

## VFMADD132PS/VFMADD213PS/VFMADD231PS—Fused Multiply-Add of Packed Single-Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F38.W0 98 /r VFMADD132PS xmm1, xmm2, xmm3/m128	A	V/V	FMA	Multiply packed single-precision floating-point values from xmm1 and xmm3/mem, add to xmm2 and put result in xmm1.
VEX.128.66.0F38.W0 A8 /r VFMADD213PS xmm1, xmm2, xmm3/m128	A	V/V	FMA	Multiply packed single-precision floating-point values from xmm1 and xmm2, add to xmm3/mem and put result in xmm1.
VEX.128.66.0F38.W0 B8 /r VFMADD231PS xmm1, xmm2, xmm3/m128	A	V/V	FMA	Multiply packed single-precision floating-point values from xmm2 and xmm3/mem, add to xmm1 and put result in xmm1.
VEX.256.66.0F38.W0 98 /r VFMADD132PS ymm1, ymm2, ymm3/m256	A	V/V	FMA	Multiply packed single-precision floating-point values from ymm1 and ymm3/mem, add to ymm2 and put result in ymm1.
VEX.256.66.0F38.W0 A8 /r VFMADD213PS ymm1, ymm2, ymm3/m256	A	V/V	FMA	Multiply packed single-precision floating-point values from ymm1 and ymm2, add to ymm3/mem and put result in ymm1.
VEX.256.66.0F38.W0 B8 /r VFMADD231PS ymm1, ymm2, ymm3/m256	A	V/V	FMA	Multiply packed single-precision floating-point values from ymm2 and ymm3/mem, add to ymm1 and put result in ymm1.
EVEX.128.66.0F38.W0 98 /r VFMADD132PS xmm1 {k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512VL AVX512F	Multiply packed single-precision floating-point values from xmm1 and xmm3/m128/m32bcst, add to xmm2 and put result in xmm1.
EVEX.128.66.0F38.W0 A8 /r VFMADD213PS xmm1 {k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512VL AVX512F	Multiply packed single-precision floating-point values from xmm1 and xmm2, add to xmm3/m128/m32bcst and put result in xmm1.
EVEX.128.66.0F38.W0 B8 /r VFMADD231PS xmm1 {k1}{z}, xmm2, xmm3/m128/m32bcst	B	V/V	AVX512VL AVX512F	Multiply packed single-precision floating-point values from xmm2 and xmm3/m128/m32bcst, add to xmm1 and put result in xmm1.
EVEX.256.66.0F38.W0 98 /r VFMADD132PS ymm1 {k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512VL AVX512F	Multiply packed single-precision floating-point values from ymm1 and ymm3/m256/m32bcst, add to ymm2 and put result in ymm1.
EVEX.256.66.0F38.W0 A8 /r VFMADD213PS ymm1 {k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512VL AVX512F	Multiply packed single-precision floating-point values from ymm1 and ymm2, add to ymm3/m256/m32bcst and put result in ymm1.
EVEX.256.66.0F38.W0 B8 /r VFMADD231PS ymm1 {k1}{z}, ymm2, ymm3/m256/m32bcst	B	V/V	AVX512VL AVX512F	Multiply packed single-precision floating-point values from ymm2 and ymm3/m256/m32bcst, add to ymm1 and put result in ymm1.
EVEX.512.66.0F38.W0 98 /r VFMADD132PS zmm1 {k1}{z}, zmm2, zmm3/m512/m32bcst{er}	B	V/V	AVX512F	Multiply packed single-precision floating-point values from zmm1 and zmm3/m512/m32bcst, add to zmm2 and put result in zmm1.
EVEX.512.66.0F38.W0 A8 /r VFMADD213PS zmm1 {k1}{z}, zmm2, zmm3/m512/m32bcst{er}	B	V/V	AVX512F	Multiply packed single-precision floating-point values from zmm1 and zmm2, add to zmm3/m512/m32bcst and put result in zmm1.
EVEX.512.66.0F38.W0 B8 /r VFMADD231PS zmm1 {k1}{z}, zmm2, zmm3/m512/m32bcst{er}	B	V/V	AVX512F	Multiply packed single-precision floating-point values from zmm2 and zmm3/m512/m32bcst, add to zmm1 and put result in zmm1.

## Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	VEX.vvvv (r)	ModRM:r/m (r)	NA
B	Full	ModRM:reg (r, w)	EVEX.vvvv (r)	ModRM:r/m (r)	NA

## Description

Performs a set of SIMD multiply-add computation on packed single-precision floating-point values using three source operands and writes the multiply-add results in the destination operand. The destination operand is also the first source operand. The second operand must be a SIMD register. The third source operand can be a SIMD register or a memory location.

**VFMADD132PS:** Multiplies the four, eight or sixteen packed single-precision floating-point values from the first source operand to the four, eight or sixteen packed single-precision floating-point values in the third source operand, adds the infinite precision intermediate result to the four, eight or sixteen packed single-precision floating-point values in the second source operand, performs rounding and stores the resulting four, eight or sixteen packed single-precision floating-point values to the destination operand (first source operand).

**VFMADD213PS:** Multiplies the four, eight or sixteen packed single-precision floating-point values from the second source operand to the four, eight or sixteen packed single-precision floating-point values in the first source operand, adds the infinite precision intermediate result to the four, eight or sixteen packed single-precision floating-point values in the third source operand, performs rounding and stores the resulting the four, eight or sixteen packed single-precision floating-point values to the destination operand (first source operand).

**VFMADD231PS:** Multiplies the four, eight or sixteen packed single-precision floating-point values from the second source operand to the four, eight or sixteen packed single-precision floating-point values in the third source operand, adds the infinite precision intermediate result to the four, eight or sixteen packed single-precision floating-point values in the first source operand, performs rounding and stores the resulting four, eight or sixteen packed single-precision floating-point values to the destination operand (first source operand).

**EVEX encoded versions:** The destination operand (also first source operand) is a ZMM register and encoded in `reg_field`. The second source operand is a ZMM register and encoded in `EVEX.vvvv`. The third source operand is a ZMM register, a 512-bit memory location, or a 512-bit vector broadcasted from a 32-bit memory location. The destination operand is conditionally updated with write mask `k1`.

**VEX.256 encoded version:** The destination operand (also first source operand) is a YMM register and encoded in `reg_field`. The second source operand is a YMM register and encoded in `VEX.vvvv`. The third source operand is a YMM register or a 256-bit memory location and encoded in `rm_field`.

**VEX.128 encoded version:** The destination operand (also first source operand) is a XMM register and encoded in `reg_field`. The second source operand is a XMM register and encoded in `VEX.vvvv`. The third source operand is a XMM register or a 128-bit memory location and encoded in `rm_field`. The upper 128 bits of the YMM destination register are zeroed.

**Operation**

In the operations below, “\*” and “+” symbols represent multiplication and addition with infinite precision inputs and outputs (no rounding).

**VFMADD132PS DEST, SRC2, SRC3**

```

IF (VEX.128) THEN
    MAXNUM ← 4
ELSEIF (VEX.256)
    MAXNUM ← 8
FI
For i = 0 to MAXNUM-1 {
    n ← 32*i;
    DEST[n+31:n] ← RoundFPControl_MXCSR(DEST[n+31:n]*SRC3[n+31:n] + SRC2[n+31:n])
}
IF (VEX.128) THEN
    DEST[MAXVL-1:128] ← 0
ELSEIF (VEX.256)
    DEST[MAXVL-1:256] ← 0
FI

```

**VFMADD213PS DEST, SRC2, SRC3**

```

IF (VEX.128) THEN
    MAXNUM ← 4
ELSEIF (VEX.256)
    MAXNUM ← 8
FI
For i = 0 to MAXNUM-1 {
    n ← 32*i;
    DEST[n+31:n] ← RoundFPControl_MXCSR(SRC2[n+31:n]*DEST[n+31:n] + SRC3[n+31:n])
}
IF (VEX.128) THEN
    DEST[MAXVL-1:128] ← 0
ELSEIF (VEX.256)
    DEST[MAXVL-1:256] ← 0
FI

```

**VFMADD231PS DEST, SRC2, SRC3**

```

IF (VEX.128) THEN
    MAXNUM ← 4
ELSEIF (VEX.256)
    MAXNUM ← 8
FI
For i = 0 to MAXNUM-1 {
    n ← 32*i;
    DEST[n+31:n] ← RoundFPControl_MXCSR(SRC2[n+31:n]*SRC3[n+31:n] + DEST[n+31:n])
}
IF (VEX.128) THEN
    DEST[MAXVL-1:128] ← 0
ELSEIF (VEX.256)
    DEST[MAXVL-1:256] ← 0
FI

```

**VFMAADD132PS DEST, SRC2, SRC3 (EVEX encoded version, when src3 operand is a register)**

(KL, VL) = (4, 128), (8, 256), (16, 512)

IF (VL = 512) AND (EVEX.b = 1)

THEN

SET\_RM(EVEX.RC);

ELSE

SET\_RM(MXCSR.RM);

FI;

FOR j ← 0 TO KL-1

i ← j \* 32

IF k1[j] OR \*no writemask\*

THEN DEST[i+31:i] ←

RoundFPControl(DEST[i+31:i]\*SRC3[i+31:i] + SRC2[i+31:i])

ELSE

IF \*merging-masking\* ; merging-masking

THEN \*DEST[i+31:i] remains unchanged\*

ELSE ; zeroing-masking

DEST[i+31:i] ← 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL] ← 0

**VFMAADD132PS DEST, SRC2, SRC3 (EVEX encoded version, when src3 operand is a memory source)**

(KL, VL) = (4, 128), (8, 256), (16, 512)

FOR j ← 0 TO KL-1

i ← j \* 32

IF k1[j] OR \*no writemask\*

THEN

IF (EVEX.b = 1)

THEN

DEST[i+31:i] ←

RoundFPControl\_MXCSR(DEST[i+31:i]\*SRC3[31:0] + SRC2[i+31:i])

ELSE

DEST[i+31:i] ←

RoundFPControl\_MXCSR(DEST[i+31:i]\*SRC3[i+31:i] + SRC2[i+31:i])

FI;

ELSE

IF \*merging-masking\* ; merging-masking

THEN \*DEST[i+31:i] remains unchanged\*

ELSE ; zeroing-masking

DEST[i+31:i] ← 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL] ← 0

**VFMADD213PS DEST, SRC2, SRC3 (EVEX encoded version, when src3 operand is a register)**

(KL, VL) = (4, 128), (8, 256), (16, 512)

IF (VL = 512) AND (EVEX.b = 1)

THEN

SET\_RM(EVEX.RC);

ELSE

SET\_RM(MXCSR.RM);

FI;

FOR j ← 0 TO KL-1

i ← j \* 32

IF k1[j] OR \*no writemask\*

THEN DEST[i+31:i] ←

RoundFPControl(SRC2[i+31:i]\*DEST[i+31:i] + SRC3[i+31:i])

ELSE

IF \*merging-masking\* ; merging-masking

THEN \*DEST[i+31:i] remains unchanged\*

ELSE ; zeroing-masking

DEST[i+31:i] ← 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL] ← 0

**VFMADD213PS DEST, SRC2, SRC3 (EVEX encoded version, when src3 operand is a memory source)**

(KL, VL) = (4, 128), (8, 256), (16, 512)

FOR j ← 0 TO KL-1

i ← j \* 32

IF k1[j] OR \*no writemask\*

THEN

IF (EVEX.b = 1)

THEN

DEST[i+31:i] ←

RoundFPControl\_MXCSR(SRC2[i+31:i]\*DEST[i+31:i] + SRC3[31:0])

ELSE

DEST[i+31:i] ←

RoundFPControl\_MXCSR(SRC2[i+31:i]\*DEST[i+31:i] + SRC3[i+31:i])

FI;

ELSE

IF \*merging-masking\* ; merging-masking

THEN \*DEST[i+31:i] remains unchanged\*

ELSE ; zeroing-masking

DEST[i+31:i] ← 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL] ← 0

**VFMAADD231PS DEST, SRC2, SRC3 (EVEX encoded version, when src3 operand is a register)**

(KL, VL) = (4, 128), (8, 256), (16, 512)

IF (VL = 512) AND (EVEX.b = 1)

THEN

SET\_RM(EVEX.RC);

ELSE

SET\_RM(MXCSR.RM);

FI;

FOR j ← 0 TO KL-1

i ← j \* 32

IF k1[j] OR \*no writemask\*

THEN DEST[i+31:i] ←

RoundFPControl(SRC2[i+31:i]\*SRC3[i+31:i] + DEST[i+31:i])

ELSE

IF \*merging-masking\* ; merging-masking

THEN \*DEST[i+31:i] remains unchanged\*

ELSE ; zeroing-masking

DEST[i+31:i] ← 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL] ← 0

**VFMAADD231PS DEST, SRC2, SRC3 (EVEX encoded version, when src3 operand is a memory source)**

(KL, VL) = (4, 128), (8, 256), (16, 512)

FOR j ← 0 TO KL-1

i ← j \* 32

IF k1[j] OR \*no writemask\*

THEN

IF (EVEX.b = 1)

THEN

DEST[i+31:i] ←

RoundFPControl\_MXCSR(SRC2[i+31:i]\*SRC3[31:0] + DEST[i+31:i])

ELSE

DEST[i+31:i] ←

RoundFPControl\_MXCSR(SRC2[i+31:i]\*SRC3[i+31:i] + DEST[i+31:i])

FI;

ELSE

IF \*merging-masking\* ; merging-masking

THEN \*DEST[i+31:i] remains unchanged\*

ELSE ; zeroing-masking

DEST[i+31:i] ← 0

FI

FI;

ENDFOR

DEST[MAXVL-1:VL] ← 0

**Intel C/C++ Compiler Intrinsic Equivalent**

```

VFMADDxxxPS __m512 __mm512_fmadd_ps(__m512 a, __m512 b, __m512 c);
VFMADDxxxPS __m512 __mm512_fmadd_round_ps(__m512 a, __m512 b, __m512 c, int r);
VFMADDxxxPS __m512 __mm512_mask_fmadd_ps(__m512 a, __mmask16 k, __m512 b, __m512 c);
VFMADDxxxPS __m512 __mm512_maskz_fmadd_ps(__mmask16 k, __m512 a, __m512 b, __m512 c);
VFMADDxxxPS __m512 __mm512_mask3_fmadd_ps(__m512 a, __m512 b, __m512 c, __mmask16 k);
VFMADDxxxPS __m512 __mm512_mask_fmadd_round_ps(__m512 a, __mmask16 k, __m512 b, __m512 c, int r);
VFMADDxxxPS __m512 __mm512_maskz_fmadd_round_ps(__mmask16 k, __m512 a, __m512 b, __m512 c, int r);
VFMADDxxxPS __m512 __mm512_mask3_fmadd_round_ps(__m512 a, __m512 b, __m512 c, __mmask16 k, int r);
VFMADDxxxPS __m256 __mm256_mask_fmadd_ps(__m256 a, __mmask8 k, __m256 b, __m256 c);
VFMADDxxxPS __m256 __mm256_maskz_fmadd_ps(__mmask8 k, __m256 a, __m256 b, __m256 c);
VFMADDxxxPS __m256 __mm256_mask3_fmadd_ps(__m256 a, __m256 b, __m256 c, __mmask8 k);
VFMADDxxxPS __m128 __mm_mask_fmadd_ps(__m128 a, __mmask8 k, __m128 b, __m128 c);
VFMADDxxxPS __m128 __mm_maskz_fmadd_ps(__mmask8 k, __m128 a, __m128 b, __m128 c);
VFMADDxxxPS __m128 __mm_mask3_fmadd_ps(__m128 a, __m128 b, __m128 c, __mmask8 k);
VFMADDxxxPS __m128 __mm_fmadd_ps (__m128 a, __m128 b, __m128 c);
VFMADDxxxPS __m256 __mm256_fmadd_ps (__m256 a, __m256 b, __m256 c);

```

**SIMD Floating-Point Exceptions**

Overflow, Underflow, Invalid, Precision, Denormal

**Other Exceptions**

VEX-encoded instructions, see Exceptions Type 2.

EVEX-encoded instructions, see Exceptions Type E2.