-bit , Double-precision to 32-bit and Double-precision to 64-bit

Half-precision to 16-bit

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	1	1	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	Pg			Zn					Zd				
															int_U																

```
FCVTZS <Zd>.H, <Pg>/M, <Zn>.H
```

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 16;
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Zd);
constant integer s_esize = 16;
constant integer d_esize = 16;
boolean unsigned = FALSE;
FPRounding rounding = FPRounding_ZERO;
```

Half-precision to 32-bit

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	1	1	0	0	1	0	1	0	1	0	1	1	1	0	0	1	0	1	Pg			Zn					Zd				
															int_U																

```
FCVTZS <Zd>.S, <Pg>/M, <Zn>.H
```

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 32;
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Zd);
constant integer s_esize = 16;
```

```
    constant integer d_esize = 32;
    boolean unsigned = FALSE;
    FPRounding rounding = FPRounding_ZERO;
```

Half-precision to 64-bit

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	1	1	0	0	1	0	1	0	1	0	1	1	1	1	0	1	0	1	Pg			Zn					Zd				
															int_U																

FCVTZS <Zd>.D, <Pg>/M, <Zn>.H

```
    if !HaveSVE() && !HaveSME() then UNDEFINED;
    constant integer esize = 64;
    integer g = UInt(Pg);
    integer n = UInt(Zn);
    integer d = UInt(Zd);
    constant integer s_esize = 16;
    constant integer d_esize = 64;
    boolean unsigned = FALSE;
    FPRounding rounding = FPRounding_ZERO;
```

Single-precision to 32-bit

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	1	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	Pg			Zn					Zd				
															int_U																

FCVTZS <Zd>.S, <Pg>/M, <Zn>.S

```
    if !HaveSVE() && !HaveSME() then UNDEFINED;
    constant integer esize = 32;
    integer g = UInt(Pg);
    integer n = UInt(Zn);
    integer d = UInt(Zd);
    constant integer s_esize = 32;
    constant integer d_esize = 32;
    boolean unsigned = FALSE;
    FPRounding rounding = FPRounding_ZERO;
```

Single-precision to 64-bit

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	1	1	0	0	1	0	1	1	1	0	1	1	1	0	0	1	0	1	Pg			Zn					Zd				
															int_U																

FCVTZS <Zd>.D, <Pg>/M, <Zn>.S

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 64;
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Zd);
constant integer s_esize = 32;
constant integer d_esize = 64;
boolean unsigned = FALSE;
FPRounding rounding = FPRounding_ZERO;
```

Double-precision to 32-bit

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	1	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0	1	Pg			Zn					Zd				
															int_U																

FCVTZS <Zd>.S, <Pg>/M, <Zn>.D

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 64;
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Zd);
constant integer s_esize = 64;
constant integer d_esize = 32;
boolean unsigned = FALSE;
FPRounding rounding = FPRounding_ZERO;
```

Double-precision to 64-bit

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	1	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0	1	Pg			Zn					Zd				
															int_U																

FCVTZS <Zd>.D, <Pg>/M, <Zn>.D

```
if !HaveSVE() && !HaveSME() then UNDEFINED;
constant integer esize = 64;
integer g = UInt(Pg);
integer n = UInt(Zn);
integer d = UInt(Zd);
constant integer s_esize = 64;
constant integer d_esize = 64;
boolean unsigned = FALSE;
FPRounding rounding = FPRounding_ZERO;
```

Assembler Symbols

<Zd>	Is the name of the destination scalable vector register, encoded in the "Zd" field.
<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.

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) operand  = if AnyActiveElement(mask, esize) then Z[n, VL] else
bits(VL) result = Z[d, VL];

for e = 0 to elements-1
    if ActivePredicateElement(mask, e, esize) then
        bits(esize) element = Elem[operand, e, esize];
        bits(d_esize) res = FPToFixed(element<s_esize-1:0>, 0, unsigned
        Elem[result, e, esize] = Extend(res, esize, unsigned);

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

Operational information

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, or be predicated using the same governing predicate register and source element size as this instruction.
- 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.

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