

LDFF1SH (scalar plus vector)

Gather load first-fault signed halfwords to vector (vector index)

Gather load with first-faulting behavior of signed halfwords to active elements of a vector register from memory addresses generated by a 64-bit scalar base plus vector index. The index values are optionally first sign or zero-extended from 32 to 64 bits and then optionally multiplied by 2. Inactive elements will not cause a read from Device memory or signal faults, and are set to zero in the destination vector.

This instruction is illegal when executed in Streaming SVE mode, unless FEAT_SME_FA64 is implemented and enabled.

It has encodings from 6 classes: [32-bit scaled offset](#) , [32-bit unpacked scaled offset](#) , [32-bit unpacked unscaled offset](#) , [32-bit unscaled offset](#) , [64-bit scaled offset](#) and [64-bit unscaled offset](#)

32-bit scaled offset

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
1	0	0	0	0	1	0	0	1	xs	1				Zm		0	0	1		Pg						Rn				Zt	
																U ff															

LDFF1SH { **<Zt>.S** }, **<Pg>/Z**, [**<Xn|SP>**, **<Zm>.S**, **<mod> #1**]

```
if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Zm);
integer g = UInt(Pg);
constant integer esize = 32;
constant integer msize = 16;
constant integer offs_size = 32;
boolean unsigned = FALSE;
boolean offs_unsigned = xs == '0';
integer scale = 1;
```

32-bit unpacked scaled offset

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
1	1	0	0	0	1	0	0	1	xs	1				Zm		0	0	1		Pg						Rn				Zt	
																U ff															

LDFF1SH { **<Zt>.D** }, **<Pg>/Z**, [**<Xn|SP>**, **<Zm>.D**, **<mod> #1**]

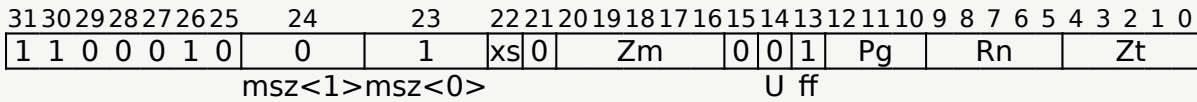
```
if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Zm);
integer g = UInt(Pg);
constant integer esize = 64;
constant integer msize = 16;
```

```

constant integer offs_size = 32;
boolean unsigned = FALSE;
boolean offs_unsigned = xs == '0';
integer scale = 1;

```

32-bit unpacked unscaled offset



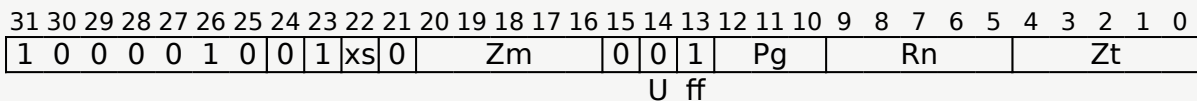
LDFF1SH { **<Zt>.D** }, **<Pg>/Z**, [**<Xn|SP>**, **<Zm>.D**, **<mod>**]

```

if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Zm);
integer g = UInt(Pg);
constant integer esize = 64;
constant integer msize = 16;
constant integer offs_size = 32;
boolean unsigned = FALSE;
boolean offs_unsigned = xs == '0';
integer scale = 0;

```

32-bit unscaled offset



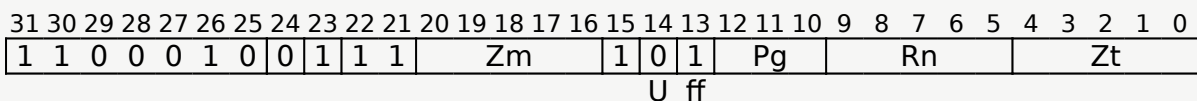
LDFF1SH { **<Zt>.S** }, **<Pg>/Z**, [**<Xn|SP>**, **<Zm>.S**, **<mod>**]

```

if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Zm);
integer g = UInt(Pg);
constant integer esize = 32;
constant integer msize = 16;
constant integer offs_size = 32;
boolean unsigned = FALSE;
boolean offs_unsigned = xs == '0';
integer scale = 0;

```

64-bit scaled offset



LDFF1SH { **<Zt>.D** }, **<Pg>/Z**, [**<Xn|SP>**, **<Zm>.D**, **LSL #1**]

```

if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);

```

```

integer m = UInt(Zm);
integer g = UInt(Pg);
constant integer esize = 64;
constant integer msize = 16;
constant integer offs_size = 64;
boolean unsigned = FALSE;
boolean offs_unsigned = TRUE;
integer scale = 1;

```

64-bit unscaled offset

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
1	1	0	0	0	1	0	0	1	1	0	Zm	1	0	1	Pg	Rn	Zt														
msz<1>msz<0>																U ff															

LDFF1SH { <Zt>.D }, <Pg>/Z, [<Xn|SP>, <Zm>.D]

```

if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Zm);
integer g = UInt(Pg);
constant integer esize = 64;
constant integer msize = 16;
constant integer offs_size = 64;
boolean unsigned = FALSE;
boolean offs_unsigned = TRUE;
integer scale = 0;

```

Assembler Symbols

- <Zt> Is the name of the scalable vector register to be transferred, encoded in the "Zt" field.
- <Pg> Is the name of the governing scalable predicate register P0-P7, encoded in the "Pg" field.
- <Xn|SP> Is the 64-bit name of the general-purpose base register or stack pointer, encoded in the "Rn" field.
- <Zm> Is the name of the offset scalable vector register, encoded in the "Zm" field.
- <mod> Is the index extend and shift specifier, encoded in "xs":

xs	<mod>
0	UXTW
1	SXTW

Operation

```

CheckNonStreamingSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;

```

```

bits(PL) mask = P[g, PL];
bits(64) base;
bits(VL) offset;
bits(VL) result;
bits(VL) orig = Z[t, VL];
bits(msize) data;
constant integer mbytes = msize DIV 8;
boolean fault = FALSE;
boolean faulted = FALSE;
boolean unknown = FALSE;
boolean contiguous = FALSE;
boolean tagchecked = TRUE;
AccessDescriptor accdesc = CreateAccDescSVEFF(contiguous, tagchecked);

if !AnyActiveElement(mask, esize) then
    if n == 31 && ConstrainUnpredictableBool(Unpredictable\_CHECKSPNONEA
        CheckSPAlignment());
else
    if n == 31 then CheckSPAlignment();
    base = if n == 31 then SP[] else X[n, 64];
    offset = Z[m, VL];

assert accdesc.first;

for e = 0 to elements-1
    if ActivePredicateElement(mask, e, esize) then
        integer off = Int(Elem[offset, e, esize]<offs_size-1:0>, offs_unsig
        bits(64) addr = base + (off << scale);
        if accdesc.first then
            // Mem[] will not return if a fault is detected for the first
            data = Mem[addr, mbytes, accdesc];
            accdesc.first = FALSE;
        else
            // MemNF[] will return fault=TRUE if access is not performed
            (data, fault) = MemNF[addr, mbytes, accdesc];
    else
        (data, fault) = (Zeros(msize), FALSE);

    // FFR elements set to FALSE following a suppressed access/fault
    faulted = faulted || fault;
    if faulted then
        ElemFFR[e, esize] = '0';

    // Value becomes CONSTRAINED UNPREDICTABLE after an FFR element is
    unknown = unknown || ElemFFR[e, esize] == '0';
    if unknown then
        if !fault && ConstrainUnpredictableBool(Unpredictable\_SVELDNFDA
            Elem[result, e, esize] = Extend(data, esize, unsigned);
        elsif ConstrainUnpredictableBool(Unpredictable\_SVELDNFZERO) then
            Elem[result, e, esize] = Zeros(esize);
        else // merge
            Elem[result, e, esize] = Elem[orig, e, esize];
    else
        Elem[result, e, esize] = Extend(data, esize, unsigned);

Z[t, VL] = result;

```

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