Intel® AVX512-FP16

Architecture Specification

June 2021

Revision 1.0

Document Number: 347407-001US

Contents

Chapter 1

CHANGES

Rev 1.0	• Initial document release

Chapter 2

CPUID

The AVX512_FP16* ISA extensions require that AVX512BW feature be implemented since the instructions for manipulating 32b masks are associated with AVX512BW.

Pre-existing related CPUID bits		
CPUID.(1.0).ECX[29]	F16C	AVX FP16 conversion instructions introduced on the Ivy Bridge (IVB) microarchitecture.
CPUID.(7.0).EBX[16]	AVX512F	AVX512 versions of the IVB conversion instructions.
New CPUID bits		
CPUID.(7.0).EDX[23]	AVX512_FP16	FP16

Chapter 3

INSTRUCTION TABLE

IVB	Intel® Xeon® processors based on Ivy Bridge microarchitecture
KNL	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
SKX	Intel® Xeon® Processor Scalable Family based on Skylake microarchitecture
SPR	Future Intel® Xeon® processors based on Sapphire Rapids microarchitecture

Figure 3.1: Microarchitecture abbreviations used in instruction table

MNEMONIC	OPERANDS	ENCSPACE	CPUID	1st INTERCEPT
VADDPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VADDPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VADDPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VADDSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VCMPPH	k1, zmm2, zmm3/m512, imm8	EVEX	AVX512-FP16	SPR
VCMPPH	k1, xmm2, xmm3/m128, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VCMPPH	k1, ymm2, ymm3/m256, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VCMPSH	k1, xmm2, xmm3/m16, imm8	EVEX	AVX512-FP16	SPR
VCOMISH	xmm1, xmm2/m16	EVEX	AVX512-FP16	SPR
VCVTDQ2PH	ymm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTDQ2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTDQ2PH	xmm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPD2PH	xmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTPD2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPD2PH	xmm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2DQ	zmm1, ymm2/m256	EVEX	AVX512-FP16	SPR
VCVTPH2DQ	xmm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2DQ	ymm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2PD	zmm1, xmm2/m128	EVEX	AVX512-FP16	SPR
VCVTPH2PD	xmm1, xmm2/m32	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2PD	ymm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2PS	zmm1, ymm2/m256	EVEX	AVX512F	SKX,KNL
VCVTPH2PS	xmm1, xmm2/m64	EVEX	AVX512F,AVX512VL	SKX
VCVTPH2PS	ymm1, xmm2/m128	EVEX	AVX512F,AVX512VL	SKX
VCVTPH2PS	xmm1, xmm2/m64	VEX	F16C	IVB
VCVTPH2PS	ymm1, xmm2/m128	VEX	F16C	IVB
VCVTPH2PSX	zmm1, ymm2/m256	EVEX	AVX512-FP16	SPR
VCVTPH2PSX	xmm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2PSX	ymm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2QQ	zmm1, xmm2/m128	EVEX	AVX512-FP16	SPR
VCVTPH2QQ	xmm1, xmm2/m32	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2QQ	ymm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2UDQ	zmm1, ymm2/m256	EVEX	AVX512-FP16	SPR
VCVTPH2UDQ	xmm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2UDQ	ymm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2UQQ	zmm1, xmm2/m128	EVEX	AVX512-FP16	SPR
VCVTPH2UQQ	xmm1, xmm2/m32	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2UQQ	ymm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2UW	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTPH2UW	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR

Table continued on next page...

MNEMONIC	OPERANDS	ENCSPACE	CPUID	1st INTERCEPT
VCVTPH2UW	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2W	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTPH2W	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPH2W	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPS2PH	m256, zmm1, imm8	EVEX	AVX512F	SKX,KNL
VCVTPS2PH	ymm1, zmm2, imm8	EVEX	AVX512F	SKX,KNL
VCVTPS2PH	m128, ymm1, imm8	EVEX	AVX512F,AVX512VL	SKX
VCVTPS2PH	m64, xmm1, imm8	EVEX	AVX512F,AVX512VL	SKX
VCVTPS2PH	xmm1, xmm2, imm8	EVEX	AVX512F,AVX512VL	SKX
VCVTPS2PH	xmm1, ymm2, imm8	EVEX	AVX512F,AVX512VL	SKX
VCVTPS2PH	xmm1/m128, ymm2, imm8	VEX	F16C	IVB
VCVTPS2PH	xmm1/m64, xmm2, imm8	VEX	F16C	IVB
VCVTPS2PHX	ymm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTPS2PHX	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTPS2PHX	xmm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTQQ2PH	xmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTQQ2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTQQ2PH	xmm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTSD2SH	xmm1, xmm2, xmm3/m64	EVEX	AVX512-FP16	SPR
VCVTSH2SD	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VCVTSH2SI	r32, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTSH2SI	r64, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTSH2SS	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VCVTSH2USI	r32, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTSH2USI	r64, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTSI2SH	xmm1, xmm2, r32/m32	EVEX	AVX512-FP16	SPR
VCVTSI2SH	xmm1, xmm2, r64/m64	EVEX	AVX512-FP16	SPR
VCVTSS2SH	xmm1, xmm2, xmm3/m32	EVEX	AVX512-FP16	SPR
VCVTTPH2DQ	zmm1, ymm2/m256	EVEX	AVX512-FP16	SPR
VCVTTPH2DQ	xmm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2DQ	ymm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2QQ	zmm1, xmm2/m128	EVEX	AVX512-FP16	SPR
VCVTTPH2QQ	xmm1, xmm2/m32	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2QQ	ymm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2UDQ	zmm1, ymm2/m256	EVEX	AVX512-FP16	SPR
VCVTTPH2UDQ	xmm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2UDQ	ymm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2UQQ	zmm1, xmm2/m128	EVEX	AVX512-FP16	SPR
VCVTTPH2UQQ	xmm1, xmm2/m32	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2UQQ	ymm1, xmm2/m64	EVEX	AVX512-FP16,AVX512VL	SPR

Table continued on next page...

MNEMONIC	OPERANDS	ENCSPACE	CPUID	1st INTERCEPT
VCVTTPH2UW	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTTPH2UW	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2UW	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2W	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTTPH2W	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTPH2W	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTTSH2SI	r32, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTTSH2SI	r64, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTTSH2USI	r32, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTTSH2USI	r64, xmm1/m16	EVEX	AVX512-FP16	SPR
VCVTUDQ2PH	ymm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTUDQ2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTUDQ2PH	xmm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTUQQ2PH	xmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTUQQ2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTUQQ2PH	xmm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTUSI2SH	xmm1, xmm2, r32/m32	EVEX	AVX512-FP16	SPR
VCVTUSI2SH	xmm1, xmm2, r64/m64	EVEX	AVX512-FP16	SPR
VCVTUW2PH	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTUW2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTUW2PH	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTW2PH	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VCVTW2PH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VCVTW2PH	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VDIVPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VDIVPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VDIVPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VDIVSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFCMADDCPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFCMADDCPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFCMADDCPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFCMADDCSH	xmm1, xmm2, xmm3/m32	EVEX	AVX512-FP16	SPR
VFCMULCPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFCMULCPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFCMULCPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFCMULCSH	xmm1, xmm2, xmm3/m32	EVEX	AVX512-FP16	SPR
VFMADD132PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADD132PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADD132PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADD132SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR

Table continued on next page...

MNEMONIC	OPERANDS	ENCSPACE	CPUID	1st INTERCEPT
VFMADD213PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADD213PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADD213PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADD213SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFMADD231PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADD231PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADD231PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADD231SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFMADDCPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADDCPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDCPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDCSH	xmm1, xmm2, xmm3/m32	EVEX	AVX512-FP16	SPR
VFMADDSUB132PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADDSUB132PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDSUB132PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDSUB213PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADDSUB213PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDSUB213PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDSUB231PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMADDSUB231PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMADDSUB231PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB132PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMSUB132PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB132PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB132SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFMSUB213PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMSUB213PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB213PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB213SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFMSUB231PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMSUB231PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB231PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUB231SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFMSUBADD132PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMSUBADD132PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUBADD132PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUBADD213PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMSUBADD213PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUBADD213PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUBADD231PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR

Table continued on next page...

MNEMONIC	OPERANDS	ENCSPACE	CPUID	1st INTERCEPT
VFMSUBADD231PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMSUBADD231PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMULCPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFMULCPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFMULCPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFMULCSH	xmm1, xmm2, xmm3/m32	EVEX	AVX512-FP16	SPR
VFNMADD132PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFNMADD132PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMADD132PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMADD132SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFNMADD213PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFNMADD213PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMADD213PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMADD213SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFNMADD231PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFNMADD231PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMADD231PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMADD231SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFNMSUB132PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFNMSUB132PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMSUB132PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMSUB132SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFNMSUB213PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFNMSUB213PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMSUB213PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMSUB213SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFNMSUB231PH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VFNMSUB231PH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMSUB231PH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VFNMSUB231SH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VFPCLASSPH	k1, zmm1/m512, imm8	EVEX	AVX512-FP16	SPR
VFPCLASSPH	k1, xmm1/m128, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VFPCLASSPH	k1, ymm1/m256, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VFPCLASSSH	k1, xmm1/m16, imm8	EVEX	AVX512-FP16	SPR
VGETEXPPH	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VGETEXPPH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VGETEXPPH	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VGETEXPSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VGETMANTPH	zmm1, zmm2/m512, imm8	EVEX	AVX512-FP16	SPR
VGETMANTPH	xmm1, xmm2/m128, imm8	EVEX	AVX512-FP16,AVX512VL	SPR

Table continued on next page...

MNEMONIC	OPERANDS	ENCSPACE	CPUID	1st INTERCEPT
VGETMANTPH	ymm1, ymm2/m256, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VGETMANTSH	xmm1, xmm2, xmm3/m16, imm8	EVEX	AVX512-FP16	SPR
VMAXPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VMAXPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VMAXPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VMAXSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VMINPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VMINPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VMINPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VMINSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VMOVSH	m16, xmm1	EVEX	AVX512-FP16	SPR
VMOVSH	xmm1, m16	EVEX	AVX512-FP16	SPR
VMOVSH	xmm1, xmm2, xmm3	EVEX	AVX512-FP16	SPR
VMOVW	reg/m16, xmm1	EVEX	AVX512-FP16	SPR
VMOVW	xmm1, reg/m16	EVEX	AVX512-FP16	SPR
VMULPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VMULPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VMULPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VMULSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VRCPPH	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VRCPPH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VRCPPH	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VRCPSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VREDUCEPH	zmm1, zmm2/m512, imm8	EVEX	AVX512-FP16	SPR
VREDUCEPH	xmm1, xmm2/m128, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VREDUCEPH	ymm1, ymm2/m256, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VREDUCESH	xmm1, xmm2, xmm3/m16, imm8	EVEX	AVX512-FP16	SPR
VRNDSCALEPH	zmm1, zmm2/m512, imm8	EVEX	AVX512-FP16	SPR
VRNDSCALEPH	xmm1, xmm2/m128, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VRNDSCALEPH	ymm1, ymm2/m256, imm8	EVEX	AVX512-FP16,AVX512VL	SPR
VRNDSCALESH	xmm1, xmm2, xmm3/m16, imm8	EVEX	AVX512-FP16	SPR
VRSQRTPH	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR
VRSQRTPH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VRSQRTPH	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VRSQRTSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VSCALEFPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VSCALEFPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VSCALEFPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VSCALEFSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VSQRTPH	zmm1, zmm2/m512	EVEX	AVX512-FP16	SPR

Table continued on next page...

MNEMONIC	**OPERANDS**	**ENCSPACE**	**CPUID**	**1st INTERCEPT**
VSQRTPH	xmm1, xmm2/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VSQRTPH	ymm1, ymm2/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VSQRTSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VSUBPH	zmm1, zmm2, zmm3/m512	EVEX	AVX512-FP16	SPR
VSUBPH	xmm1, xmm2, xmm3/m128	EVEX	AVX512-FP16,AVX512VL	SPR
VSUBPH	ymm1, ymm2, ymm3/m256	EVEX	AVX512-FP16,AVX512VL	SPR
VSUBSH	xmm1, xmm2, xmm3/m16	EVEX	AVX512-FP16	SPR
VUCOMISH	xmm1, xmm2/m16	EVEX	AVX512-FP16	SPR

Chapter 4

INTRODUCTION

This is a comprehensive floating point instruction set extension for the FP16 data type, comparable to FP32/FP64 support. The largest difference from FP32/FP64 support is the full speed handling of denormal FP16 values. Denormal FP16 operands in the AVX512-FP16 architecture are handled at full speed to facilitate using the full dynamic range of FP16 numbers. Unlike FP32 and FP64 numbers, the FP16 operands in the AVX512-FP16 ISA are neither conditionally flushed to zero (MXCSR.FTZ) nor conditionally treated as zero (MXCSR.DAZ) based on MXCSR settings.

Conversion instructions that use FP32 or FP64 sources continue to use MXCSR.DAZ to control denormal handling for their inputs. Conversion instructions that create FP32 or FP64 outputs continue to use MXCSR.FTZ to control denormal handling for their outputs.

Instructions that were defined to ignore MXCSR.DAZ or MXCSR.FTZ continue to do so. No changes are made to the behavior of pre-existing instructions.

4.1 EVEX decoding maps

EVEX maps 3, 5 and 6 are used for encoding FP16 instructions.

Figure 4.1 shows the layout of the EVEX encoding prefix for different situations. The EVEX prefix byte, 0x62, is followed by 3 payload bytes P0, P1, and P2. The decoding map is encoded in the low 3 bits of the P0 byte, P[2:0]. The current Intel Software Developer's Manual (SDM) lists the EVEX map field (EVEX.mm) encoded in P[1:0] as 2 bits, with the next two bits P[3:2] being reserved as zero. The FP16 ISA extension uses the bit P[2] to access the new decoding maps 5 and 6. EVEX maps 0, 4 and 7 are reserved for future use. A future update of the Intel SDM will update the EVEX description when the AVX512-FP16 information is incorporated. The other bits in Figure 4.1 are described in the Intel SDM.

Map 5 and Map 6 are regular fixed length maps, like map 2 (and map 3). In regular fixed length maps, the opcode plays no role in determining the overall instruction length. All instructions in map 2, 3, 5 and 6 have a MODRM byte. All instructions in map 3 also require an immediate byte. The older map maps 0 and

1 are irregular variable length maps since the overall instruction length is determined more complex logic that includes the opcode.

EVEX prefix	P0	P1	P2	Description
0x62	RXBR' 0mmm	WVVVV1pp	zL'LbV'aaa	AVX512 Load-op
0x62	RXBR' 0mmm	WVVVV1pp	zL'L0V'aaa	AVX512 Loads and stores (MODRM.MOD != 0b11)
0x62	RXBR' 0mmm	WVVVV1pp	zL'L0V'aaa	AVX512 reg-reg without rounding control (MODRM.MOD = 0b11)
0x62	RXBR' 0mmm	WVVVV1pp	zRC1V'aaa	AVX512 reg-reg with rounding control (MODRM.MOD = 0b11)

Figure 4.1: EVEX decoding

4.2 Displacement Scaling

With EVEX encodings, 1-byte memory displacements are scaled based on the tuple code and vector length.

The official tuple scaling information is in the public Software Developers Manual. This section just lists the additions for handling the FP16 instructions.

In the following, N refers to the scale factor applied to the signed 1-byte memory displacement. The units of accessing memory are always measured in bytes. The following only applies to 16b input sizes for load-type operations (or 16b output sizes for store-type operations).

FULL If broadcasting, N=2 for FP16 inputs and N=4 for complex FP16 inputs. Otherwise N=16, 32, or 64 corresponding to the full vector length, in bytes.

FULLMEM N=16, 32, or 64 corresponding to the full vector length, in bytes.

SCALAR N=2 always for FP16 inputs and N=4 for complex FP16 inputs.

HALF If broadcasting, N=2. Otherwise N=8, 16, or 32 corresponding to half the vector length, in bytes.

HALFMEM N=8, 16, or 32 corresponding to half the vector length, in bytes.

QUARTER If broadcasting, N=2. Otherwise N=4, 8, or 16 corresponding to one-quarter the vector length, in bytes.

4.3 Rounding of Denormal Numbers

IEEE-754 does not define the setting of the MXCSR.PE bit when underflow exception is unmasked.

If the computation result is underflow (i.e. tiny results) and cannot be accurately represented in the destination format, the MXCSR.PE bit will be set regardless of the underflow mask status (i.e. can get underflow trap with both UE=1 and PE=1). This is different than the handling of FP32/FP64 operations in the Intel Architecture.

4.4 Integer Indefinite Values

When converting floating point values to integers, the output integer format may be signed or unsigned. There are two reasons when the destination value cannot be represented in the output integer format:

- The value is too big.
- The value is negative and the output integer format is unsigned.

If the output integer format is signed and the value is too big, 0x800... is returned (minimal signed number).

If the output integer format is unsigned and the value is too big, 0xfff... is returned (maximal unsigned number).

If the output integer format is unsigned and the value is negative, 0xfff... is returned.

The width of the destination value is determined by the output element size of the instruction.

4.5 NOTATION

In the instruction encoding, the MODRM byte is represented several ways depending on the role it plays. The MODRM byte has 3 fields: 2-bit MODRM.MOD field, a 3-bit MODRM.REG field and a 3-bit MODRM.RM field. When all bits of the MODRM byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the MODRM byte must contain fixed values, those values are specified as follows. If only the MODRM.MOD must be 0b11, and MODRM.REG and MODRM.RM fields are unrestricted; This is denoted “11:rrr:bbb”. The “rrr” correspond to the 3-bits of the MODRM.REG field and the “bbb” correspond to the 3-bits of the MODRM.RM field. If the MODRM.MOD field is constrained to be a value other than 0b11 – that is it must be one of 0b00, 0b01, or 0b10 – then we use the notation ’!(11)’. If for example only the MODRM.REG field had a specific required value, for example 0b101, that would be denoted “mm:101:bbb”. Historically the Software Developers Manual (SDM) only specified the MODRM.REG field restrictions with the notation /0 ... /7 and failed to specify restrictions on the MODRM.MOD and MODRM.RM fields in the encoding boxes.

4.6 INTRINSICS

The compiler intrinsic functions are documented in the Intel Intrinsics Guide https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

Chapter 5

INSTRUCTIONS

5.1 VADDPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 58 /r VADDPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 58 /r VADDPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 58 /r VADDPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.1.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.1.2 Description

Adds packed FP16 values from source operands and stores the packed FP16 result in the destination operand. The destination elements are updated according to the writemask.

5.1.3 Operation

```
VADDPH (EVEX encoded versions) when src2 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      DEST.fp16[j] := SRC1.fp16[j] + SRC2.fp16[j]
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VADDPH (EVEX encoded versions) when src2 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
          DEST.fp16[j] := SRC1.fp16[j] + SRC2.fp16[0]
      ELSE:
          DEST.fp16[j] := SRC1.fp16[j] + SRC2.fp16[j]
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.1.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VADDPH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VADDPH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VADDPH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.2 VADDSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 58 /r VADDSH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.2.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.2.2 Description

Adds low FP16 value from the source operands and stores the FP16 result in the destination operand.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.2.3 Operation

```
VADDSH (EVEX encoded versions)
IF EVEX.b = 1 and SRC2 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   DEST.fp16[0] := SRC1.fp16[0] + SRC2.fp16[0]
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.2.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VADDSH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16

5.3 VCMPPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.0F3A.W0 C2 /r /ib VCMPPH k1{k2}, xmm2, xmm3/m128/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.0F3A.W0 C2 /r /ib VCMPPH k1{k2}, ymm2, ymm3/m256/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.0F3A.W0 C2 /r /ib VCMPPH k1{k2}, zmm2, zmm3/m512/m16bcst {sae}, imm8	A	V/V	AVX512-FP16

5.3.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	IMM8(r)

5.3.2 Description

Compare packed FP16 values from source operands and stores the result in the destination mask operand. The comparison predicate operand (immediate byte bits 4:0) specifies the type of comparison performed on each of the pairs of packed values. The destination elements are updated according to the writemask.

5.3.3 Operation

```
CASE (imm8 & 0x1F) OF
0:  CMP_OPERATOR := EQ_OQ;
1:  CMP_OPERATOR := LT_OS;
2:  CMP_OPERATOR := LE_OS;
3:  CMP_OPERATOR := UNORD_Q;
4:  CMP_OPERATOR := NEQ_UQ;
5:  CMP_OPERATOR := NLT_US;
6:  CMP_OPERATOR := NLE_US;
7:  CMP_OPERATOR := ORD_Q;
8:  CMP_OPERATOR := EQ_UQ;
9:  CMP_OPERATOR := NGE_US;
10: CMP_OPERATOR := NGT_US;
11: CMP_OPERATOR := FALSE_OQ;
12: CMP_OPERATOR := NEQ_OQ;
13: CMP_OPERATOR := GE_OS;
14: CMP_OPERATOR := GT_OS;
15: CMP_OPERATOR := TRUE_UQ;
16: CMP_OPERATOR := EQ_OS;
17: CMP_OPERATOR := LT_OQ;
18: CMP_OPERATOR := LE_OQ;
19: CMP_OPERATOR := UNORD_S;
20: CMP_OPERATOR := NEQ_US;
21: CMP_OPERATOR := NLT_UQ;
22: CMP_OPERATOR := NLE_UQ;
23: CMP_OPERATOR := ORD_S;
24: CMP_OPERATOR := EQ_US;
25: CMP_OPERATOR := NGE_UQ;
26: CMP_OPERATOR := NGT_UQ;
27: CMP_OPERATOR := FALSE_OS;
28: CMP_OPERATOR := NEQ_OS;
29: CMP_OPERATOR := GE_OQ;
30: CMP_OPERATOR := GT_OQ;
31: CMP_OPERATOR := TRUE_US;
ESAC
```

```
VCMPPH (EVEX encoded versions)
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k2[j] OR *no writemask*:
      IF EVEX.b = 1:
          tsrc2 := SRC2.fp16[0]
      ELSE:
          tsrc2 := SRC2.fp16[j]
      DEST.bit[j] := SRC1.fp16[j] CMP_OPERATOR tsrc2
   ELSE
      DEST.bit[j] := 0

DEST[MAXKL-1:KL] := 0
```

5.3.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCMPPH k1, xmm2, xmm3/m128, imm8	E2	ID	AVX512-FP16, AVX512VL
VCMPPH k1, ymm2, ymm3/m256, imm8	E2	ID	AVX512-FP16, AVX512VL
VCMPPH k1, zmm2, zmm3/m512, imm8	E2	ID	AVX512-FP16

5.4 VCMPSH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.LLIG.F3.0F3A.W0 C2 /r /ib VCMPSH k1{k2}, xmm2, xmm3/m16 {sae}, imm8	A	V/V	AVX512-FP16

5.4.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	IMM8(r)

5.4.2 Description

Compare the FP16 values from the lowest element of the source operands and stores the result in the destination mask operand. The comparison predicate operand (immediate byte bits 4:0) specifies the type of comparison performed on the pair of packed FP16 values. The low destination bit is updated according to the writemask. Bits MAX_KL-1:1 of the destination operand are zeroed.

5.4.3 Operation

```
CASE (imm8 & 0x1F) OF
0:  CMP_OPERATOR := EQ_OQ;
1:  CMP_OPERATOR := LT_OS;
2:  CMP_OPERATOR := LE_OS;
3:  CMP_OPERATOR := UNORD_Q;
4:  CMP_OPERATOR := NEQ_UQ;
5:  CMP_OPERATOR := NLT_US;
6:  CMP_OPERATOR := NLE_US;
7:  CMP_OPERATOR := ORD_Q;
8:  CMP_OPERATOR := EQ_UQ;
9:  CMP_OPERATOR := NGE_US;
10: CMP_OPERATOR := NGT_US;
11: CMP_OPERATOR := FALSE_OQ;
12: CMP_OPERATOR := NEQ_OQ;
13: CMP_OPERATOR := GE_OS;
14: CMP_OPERATOR := GT_OS;
15: CMP_OPERATOR := TRUE_UQ;
16: CMP_OPERATOR := EQ_OS;
17: CMP_OPERATOR := LT_OQ;
18: CMP_OPERATOR := LE_OQ;
19: CMP_OPERATOR := UNORD_S;
20: CMP_OPERATOR := NEQ_US;
21: CMP_OPERATOR := NLT_UQ;
22: CMP_OPERATOR := NLE_UQ;
23: CMP_OPERATOR := ORD_S;
24: CMP_OPERATOR := EQ_US;
25: CMP_OPERATOR := NGE_UQ;
26: CMP_OPERATOR := NGT_UQ;
27: CMP_OPERATOR := FALSE_OS;
28: CMP_OPERATOR := NEQ_OS;
29: CMP_OPERATOR := GE_OQ;
30: CMP_OPERATOR := GT_OQ;
31: CMP_OPERATOR := TRUE_US;
ESAC
```

```
VCMPSH (EVEX encoded versions)
IF k2[0] OR *no writemask*:
   DEST.bit[0] := SRC1.fp16[0] CMP_OPERATOR SRC2.fp16[0]
ELSE
   DEST.bit[0] := 0

DEST[MAXKL-1:1] := 0
```

5.4.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCMPSH k1, xmm2, xmm3/m16, imm8	E3	ID	AVX512-FP16

5.5 VCOMISH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.MAP5.W0 2F /r VCOMISH xmm1, xmm2/m16 {sae}	A	V/V	AVX512-FP16

5.5.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(r)	MODRM.R/M(r)	N/A	N/A

5.5.2 Description

Compares the FP16 values in the low word of operand 1 (first operand) and operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN).

Operand 1 is an XMM register; operand 2 can be an XMM register or a 16-bit memory location.

The VCOMISH instruction differs from the VUCOMISH instruction in that it signals a SIMD floating-point invalid operation exception (#I) when a source operand is either a QNaN or SNaN. The VUCOMISH instruction signals an invalid numeric exception only if a source operand is an SNaN.

The EFLAGS register is not updated if an unmasked SIMD floating-point exception is generated.

EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

5.5.3 Operation

```
VCOMISH src1, src2

RESULT := OrderedCompare(SRC1.fp16[0],SRC2.fp16[0])
if RESULT is UNORDERED:
    ZF, PF, CF := 1, 1, 1
else if RESULT is GREATER_THAN:
    ZF, PF, CF := 0, 0, 0
else if RESULT is LESS_THAN:
    ZF, PF, CF := 0, 0, 1
else: // RESULT is EQUALS
    ZF, PF, CF := 1, 0, 0

OF, AF, SF := 0, 0, 0
```

5.5.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCOMISH xmm1, xmm2/m16	E3NF	ID	AVX512-FP16

5.6 VCVTDQ2PH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.MAP5.W0 5B /r VCVTDQ2PH xmm1{k1}{z}, xmm2/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 5B /r VCVTDQ2PH xmm1{k1}{z}, ymm2/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 5B /r VCVTDQ2PH ymm1{k1}{z}, zmm2/m512/m32bcst {er}	A	V/V	AVX512-FP16

5.6.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.6.2 Description

Converts four, eight or sixteen packed signed doubleword integers in the source operand to four, eight or sixteen packed FP16 values in the destination operand.

EVEX encoded versions: The source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcast from a 32-bit memory location. The destination operand is a YMM/XMM register conditionally updated with writemask k1.

EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.6.3 Operation

```
VCVTDQ2PH  dest, src
VL = 128, 256 or 512

KL := VL / 32
IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.dword[0]
      ELSE
         tsrc := SRC.dword[j]

      DEST.fp16[j] := Convert_integer32_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/2] := 0
```

5.6.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTDQ2PH xmm1, xmm2/m128	E2	PO	AVX512-FP16, AVX512VL
VCVTDQ2PH xmm1, ymm2/m256	E2	PO	AVX512-FP16, AVX512VL
VCVTDQ2PH ymm1, zmm2/m512	E2	PO	AVX512-FP16

5.7 VCVTPD2PH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.W1 5A /r VCVTPD2PH xmm1{k1}{z}, xmm2/m128/m64bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W1 5A /r VCVTPD2PH xmm1{k1}{z}, ymm2/m256/m64bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W1 5A /r VCVTPD2PH xmm1{k1}{z}, zmm2/m512/m64bcst {er}	A	V/V	AVX512-FP16

5.7.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.7.2 Description

Converts two, four or eight packed double-precision floating-point values in the source operand (second operand) to two, four or eight packed FP16 values in the destination operand (first operand). When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits.

EVEX encoded versions: The source operand is a ZMM/YMM/XMM register, a 512/256/128-bit memory location, or a 512/256/128-bit vector broadcasts from a 64-bit memory location. The destination operand is a XMM register conditionally updated with writemask k1. The upper bits (`MAXVL-1:128/64/32`) of the corresponding destination are zeroed.

EVEX.vvvv are reserved and must be 1111b otherwise instructions will #UD.

This instruction uses MXCSR.DAZ for handling FP64 inputs. FP16 outputs can be normal or denormal, and are not conditionally flushed to zero.

5.7.3 Operation

```
VCVTPD2PH  dest, src
VL = 128, 256 or 512

KL := VL / 64

IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.double[0]
      ELSE
         tsrc := SRC.double[j]

      DEST.fp16[j] := Convert_fp64_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/4] := 0
```

5.7.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPD2PH xmm1, xmm2/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VCVTPD2PH xmm1, ymm2/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VCVTPD2PH xmm1, zmm2/m512	E2	IOUPD	AVX512-FP16

5.8 VCVTPH2DQ

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.W0 5B /r VCVTPH2DQ xmm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 5B /r VCVTPH2DQ ymm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 5B /r VCVTPH2DQ zmm1{k1}{z}, ymm2/m256/m16bcst {er}	A	V/V	AVX512-FP16

5.8.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	HALF	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.8.2 Description

Converts packed FP16 values in the source operand to signed doubleword integers in destination operand.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.8.3 Operation

```
VCVTPH2DQ  dest, src
VL = 128, 256 or 512

KL := VL / 32
IF *SRC is a register* and (VL = 512) and (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.dword[j] := Convert_fp16_to_integer32(tsrc)
   ELSE   IF *zeroing*:
      DEST.dword[j] := 0
   // else dest.dword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.8.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2DQ xmm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2DQ ymm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2DQ zmm1, ymm2/m256	E2	IP	AVX512-FP16

5.9 VCVTPH2PD

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.MAP5.W0 5A /r VCVTPH2PD xmm1{k1}{z}, xmm2/m32/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 5A /r VCVTPH2PD ymm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 5A /r VCVTPH2PD zmm1{k1}{z}, xmm2/m128/m16bcst {sae}	A	V/V	AVX512-FP16

5.9.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	QUARTER	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.9.2 Description

Convert packed FP16 values to FP64 values in the destination register. The destination elements are updated according to the writemask.

This instruction handles both normal and denormal FP16 inputs.

5.9.3 Operation

```
VCVTPH2PD  dest, src
VL  = 128, 256, or 512

KL := VL/64

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.fp64[j] := Convert_fp16_to_fp64(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp64[j] := 0
   // else dest.fp64[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.9.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2PD xmm1, xmm2/m32	E2	ID	AVX512-FP16, AVX512VL
VCVTPH2PD ymm1, xmm2/m64	E2	ID	AVX512-FP16, AVX512VL
VCVTPH2PD zmm1, xmm2/m128	E2	ID	AVX512-FP16

5.10 VCVTPH2PS[,X]

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
VEX.128.66.0F38.W0 13 /r VCVTPH2PS xmm1, xmm2/m64	A	V/V	F16C
VEX.256.66.0F38.W0 13 /r VCVTPH2PS ymm1, xmm2/m128	A	V/V	F16C
EVEX.128.66.0F38.W0 13 /r VCVTPH2PS xmm1{k1}{z}, xmm2/m64	B	V/V	AVX512F AVX512VL
EVEX.256.66.0F38.W0 13 /r VCVTPH2PS ymm1{k1}{z}, xmm2/m128	B	V/V	AVX512F AVX512VL
EVEX.512.66.0F38.W0 13 /r VCVTPH2PS zmm1{k1}{z}, ymm2/m256 {sae}	B	V/V	AVX512F
EVEX.128.66.MAP6.W0 13 /r VCVTPH2PSX xmm1{k1}{z}, xmm2/m64/m16bcst	C	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 13 /r VCVTPH2PSX ymm1{k1}{z}, xmm2/m128/m16bcst	C	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 13 /r VCVTPH2PSX zmm1{k1}{z}, ymm2/m256/m16bcst {sae}	C	V/V	AVX512-FP16

5.10.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A
B	HALFMEM	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A
C	HALF	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.10.2 Description

VCVTPH2PSX is a new form of the PH to PS conversion instruction, encoded in map 6. The older version of VCVTPH2PS that is present in AVX512F (encoded in map 2, 0F38) does not support embedded broadcasting. VCVTPH2PSX has the embedded broadcasting option available.

The instructions associated with AVX512_FP16 always handle FP16 denormal number inputs; denormal inputs are not treated as zero.

5.10.3 Operation

```
VCVTPH2PSX  dest, src
VL  = 128, 256, or 512

KL := VL/32

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.fp32[j] := Convert_fp16_to_fp32(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp32[j] := 0
   // else dest.fp32[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.10.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2PS xmm1, xmm2/m64	11	I	F16C
VCVTPH2PS ymm1, xmm2/m128	11	I	F16C
VCVTPH2PS xmm1, xmm2/m64	E11	I	AVX512F, AVX512VL
VCVTPH2PS ymm1, xmm2/m128	E11	I	AVX512F, AVX512VL
VCVTPH2PS zmm1, ymm2/m256	E11	I	AVX512F
VCVTPH2PSX xmm1, xmm2/m64	E2	ID	AVX512-FP16, AVX512VL
VCVTPH2PSX ymm1, xmm2/m128	E2	ID	AVX512-FP16, AVX512VL
VCVTPH2PSX zmm1, ymm2/m256	E2	ID	AVX512-FP16

5.11 VCVTPH2QQ

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.66.MAP5.W0 7B /r VCVTPH2QQ xmm1{k1}{z}, xmm2/m32/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 7B /r VCVTPH2QQ ymm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 7B /r VCVTPH2QQ zmm1{k1}{z}, xmm2/m128/m16bcst {er}	A	V/V	AVX512-FP16

5.11.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	QUARTER	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.11.2 Description

Converts packed FP16 values in the source operand to signed quadword integers in destination operand.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.11.3 Operation

```
VCVTPH2QQ  dest, src
VL = 128, 256 or 512

KL := VL / 64
IF *SRC is a register* and (VL = 512) and (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.qword[j] := Convert_fp16_to_integer64(tsrc)
   ELSE   IF *zeroing*:
      DEST.qword[j] := 0
   // else dest.qword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.11.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2QQ xmm1, xmm2/m32	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2QQ ymm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2QQ zmm1, xmm2/m128	E2	IP	AVX512-FP16

5.12 VCVTPH2UDQ

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.MAP5.W0 79 /r VCVTPH2UDQ xmm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 79 /r VCVTPH2UDQ ymm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 79 /r VCVTPH2UDQ zmm1{k1}{z}, ymm2/m256/m16bcst {er}	A	V/V	AVX512-FP16

5.12.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	HALF	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.12.2 Description

Converts packed FP16 values in the source operand to unsigned doubleword integers in destination operand.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.12.3 Operation

```
VCVTPH2UDQ  dest, src
VL = 128, 256 or 512

KL := VL / 32
IF *SRC is a register* and (VL = 512) and (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.dword[j] := Convert_fp16_to_unsigned_integer32(tsrc)
   ELSE   IF *zeroing*:
      DEST.dword[j] := 0
   // else dest.dword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.12.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2UDQ xmm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2UDQ ymm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2UDQ zmm1, ymm2/m256	E2	IP	AVX512-FP16

5.13 VCVTPH2UQQ

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.W0 79 /r VCVTPH2UQQ xmm1{k1}{z}, xmm2/m32/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 79 /r VCVTPH2UQQ ymm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 79 /r VCVTPH2UQQ zmm1{k1}{z}, xmm2/m128/m16bcst {er}	A	V/V	AVX512-FP16

5.13.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	QUARTER	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.13.2 Description

Converts packed FP16 values in the source operand to unsigned quadword integers in destination operand.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.13.3 Operation

```
VCVTPH2UQQ  dest, src
VL = 128, 256 or 512

KL := VL / 64
IF *SRC is a register* and (VL = 512) and (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.qword[j] := Convert_fp16_to_unsigned_integer64(tsrc)
   ELSE   IF *zeroing*:
      DEST.qword[j] := 0
   // else dest.qword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.13.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2UQQ xmm1, xmm2/m32	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2UQQ ymm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2UQQ zmm1, xmm2/m128	E2	IP	AVX512-FP16

5.14 VCVTPH2UW

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.MAP5.W0 7D /r VCVTPH2UW xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 7D /r VCVTPH2UW ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 7D /r VCVTPH2UW zmm1{k1}{z}, zmm2/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.14.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.14.2 Description

Converts packed FP16 values in the source operand to unsigned word integers in destination operand.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.14.3 Operation

```
VCVTPH2UW  dest, src
VL = 128, 256 or 512

KL := VL / 16
IF *SRC is a register* and (VL = 512) and (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.word[j] := Convert_fp16_to_unsigned_integer16(tsrc)
   ELSE   IF *zeroing*:
      DEST.word[j] := 0
   // else dest.word[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.14.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2UW xmm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2UW ymm1, ymm2/m256	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2UW zmm1, zmm2/m512	E2	IP	AVX512-FP16

5.15 VCVTPH2W

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.W0 7D /r VCVTPH2W xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 7D /r VCVTPH2W ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 7D /r VCVTPH2W zmm1{k1}{z}, zmm2/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.15.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.15.2 Description

Converts packed FP16 values in the source operand to signed word integers in destination operand.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.15.3 Operation

```
VCVTPH2W  dest, src
VL = 128, 256 or 512

KL := VL / 16
IF *SRC is a register* and (VL = 512) and (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.word[j] := Convert_fp16_to_integer16(tsrc)
   ELSE   IF *zeroing*:
      DEST.word[j] := 0
   // else dest.word[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.15.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTPH2W xmm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2W ymm1, ymm2/m256	E2	IP	AVX512-FP16, AVX512VL
VCVTPH2W zmm1, zmm2/m512	E2	IP	AVX512-FP16

5.16 VCVTPS2PH[,X]

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
VEX.128.66.0F3A.W0 1D /r /ib VCVTPS2PH xmm1/m64, xmm2, imm8	A	V/V	F16C
VEX.256.66.0F3A.W0 1D /r /ib VCVTPS2PH xmm1/m128, ymm2, imm8	A	V/V	F16C
EVEX.128.66.0F3A.W0 1D 11:rrr:bbb /ib VCVTPS2PH xmm1{k1}{z}, xmm2, imm8	B	V/V	AVX512F AVX512VL
EVEX.128.66.0F3A.W0 1D !(11):rrr:bbb /ib VCVTPS2PH m64{k1}, xmm1, imm8	B	V/V	AVX512F AVX512VL
EVEX.256.66.0F3A.W0 1D 11:rrr:bbb /ib VCVTPS2PH xmm1{k1}{z}, ymm2, imm8	B	V/V	AVX512F AVX512VL
EVEX.256.66.0F3A.W0 1D !(11):rrr:bbb /ib VCVTPS2PH m128{k1}, ymm1, imm8	B	V/V	AVX512F AVX512VL
EVEX.512.66.0F3A.W0 1D 11:rrr:bbb /ib VCVTPS2PH ymm1{k1}{z}, zmm2 {sae}, imm8	B	V/V	AVX512F
EVEX.512.66.0F3A.W0 1D !(11):rrr:bbb /ib VCVTPS2PH m256{k1}, zmm1, imm8	B	V/V	AVX512F
EVEX.128.66.MAP5.W0 1D /r VCVTPS2PHX xmm1{k1}{z}, xmm2/m128/m32bcst	C	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 1D /r VCVTPS2PHX xmm1{k1}{z}, ymm2/m256/m32bcst	C	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 1D /r VCVTPS2PHX ymm1{k1}{z}, zmm2/m512/m32bcst {er}	C	V/V	AVX512-FP16

5.16.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	MODRM.R/M(w)	MODRM.REG(r)	IMM8(r)	N/A
B	HALFMEM	MODRM.R/M(w)	MODRM.REG(r)	IMM8(r)	N/A
C	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.16.2 Description

The load form of the VCVTPS2PHX instruction, without imm8, is new. VCVTPS2PHX supports broadcasting. The store (with imm8) versions (VCVTPS2PH) were introduced with AVX512F and F16C.

The instructions associated with AVX512_FP16 use MXCSR.DAZ for handling FP32 inputs. FP16 outputs can be normal or denormal numbers, and are not conditionally flushed based on MXCSR settings.

The legacy VCVTPS2PH instructions associated with the CPUID bits F16C and AVX512F continue to use MXCSR.DAZ for their inputs.

5.16.3 Operation

```
VCVTPS2PHX  dest, src   (AVX512_FP16 load version with broadcast support)
VL  = 128, 256, or 512

KL := VL / 32

IF *SRC is a register* and (VL == 512) and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp32[0]
      ELSE
         tsrc := SRC.fp32[j]

      DEST.fp16[j] := Convert_fp32_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/2] := 0
```

5.16.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VCVTPS2PH xmm1/m64, xmm2, imm8	11	IOUPD	F16C
VCVTPS2PH xmm1/m128, ymm2, imm8	11	IOUPD	F16C
VCVTPS2PH xmm1, xmm2, imm8	E11	IOUPD	AVX512F, AVX512VL
VCVTPS2PH m64, xmm1, imm8	E11	IOUPD	AVX512F, AVX512VL
VCVTPS2PH xmm1, ymm2, imm8	E11	IOUPD	AVX512F, AVX512VL
VCVTPS2PH m128, ymm1, imm8	E11	IOUPD	AVX512F, AVX512VL
VCVTPS2PH ymm1, zmm2, imm8	E11	IOUPD	AVX512F
VCVTPS2PH m256, zmm1, imm8	E11	IOUPD	AVX512F
VCVTPS2PHX xmm1, xmm2/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VCVTPS2PHX xmm1, ymm2/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VCVTPS2PHX ymm1, zmm2/m512	E2	IOUPD	AVX512-FP16

5.17 VCVTQQ2PH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.MAP5.W1 5B /r VCVTQQ2PH xmm1{k1}{z}, xmm2/m128/m64bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W1 5B /r VCVTQQ2PH xmm1{k1}{z}, ymm2/m256/m64bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W1 5B /r VCVTQQ2PH xmm1{k1}{z}, zmm2/m512/m64bcst {er}	A	V/V	AVX512-FP16

5.17.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.17.2 Description

Converts packed quadword integers in the source operand to packed FP16 values in the destination operand. The destination elements are updated according to the writemask.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.17.3 Operation

```
VCVTQQ2PH  dest, src
VL = 128, 256 or 512

KL := VL / 64
IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.qword[0]
      ELSE
         tsrc := SRC.qword[j]

      DEST.fp16[j] := Convert_integer64_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/4] := 0
```

5.17.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTQQ2PH xmm1, xmm2/m128	E2	PO	AVX512-FP16, AVX512VL
VCVTQQ2PH xmm1, ymm2/m256	E2	PO	AVX512-FP16, AVX512VL
VCVTQQ2PH xmm1, zmm2/m512	E2	PO	AVX512-FP16

5.18 VCVTSD2SH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F2.MAP5.W1 5A /r VCVTSD2SH xmm1{k1}{z}, xmm2, xmm3/m64 {er}	A	V/V	AVX512-FP16

5.18.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.18.2 Description

Converts the low FP64 value in the second source operand to a FP16 value in the low element of the destination operand.

When the conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.18.3 Operation

```
VCVTSD2SH  dest, src1, src2

IF *SRC2 is a register* and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   DEST.fp16[0] := Convert_fp64_to_fp16(SRC2.fp64[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.18.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTSD2SH xmm1, xmm2, xmm3/m64	E3	IOUPD	AVX512-FP16

5.19 VCVTSH2SD

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 5A /r VCVTSH2SD xmm1{k1}{z}, xmm2, xmm3/m16 {sae}	A	V/V	AVX512-FP16

5.19.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.19.2 Description

Converts the low FP16 element in the second source operand to a FP64 element in the low element of the destination operand.

Bits 127:64 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP64 element of the destination is updated according to the writemask.

5.19.3 Operation

```
VCVTSH2SD  dest, src1, src2

IF k1[0] OR *no writemask*:
   DEST.fp64[0] := Convert_fp16_to_fp64(SRC2.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp64[0] := 0
// else dest.fp64[0] remains unchanged

DEST[127:64] := SRC1[127:64]
DEST[MAXVL-1:128] := 0
```

5.19.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTSH2SD xmm1, xmm2, xmm3/m16	E3	ID	AVX512-FP16

5.20 VCVTSH2SI

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 2D /r VCVTSH2SI r32, xmm1/m16 {er}	A	V/V[1]	AVX512-FP16
EVEX.LLIG.F3.MAP5.W1 2D /r VCVTSH2SI r64, xmm1/m16 {er}	A	V/N.E.	AVX512-FP16
Notes: 1: Outside of 64b mode, the EVEX.W field is ignored. The instruction behaves as if W=0 were used.			

5.20.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.20.2 Description

Converts the low FP16 element in the source operand to a signed integer in the destination general purpose register.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

5.20.3 Operation

```
VCVTSH2SI  dest, src

IF *SRC is a register* and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

IF 64-mode and OperandSize == 64:
  DEST.qword := Convert_fp16_to_integer64(SRC.fp16[0])
ELSE:
  DEST.dword := Convert_fp16_to_integer32(SRC.fp16[0])
```

5.20.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VCVTSH2SI r32, xmm1/m16	E3NF	IP	AVX512-FP16
VCVTSH2SI r64, xmm1/m16	E3NF	IP	AVX512-FP16

5.21 VCVTSH2SS

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.MAP6.W0 13 /r VCVTSH2SS xmm1{k1}{z}, xmm2, xmm3/m16 {sae}	A	V/V	AVX512-FP16

5.21.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.21.2 Description

Converts the low FP16 element in the second source operand to the low FP32 element of the destination operand.

Bits 127:32 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.21.3 Operation

```
VCVTSH2SS  dest, src1, src2

IF k1[0] OR *no writemask*:
   DEST.fp32[0] := Convert_fp16_to_fp32(SRC2.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp32[0] := 0
// else dest.fp32[0] remains unchanged

DEST[127:32] := SRC1[127:32]
DEST[MAXVL-1:128] := 0
```

5.21.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTSH2SS xmm1, xmm2, xmm3/m16	E3	ID	AVX512-FP16

5.22 VCVTSH2USI

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 79 /r VCVTSH2USI r32, xmm1/m16 {er}	A	V/V[1]	AVX512-FP16
EVEX.LLIG.F3.MAP5.W1 79 /r VCVTSH2USI r64, xmm1/m16 {er}	A	V/N.E.	AVX512-FP16
Notes: 1: Outside of 64b mode, the EVEX.W field is ignored. The instruction behaves as if W=0 were used.			

5.22.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.22.2 Description

Converts the low FP16 element in the source operand to an unsigned integer in the destination general purpose register.

When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or the embedded rounding control bits. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

5.22.3 Operation

```
VCVTSH2USI  dest, src

// SET_RM() sets the rounding mode used for this instruction.
IF *SRC is a register* and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

IF 64-mode and OperandSize == 64:
  DEST.qword := Convert_fp16_to_unsigned_integer64(SRC.fp16[0])
ELSE:
  DEST.dword := Convert_fp16_to_unsigned_integer32(SRC.fp16[0])
```

5.22.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VCVTSH2USI r32, xmm1/m16	E3NF	IP	AVX512-FP16
VCVTSH2USI r64, xmm1/m16	E3NF	IP	AVX512-FP16

5.23 VCVTSI2SH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 2A /r VCVTSI2SH xmm1, xmm2, r32/m32 {er}	A	V/V[1]	AVX512-FP16
EVEX.LLIG.F3.MAP5.W1 2A /r VCVTSI2SH xmm1, xmm2, r64/m64 {er}	A	V/N.E.	AVX512-FP16
Notes: 1: Outside of 64b mode, the EVEX.W field is ignored. The instruction behaves as if W=0 were used.			

5.23.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.23.2 Description

Converts an signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the second source operand to a FP16 value in the destination operand. The result is stored in the low word of the destination operand. When conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or embedded rounding controls.

The second source operand can be a general-purpose register or a 32/64-bit memory location. The first source and destination operands are XMM registers. Bits 127:16 of the XMM register destination are copied from corresponding bits in the first source operand. Bits MAXVL-1:128 of the destination register are zeroed.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.23.3 Operation

```
VCVTSI2SH  dest, src1, src2

IF *SRC2 is a register* and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

IF 64-mode and OperandSize == 64:
   DEST.fp16[0] := Convert_integer64_to_fp16(SRC2.qword)
ELSE:
   DEST.fp16[0] := Convert_integer32_to_fp16(SRC2.dword)

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.23.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTSI2SH xmm1, xmm2, r32/m32	E3NF	PO	AVX512-FP16
VCVTSI2SH xmm1, xmm2, r64/m64	E3NF	PO	AVX512-FP16

5.24 VCVTSS2SH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.LLIG.NP.MAP5.W0 1D /r VCVTSS2SH xmm1{k1}{z}, xmm2, xmm3/m32 {er}	A	V/V	AVX512-FP16

5.24.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.24.2 Description

Converts the low FP32 value in the second source operand to a FP16 value in the low element of the destination operand.

When the conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.24.3 Operation

```
VCVTSS2SH  dest, src1, src2

IF *SRC2 is a register* and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   DEST.fp16[0] := Convert_fp32_to_fp16(SRC2.fp32[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.24.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTSS2SH xmm1, xmm2, xmm3/m32	E3	IOUPD	AVX512-FP16

5.25 VCVTTPH2DQ

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.F3.MAP5.W0 5B /r VCVTTPH2DQ xmm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F3.MAP5.W0 5B /r VCVTTPH2DQ ymm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F3.MAP5.W0 5B /r VCVTTPH2DQ zmm1{k1}{z}, ymm2/m256/m16bcst {sae}	A	V/V	AVX512-FP16

5.25.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	HALF	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.25.2 Description

Converts packed FP16 values in the source operand to signed doubleword integers in destination operand.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result is larger than the maximum signed doubleword integer, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.25.3 Operation

```
VCVTTPH2DQ  dest, src
VL = 128, 256 or 512

KL := VL / 32

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.fp32[j] := Convert_fp16_to_integer32_truncate(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp32[j] := 0
   // else dest.fp32[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.25.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTPH2DQ xmm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2DQ ymm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2DQ zmm1, ymm2/m256	E2	IP	AVX512-FP16

5.26 VCVTTPH2QQ

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.W0 7A /r VCVTTPH2QQ xmm1{k1}{z}, xmm2/m32/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 7A /r VCVTTPH2QQ ymm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 7A /r VCVTTPH2QQ zmm1{k1}{z}, xmm2/m128/m16bcst {sae}	A	V/V	AVX512-FP16

5.26.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	QUARTER	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.26.2 Description

Converts packed FP16 values in the source operand to signed quadword integers in destination operand.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.26.3 Operation

```
VCVTTPH2QQ  dest, src
VL = 128, 256 or 512

KL := VL / 64

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.qword[j] := Convert_fp16_to_integer64_truncate(tsrc)
   ELSE   IF *zeroing*:
      DEST.qword[j] := 0
   // else dest.qword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.26.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTPH2QQ xmm1, xmm2/m32	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2QQ ymm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2QQ zmm1, xmm2/m128	E2	IP	AVX512-FP16

5.27 VCVTTPH2UDQ

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 78 /r VCVTTPH2UDQ xmm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 78 /r VCVTTPH2UDQ ymm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 78 /r VCVTTPH2UDQ zmm1{k1}{z}, ymm2/m256/m16bcst {sae}	A	V/V	AVX512-FP16

5.27.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	HALF	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.27.2 Description

Converts packed FP16 values in the source operand to unsigned doubleword integers in destination operand.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.27.3 Operation

```
VCVTTPH2UDQ  dest, src
VL = 128, 256 or 512

KL := VL / 32

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.dword[j] := Convert_fp16_to_unsigned_integer32_truncate(tsrc)
   ELSE   IF *zeroing*:
      DEST.dword[j] := 0
   // else dest.dword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.27.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTPH2UDQ xmm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2UDQ ymm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2UDQ zmm1, ymm2/m256	E2	IP	AVX512-FP16

5.28 VCVTTPH2UQQ

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.66.MAP5.W0 78 /r VCVTTPH2UQQ xmm1{k1}{z}, xmm2/m32/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 78 /r VCVTTPH2UQQ ymm1{k1}{z}, xmm2/m64/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 78 /r VCVTTPH2UQQ zmm1{k1}{z}, xmm2/m128/m16bcst {sae}	A	V/V	AVX512-FP16

5.28.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	QUARTER	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.28.2 Description

Converts packed FP16 values in the source operand to unsigned quadword integers in destination operand.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.28.3 Operation

```
VCVTTPH2UQQ  dest, src
VL = 128, 256 or 512

KL := VL / 64

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.qword[j] := Convert_fp16_to_unsigned_integer64_truncate(tsrc)
   ELSE   IF *zeroing*:
      DEST.qword[j] := 0
   // else dest.qword[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.28.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTPH2UQQ xmm1, xmm2/m32	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2UQQ ymm1, xmm2/m64	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2UQQ zmm1, xmm2/m128	E2	IP	AVX512-FP16

5.29 VCVTTPH2UW

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 7C /r VCVTTPH2UW xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 7C /r VCVTTPH2UW ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 7C /r VCVTTPH2UW zmm1{k1}{z}, zmm2/m512/m16bcst {sae}	A	V/V	AVX512-FP16

5.29.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.29.2 Description

Converts packed FP16 values in the source operand to unsigned word integers in destination operand.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.29.3 Operation

```
VCVTTPH2UW  dest, src
VL = 128, 256 or 512

KL := VL / 16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.word[j] := Convert_fp16_to_unsigned_integer16_truncate(tsrc)
   ELSE   IF *zeroing*:
      DEST.word[j] := 0
   // else dest.word[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.29.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTPH2UW xmm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2UW ymm1, ymm2/m256	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2UW zmm1, zmm2/m512	E2	IP	AVX512-FP16

5.30 VCVTTPH2W

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.W0 7C /r VCVTTPH2W xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP5.W0 7C /r VCVTTPH2W ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP5.W0 7C /r VCVTTPH2W zmm1{k1}{z}, zmm2/m512/m16bcst {sae}	A	V/V	AVX512-FP16

5.30.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.30.2 Description

Converts packed FP16 values in the source operand to signed word integers in destination operand.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

The destination elements are updated according to the writemask.

5.30.3 Operation

```
VCVTTPH2W  dest, src
VL = 128, 256 or 512

KL := VL / 16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.fp16[0]
      ELSE
         tsrc := SRC.fp16[j]

      DEST.word[j] := Convert_fp16_to_integer16_truncate(tsrc)
   ELSE   IF *zeroing*:
      DEST.word[j] := 0
   // else dest.word[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.30.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTPH2W xmm1, xmm2/m128	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2W ymm1, ymm2/m256	E2	IP	AVX512-FP16, AVX512VL
VCVTTPH2W zmm1, zmm2/m512	E2	IP	AVX512-FP16

5.31 VCVTTSH2SI

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 2C /r VCVTTSH2SI r32, xmm1/m16 {sae}	A	V/V[1]	AVX512-FP16
EVEX.LLIG.F3.MAP5.W1 2C /r VCVTTSH2SI r64, xmm1/m16 {sae}	A	V/N.E.	AVX512-FP16
Notes: 1: Outside of 64b mode, the EVEX.W field is ignored. The instruction behaves as if W=0 were used.			

5.31.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.31.2 Description

Converts the low FP16 element in the source operand to a signed integer in the destination general purpose register.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

5.31.3 Operation

```
VCVTTSH2SI  dest, src

IF 64-mode and OperandSize == 64:
  DEST.qword := Convert_fp16_to_integer64_truncate(SRC.fp16[0])
ELSE:
  DEST.dword := Convert_fp16_to_integer32_truncate(SRC.fp16[0])
```

5.31.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTSH2SI r32, xmm1/m16	E3NF	IP	AVX512-FP16
VCVTTSH2SI r64, xmm1/m16	E3NF	IP	AVX512-FP16

5.32 VCVTTSH2USI

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 78 /r VCVTTSH2USI r32, xmm1/m16 {sae}	A	V/V[1]	AVX512-FP16
EVEX.LLIG.F3.MAP5.W1 78 /r VCVTTSH2USI r64, xmm1/m16 {sae}	A	V/N.E.	AVX512-FP16
Notes: 1: Outside of 64b mode, the EVEX.W field is ignored. The instruction behaves as if W=0 were used.			

5.32.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.32.2 Description

Converts the low FP16 element in the source operand to an unsigned integer in the destination general purpose register.

When a conversion is inexact, a truncated (round toward zero) value is returned. If a converted result cannot be represented in the destination format, the floating-point invalid exception is raised, and if this exception is masked, the integer indefinite value is returned; See Section 4.4.

5.32.3 Operation

```
VCVTTSH2USI  dest, src

IF 64-mode and OperandSize == 64:
  DEST.qword := Convert_fp16_to_unsigned_integer64_truncate(SRC.fp16[0])
ELSE:
  DEST.dword := Convert_fp16_to_unsigned_integer32_truncate(SRC.fp16[0])
```

5.32.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTTSH2USI r32, xmm1/m16	E3NF	IP	AVX512-FP16
VCVTTSH2USI r64, xmm1/m16	E3NF	IP	AVX512-FP16

5.33 VCVTUDQ2PH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.F2.MAP5.W0 7A /r VCVTUDQ2PH xmm1{k1}{z}, xmm2/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F2.MAP5.W0 7A /r VCVTUDQ2PH xmm1{k1}{z}, ymm2/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F2.MAP5.W0 7A /r VCVTUDQ2PH ymm1{k1}{z}, zmm2/m512/m32bcst {er}	A	V/V	AVX512-FP16

5.33.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.33.2 Description

Converts packed unsigned doubleword integers in the source operand to packed FP16 values in the destination operand. The destination elements are updated according to the writemask.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.33.3 Operation

```
VCVTUDQ2PH  dest, src
VL = 128, 256 or 512

KL := VL / 32
IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.dword[0]
      ELSE
         tsrc := SRC.dword[j]

      DEST.fp16[j] := Convert_unsigned_integer32_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/2] := 0
```

5.33.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTUDQ2PH xmm1, xmm2/m128	E2	PO	AVX512-FP16, AVX512VL
VCVTUDQ2PH xmm1, ymm2/m256	E2	PO	AVX512-FP16, AVX512VL
VCVTUDQ2PH ymm1, zmm2/m512	E2	PO	AVX512-FP16

5.34 VCVTUQQ2PH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.F2.MAP5.W1 7A /r VCVTUQQ2PH xmm1{k1}{z}, xmm2/m128/m64bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F2.MAP5.W1 7A /r VCVTUQQ2PH xmm1{k1}{z}, ymm2/m256/m64bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F2.MAP5.W1 7A /r VCVTUQQ2PH xmm1{k1}{z}, zmm2/m512/m64bcst {er}	A	V/V	AVX512-FP16

5.34.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.34.2 Description

Converts packed unsigned quadword integers in the source operand to packed FP16 values in the destination operand. The destination elements are updated according to the writemask.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.34.3 Operation

```
VCVTUQQ2PH  dest, src
VL = 128, 256 or 512

KL := VL / 64
IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.qword[0]
      ELSE
         tsrc := SRC.qword[j]

      DEST.fp16[j] := Convert_unsigned_integer64_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/4] := 0
```

5.34.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTUQQ2PH xmm1, xmm2/m128	E2	PO	AVX512-FP16, AVX512VL
VCVTUQQ2PH xmm1, ymm2/m256	E2	PO	AVX512-FP16, AVX512VL
VCVTUQQ2PH xmm1, zmm2/m512	E2	PO	AVX512-FP16

5.35 VCVTUSI2SH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 7B /r VCVTUSI2SH xmm1, xmm2, r32/m32 {er}	A	V/V[1]	AVX512-FP16
EVEX.LLIG.F3.MAP5.W1 7B /r VCVTUSI2SH xmm1, xmm2, r64/m64 {er}	A	V/N.E.	AVX512-FP16
Notes: 1: Outside of 64b mode, the EVEX.W field is ignored. The instruction behaves as if W=0 were used.			

5.35.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.35.2 Description

Converts an unsigned doubleword integer (or unsigned quadword integer if operand size is 64 bits) in the second source operand to a FP16 value in the destination operand. The result is stored in the low word of the destination operand. When conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or embedded rounding controls.

The second source operand can be a general-purpose register or a 32/64-bit memory location. The first source and destination operands are XMM registers. Bits 127:16 of the XMM register destination are copied from corresponding bits in the first source operand. Bits MAXVL-1:128 of the destination register are zeroed.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.35.3 Operation

```
VCVTUSI2SH  dest, src1, src2

IF *SRC2 is a register* and  (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

IF 64-mode and OperandSize == 64:
   DEST.fp16[0] := Convert_unsigned_integer64_to_fp16(SRC2.qword)
ELSE:
   DEST.fp16[0] := Convert_unsigned_integer32_to_fp16(SRC2.dword)

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.35.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VCVTUSI2SH xmm1, xmm2, r32/m32	E3NF	PO	AVX512-FP16
VCVTUSI2SH xmm1, xmm2, r64/m64	E3NF	PO	AVX512-FP16

5.36 VCVTUW2PH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.F2.MAP5.W0 7D /r VCVTUW2PH xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F2.MAP5.W0 7D /r VCVTUW2PH ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F2.MAP5.W0 7D /r VCVTUW2PH zmm1{k1}{z}, zmm2/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.36.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.36.2 Description

Converts packed unsigned word integers in the source operand to FP16 values in the destination operand. When conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or embedded rounding controls.

The destination elements are updated according to the writemask.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

5.36.3 Operation

```
VCVTUW2PH  dest, src
VL = 128, 256 or 512

KL := VL / 16
IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.word[0]
      ELSE
         tsrc := SRC.word[j]

      DEST.fp16[j] := Convert_unsignd_integer16_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.36.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTUW2PH xmm1, xmm2/m128	E2	PO	AVX512-FP16, AVX512VL
VCVTUW2PH ymm1, ymm2/m256	E2	PO	AVX512-FP16, AVX512VL
VCVTUW2PH zmm1, zmm2/m512	E2	PO	AVX512-FP16

5.37 VCVTW2PH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.F3.MAP5.W0 7D /r VCVTW2PH xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F3.MAP5.W0 7D /r VCVTW2PH ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F3.MAP5.W0 7D /r VCVTW2PH zmm1{k1}{z}, zmm2/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.37.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.37.2 Description

Converts packed signed word integers in the source operand to FP16 values in the destination operand. When conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register or embedded rounding controls.

The destination elements are updated according to the writemask.

5.37.3 Operation

```
VCVTW2PH  dest, src
VL = 128, 256 or 512

KL := VL / 16
IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE:
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *SRC is memory* and EVEX.b = 1:
         tsrc := SRC.word[0]
      ELSE
         tsrc := SRC.word[j]

      DEST.fp16[j] := Convert_integer16_to_fp16(tsrc)
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.37.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VCVTW2PH xmm1, xmm2/m128	E2	P	AVX512-FP16, AVX512VL
VCVTW2PH ymm1, ymm2/m256	E2	P	AVX512-FP16, AVX512VL
VCVTW2PH zmm1, zmm2/m512	E2	P	AVX512-FP16

5.38 VDIVPH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.MAP5.W0 5E /r VDIVPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 5E /r VDIVPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 5E /r VDIVPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.38.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.38.2 Description

Divide packed FP16 values from first source operand by the corresponding elements in the second source operand, storing the packed FP16 result in the destination operand. The destination elements are updated according to the writemask.

5.38.3 Operation

```
VDIVPH (EVEX encoded versions) when src2 operand is a register
VL = 128, 256 or 512
KL := VL/16

IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      DEST.fp16[j] := SRC1.fp16[j] / SRC2.fp16[j]
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VDIVPH (EVEX encoded versions) when src2 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
          DEST.fp16[j] := SRC1.fp16[j] / SRC2.fp16[0]
      ELSE:
          DEST.fp16[j] := SRC1.fp16[j] / SRC2.fp16[j]
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.38.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VDIVPH xmm1, xmm2, xmm3/m128	E2	IOUPDZ	AVX512-FP16, AVX512VL
VDIVPH ymm1, ymm2, ymm3/m256	E2	IOUPDZ	AVX512-FP16, AVX512VL
VDIVPH zmm1, zmm2, zmm3/m512	E2	IOUPDZ	AVX512-FP16

5.39 VDIVSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 5E /r VDIVSH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.39.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.39.2 Description

Divide the low FP16 value from the first source operand by the corresponding value in the second source operand, storing the FP16 result in the destination operand. Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.39.3 Operation

```
VDIVSH (EVEX encoded versions)
IF EVEX.b = 1 and SRC2 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   DEST.fp16[0] := SRC1.fp16[0] / SRC2.fp16[0]
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.39.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VDIVSH xmm1, xmm2, xmm3/m16	E3	IOUPDZ	AVX512-FP16

5.40 VF[,C]MADDCPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.F2.MAP6.W0 56 /r VFCMADDCPH xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F2.MAP6.W0 56 /r VFCMADDCPH ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F2.MAP6.W0 56 /r VFCMADDCPH zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst {er}	A	V/V	AVX512-FP16
EVEX.128.F3.MAP6.W0 56 /r VFMADDCPH xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F3.MAP6.W0 56 /r VFMADDCPH ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F3.MAP6.W0 56 /r VFMADDCPH zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst {er}	A	V/V	AVX512-FP16

5.40.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.40.2 Description

This instruction performs a complex multiply and accumulate operation. There are normal and complex conjugate forms of the operation.

The broadcasting and masking for this operation is done on 32-bit quantities representing a pair of FP16 values.

Rounding is performed at every FMA (fused multiply and add) boundary. Execution occurs as if all MXCSR exceptions are masked. MXCSR status bits are updated to reflect exceptional conditions.

5.40.3 Operation

```
VFMADDCPH dest{k1}, src1, src2  (AVX512)
vl = 128, 256, 512

kl := vl / 32

for i := 0 to kl-1:
    if k1[i] or  *no writemask*:
        if broadcasting and src2 is memory:
            tsrc2.fp16[2*i+0] := src2.fp16[0]
            tsrc2.fp16[2*i+1] := src2.fp16[1]
        else:
            tsrc2.fp16[2*i+0] := src2.fp16[2*i+0]
            tsrc2.fp16[2*i+1] := src2.fp16[2*i+1]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        tmp[2*i+0] := dest.fp16[2*i+0] +
             src1.fp16[2*i+0] * tsrc2.fp16[2*i+0]
        tmp[2*i+1] := dest.fp16[2*i+1] +
             src1.fp16[2*i+1] * tsrc2.fp16[2*i+0]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        // non-conjugate version subtracts even term
        dest.fp16[2*i+0] := tmp[2*i+0] -
             src1.fp16[2*i+1] * tsrc2.fp16[2*i+1]
        dest.fp16[2*i+1] := tmp[2*i+1] +
             src1.fp16[2*i+0] * tsrc2.fp16[2*i+1]
    else if *zeroing*:
        dest.fp16[2*i+0] := 0
        dest.fp16[2*i+1] := 0

dest[MAXVL-1:vl] := 0
```

```
VFCMADDCPH dest{k1}, src1, src2  (AVX512)
vl = 128, 256, 512

kl := vl / 32

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        if broadcasting and src2 is memory:
            tsrc2.fp16[2*i+0] := src2.fp16[0]
            tsrc2.fp16[2*i+1] := src2.fp16[1]
        else:
            tsrc2.fp16[2*i+0] := src2.fp16[2*i+0]
            tsrc2.fp16[2*i+1] := src2.fp16[2*i+1]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        tmp[2*i+0] := dest.fp16[2*i+0] +
             src1.fp16[2*i+0] * tsrc2.fp16[2*i+0]
        tmp[2*i+1] := dest.fp16[2*i+1] +
             src1.fp16[2*i+1] * tsrc2.fp16[2*i+0]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        // conjugate version subtracts odd final term
        dest.fp16[2*i+0] := tmp[2*i+0] +
             src1.fp16[2*i+1] * tsrc2.fp16[2*i+1]
        dest.fp16[2*i+1] := tmp[2*i+1] -
             src1.fp16[2*i+0] * tsrc2.fp16[2*i+1]
    else if *zeroing*:
        dest.fp16[2*i+0] := 0
        dest.fp16[2*i+1] := 0

dest[MAXVL-1:vl] := 0
```

5.40.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFCMADDCPH xmm1, xmm2, xmm3/m128	E4*	IOUPD	AVX512-FP16, AVX512VL
VFCMADDCPH ymm1, ymm2, ymm3/m256	E4*	IOUPD	AVX512-FP16, AVX512VL
VFCMADDCPH zmm1, zmm2, zmm3/m512	E4*	IOUPD	AVX512-FP16
VFMADDCPH xmm1, xmm2, xmm3/m128	E4*	IOUPD	AVX512-FP16, AVX512VL
VFMADDCPH ymm1, ymm2, ymm3/m256	E4*	IOUPD	AVX512-FP16, AVX512VL
VFMADDCPH zmm1, zmm2, zmm3/m512	E4*	IOUPD	AVX512-FP16

Additionally:

```
#UD if (dest_reg == src1_reg) or ( dest_reg == src2_reg)
```

5.41 VF[,C]MADDCSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F2.MAP6.W0 57 /r VFCMADDCSH xmm1{k1}{z}, xmm2, xmm3/m32 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.F3.MAP6.W0 57 /r VFMADDCSH xmm1{k1}{z}, xmm2, xmm3/m32 {er}	A	V/V	AVX512-FP16

5.41.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.41.2 Description

This instruction performs a complex multiply and accumulate operation. There are normal and complex conjugate forms of the operation.

The masking for this operation is done on 32-bit quantities representing a pair of FP16 values.

Bits 127:32 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

Rounding is performed at every FMA (fused multiply and add) boundary. Execution occurs as if all MXCSR exceptions are masked. MXCSR status bits are updated to reflect exceptional conditions.

5.41.3 Operation

```
VFMADDCSH dest{k1}, src1, src2  (AVX512)

if k1[0] or *no writemask*:

    tmp[0] := dest.fp16[0] +
         src1.fp16[0] * src2.fp16[0]
    tmp[1] := dest.fp16[1] +
         src1.fp16[1] * src2.fp16[0]

    // non-conjugate version subtracts last even term
    dest.fp16[0] := tmp[0] -
         src1.fp16[1] * src2.fp16[1]
    dest.fp16[1] := tmp[1] +
         src1.fp16[0] * src2.fp16[1]
else if *zeroing*:
    dest.fp16[0] := 0
    dest.fp16[1] := 0

dest[127:32] := src1[127:32] // copy upper part of src1
dest[MAXVL-1:128] := 0
```

```
VFCMADDCSH dest{k1}, src1, src2  (AVX512)

if k1[0] or *no writemask*:
    tmp[0] := dest.fp16[0] +
         src1.fp16[0] * src2.fp16[0]
    tmp[1] := dest.fp16[1] +
         src1.fp16[1] * src2.fp16[0]

    // conjugate version subtracts odd final term
    dest.fp16[0] := tmp[0] +
         src1.fp16[1] * src2.fp16[1]
    dest.fp16[1] := tmp[1] -
         src1.fp16[0] * src2.fp16[1]
else if *zeroing*:
    dest.fp16[0] := 0
    dest.fp16[1] := 0


dest[127:32] := src1[127:32] // copy upper part of src1
dest[MAXVL-1:128] := 0
```

5.41.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFCMADDCSH xmm1, xmm2, xmm3/m32	E10*	IOUPD	AVX512-FP16
VFMADDCSH xmm1, xmm2, xmm3/m32	E10*	IOUPD	AVX512-FP16

Additionally:

```
#UD if (dest_reg == src1_reg) or ( dest_reg == src2_reg)
```

5.42 VF[,C]MULCPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.F2.MAP6.W0 D6 /r VFCMULCPH xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F2.MAP6.W0 D6 /r VFCMULCPH ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F2.MAP6.W0 D6 /r VFCMULCPH zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst {er}	A	V/V	AVX512-FP16
EVEX.128.F3.MAP6.W0 D6 /r VFMULCPH xmm1{k1}{z}, xmm2, xmm3/m128/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.F3.MAP6.W0 D6 /r VFMULCPH ymm1{k1}{z}, ymm2, ymm3/m256/m32bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.F3.MAP6.W0 D6 /r VFMULCPH zmm1{k1}{z}, zmm2, zmm3/m512/m32bcst {er}	A	V/V	AVX512-FP16

5.42.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.42.2 Description

This instruction performs a complex multiply operation. There are normal and complex conjugate forms of the operation. The broadcasting and masking for this operation is done on 32-bit quantities representing a pair of FP16 values.

Rounding is performed at every FMA (fused multiply and add) boundary. Execution occurs as if all MXCSR exceptions are masked. MXCSR status bits are updated to reflect exceptional conditions.

5.42.3 Operation

```
VFMULCPH dest{k1}, src1, src2  (AVX512)
vl = 128, 256, 512

kl := vl / 32

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        if broadcasting and src2 is memory:
            tsrc2.fp16[2*i+0] := src2.fp16[0]
            tsrc2.fp16[2*i+1] := src2.fp16[1]
        else:
            tsrc2.fp16[2*i+0] := src2.fp16[2*i+0]
            tsrc2.fp16[2*i+1] := src2.fp16[2*i+1]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        tmp.fp16[2*i+0] := src1.fp16[2*i+0] * tsrc2.fp16[2*i+0]
        tmp.fp16[2*i+1] := src1.fp16[2*i+1] * tsrc2.fp16[2*i+0]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        // non-conjugate version subtracts last even term
        dest.fp16[2*i+0] := tmp.fp16[2*i+0] -
             src1.fp16[2*i+1] * tsrc2.fp16[2*i+1]
        dest.fp16[2*i+1] := tmp.fp16[2*i+1] +
             src1.fp16[2*i+0] * tsrc2.fp16[2*i+1]
    else if *zeroing*:
        dest.fp16[2*i+0] := 0
        dest.fp16[2*i+1] := 0

dest[MAXVL-1:vl] := 0
```

```
VFCMULCPH dest{k1}, src1, src2  (AVX512)
vl = 128, 256, 512

kl := vl / 32

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        if broadcasting and src2 is memory:
            tsrc2.fp16[2*i+0] := src2.fp16[0]
            tsrc2.fp16[2*i+1] := src2.fp16[1]
        else:
            tsrc2.fp16[2*i+0] := src2.fp16[2*i+0]
            tsrc2.fp16[2*i+1] := src2.fp16[2*i+1]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        tmp.fp16[2*i+0] := src1.fp16[2*i+0] * tsrc2.fp16[2*i+0]
        tmp.fp16[2*i+1] := src1.fp16[2*i+1] * tsrc2.fp16[2*i+0]

for i := 0 to kl-1:
    if k1[i] or *no writemask*:
        // conjugate version subtracts odd final term
        dest.fp16[2*i] := tmp.fp16[2*i+0] +
             src1.fp16[2*i+1] * tsrc2.fp16[2*i+1]
        dest.fp16[2*i+1] := tmp.fp16[2*i+1] -
             src1.fp16[2*i+0] * tsrc2.fp16[2*i+1]
    else if *zeroing*:
        dest.fp16[2*i+0] := 0
        dest.fp16[2*i+1] := 0

dest[MAXVL-1:vl] := 0
```

5.42.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFCMULCPH xmm1, xmm2, xmm3/m128	E4*	IOUPD	AVX512-FP16, AVX512VL
VFCMULCPH ymm1, ymm2, ymm3/m256	E4*	IOUPD	AVX512-FP16, AVX512VL
VFCMULCPH zmm1, zmm2, zmm3/m512	E4*	IOUPD	AVX512-FP16
VFMULCPH xmm1, xmm2, xmm3/m128	E4*	IOUPD	AVX512-FP16, AVX512VL
VFMULCPH ymm1, ymm2, ymm3/m256	E4*	IOUPD	AVX512-FP16, AVX512VL
VFMULCPH zmm1, zmm2, zmm3/m512	E4*	IOUPD	AVX512-FP16

Additionally:

```
#UD if (dest_reg == src1_reg) or ( dest_reg == src2_reg)
```

5.43 VF[,C]MULCSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F2.MAP6.W0 D7 /r VFCMULCSH xmm1{k1}{z}, xmm2, xmm3/m32 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.F3.MAP6.W0 D7 /r VFMULCSH xmm1{k1}{z}, xmm2, xmm3/m32 {er}	A	V/V	AVX512-FP16

5.43.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.43.2 Description

This instruction performs a complex multiply operation. There are normal and complex conjugate forms of the operation. The masking for this operation is done on 32-bit quantities representing a pair of FP16 values.

Bits 127:32 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

Rounding is performed at every FMA (fused multiply and add) boundary. Execution occurs as if all MXCSR exceptions are masked. MXCSR status bits are updated to reflect exceptional conditions.

5.43.3 Operation

```
VFMULCSH dest{k1}, src1, src2  (AVX512)
kl := vl / 32

if k1[0] or *no writemask*:
    // non-conjugate version subtracts last even term
    tmp.fp16[0] := src1.fp16[0] * src2.fp16[0]
    tmp.fp16[1] := src1.fp16[1] * src2.fp16[0]

    dest.fp16[0] := tmp.fp16[0] -
         src1.fp16[1] * src2.fp16[1]
    dest.fp16[1] := tmp.fp16[1] +
         src1.fp16[0] * src2.fp16[1]
else if *zeroing*:
    dest.fp16[0] := 0
    dest.fp16[1] := 0

dest[127:32] := src1[127:32] // copy upper part of src1
dest[MAXVL-1:128] := 0
```

```
VFCMULCSH dest{k1}, src1, src2  (AVX512)
kl := vl / 32

if k1[0] or *no writemask*:

    tmp.fp16[0] := src1.fp16[0] * src2.fp16[0]
    tmp.fp16[1] := src1.fp16[1] * src2.fp16[0]

    // conjugate version subtracts odd final term
    dest.fp16[0] := tmp.fp16[0] +
         src1.fp16[1] * src2.fp16[1]
    dest.fp16[1] := tmp.fp16[1] -
         src1.fp16[0] * src2.fp16[1]
else if *zeroing*:
    dest.fp16[0] := 0
    dest.fp16[1] := 0

dest[127:32] := src1[127:32] // copy upper part of src1
dest[MAXVL-1:128] := 0
```

5.43.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFCMULCSH xmm1, xmm2, xmm3/m32	E10*	IOUPD	AVX512-FP16
VFMULCSH xmm1, xmm2, xmm3/m32	E10*	IOUPD	AVX512-FP16

Additionally:

```
#UD if (dest_reg == src1_reg) or ( dest_reg == src2_reg)
```

5.44 VFMADDSUB[132,213,231]PH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 96 /r VFMADDSUB132PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 96 /r VFMADDSUB132PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 96 /r VFMADDSUB132PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 A6 /r VFMADDSUB213PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 A6 /r VFMADDSUB213PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 A6 /r VFMADDSUB213PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 B6 /r VFMADDSUB231PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 B6 /r VFMADDSUB231PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 B6 /r VFMADDSUB231PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.44.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.44.2 Description

Performs a packed multiply-add (odd elements) or multiply-subtract (even elements) computation on FP16 values using three source operands and writes the results in the destination operand. The destination operand is also the first source operand. The notation' “132”, “213” and “231” indicate the use of the operands in $A * B \pm C$, where each digit corresponds to the operand number, with the destination being operand 1.

	odd elements	even elements
132	dest = dest*src3+src2	dest = dest*src3-src2
231	dest = src2*src3+dest	dest = src2*src3-dest
213	dest = src2*dest+src3	dest = src2*dest-src3

The destination elements are updated according to the writemask.

5.44.3 Operation

```
VFMADDSUB132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16

IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * SRC3.fp16[j] - SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * SRC3.fp16[j] + SRC2.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMADDSUB132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * t3 - SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * t3 + SRC2.fp16[j])
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMADDSUB213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
       IF *j is even*:
         DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*DEST.fp16[j] - SRC3.fp16[j])
       ELSE
         DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*DEST.fp16[j] + SRC3.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMADDSUB213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * DEST.fp16[j] - t3)
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * DEST.fp16[j] + t3)
   ELSE IF *zeroing*:
       DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMADDSUB231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
    IF k1[j] OR *no writemask*:
        IF *j is even:
            DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * SRC3.fp16[j] - DEST.fp16[j])
        ELSE:
            DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * SRC3.fp16[j] + DEST.fp16[j])
    ELSE IF *zeroing*:
        DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMADDSUB231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * t3 - DEST.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * t3 + DEST.fp16[j])
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.44.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFMADDSUB132PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADDSUB132PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADDSUB132PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMADDSUB213PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADDSUB213PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADDSUB213PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMADDSUB231PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADDSUB231PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADDSUB231PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.45 VFMSUBADD[132,213,231]PH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 97 /r VFMSUBADD132PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 97 /r VFMSUBADD132PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 97 /r VFMSUBADD132PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 A7 /r VFMSUBADD213PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 A7 /r VFMSUBADD213PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 A7 /r VFMSUBADD213PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 B7 /r VFMSUBADD231PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 B7 /r VFMSUBADD231PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 B7 /r VFMSUBADD231PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.45.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.45.2 Description

Performs a packed multiply-add (even elements) or multiply-subtract (odd elements) computation on FP16 values using three source operands and writes the results in the destination operand. The destination operand is also the first source operand. The notation' “132”, “213” and “231” indicate the use of the operands in $A * B \pm C$, where each digit corresponds to the operand number, with the destination being operand 1.

	odd elements	even elements
132	dest = dest*src3-src2	dest = dest*src3+src2
231	dest = src2*src3-dest	dest = src2*src3+dest
213	dest = src2*dest-src3	dest = src2*dest+src3

The destination elements are updated according to the writemask.

5.45.3 Operation

```
VFMSUBADD132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j]*SRC3.fp16[j] + SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j]*SRC3.fp16[j] - SRC2.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMSUBADD132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * t3 + SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * t3 - SRC2.fp16[j])
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMSUBADD213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
       IF *j is even*:
         DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*DEST.fp16[j] + SRC3.fp16[j])
       ELSE
         DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*DEST.fp16[j] - SRC3.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMSUBADD213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * DEST.fp16[j] + t3 )
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * DEST.fp16[j] - t3 )
   ELSE IF *zeroing*:
       DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMSUBADD231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
    IF k1[j] OR *no writemask*:
        IF *j is even:
            DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*SRC3.fp16[j] + DEST.fp16[j])
        ELSE:
            DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*SRC3.fp16[j] - DEST.fp16[j])
    ELSE IF *zeroing*:
        DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VFMSUBADD231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *j is even*:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * t3 + DEST.fp16[j] )
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * t3 - DEST.fp16[j] )
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.45.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFMSUBADD132PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUBADD132PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUBADD132PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMSUBADD213PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUBADD213PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUBADD213PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMSUBADD231PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUBADD231PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUBADD231PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.46 VF[,N]MADD[132,213,231]PH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 98 /r VFMADD132PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 98 /r VFMADD132PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 98 /r VFMADD132PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 A8 /r VFMADD213PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 A8 /r VFMADD213PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 A8 /r VFMADD213PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 B8 /r VFMADD231PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 B8 /r VFMADD231PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 B8 /r VFMADD231PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 9C /r VFNMADD132PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 9C /r VFNMADD132PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 9C /r VFNMADD132PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 AC /r VFNMADD213PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 AC /r VFNMADD213PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 AC /r VFNMADD213PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

Table continued on next page...

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 BC /r VFNMADD231PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 BC /r VFNMADD231PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 BC /r VFNMADD231PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.46.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.46.2 Description

Performs a packed multiply-add or negated multiply-add computation on FP16 values using three source operands and writes the results in the destination operand. The destination operand is also the first source operand. The “N” (negated) forms of this instruction add the negated infinite precision intermediate product from the corresponding remaining operand. The notation’ “132”, “213” and “231” indicate the use of the operands in $\pm A * B + C$, where each digit corresponds to the operand number, with the destination being operand 1.

132	dest = $\pm$ dest*src3+src2
231	dest = $\pm$ src2*src3+dest
213	dest = $\pm$ src2*dest+src3

The destination elements are updated according to the writemask.

5.46.3 Operation

```
VF[,N]MADD132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-DEST.fp16[j]*SRC3.fp16[j] + SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j]*SRC3.fp16[j] + SRC2.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MADD132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16



FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-DEST.fp16[j] * t3 + SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * t3 + SRC2.fp16[j])
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MADD213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16

IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
       IF *negative form*:
         DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j]*DEST.fp16[j] + SRC3.fp16[j])
       ELSE
         DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*DEST.fp16[j] + SRC3.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MADD213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j] * DEST.fp16[j] + t3 )
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * DEST.fp16[j] + t3 )
   ELSE IF *zeroing*:
       DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MADD231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16

IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
    IF k1[j] OR *no writemask*:
        IF *negative form:
            DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j]*SRC3.fp16[j] + DEST.fp16[j])
        ELSE:
            DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*SRC3.fp16[j] + DEST.fp16[j])
    ELSE IF *zeroing*:
        DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MADD231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j] * t3 + DEST.fp16[j] )
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * t3 + DEST.fp16[j] )
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.46.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFMADD132PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADD132PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADD132PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMADD213PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADD213PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADD213PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMADD231PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADD231PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMADD231PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFNMADD132PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMADD132PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMADD132PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFNMADD213PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMADD213PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMADD213PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFNMADD231PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMADD231PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMADD231PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.47 VF[,N]MADD[132,213,231]SH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.66.MAP6.W0 99 /r VFMADD132SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 A9 /r VFMADD213SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 B9 /r VFMADD231SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 9D /r VFNMADD132SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 AD /r VFNMADD213SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 BD /r VFNMADD231SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.47.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.47.2 Description

Performs a scalar multiply-add or negated multiply-add computation on the low FP16 values using three source operands and writes the result in the destination operand. The destination operand is also the first source operand. The "N" (negated) forms of this instruction add the negated infinite precision intermediate product from the corresponding remaining operand. The notation' "132", "213" and "231" indicate the use of the operands in $\pm A * B + C$, where each digit corresponds to the operand number, with the destination being operand 1.

132	dest = $\pm$ dest*src3+src2
231	dest = $\pm$ src2*src3+dest
213	dest = $\pm$ src2*dest+src3

Bits 127:16 of the destination operand are preserved. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.47.3 Operation

```
VF[,N]MADD132SH DEST, SRC2, SRC3 (EVEX encoded versions)
IF EVEX.b = 1 and SRC3 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   IF *negative form*:
       DEST.fp16[0] := RoundFPControl(-DEST.fp16[0]*SRC3.fp16[0] + SRC2.fp16[0])
   ELSE:
       DEST.fp16[0] := RoundFPControl(DEST.fp16[0]*SRC3.fp16[0] + SRC2.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else DEST.fp16[0] remains unchanged

//DEST[127:16] remains unchanged
DEST[MAXVL-1:128] := 0
```

```
VF[,N]MADD213SH DEST, SRC2, SRC3 (EVEX encoded versions)
IF EVEX.b = 1 and SRC3 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   IF *negative form:
       DEST.fp16[0] := RoundFPControl(-SRC2.fp16[0]*DEST.fp16[0] + SRC3.fp16[0])
   ELSE:
       DEST.fp16[0] := RoundFPControl(SRC2.fp16[0]*DEST.fp16[0] + SRC3.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else DEST.fp16[0] remains unchanged

//DEST[127:16] remains unchanged
DEST[MAXVL-1:128] := 0
```

```
VF[,N]MADD231SH DEST, SRC2, SRC3 (EVEX encoded versions)
IF EVEX.b = 1 and SRC3 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   IF *negative form*:
       DEST.fp16[0] := RoundFPControl(-SRC2.fp16[0]*SRC3.fp16[0] + DEST.fp16[0])
   ELSE:
       DEST.fp16[0] := RoundFPControl(SRC2.fp16[0]*SRC3.fp16[0] + DEST.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else DEST.fp16[0] remains unchanged

//DEST[127:16] remains unchanged
DEST[MAXVL-1:128] := 0
```

5.47.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFMADD132SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFMADD213SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFMADD231SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFNMADD132SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFNMADD213SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFNMADD231SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16

5.48 VF[,N]MSUB[132,213,231]PH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.66.MAP6.W0 9A /r VFMSUB132PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 9A /r VFMSUB132PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 9A /r VFMSUB132PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 AA /r VFMSUB213PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 AA /r VFMSUB213PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 AA /r VFMSUB213PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 BA /r VFMSUB231PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 BA /r VFMSUB231PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 BA /r VFMSUB231PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 9E /r VFNMSUB132PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 9E /r VFNMSUB132PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 9E /r VFNMSUB132PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16
EVEX.128.66.MAP6.W0 AE /r VFNMSUB213PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 AE /r VFNMSUB213PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 AE /r VFNMSUB213PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

Table continued on next page...

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.66.MAP6.W0 BE /r VFNMSUB231PH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 BE /r VFNMSUB231PH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 BE /r VFNMSUB231PH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.48.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	FULL	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.48.2 Description

Performs a packed multiply-subtract or a negated multiply-subtract computation on FP16 values using three source operands and writes the results in the destination operand. The destination operand is also the first source operand. The “N” (negated) forms of this instruction subtract the remaining operand from the negated infinite precision intermediate product. The notation’ “132”, “213” and “231” indicate the use of the operands in $\pm A * B - C$, where each digit corresponds to the operand number, with the destination being operand 1.

132	dest = $\pm$ dest*src3-src2
231	dest = $\pm$ src2*src3-dest
213	dest = $\pm$ src2*dest-src3

The destination elements are updated according to the writemask.

5.48.3 Operation

```
VF[,N]MSUB132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-DEST.fp16[j]*SRC3.fp16[j] - SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j]*SRC3.fp16[j] - SRC2.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MSUB132PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-DEST.fp16[j] * t3 - SRC2.fp16[j])
      ELSE:
          DEST.fp16[j] := RoundFPControl(DEST.fp16[j] * t3 - SRC2.fp16[j])
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MSUB213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
       IF *negative form*:
         DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j]*DEST.fp16[j] - SRC3.fp16[j])
       ELSE
         DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*DEST.fp16[j] - SRC3.fp16[j])
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MSUB213PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j] * DEST.fp16[j] - t3 )
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * DEST.fp16[j] - t3 )
   ELSE IF *zeroing*:
       DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MSUB231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
    IF k1[j] OR *no writemask*:
        IF *negative form:
            DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j]*SRC3.fp16[j] - DEST.fp16[j])
        ELSE:
            DEST.fp16[j] := RoundFPControl(SRC2.fp16[j]*SRC3.fp16[j] - DEST.fp16[j])
    ELSE IF *zeroing*:
        DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VF[,N]MSUB231PH DEST, SRC2, SRC3 (EVEX encoded versions) when src3 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
         t3 := SRC3.fp16[0]
      ELSE:
         t3 := SRC3.fp16[j]
      IF *negative form*:
          DEST.fp16[j] := RoundFPControl(-SRC2.fp16[j] * t3 - DEST.fp16[j] )
      ELSE:
          DEST.fp16[j] := RoundFPControl(SRC2.fp16[j] * t3 - DEST.fp16[j] )
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.48.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFMSUB132PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUB132PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUB132PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMSUB213PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUB213PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUB213PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFMSUB231PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUB231PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFMSUB231PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFNMSUB132PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMSUB132PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMSUB132PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFNMSUB213PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMSUB213PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMSUB213PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16
VFNMSUB231PH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMSUB231PH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VFNMSUB231PH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.49 VF[,N]MSUB[132,213,231]SH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.66.MAP6.W0 9B /r VFMSUB132SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 AB /r VFMSUB213SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 BB /r VFMSUB231SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 9F /r VFNMSUB132SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 AF /r VFNMSUB213SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16
EVEX.LLIG.66.MAP6.W0 BF /r VFNMSUB231SH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.49.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(rw)	VVVV(r)	MODRM.R/M(r)	N/A

5.49.2 Description

Performs a scalar multiply-subtract or negated multiply-subtract computation on the low FP16 values using three source operands and writes the result in the destination operand. The destination operand is also the first source operand. The "N" (negated) forms of this instruction subtract the remaining operand from the negated infinite precision intermediate product. The notation' "132", "213" and "231" indicate the use of the operands in $\pm A * B - C$, where each digit corresponds to the operand number, with the destination being operand 1.

132	dest = $\pm$ dest*src3-src2
231	dest = $\pm$ src2*src3-dest
213	dest = $\pm$ src2*dest-src3

Bits 127:16 of the destination operand are preserved. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.49.3 Operation

```
VF[,N]MSUB132SH DEST, SRC2, SRC3 (EVEX encoded versions)
IF EVEX.b = 1 and SRC3 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   IF *negative form*:
       DEST.fp16[0] := RoundFPControl(-DEST.fp16[0]*SRC3.fp16[0] - SRC2.fp16[0])
   ELSE:
       DEST.fp16[0] := RoundFPControl(DEST.fp16[0]*SRC3.fp16[0] - SRC2.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else DEST.fp16[0] remains unchanged

//DEST[127:16] remains unchanged
DEST[MAXVL-1:128] := 0
```

```
VF[,N]MSUB213SH DEST, SRC2, SRC3 (EVEX encoded versions)
IF EVEX.b = 1 and SRC3 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   IF *negative form:
       DEST.fp16[0] := RoundFPControl(-SRC2.fp16[0]*DEST.fp16[0] - SRC3.fp16[0])
   ELSE:
       DEST.fp16[0] := RoundFPControl(SRC2.fp16[0]*DEST.fp16[0] - SRC3.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else DEST.fp16[0] remains unchanged

//DEST[127:16] remains unchanged
DEST[MAXVL-1:128] := 0
```

```
VF[,N]MSUB231SH DEST, SRC2, SRC3 (EVEX encoded versions)
IF EVEX.b = 1 and SRC3 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   IF *negative form*:
       DEST.fp16[0] := RoundFPControl(-SRC2.fp16[0]*SRC3.fp16[0] - DEST.fp16[0])
   ELSE:
       DEST.fp16[0] := RoundFPControl(SRC2.fp16[0]*SRC3.fp16[0] - DEST.fp16[0])
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else DEST.fp16[0] remains unchanged

//DEST[127:16] remains unchanged
DEST[MAXVL-1:128] := 0
```

5.49.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFMSUB132SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFMSUB213SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFMSUB231SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFNMSUB132SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFNMSUB213SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16
VFNMSUB231SH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16

5.50 VFPCLASSPH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.128.NP.0F3A.W0 66 /r /ib VFPCLASSPH k1{k2}, xmm1/m128/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.0F3A.W0 66 /r /ib VFPCLASSPH k1{k2}, ymm1/m256/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.0F3A.W0 66 /r /ib VFPCLASSPH k1{k2}, zmm1/m512/m16bcst, imm8	A	V/V	AVX512-FP16

5.50.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	IMM8(r)	N/A

5.50.2 Description

Checks the packed FP16 values in the source operand for special categories, specified by the set bits in the imm8 byte. Each set bit in imm8 specifies a category of floating-point values that the input data element is classified against. See Figure 5.1 for the categories. The classified results of all specified categories of an input value are ORed together to form the final boolean result for the input element. The result is written to the corresponding bits in the destination mask register according to the writemask.

Bits	Category	Classifier
imm8[0]	QNAN	Checks for QNAN
imm8[1]	PosZero	Checks +0
imm8[2]	NegZero	Checks for -0
imm8[3]	PosINF	Checks for $+\infty$
imm8[4]	NegINF	Checks for $-\infty$
imm8[5]	Denormal	Checks for Denormal
imm8[6]	Negative	Checks for Negative finite
imm8[7]	SNAN	Checks for SNAN

Figure 5.1: Classifier operations for VFPCLASS[PH,SH]

5.50.3 Operation

```
def check_fp_class_fp16(tsrc, imm8):
   negative := tsrc[15]
   exponent_all_ones := (tsrc[14:10] == 0x1F)
   exponent_all_zeros := (tsrc[14:10] == 0)
   mantissa_all_zeros := (tsrc[9:0] == 0)
   zero := exponent_all_zeros and mantissa_all_zeros
   signaling_bit := tsrc[9]

   snan := exponent_all_ones and not(mantissa_all_zeros) and not(signaling_bit)
   qnan := exponent_all_ones and not(mantissa_all_zeros) and signaling_bit
   positive_zero := not(negative) and zero
   negative_zero := negative      and zero
   positive_infinity := not(negative) and exponent_all_ones and mantissa_all_zeros
   negative_infinity := negative      and exponent_all_ones and mantissa_all_zeros
   denormal := exponent_all_zeros and not(mantissa_all_zeros)
   finite_negative := negative and not(exponent_all_ones) and not(zero)

   return (imm8[0] and qnan) OR
          (imm8[1] and positive_zero)  OR
          (imm8[2] and negative_zero) OR
          (imm8[3] and positive_infinity) OR
          (imm8[4] and negative_infinity) OR
          (imm8[5] and denormal) OR
          (imm8[6] and finite_negative) OR
          (imm8[7] and snan)
```

```
VFPCLASSPH dest{k2}, src, imm8
VL = 128, 256, or 512

KL := VL / 16

FOR i := 0 to KL-1:
    IF k2[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := SRC.fp16[0]
       ELSE:
         tsrc := SRC.fp16[i]
       DEST.bit[i] := check_fp_class_fp16(tsrc, imm8)
    ELSE:
       DEST.bit[i] := 0


DEST[MAXKL-1:kl] := 0
```

5.50.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VFPCLASSPH k1, xmm1/m128, imm8	E4	N/A	AVX512-FP16, AVX512VL
VFPCLASSPH k1, ymm1/m256, imm8	E4	N/A	AVX512-FP16, AVX512VL
VFPCLASSPH k1, zmm1/m512, imm8	E4	N/A	AVX512-FP16

5.51 VFPCLASSSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.0F3A.W0 67 /r /ib VFPCLASSSH k1{k2}, xmm1/m16, imm8	A	V/V	AVX512-FP16

5.51.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	IMM8(r)	N/A

5.51.2 Description

Checks the low FP16 value in the source operand for special categories, specified by the set bits in the imm8 byte. Each set bit in imm8 specifies a category of floating-point values that the input data element is classified against. See Figure 5.1 for the categories. The classified results of all specified categories of an input value are ORed together to form the final boolean result for the input element. The result is written to the low bit in the destination mask register according to the writemask. The other bits in the destination mask register are zeroed.

5.51.3 Operation

```
VFPCLASSSH dest{k2}, src, imm8

IF k2[0] or  *no writemask*:
   DEST.bit[0] := check_fp_class_fp16(src.fp16[0], imm8) // see VFPCLASSPH
ELSE:
   DEST.bit[0] := 0

DEST[MAXKL-1:1] := 0
```

5.51.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VFPCLASSSH k1, xmm1/m16, imm8	E10	N/A	AVX512-FP16

5.52 VGETEXPPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 42 /r VGETEXPPH xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 42 /r VGETEXPPH ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 42 /r VGETEXPPH zmm1{k1}{z}, zmm2/m512/m16bcst {sae}	A	V/V	AVX512-FP16

5.52.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.52.2 Description

Extracts the biased exponents from the normalized FP16 representation of each word element of the source operand (the second operand) as unbiased signed integer value, or convert the denormal representation of input data to unbiased negative integer values. Each integer value of the unbiased exponent is converted to an FP16 value and written to the corresponding word elements of the destination operand (the first operand) as FP16 numbers.

The destination elements are updated according to the writemask.

Each GETEXP operation converts the exponent value into a FP number (permitting input value in denormal representation). Special cases of input values are listed in Figure 5.2.

The formula is:
GETEXP(x) = floor(log_2(|x|))
Notation floor(x) stands for maximal integer not exceeding real number x.

Software usage of VGETEXPxx and VGETMANTxx instructions generally involve a combination of GETEXP operation and GETMANT operation (see VGETMANTPH). Thus VGETEXPxx instruction do not require software to handle SIMD FP exceptions.

input operand	result	comments
$src1$ = NaN	QNaN(src1)	if SRC == SNaN, then #IE.
$0 < \|src1\| < \infty$	floor(log_2(|src1|)	if SRC == denormal, then #DE
$\|src1\| = +\infty$	$+\infty$	
$\|src1\| = 0$	$-\infty$	

Figure 5.2: VGETEXPPH special cases

5.52.3 Operation

```
def normalize_exponent_tiny_fp16(src):
  jbit := 0
  // src & dst are FP16 numbers with sign(1b), exp(5b) and fraction (10b) fields
  dst.exp := 1    // write bits 14:10
  dst.fraction := src.fraction  // copy bits 9:0
  while jbit == 0:
      jbit := dst.fraction[9]    // msb of the fraction
      dst.fraction := dst.fraction << 1
      dst.exp := dst.exp - 1
  dst.fraction := 0
  return dst
```

```
def getexp_fp16(src):
   src.sign := 0  // make positive
   exponent_all_ones := (src[14:10] == 0x1F)
   exponent_all_zeros := (src[14:10] == 0)
   mantissa_all_zeros := (src[9:0] == 0)
   zero := exponent_all_zeros and mantissa_all_zeros
   signaling_bit := src[9]

   nan  := exponent_all_ones and not(mantissa_all_zeros)
   snan := nan and not(signaling_bit)
   qnan := nan and signaling_bit
   positive_infinity := not(negative) and exponent_all_ones and mantissa_all_zeros
   denormal := exponent_all_zeros and not(mantissa_all_zeros)

   if nan:
      if snan:
          MXCSR.IE := 1
      return qnan(src)     // convert snan to a qnan
   if positive_infinity:
      return src
   if zero:
      return -INF
   if denormal:
      tmp := normalize_exponent_tiny_fp16(src)
      MXCSR.DE := 1
   else:
      tmp := src
   tmp := SAR(tmp, 10) // shift arithmetic right
   tmp := tmp - 15     // subtract bias
   return convert_integer_to_fp16(tmp)
```

```
VGETEXPPH dest{k1}, src
VL = 128, 256, or 512

KL := VL / 16

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := getexp_fp16(tsrc)
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.52.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VGETEXPPH xmm1, xmm2/m128	E2	ID	AVX512-FP16, AVX512VL
VGETEXPPH ymm1, ymm2/m256	E2	ID	AVX512-FP16, AVX512VL
VGETEXPPH zmm1, zmm2/m512	E2	ID	AVX512-FP16

5.53 VGETEXPSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.66.MAP6.W0 43 /r VGETEXPSH xmm1{k1}{z}, xmm2, xmm3/m16 {sae}	A	V/V	AVX512-FP16

5.53.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.53.2 Description

Extracts the biased exponents from the normalized FP16 representation of the low word element of the source operand (the second operand) as unbiased signed integer value, or convert the denormal representation of input data to an unbiased negative integer value. The integer value of the unbiased exponent is converted to an FP16 value and written to the low word element of the destination operand (the first operand) as an FP16 number.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

Each GETEXP operation converts the exponent value into a FP number (permitting input value in denormal representation). Special cases of input values are listed in Figure 5.2.

The formula is:
GETEXP(x) = floor(log_2(|x|))
Notation floor(x) stands for maximal integer not exceeding real number x.

Software usage of VGETEXPxx and VGETMANTxx instructions generally involve a combination of GETEXP operation and GETMANT operation (see VGETMANTSH). Thus VGETEXPxx instruction do not require software to handle SIMD FP exceptions.

5.53.3 Operation

```
VGETEXPSH dest{k1}, src1, src2

IF k1[0] or  *no writemask*:
   DEST.fp16[0] := getexp_fp16(src2.fp16[0])         // see VGETEXPPH
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
//else DEST.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.53.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VGETEXPSH xmm1, xmm2, xmm3/m16	E3	ID	AVX512-FP16

5.54 VGETMANTPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.0F3A.W0 26 /r /ib VGETMANTPH xmm1{k1}{z}, xmm2/m128/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.0F3A.W0 26 /r /ib VGETMANTPH ymm1{k1}{z}, ymm2/m256/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.0F3A.W0 26 /r /ib VGETMANTPH zmm1{k1}{z}, zmm2/m512/m16bcst {sae}, imm8	A	V/V	AVX512-FP16

5.54.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	IMM8(r)	N/A

5.54.2 Description

Convert FP16 values in the source operand (the second operand) to FP16 values with the mantissa normalization and sign control specified by the imm8 byte, see Figure 5.3. The converted results are written to the destination operand (the first operand) using writemask k1. The normalized mantissa is specified by interv (imm8[1:0]) and the sign control (SC) is specified by bits 3:2 of the immediate byte.

The destination elements are updated according to the writemask.

imm8[3:2]	Sign Control (SC) 0b00 -> sign(SRC) 0b01 -> 0 0b1x -> qNaN_Indefinite if sign(SRC)!=0
imm8[1:0]	Interv 0b00 -> interval is $[1, 2)$ 0b01 -> interval is $[1/2, 2)$ 0b10 -> interval is $[1/2, 1)$ 0b11 -> interval is $[3/4, 3/2)$

Figure 5.3: Imm8 controls for vgetmant*

For each input FP16 value x, The conversion operation is:
GetMant(x) = $\pm\ 2^k|x.significand|$
where: $1 \leq |x.significand| < 2$.

Unbiased exponent k depends on the interval range defined by interv and whether the exponent of the source is even or odd. The sign of the final result is determined by sc and the source sign and the leading fraction bit.

The encoded value of imm8[1:0] and sign control are shown in Figure 5.3.

Each converted FP16 result is encoded according to the sign control, the unbiased exponent k (adding bias) and a mantissa normalized to the range specified by interv.

The GetMant() function follows Figure 5.4 when dealing with floating-point special numbers.

Input	**Result**	**Exceptions, Comments**
NaN	QNaN(SRC)	Ignore Interv. If (SRC=SNaN then #IE.
$+\infty$	1.0	Ignore Interv.
$+0$	1.0	Ignore Interv.
-0	if SC[0] then +1.0 else -1.0	Ignore Interv.
$-\infty$	if SC[1] then QNaN_Indefinite, else if SC[0] then +1.0 else -1.0	Ignore Interv. If SC[1] then #IE.
negative	SC[1] ? QNaN_Indefinite : $Getmant(SRC)^1$	if SC[1] then #IE

NOTES:
1. In case SC[1]==0, the sign of Getmant(SRC) is declared according to SC[0].

Figure 5.4: Special cases vgetmant*

5.54.3 Operation

```
def getmant_fp16(src, sign_control, normalization_interval):
   bias          := 15
   dst.sign      := sign_control[0] ? 0 : src.sign
   signed_one    := sign_control[0] ? +1.0 : -1.0
   dst.exp       := src.exp
   dst.fraction  := src.fraction
   zero          := (dst.exp = 0) and (dst.fraction = 0)
   denormal      := (dst.exp = 0) and (dst.fraction != 0)
   infinity      := (dst.exp = 0x1F) and (dst.fraction = 0)
   nan           := (dst.exp = 0x1F) and (dst.fraction != 0)
   src_signaling := src.fraction[9]
   snan          := nan and (src_signaling = 0)
   positive      := (src.sign = 0)
   negative      := (src.sign = 1)
   if nan:
       if snan:
          MXCSR.IE := 1
       return qnan(src)

   if positive and (zero or infinity):
       return 1.0
   if negative:
      if zero:
          return signed_one
      if infinity:
          if sign_control[1]:
             MXCSR.IE := 1
             return QNaN_Indefinite
          return signed_one
      if sign_control[1]:
          MXCSR.IE := 1
          return QNaN_Indefinite

   // continued on next page...
   if denormal:
      jbit := 0
      dst.exp := bias  // set exponent to bias value
      while jbit = 0:
          jbit := dst.fraction[9]
          dst.fraction := dst.fraction << 1
          dst.exp : = dst.exp - 1
      MXCSR.DE := 1

   unbaiased_exp := dst.exp - bias
   odd_exp       := unbaiased_exp[0]
   signaling_bit := dst.fraction[9]
   if normalization_interval = 0b00:
       dst.exp := bias
   else if normalization_interval = 0b01:
       dst.exp := odd_exp ? bias-1 : bias
   else if normalization_interval = 0b10:
       dst.exp := bias-1
   else if normalization_interval = 0b11:
       dst.exp := signaling_bit ? bias-1 : bias
   return dst
```

```
VGETMANTPH dest{k1}, src, imm8
VL = 128, 256, or 512
KL := VL / 16
sign_control           := imm8[3:2]
normalization_interval := imm8[1:0]

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := getmant_fp16(tsrc, sign_control, normalization_interval)
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.54.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VGETMANTPH xmm1, xmm2/m128, imm8	E2	ID	AVX512-FP16, AVX512VL
VGETMANTPH ymm1, ymm2/m256, imm8	E2	ID	AVX512-FP16, AVX512VL
VGETMANTPH zmm1, zmm2/m512, imm8	E2	ID	AVX512-FP16

5.55 VGETMANTSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.0F3A.W0 27 /r /ib VGETMANTSH xmm1{k1}{z}, xmm2, xmm3/m16 {sae}, imm8	A	V/V	AVX512-FP16

5.55.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	IMM8(r)

5.55.2 Description

Convert FP16 value in the low element of the second source operand to FP16 values with the mantissa normalization and sign control specified by the imm8 byte, see Figure 5.3. The converted result is written to the low element of the destination operand using writemask k1. The normalized mantissa is specified by interv (imm8[1:0]) and the sign control (SC) is specified by bits 3:2 of the immediate byte.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

For each input FP16 value x, The conversion operation is:
GetMant(x) = $\pm\ 2^k|x.significand|$
where: $1 \leq |x.significand| < 2$.

Unbiased exponent k depends on the interval range defined by interv and whether the exponent of the source is even or odd. The sign of the final result is determined by sc and the source sign and the leading fraction bit.

The encoded value of imm8[1:0] and sign control are shown in Figure 5.3.

Each converted FP16 result is encoded according to the sign control, the unbiased exponent k (adding bias) and a mantissa normalized to the range specified by interv.

The GetMant() function uses Figure 5.4 when dealing with floating-point special numbers.

5.55.3 Operation

```
VGETMANTSH dest{k1}, src1, src2, imm8
sign_control           := imm8[3:2]
normalization_interval := imm8[1:0]

IF k1[0] or  *no writemask*:
   dest.fp16[0] := getmant_fp16(src2.fp16[0],   // see VGETMANTPH
                               sign_control,
                               normalization_interval)
ELSE IF *zeroing*:
   dest.fp16[0] := 0
//else dest.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.55.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VGETMANTSH xmm1, xmm2, xmm3/m16, imm8	E3	ID	AVX512-FP16

5.56 VMAXPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 5F /r VMAXPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 5F /r VMAXPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 5F /r VMAXPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {sae}	A	V/V	AVX512-FP16

5.56.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.56.2 Description

Performs a SIMD compare of the packed FP16 values in the first source operand and the second source operand and returns the maximum value for each pair of values to the destination operand.

If the values being compared are both 0.0s (of either sign), the value in the second operand (source operand) is returned. If a value in the second operand is an SNaN, then SNaN is forwarded unchanged to the destination (that is, a QNaN version of the SNaN is not returned).

If only one value is a NaN (SNaN or QNaN) for this instruction, the second operand (source operand), either a NaN or a valid floating-point value, is written to the result. If instead of this behavior, it is required that the NaN source operand (from either the first or second operand) be returned, the action of VMAXPH can be emulated using a sequence of instructions, such as, a comparison followed by AND, ANDN and OR.

EVEX encoded versions: The first source operand (the second operand) is a ZMM/YMM/XMM register. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcast from a 16-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1.

5.56.3 Operation

```
def MAX(SRC1, SRC2):
  IF (SRC1 = 0.0) and (SRC2 = 0.0):
      DEST := SRC2
  ELSE IF (SRC1 = NaN):
      DEST := SRC2
  ELSE IF (SRC2 = NaN):
      DEST := SRC2
  ELSE IF (SRC1 > SRC2):
      DEST := SRC1
  ELSE:
      DEST := SRC2



VMAXPH  dest, src1, src2
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
           tsrc2 := SRC2.fp16[0]
       ELSE:
           tsrc2 := SRC2.fp16[j]
       DEST.fp16[j] := MAX(SRC1.fp16[j], tsrc2)
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.56.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMAXPH xmm1, xmm2, xmm3/m128	E2	ID	AVX512-FP16, AVX512VL
VMAXPH ymm1, ymm2, ymm3/m256	E2	ID	AVX512-FP16, AVX512VL
VMAXPH zmm1, zmm2, zmm3/m512	E2	ID	AVX512-FP16

5.57 VMAXSH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.LLIG.F3.MAP5.W0 5F /r VMAXSH xmm1{k1}{z}, xmm2, xmm3/m16 {sae}	A	V/V	AVX512-FP16

5.57.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.57.2 Description

Performs a compare of the low packed FP16 values in the first source operand and the second source operand and returns the maximum value for the pair of values to the destination operand.

If the values being compared are both 0.0s (of either sign), the value in the second operand (source operand) is returned. If a value in the second operand is an SNaN, then SNaN is forwarded unchanged to the destination (that is, a QNaN version of the SNaN is not returned).

If only one value is a NaN (SNaN or QNaN) for this instruction, the second operand (source operand), either a NaN or a valid floating-point value, is written to the result. If instead of this behavior, it is required that the NaN source operand (from either the first or second operand) be returned, the action of VMAXSH can be emulated using a sequence of instructions, such as, a comparison followed by AND, ANDN and OR.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.57.3 Operation

```
def MAX(SRC1, SRC2):
  IF (SRC1 = 0.0) and (SRC2 = 0.0):
      DEST := SRC2
  ELSE IF (SRC1 = NaN):
      DEST := SRC2
  ELSE IF (SRC2 = NaN):
      DEST := SRC2
  ELSE IF (SRC1 > SRC2):
      DEST := SRC1
  ELSE:
      DEST := SRC2


VMAXSH  dest, src1, src2
IF k1[0] OR *no writemask*:
    DEST.fp16[0] := MAX(SRC1.fp16[0], SRC2.fp16[0])
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[j] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.57.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMAXSH xmm1, xmm2, xmm3/m16	E3	ID	AVX512-FP16

5.58 VMINPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 5D /r VMINPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 5D /r VMINPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 5D /r VMINPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {sae}	A	V/V	AVX512-FP16

5.58.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.58.2 Description

Performs a SIMD compare of the packed FP16 values in the first source operand and the second source operand and returns the minimum value for each pair of values to the destination operand.

If the values being compared are both 0.0s (of either sign), the value in the second operand (source operand) is returned. If a value in the second operand is an SNaN, then SNaN is forwarded unchanged to the destination (that is, a QNaN version of the SNaN is not returned).

If only one value is a NaN (SNaN or QNaN) for this instruction, the second operand (source operand), either a NaN or a valid floating-point value, is written to the result. If instead of this behavior, it is required that the NaN source operand (from either the first or second operand) be returned, the action of VMINPH can be emulated using a sequence of instructions, such as, a comparison followed by AND, ANDN and OR.

EVEX encoded versions: The first source operand (the second operand) is a ZMM/YMM/XMM register. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcast from a 16-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1.

5.58.3 Operation

```
def MIN(SRC1, SRC2):
  IF (SRC1 = 0.0) and (SRC2 = 0.0):
      DEST := SRC2
  ELSE IF (SRC1 = NaN):
      DEST := SRC2
  ELSE IF (SRC2 = NaN):
      DEST := SRC2
  ELSE IF (SRC1 < SRC2):
      DEST := SRC1
  ELSE:
      DEST := SRC2

VMINPH  dest, src1, src2
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
           tsrc2 := SRC2.fp16[0]
       ELSE:
           tsrc2 := SRC2.fp16[j]
       DEST.fp16[j] := MIN(SRC1.fp16[j], tsrc2)
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.58.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMINPH xmm1, xmm2, xmm3/m128	E2	ID	AVX512-FP16, AVX512VL
VMINPH ymm1, ymm2, ymm3/m256	E2	ID	AVX512-FP16, AVX512VL
VMINPH zmm1, zmm2, zmm3/m512	E2	ID	AVX512-FP16

5.59 VMINSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 5D /r VMINSH xmm1{k1}{z}, xmm2, xmm3/m16 {sae}	A	V/V	AVX512-FP16

5.59.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.59.2 Description

Performs a compare of the low packed FP16 values in the first source operand and the second source operand and returns the minimum value for the pair of values to the destination operand.

If the values being compared are both 0.0s (of either sign), the value in the second operand (source operand) is returned. If a value in the second operand is an SNaN, then SNaN is forwarded unchanged to the destination (that is, a QNaN version of the SNaN is not returned).

If only one value is a NaN (SNaN or QNaN) for this instruction, the second operand (source operand), either a NaN or a valid floating-point value, is written to the result. If instead of this behavior, it is required that the NaN source operand (from either the first or second operand) be returned, the action of VMINSH can be emulated using a sequence of instructions, such as, a comparison followed by AND, ANDN and OR.

EVEX encoded versions: The first source operand (the second operand) is a ZMM/YMM/XMM register. The second source operand can be a ZMM/YMM/XMM register, a 512/256/128-bit memory location or a 512/256/128-bit vector broadcast from a 16-bit memory location. The destination operand is a ZMM/YMM/XMM register conditionally updated with writemask k1.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.59.3 Operation

```
def MIN(SRC1, SRC2):
  IF (SRC1 = 0.0) and (SRC2 = 0.0):
      DEST := SRC2
  ELSE IF (SRC1 = NaN):
      DEST := SRC2
  ELSE IF (SRC2 = NaN):
      DEST := SRC2
  ELSE IF (SRC1 < SRC2):
      DEST := SRC1
  ELSE:
      DEST := SRC2


VMINSH  dest, src1, src2
IF k1[0] OR *no writemask*:
    DEST.fp16[0] := MIN(SRC1.fp16[0], SRC2.fp16[0])
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[j] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.59.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMINSH xmm1, xmm2, xmm3/m16	E3	ID	AVX512-FP16

5.60 VMOVSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 10 /r VMOVSH xmm1{k1}{z}, m16	A	V/V	AVX512-FP16
EVEX.LLIG.F3.MAP5.W0 11 /r VMOVSH m16{k1}, xmm1	B	V/V	AVX512-FP16
EVEX.LLIG.F3.MAP5.W0 10 /r VMOVSH xmm1{k1}{z}, xmm2, xmm3	C	V/V	AVX512-FP16
EVEX.LLIG.F3.MAP5.W0 11 /r VMOVSH xmm1{k1}{z}, xmm2, xmm3	D	V/V	AVX512-FP16

5.60.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A
B	SCALAR	MODRM.R/M(w)	MODRM.REG(r)	N/A	N/A
C	N/A	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A
D	N/A	MODRM.R/M(w)	VVVV(r)	MODRM.REG(r)	N/A

5.60.2 Description

Move a FP16 value to a register or memory.

The two register-only forms are aliases and differ only in where their operands are encoded. This is a side effect of the encodings selected.

5.60.3 Operation

```
VMOVSH  dest, src    // TWO OPERAND LOAD

IF k1[0] or no writemask:
   DEST.fp16[0] := SRC.fp16[0]
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
// ELSE DEST.fp16[0] remains unchanged

DEST[MAXVL:16] := 0
```

```
VMOVSH  dest, src    // TWO OPERAND STORE

IF k1[0] or no writemask:
   DEST.fp16[0] := SRC.fp16[0]
// ELSE DEST.fp16[0] remains unchanged
```

```
VMOVSH  dest, src1, src2    // THREE OPERAND COPY

IF k1[0] or no writemask:
   DEST.fp16[0] := SRC2.fp16[0]
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
// ELSE DEST.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL:128] := 0
```

5.60.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMOVSH xmm1, m16	E5	N/A	AVX512-FP16
VMOVSH m16, xmm1	E5	N/A	AVX512-FP16
VMOVSH xmm1, xmm2, xmm3	E5	N/A	AVX512-FP16

5.61 VMOVW

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP5.WIG 6E /r VMOVW xmm1, reg/m16	A	V/V	AVX512-FP16
EVEX.128.66.MAP5.WIG 7E /r VMOVW reg/m16, xmm1	B	V/V	AVX512-FP16

5.61.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A
B	SCALAR	MODRM.R/M(w)	MODRM.REG(r)	N/A	N/A

5.61.2 Description

This instruction either (a) copies a one FP16 element from an XMM register to a general purpose register or memory location or (b) copies one FP16 element from a general purpose register or memory location to an XMM register. When writing a general purpose register, the lower 16-bits of the register will contain the FP16 value. The upper bits of the general purpose register are written with zeros.

5.61.3 Operation

```
VMOVW  dest, src    // TWO OPERAND LOAD

DEST.fp16[0] := SRC.fp16[0]
DEST[MAXVL:16] := 0
```

```
VMOVW  dest, src    // TWO OPERAND STORE

DEST.fp16[0] := SRC.fp16[0]
// upper bits of GPR DEST are zero'd
```

5.61.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMOVW xmm1, reg/m16	E9NF	N/A	AVX512-FP16
VMOVW reg/m16, xmm1	E9NF	N/A	AVX512-FP16

5.62 VMULPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 59 /r VMULPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 59 /r VMULPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 59 /r VMULPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.62.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.62.2 Description

Multiply packed FP16 values from source operands and stores the packed FP16 result in the destination operand. The destination elements are updated according to the writemask.

5.62.3 Operation

```
VMULPH (EVEX encoded versions) when src2 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      DEST.fp16[j] := SRC1.fp16[j] * SRC2.fp16[j]
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VMULPH (EVEX encoded versions) when src2 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
          DEST.fp16[j] := SRC1.fp16[j] * SRC2.fp16[0]
      ELSE:
          DEST.fp16[j] := SRC1.fp16[j] * SRC2.fp16[j]
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.62.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VMULPH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VMULPH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VMULPH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.63 VMULSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 59 /r VMULSH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.63.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.63.2 Description

Multiply the low FP16 value from the source operands and stores the FP16 result in the destination operand. Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.63.3 Operation

```
VMULSH (EVEX encoded versions)
IF EVEX.b = 1 and SRC2 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   DEST.fp16[0] := SRC1.fp16[0] * SRC2.fp16[0]
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:VL] := 0
```

5.63.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VMULSH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16

5.64 VRCPPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 4C /r VRCPPH xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 4C /r VRCPPH ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 4C /r VRCPPH zmm1{k1}{z}, zmm2/m512/m16bcst	A	V/V	AVX512-FP16

5.64.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.64.2 Description

This instruction performs a SIMD computation of the approximate reciprocals of 8/16/32 packed FP16 values in the source operand (the second operand) and stores the packed FP16 results in the destination operand. The maximum relative error for this approximation is less than $2^{-11} + 2^{-14}$.

For special cases, see Figure 5.5.

Input Value	Result value	Comments
$0 \leq X \leq 2^{-16}$	∞	Very small denormal
$-2^{-16} \leq X \leq -0$	$-\infty$	Very small denormal
$X = +\infty$	$+0$	
$X = -\infty$	-0	
$X = 2^{-n}$	2^n	
$X = -2^{-n}$	-2^n	

Figure 5.5: vrcp[ph,sh] special cases

5.64.3 Operation

```
VRCPPH dest{k1}, src
VL = 128, 256, or 512

KL := VL / 16

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := APPROXIMATE(1.0 / tsrc)
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.64.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VRCPPH xmm1, xmm2/m128	E4	N/A	AVX512-FP16, AVX512VL
VRCPPH ymm1, ymm2/m256	E4	N/A	AVX512-FP16, AVX512VL
VRCPPH zmm1, zmm2/m512	E4	N/A	AVX512-FP16

5.65 VRCPSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.66.MAP6.W0 4D /r VRCPSH xmm1{k1}{z}, xmm2, xmm3/m16	A	V/V	AVX512-FP16

5.65.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.65.2 Description

This instruction performs a SIMD computation of the approximate reciprocal of the low FP16 value in the second source operand (the third operand) and stores the result in the low word element of the destination operand (the first operand) according to the writemask k1. Bits 127:16 of the XMM register destination are copied from corresponding bits in the first source operand (the second operand). The maximum relative error for this approximation is less than $2^{-11} + 2^{-14}$.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

For special cases, see Figure 5.5.

5.65.3 Operation

```
VRCPSH dest{k1}, src1, src2

IF k1[0] or  *no writemask*:
   DEST.fp16[0] := APPROXIMATE(1.0 / src2.fp16[0])
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
//else DEST.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.65.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VRCPSH xmm1, xmm2, xmm3/m16	E10	N/A	AVX512-FP16

5.66 VREDUCEPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.0F3A.W0 56 /r /ib VREDUCEPH xmm1{k1}{z}, xmm2/m128/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.0F3A.W0 56 /r /ib VREDUCEPH ymm1{k1}{z}, ymm2/m256/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.0F3A.W0 56 /r /ib VREDUCEPH zmm1{k1}{z}, zmm2/m512/m16bcst {sae}, imm8	A	V/V	AVX512-FP16

5.66.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	IMM8(r)	N/A

5.66.2 Description

Perform reduction transformation of the packed binary encoded FP16 values in the source operand (the second operand) and store the reduced results in binary FP format to the destination operand (the first operand) under the writemask k1.

The reduction transformation subtracts the integer part and the leading M fractional bits from the binary FP source value, where M is a unsigned integer specified by imm8[7:4]. Specifically, the reduction transformation can be expressed as:
$dest = src - (ROUND(2^M * src)) * 2^{-M}$
where ROUND() treats src, 2^M and their product as binary FP numbers with normalized significand and biased exponents.

The magnitude of the reduced result can be expressed by considering $src = 2^p * man2$, where $man2$ is the normalized significand and p is the unbiased exponent.
Then if RC=RNE: $0 \leq |ReducedResult| \leq 2^{-M-1}$
Then if RC $\neq$ RNE: $0 \leq |ReducedResult| < 2^{-M}$
This instruction might end up with a precision exception set. However, in case of SPE set (i.e. Suppress Precision Exception, which is imm8[3]=1), no precision exception is reported.

This instruction may generate tiny non-zero result. If it does so, it does not report underflow exception, even if underflow exceptions are unmasked (UM flag in MXCSR register is 0).

For special cases see Figure 5.6.

	Round mode	**Returned value**
$\|Src1\| < 2^{-M-1}$	RNE	$Src1$
$\|Src1\| < 2^{-M}$	RU, $Src1 > 0$	$Round(Src1 - 2^{-M})$
	RU, $Src1 \leq 0$	$Src1$
	RD, $Src1 \geq 0$	$Src1$
	RD, $Src1 < 0$	$Round(Src1 + 2^{-M})$
$Src1 = \pm 0$ or $Dest = \pm 0 (Src1 \neq \infty)$	NOT RD	$+0.0$
	RD	-0.0
$Src1 = \pm\infty$	any	$+0.0$
$Src1 = \pm NAN$	any	$QNaN(Src1)$

Figure 5.6: VREDUCEPH, VREDUCESH special cases. The $Round(.)$ function uses rounding controls specified by (imm8[2]? MXCSR.RC : imm8[1:0]).

5.66.3 Operation

```
def reduce_fp16(src, imm8):
   nan := (src.exp = 0x1F) and (src.fraction != 0)
   if nan:
      return QNAN(src)
   m := imm8[7:4]
   rc := imm8[1:0]
   rc_source := imm8[2]
   spe := imm[3]  // suppress precision exception
   tmp := 2^(-m) * ROUND(2^m * src, spe, rc_source, rc)
   tmp := src - tmp  // using same RC, SPE controls
   return tmp
```

```
VREDUCEPH dest{k1}, src, imm8
VL = 128, 256, or 512
KL := VL / 16

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := reduce_fp16(tsrc, imm8)
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.66.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VREDUCEPH xmm1, xmm2/m128, imm8	E2	IP	AVX512-FP16, AVX512VL
VREDUCEPH ymm1, ymm2/m256, imm8	E2	IP	AVX512-FP16, AVX512VL
VREDUCEPH zmm1, zmm2/m512, imm8	E2	IP	AVX512-FP16

5.67 VREDUCESH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.0F3A.W0 57 /r /ib VREDUCESH xmm1{k1}{z}, xmm2, xmm3/m16 {sae}, imm8	A	V/V	AVX512-FP16

5.67.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	IMM8(r)

5.67.2 Description

Perform reduction transformation of the low binary encoded FP16 value in the source operand (the second operand) and store the reduced result in binary FP format to the low element of the destination operand (the first operand) under the writemask k1. For further details see the description of VREDUCEPH.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

This instruction might end up with a precision exception set. However, in case of SPE set (i.e. Suppress Precision Exception, which is imm8[3]=1), no precision exception is reported.

This instruction may generate tiny non-zero result. If it does so, it does not report underflow exception, even if underflow exceptions are unmasked (UM flag in MXCSR register is 0).

For special cases see Figure 5.6.

5.67.3 Operation

```
VREDUCESH dest{k1}, src, imm8

IF k1[0] or  *no writemask*:
   dest.fp16[0] := reduce_fp16(src2.fp16[0], imm8)   // see VREDUCEPH
ELSE IF *zeroing*:
   dest.fp16[0] := 0
//else dest.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.67.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VREDUCESH xmm1, xmm2, xmm3/m16, imm8	E3	IP	AVX512-FP16

5.68 VRNDSCALEPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.0F3A.W0 08 /r /ib VRNDSCALEPH xmm1{k1}{z}, xmm2/m128/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.0F3A.W0 08 /r /ib VRNDSCALEPH ymm1{k1}{z}, ymm2/m256/m16bcst, imm8	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.0F3A.W0 08 /r /ib VRNDSCALEPH zmm1{k1}{z}, zmm2/m512/m16bcst {sae}, imm8	A	V/V	AVX512-FP16

5.68.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	IMM8(r)	N/A

5.68.2 Description

Round the FP16 values in the source operand by the rounding mode specified in the immediate operand (see Figure 5.7) and places the result in the destination operand. The destination operand is conditionally updated according to the writemask.

The rounding process rounds the input to an integral value, plus number bits of fraction that are specified by imm8[7:4] (to be included in the result) and returns the result as a FP16 value.

It should be noticed that no overflow is induced while executing this instruction (although the source is scaled by the imm8[7:4] value).

The immediate operand also specifies control fields for the rounding operation, three bit fields are defined and shown in the “Immediate Control Description” figure below. Bit 3 of the immediate byte controls the processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Immediate control table below lists the encoded values for rounding-mode field).

The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN.

The sign of the result of this instruction is preserved, including the sign of zero. Special cases are described in Figure 5.8.

The formula of the operation on each data element for VRNDSCALEPH is
ROUND(x) = 2^{-M}*Round_to_INT($x * 2^M$, round_ctrl),

round_ctrl = imm[3:0];
M=imm[7:4];
The operation of $x * 2^M$ is computed as if the exponent range is unlimited (i.e. no overflow ever occurs).

If this instruction encoding's SPE bit (bit 3) in the immediate operand is 1, VRNDSCALEPH can set MXCSR.UE without MXCSR.PE.

EVEX.vvvv is reserved and must be 1111b, otherwise instructions will #UD.

imm8[7:4]	Number of fixed points to preserve
imm8[3]	Suppress Precision Exception (SPE). 0 implies use MXCSR exception mask. 1 implies suppress.
imm8[2]	Round Select (RS). 0 implies use imm8[1:0]. 1 implies use MXCSR
imm8[1:0]	Round Control Override. 0b00=Nearest Even 0b01=Round Down 0b10=Round Up 0b11=Truncate

Figure 5.7: Imm8 controls for VRNDSCALE*

Input	Returned Value
SRC1=$\pm\infty$	Src1
SRC1=$\pm$ NaN	QNaN(Src1)
SRC1=$\pm$ 0	Src1

Figure 5.8: VRNDSCALE* Special cases

5.68.3 Operation

```
def round_fp16_to_integer(src, imm8):
   if imm8[2] = 1:
       rounding_direction := MXCSR.RC
   else:
       rounding_direction := imm8[1:0]
   m := imm8[7:4]   // scaling factor

   tsrc1 := 2^m * src

   if rounding_direction = 0b00:
      tmp := round_to_nearest_even_integer(trc1)
   else if rounding_direction = 0b01:
      tmp := round_to_equal_or_smaller_integer(trc1)
   else if rounding_direction = 0b10:
      tmp := round_to_equal_or_larger_integer(trc1)
   else if rounding_direction = 0b11:
      tmp := round_to_smallest_magnitude_integer(trc1)

   dst := 2^(-m) * tmp

   if imm8[3]==0: // check SPE
       if src != dst:
           MXCSR.PE := 1
   return dst
```

```
VRNDSCALEPH dest{k1}, src, imm8
VL = 128, 256, or 512
KL := VL / 16

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := round_fp16_to_integer(tsrc, imm8)
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.68.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VRNDSCALEPH xmm1, xmm2/m128, imm8	E2	IPU	AVX512-FP16, AVX512VL
VRNDSCALEPH ymm1, ymm2/m256, imm8	E2	IPU	AVX512-FP16, AVX512VL
VRNDSCALEPH zmm1, zmm2/m512, imm8	E2	IPU	AVX512-FP16

5.69 VRNDSCALESH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.0F3A.W0 0A /r /ib VRNDSCALESH xmm1{k1}{z}, xmm2, xmm3/m16 {sae}, imm8	A	V/V	AVX512-FP16

5.69.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	IMM8(r)

5.69.2 Description

Round the low FP16 value in the second source operand by the rounding mode specified in the immediate operand (see Figure 5.7) and places the result in the destination operand.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

The rounding process rounds the input to an integral value, plus number bits of fraction that are specified by imm8[7:4] (to be included in the result) and returns the result as a FP16 value.

It should be noticed that no overflow is induced while executing this instruction (although the source is scaled by the imm8[7:4] value).

The immediate operand also specifies control fields for the rounding operation, three bit fields are defined and shown in the “Immediate Control Description” figure below. Bit 3 of the immediate byte controls the processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Immediate control table in Figure 5.7 lists the encoded values for rounding-mode field).

The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN.

The sign of the result of this instruction is preserved, including the sign of zero. Special cases are described in Figure 5.8.

If this instruction encoding’s SPE bit (bit 3) in the immediate operand is 1, VRNDSCALESH can set MXCSR.UE without MXCSR.PE.

The formula of the operation on each data element for VRNDSCALESH is

ROUND(x) = 2^{-M}*Round_to_INT($x * 2^M$, round_ctrl),
round_ctrl = imm[3:0];
M=imm[7:4];
The operation of $x * 2^M$ is computed as if the exponent range is unlimited (i.e. no overflow ever occurs).

5.69.3 Operation

```
VRNDSCALESH dest{k1}, src1, src2, imm8

IF k1[0] or  *no writemask*:
   DEST.fp16[0] := round_fp16_to_integer(src2.fp16[0], imm8)  // see VRNDSCALEPH
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
//else DEST.fp16[0] remains unchanged

dest[127:16] = src1[127:16]
dest[MAXVL-1:128] := 0
```

5.69.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VRNDSCALESH xmm1, xmm2, xmm3/m16, imm8	E3	IPU	AVX512-FP16

5.70 VRSQRTPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 4E /r VRSQRTPH xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 4E /r VRSQRTPH ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 4E /r VRSQRTPH zmm1{k1}{z}, zmm2/m512/m16bcst	A	V/V	AVX512-FP16

5.70.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.70.2 Description

This instruction performs a SIMD computation of the approximate reciprocals square-root of 8/16/32 packed FP16 floating-point values in the source operand (the second operand) and stores the packed FP16 floating-point results in the destination operand.

The maximum relative error for this approximation is less than 2^{-14}.

For special cases, see Figure 5.9.

The destination elements are updated according to the writemask.

Input Value	Result value	Comments
Any denormal	Normal	Cannot generate overflow
$X = 2^{-2n}$	2^n	
$X < 0$	QNaN_Indefinite	Including $-\infty$
$X = -0$	$-\infty$	
$X = +0$	$+\infty$	
$X = +\infty$	$+0$	

Figure 5.9: vrsqrt[ph,sh] special cases

5.70.3 Operation

```
VRSQRTPH dest{k1}, src
VL = 128, 256, or 512

KL := VL / 16

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := APPROXIMATE(1.0 / SQRT(tsrc) )
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.70.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VRSQRTPH xmm1, xmm2/m128	E4	N/A	AVX512-FP16, AVX512VL
VRSQRTPH ymm1, ymm2/m256	E4	N/A	AVX512-FP16, AVX512VL
VRSQRTPH zmm1, zmm2/m512	E4	N/A	AVX512-FP16

5.71 VRSQRTSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.66.MAP6.W0 4F /r VRSQRTSH xmm1{k1}{z}, xmm2, xmm3/m16	A	V/V	AVX512-FP16

5.71.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.71.2 Description

This instruction performs the computation of the approximate reciprocal square-root of the low FP16 value in the second source operand (the third operand) and stores the result in the low word element of the destination operand (the first operand) according to the writemask k1.

The maximum relative error for this approximation is less than 2^{-14}.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed.

For special cases, see Figure 5.9.

5.71.3 Operation

```
VRCPSH dest{k1}, src1, src2

IF k1[0] or  *no writemask*:
   DEST.fp16[0] := APPROXIMATE(1.0 / SQRT(src2.fp16[0]))
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
//else DEST.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.71.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VRSQRTSH xmm1, xmm2, xmm3/m16	E10	N/A	AVX512-FP16

5.72 VSCALEFPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.66.MAP6.W0 2C /r VSCALEFPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.66.MAP6.W0 2C /r VSCALEFPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.66.MAP6.W0 2C /r VSCALEFPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.72.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.72.2 Description

Performs a floating-point scale of the packed FP16 values in the first source operand by multiplying it by 2 power of the FP16 values in second source operand. The destination elements are updated according to the writemask.

The equation of this operation is given by:
zmm1 := $zmm2 * 2^{floor(zmm3)}$.
Floor(zmm3) means maximum integer value $\leq$ zmm3.

If the result cannot be represented in FP16, then the proper overflow response (for positive scaling operand), or the proper underflow response (for negative scaling operand) is issued. The overflow and underflow responses are dependent on the rounding mode (for IEEE-compliant rounding), as well as on other settings in MXCSR (exception mask bits), and on the SAE bit.

Handling of special-case input values are listed in Figure 5.10 and Figure 5.11.

5.72.3 Operation

```
def scale_fp16(src1,src2):
   tmp1 := src1
   tmp2 := src2
   return tmp1 * POW(2, FLOOR(tmp2))
```

Src1	**Src2**				**Set IE**
	$\pm NaN$	$+\infty$	$-\infty$	**0/Norm/Denorm**	
$\pm QNaN$	QNaN(Src1)	$+\infty$	$+0$	QNaN(src1)	If either source is SNaN
$\pm SNaN$	QNaN(Src1)	QNaN(src1)	QNaN(src1)	QNaN(src1)	YES
$\pm\infty$	QNaN(Src2)	Src1	QNaN_Indefinite	Src1	If src2 is SNaN or $-\infty$
± 0	QNaN(Src2)	QNaN_Indefinite	Src1	Src1	If src2 is SNaN or $+\infty$
Denorm/Norm	QNaN(Src2)	$\pm\infty$ (src1 sign)	± 0(src1 sign)	Compute Result	If src2 is SNaN

Figure 5.10: VSCALEF* special cases

Special Case	**Returned value**	**Faults**
$\|result\| < 2^{-24}$	± 0 or $\pm$ Min-Denormal (src1 sign)	Underflow
$\|result\| \geq 2^{16}$	$\pm\infty$ (src1 sign) or $\pm$ Max-Denormal (src1 sign)	Overflow

Figure 5.11: Additional VSCALEF* special cases

```
VSCALEFPH dest{k1}, src1, src2
VL = 128, 256, or 512
KL := VL / 16

IF (VL = 512) AND (EVEX.b = 1) and no memory operand:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC2 is memory and (EVEX.b = 1):
         tsrc := src2.fp16[0]
       ELSE:
         tsrc := src2.fp16[i]
       dest.fp16[i] := scale_fp16(src1.fp16[i],tsrc)
    ELSE IF *zeroing*:
       dest.fp16[i] := 0
    //else dest.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.72.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VSCALEFPH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VSCALEFPH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VSCALEFPH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

Denormal-operand exception (#D) is checked and signaled for src1 operand, but not for src2 operand. The denormal-operand exception is checked for src1 operand only if the src2 operand is not NaN. If the src2 operand is NaN, the processor generates NaN and does not signal denormal-operand exception, even if src1 operand is denormal.

5.73 VSCALEFSH

Encoding / Instruction	**Op/En**	**64/32-bit mode**	**CPUID**
EVEX.LLIG.66.MAP6.W0 2D /r VSCALEFSH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.73.1 Instruction Operand Encoding

Op/En	**Tuple**	**Operand 1**	**Operand 2**	**Operand 3**	**Operand 4**
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.73.2 Description

Performs a floating-point scale of the low FP16 element in the first source operand by multiplying it by 2 power of the low FP16 element in second source operand, storing the result in the low element of the destination operand.

Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

The equation of this operation is given by:
xmm1 := $xmm2 * 2^{floor(xmm3)}$.
Floor(xmm3) means maximum integer value $\leq$ xmm3.

If the result cannot be represented in FP16, then the proper overflow response (for positive scaling operand), or the proper underflow response (for negative scaling operand) is issued. The overflow and underflow responses are dependent on the rounding mode (for IEEE-compliant rounding), as well as on other settings in MXCSR (exception mask bits, FTZ bit), and on the SAE bit.

Handling of special-case input values are listed in Figure 5.10 and Figure 5.11.

5.73.3 Operation

```
VSCALEFSS dest{k1}, src1, src2

IF (EVEX.b = 1) and no memory operand:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] or  *no writemask*:
   dest.fp16[0] := scale_fp16(src1.fp16[0], src2.fp16[0]) // see VSCALEFPH
ELSE IF *zeroing*:
   dest.fp16[0] := 0
//else DEST.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.73.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VSCALEFSH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16

Denormal-operand exception (#D) is checked and signaled for src1 operand, but not for src2 operand. The denormal-operand exception is checked for src1 operand only if the src2 operand is not NaN. If the src2 operand is NaN, the processor generates NaN and does not signal denormal-operand exception, even if src1 operand is denormal.

5.74 VSQRTPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 51 /r VSQRTPH xmm1{k1}{z}, xmm2/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 51 /r VSQRTPH ymm1{k1}{z}, ymm2/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 51 /r VSQRTPH zmm1{k1}{z}, zmm2/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.74.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	MODRM.R/M(r)	N/A	N/A

5.74.2 Description

Performs a packed FP16 square-root computation on the values from source operand and stores the packed FP16 result in the destination operand. The destination elements are updated according to the writemask.

5.74.3 Operation

```
VSQRTPH dest{k1}, src
VL = 128, 256, or 512

KL := VL / 16

FOR i := 0 to KL-1:
    IF k1[i] or  *no writemask*:
       IF SRC is memory and (EVEX.b = 1):
         tsrc := src.fp16[0]
       ELSE:
         tsrc := src.fp16[i]
       DEST.fp16[i] := SQRT(tsrc)
    ELSE IF *zeroing*:
       DEST.fp16[i] := 0
    //else DEST.fp16[i] remains unchanged

dest[MAXVL-1:vl] := 0
```

5.74.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VSQRTPH xmm1, xmm2/m128	E2	IPD	AVX512-FP16, AVX512VL
VSQRTPH ymm1, ymm2/m256	E2	IPD	AVX512-FP16, AVX512VL
VSQRTPH zmm1, zmm2/m512	E2	IPD	AVX512-FP16

5.75 VSQRTSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 51 /r VSQRTSH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.75.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.75.2 Description

Performs a scalar FP16 square-root computation on the source operand and stores the FP16 result in the destination operand. Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.75.3 Operation

```
VSQRTSH dest{k1}, src1, src2

IF k1[0] or  *no writemask*:
   DEST.fp16[0] := SQRT(src2.fp16[0])
ELSE IF *zeroing*:
   DEST.fp16[0] := 0
//else DEST.fp16[0] remains unchanged

dest[127:16] := src1[127:16]
dest[MAXVL-1:128] := 0
```

5.75.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VSQRTSH xmm1, xmm2, xmm3/m16	E3	IPD	AVX512-FP16

5.76 VSUBPH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.128.NP.MAP5.W0 5C /r VSUBPH xmm1{k1}{z}, xmm2, xmm3/m128/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.256.NP.MAP5.W0 5C /r VSUBPH ymm1{k1}{z}, ymm2, ymm3/m256/m16bcst	A	V/V	AVX512-FP16 AVX512VL
EVEX.512.NP.MAP5.W0 5C /r VSUBPH zmm1{k1}{z}, zmm2, zmm3/m512/m16bcst {er}	A	V/V	AVX512-FP16

5.76.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	FULL	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.76.2 Description

Subtract packed FP16 values from second source operand from the corresponding elements in the first source operand, storing the packed FP16 result in the destination operand. The destination elements are updated according to the writemask.

5.76.3 Operation

```
VSUBPH (EVEX encoded versions) when src2 operand is a register
VL = 128, 256 or 512
KL := VL/16
IF (VL = 512) AND (EVEX.b = 1):
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      DEST.fp16[j] := SRC1.fp16[j] - SRC2.fp16[j]
   ELSE   IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

```
VSUBPH (EVEX encoded versions) when src2 operand is a memory source
VL = 128, 256 or 512
KL := VL/16

FOR j := 0 TO KL-1:
   IF k1[j] OR *no writemask*:
      IF EVEX.b = 1:
          DEST.fp16[j] := SRC1.fp16[j] - SRC2.fp16[0]
      ELSE:
          DEST.fp16[j] := SRC1.fp16[j] - SRC2.fp16[j]
   ELSE IF *zeroing*:
      DEST.fp16[j] := 0
   // else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL] := 0
```

5.76.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VSUBPH xmm1, xmm2, xmm3/m128	E2	IOUPD	AVX512-FP16, AVX512VL
VSUBPH ymm1, ymm2, ymm3/m256	E2	IOUPD	AVX512-FP16, AVX512VL
VSUBPH zmm1, zmm2, zmm3/m512	E2	IOUPD	AVX512-FP16

5.77 VSUBSH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.F3.MAP5.W0 5C /r VSUBSH xmm1{k1}{z}, xmm2, xmm3/m16 {er}	A	V/V	AVX512-FP16

5.77.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(w)	VVVV(r)	MODRM.R/M(r)	N/A

5.77.2 Description

Subtract the low FP16 value from the second source operand from the corresponding value in the first source operand, storing the FP16 result in the destination operand. Bits 127:16 of the destination operand are copied from the corresponding bits of the first source operand. Bits MAXVL-1:128 of the destination operand are zeroed. The low FP16 element of the destination is updated according to the writemask.

5.77.3 Operation

```
VSUBSH (EVEX encoded versions)
IF EVEX.b = 1 and SRC2 is a register:
       SET_RM(EVEX.RC)
ELSE
       SET_RM(MXCSR.RC)

IF k1[0] OR *no writemask*:
   DEST.fp16[0] := SRC1.fp16[0] - SRC2.fp16[0]
ELSE   IF *zeroing*:
   DEST.fp16[0] := 0
// else dest.fp16[0] remains unchanged

DEST[127:16] := SRC1[127:16]
DEST[MAXVL-1:128] := 0
```

5.77.4 Exceptions

Instruction	**Exception Type**	**Arithmetic Flags**	**CPUID**
VSUBSH xmm1, xmm2, xmm3/m16	E3	IOUPD	AVX512-FP16

5.78 VUCOMISH

Encoding / Instruction	Op/En	64/32-bit mode	CPUID
EVEX.LLIG.NP.MAP5.W0 2E /r VUCOMISH xmm1, xmm2/m16 {sae}	A	V/V	AVX512-FP16

5.78.1 Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	SCALAR	MODRM.REG(r)	MODRM.R/M(r)	N/A	N/A

5.78.2 Description

Compares the FP16 values in the low word of operand 1 (first operand) and operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN).

Operand 1 is an XMM register; operand 2 can be an XMM register or a 16-bit memory location.

The VUCOMISH instruction differs from the VCOMISH instruction in that it signals a SIMD floating-point invalid operation exception (#I) only if a source operand is an SNaN. The COMISS instruction signals an invalid numeric exception when a source operand is either a QNaN or SNaN.

The EFLAGS register is not updated if an unmasked SIMD floating-point exception is generated.

EVEX.vvvv are reserved and must be 1111b, otherwise instructions will #UD.

5.78.3 Operation

```
VUCOMISH

RESULT := UnorderedCompare(SRC1.fp16[0],SRC2.fp16[0])
if RESULT is UNORDERED:
    ZF, PF, CF := 1, 1, 1
else if RESULT is GREATER_THAN:
    ZF, PF, CF := 0, 0, 0
else if RESULT is LESS_THAN:
    ZF, PF, CF := 0, 0, 1
else: // RESULT is EQUALS
    ZF, PF, CF := 1, 0, 0

OF, AF, SF := 0, 0, 0
```

5.78.4 Exceptions

Instruction	Exception Type	Arithmetic Flags	CPUID
VUCOMISH xmm1, xmm2/m16	E3NF	ID	AVX512-FP16