



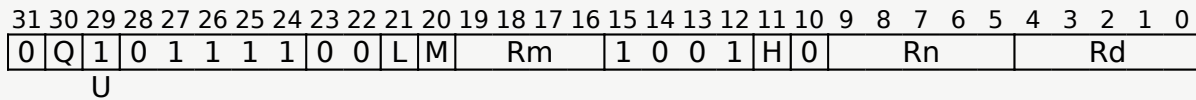
**FMULX** <V><d>, <V><n>, <Vm>.<Ts>[<index>]

```
constant integer idxdsize = 64 << UInt(H);
integer index;
bit Rmhi = M;
case sz:L of
  when '0x' index = UInt(H:L);
  when '10' index = UInt(H);
  when '11' UNDEFINED;

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rmhi:Rm);

constant integer esize = 32 << UInt(sz);
constant integer datasize = esize;
integer elements = 1;
boolean mulx_op = (U == '1');
```

### Vector, half-precision (FEAT\_FP16)



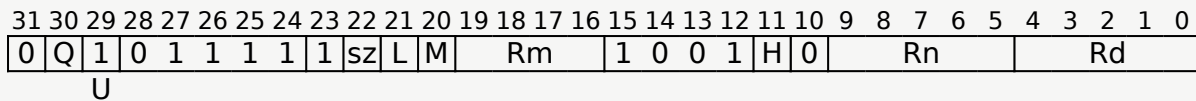
**FMULX** <Vd>.<T>, <Vn>.<T>, <Vm>.<H>[<index>]

```
if !IsFeatureImplemented(FEAT_FP16) then UNDEFINED;

constant integer idxdsize = 64 << UInt(H);
integer n = UInt(Rn);
integer m = UInt(Rm);
integer d = UInt(Rd);
integer index = UInt(H:L:M);

constant integer esize = 16;
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
boolean mulx_op = (U == '1');
```

### Vector, single-precision and double-precision



**FMULX** <Vd>.<T>, <Vn>.<T>, <Vm>.<Ts>[<index>]

```
constant integer idxdsize = 64 << UInt(H);
integer index;
bit Rmhi = M;
case sz:L of
  when '0x' index = UInt(H:L);
  when '10' index = UInt(H);
  when '11' UNDEFINED;
```

```

integer d = UInt(Rd);
integer n = UInt(Rn);
integer m = UInt(Rmhi:Rm);

if sz:Q == '10' then UNDEFINED;
constant integer esize = 32 << UInt(sz);
constant integer datasize = 64 << UInt(Q);
integer elements = datasize DIV esize;
boolean mulx_op = (U == '1');

```

## Assembler Symbols

<Hd> Is the 16-bit name of the SIMD&FP destination register, encoded in the "Rd" field.

<Hn> Is the 16-bit name of the first SIMD&FP source register, encoded in the "Rn" field.

<V> Is a width specifier, encoded in "sz":

sz	<V>
0	S
1	D

<d> Is the number of the SIMD&FP destination register, encoded in the "Rd" field.

<n> Is the number of the first SIMD&FP source register, encoded in the "Rn" field.

<Vd> Is the name of the SIMD&FP destination register, encoded in the "Rd" field.

<T> For the half-precision variant: is an arrangement specifier, encoded in "Q":

Q	<T>
0	4H
1	8H

For the single-precision and double-precision variant: is an arrangement specifier, encoded in "Q:sz":

Q	sz	<T>
0	0	2S
0	1	RESERVED
1	0	4S
1	1	2D

**<Vn>** Is the name of the first SIMD&FP source register, encoded in the "Rn" field.

**<Vm>** For the half-precision variant: is the name of the second SIMD&FP source register, in the range V0 to V15, encoded in the "Rm" field.

For the single-precision and double-precision variant: is the name of the second SIMD&FP source register, encoded in the "M:Rm" fields.

**<Ts>** Is an element size specifier, encoded in "sz":

sz	<Ts>
0	S
1	D

**<index>** For the half-precision variant: is the element index, in the range 0 to 7, encoded in the "H:L:M" fields.

For the single-precision and double-precision variant: is the element index, encoded in "sz:L:H":

sz	L	<index>
0	x	H:L
1	0	H
1	1	RESERVED

## Operation

```

CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = V[n, datasize];
bits(idxdsize) operand2 = V[m, idxdsize];
bits(esize) element1;
bits(esize) element2 = Elem[operand2, index, esize];
FPCRType fpcr = FPCR[];
boolean merge = elements == 1 && IsMerging(fpcr);
bits(128) result = if merge then V[n, 128] else Zeros(128);

for e = 0 to elements-1
    element1 = Elem[operand1, e, esize];
    if mulx_op then
        Elem[result, e, esize] = FPMulX(element1, element2, fpcr);
    else
        Elem[result, e, esize] = FPMul(element1, element2, fpcr);
V[d, 128] = 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.