LDFF1B (scalar plus scalar)

Contiguous load first-fault unsigned bytes to vector (scalar index)

Contiguous load with first-faulting behavior of unsigned bytes to elements of a vector register from the memory address generated by a 64-bit scalar base and scalar index which is added to the base address. After each element access the index value is incremented, but the index register is not updated. Inactive elements will not cause a read from Device memory or signal a fault, 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 4 classes: 8-bit element , 16-bit element , 32-bit element and 64-bit element

8-bit element

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	1	0	0	1	0	0	0	0	0	Rm	0	1	1	Pg	Rn	Zt
							dtype<3:1>			dtype<0>							

```
LDFF1B { <Zt>.B }, <Pg>/Z, [<Xn|SP>{, <Xm>}]
```

```
if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer g = UInt(Pg);
constant integer esize = 8;
constant integer msize = 8;
boolean unsigned = TRUE;
```

16-bit element

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	1	0	0	1	0	0	0	0	1	Rm	0	1	1	Pg	Rn	Zt
							dtype<3:1>			dtype<0>							

```
LDFF1B { <Zt>.H }, <Pg>/Z, [<Xn|SP>{, <Xm>}]
```

```
if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer g = UInt(Pg);
constant integer esize = 16;
constant integer msize = 8;
boolean unsigned = TRUE;
```

32-bit element

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 1 0 0 1 0	0 0 1	0	Rm	0 1 1	Pg	Rn	Zt
	dtype<3:1>	dtype<0>					

LDFF1B { <Zt>.S }, <Pg>/Z, [<Xn|SP>{, <Xm>}]

```
if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer g = UInt(Pg);
constant integer esize = 32;
constant integer msize = 8;
boolean unsigned = TRUE;
```

64-bit element

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 1 0 0 1 0	0 0 1	1	Rm	0 1 1	Pg	Rn	Zt
	dtype<3:1>	dtype<0>					

LDFF1B { <Zt>.D }, <Pg>/Z, [<Xn|SP>{, <Xm>}]

```
if !HaveSVE() then UNDEFINED;
integer t = UInt(Zt);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer g = UInt(Pg);
constant integer esize = 64;
constant integer msize = 8;
boolean unsigned = TRUE;
```

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.
<Xm>	Is the optional 64-bit name of the general-purpose offset register, defaulting to XZR, encoded in the "Rm" field.

Operation

```
CheckNonStreamingSVEEnabled();
constant integer VL = CurrentVL;
constant integer PL = VL DIV 8;
constant integer elements = VL DIV esize;
bits(64) base;
bits(PL) mask = P[g, PL];
bits(VL) result;
bits(VL) orig = Z[t, VL];
bits(msize) data;
bits(64) offset;
constant integer mbytes = msize DIV 8;
boolean fault = FALSE;
boolean faulted = FALSE;
boolean unknown = FALSE;
boolean contiguous = TRUE;
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 = X[m, 64];

assert accdesc.first;

for e = 0 to elements-1
    if ActivePredicateElement(mask, e, esize) then
        bits(64) addr = base + (UInt(offset) + e) * mbytes;
        if accdesc.first then
            // Mem[] will not return if a fault is detected for the firs
            data = Mem[addr, mbytes, accdesc];
            accdesc.first = FALSE;
        else
            // MemNF[] will return fault=TRUE if access is not performe
            (data, fault) = MemNF[addr, mbytes, accdesc];
    else
        (data, fault) = (Zeros(msize), FALSE);

    // FFR elements set to FALSE following a supressed 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) the
            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;
```

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