AMD64 Technology

AMD64 Architecture Programmer's Manual

Volume 4: 128-Bit and 256-Bit Media Instructions

Publication No.	Revision	Date
26568	3.11	December 2010

Advanced Micro Devices

Contents

Figures

Tables

Revision History

Date	Revision	Description
December 2010	3.11	Complete revision and reformat accommodating 128-bit and 256-bit media instructions. Includes revised definitions of legacy SSE, SSE2, SSE3, SSE4.1, SSE4.2, and SSSE3 instructions, as well as new definitions of extended AES, AVX, CLMUL, FMA4, and XOP instructions. Introduction includes supplemental information concerning encoding of extended instructions, enhanced processor state management provided by the XSAVE/XRSTOR instructions, cryptographic capabilities of the AES instructions, and functionality of extended string comparison instructions.
September 2007	3.10	Added minor clarifications and corrected typographical and formatting errors.
July 2007	3.09	Added the following instructions: EXTRQ on page 105, INSERTQ on page 121, MOVNTSD on page 183, and MOVNTSS on page 185. Added misaligned exception mask (MXCSR.MM) information. Added imm8 values with corresponding mnemonics to CMPPD on page 25, CMPPS on page 29, CMPSD on page 32, and CMPSS on page 35. Reworded CPUID information in condition tables. Added minor clarifications and corrected typographical and formatting errors.
September 2006	3.08	Made minor corrections.
December 2005	3.07	Made minor editorial and formatting changes.
January 2005	3.06	Added documentation on SSE3 instructions. Corrected numerous minor factual errors and typos.
September 2003	3.05	Made numerous small factual corrections.
April 2003	3.04	Made minor corrections.

Preface

About This Book

This book is part of a multivolume work entitled the *AMD64 Architecture Programmer's Manual*. The complete set includes the following volumes.

Title	Order No.
Volume 1: Application Programming	24592
Volume 2: System Programming	24593
Volume 3: General-Purpose and System Instructions	24594
Volume 4: 128-Bit and 256-Bit Media Instructions	26568
Volume 5: 64-Bit Media and x87 Floating-Point Instructions	26569

Audience

This volume is intended for programmers who develop application or system software.

Organization

Volumes 3, 4, and 5 describe the AMD64 instruction set in detail, providing mnemonic syntax, opcodes, functions, affected flags, and possible exceptions.

The AMD64 instruction set is divided into five subsets:

- General-purpose instructions
- System instructions
- 128-bit and 256-bit media instructions
- 64-bit media instructions
- x87 floating-point instructions

Several instructions belong to, and are described identically in, multiple instruction subsets.

This volume describes the 128-bit and 256-bit media instructions, including both legacy and extended forms of the instructions. The index at the end cross-references topics within this volume. For other topics relating to the AMD64 architecture, and for information on instructions in other subsets, see the tables of contents and indexes of the other volumes.

Definitions

Many of the following definitions assume an in-depth knowledge of the legacy x86 architecture. See "Related Documents" for descriptions of the legacy x86 architecture.

Terminology

128-bit media instructions

Instructions that use the 128-bit XMM registers.

256-bit media instructions

Instructions that use the 256-bit YMM registers.

16-bit mode

Legacy mode or compatibility mode in which a 16-bit address size is active. See *legacy mode* and *compatibility mode*.

32-bit mode

Legacy mode or compatibility mode in which a 32-bit address size is active. See *legacy mode* and *compatibility mode*.

64-bit mode

A submode of *long mode*. In 64-bit mode, the default address size is 64 bits and new features, such as register extensions, are supported for system and application software.

absolute

A displacement that references the base of a code segment rather than an instruction pointer. See *relative*.

biased exponent

The sum of a floating-point value's exponent and a constant bias for a particular floating-point data type. The bias makes the range of the biased exponent always positive, which allows reciprocation without overflow.

byte

Eight bits.

clear, cleared

To write the value 0 to a bit or a range of bits. See *set*.

compatibility mode

A submode of *long mode*. In compatibility mode, the default address size is 32 bits, and legacy 16-bit and 32-bit applications run without modification.

commit

To irreversibly write, in program order, an instruction's result to software-visible storage, such as a register (including flags), the data cache, an internal write buffer, or memory.

CPL

Current privilege level.

direct

Referencing a memory address included in the instruction syntax as an immediate operand. The address may be an absolute or relative address. See *indirect*.

dirty data

Data in processor caches or internal buffers that is more recent than the copy held in main memory.

displacement

A signed value that is added to the base of a segment (absolute addressing) or an instruction pointer (relative addressing). Same as *offset*.

doubleword

Two words, or four bytes, or 32 bits.

double quadword

Eight words, or 16 bytes, or 128 bits. Also called *octword*.

effective address size

The address size for the current instruction after accounting for the default address size and any address-size override prefix.

effective operand size

The operand size for the current instruction after accounting for the default operand size and any operand-size override prefix.

element

See *vector*.

exception

An abnormal condition that occurs as the result of instruction execution. Processor response to an exception depends on the type of exception. For exceptions other than SIMD floating-point exceptions and x87 floating-point exceptions, control is transferred to a handler (service routine) by means of an exception vector. For floating-point exceptions defined by the IEEE 754 standard, there are both masked and unmasked responses. When unmasked, the exception handler is called, and when masked, a default response is provided instead of calling the handler.

extended instruction

An AVX, FMA, or XOP media instruction. See *legacy instruction*.

flush

An often ambiguous term meaning (1) writeback, if modified, and invalidate, as in “flush the cache line,” or (2) invalidate, as in “flush the pipeline,” or (3) change a value, as in “flush to zero.”

GDT

Global descriptor table.

GIF

Global interrupt flag.

IDT

Interrupt descriptor table.

indirect

Referencing a memory location whose address is in a register or other memory location. The address may be an absolute or relative address. See *direct*.

IRB

The virtual-8086 mode interrupt-redirection bitmap.

IST

The long-mode interrupt-stack table.

IVT

The real-address mode interrupt-vector table.

LDT

Local descriptor table.

legacy instruction

Any version of SSE media instruction. See *extended instruction*.

legacy x86

The legacy x86 architecture.

legacy mode

An operating mode of the AMD64 architecture in which existing 16-bit and 32-bit applications and operating systems run without modification. A processor implementation of the AMD64 architecture can run in either *long mode* or *legacy mode*. Legacy mode has three submodes, *real mode*, *protected mode*, and *virtual-8086 mode*.

long mode

An operating mode unique to the AMD64 architecture. A processor implementation of the AMD64 architecture can run in either *long mode* or *legacy mode*. Long mode has two submodes, *64-bit mode* and *compatibility mode*.

lsb

Least-significant bit.

LSB

Least-significant byte.

main memory

Physical memory, such as RAM and ROM (but not cache memory) that is installed in a particular computer system.

mask

(1) A control bit that prevents the occurrence of a floating-point exception from invoking an exception-handling routine. (2) A field of bits used for a control purpose.

MBZ

Must be zero. If software attempts to set an MBZ bit to 1, a general-protection exception (#GP) occurs.

memory

Unless otherwise specified, *main memory.*

moffset

A 16, 32, or 64-bit offset that specifies a memory operand directly, without using a ModRM or SIB byte.

msb

Most-significant bit.

MSB

Most-significant byte.

octword

Same as *double quadword.*

offset

Same as *displacement*.

overflow

The condition in which a floating-point number is larger in magnitude than the largest, finite, positive or negative number that can be represented in the data-type format being used.

packed

See *vector.*

PAE

Physical-address extensions.

physical memory

Actual memory, consisting of *main memory* and cache.

probe

A check for an address in processor caches or internal buffers. *External probes* originate outside the processor, and *internal probes* originate within the processor.

protected mode

A submode of *legacy mode*.

quadword

Four words, eight bytes, or 64 bits.

RAZ

Read as zero (0), regardless of what is written.

real-address mode, real mode

A short name for *real-address mode,* a submode of *legacy mode*.

relative

Referencing with a *displacement* (*offset*) from an instruction pointer rather than the base of a code segment. See *absolute*.

reserved

Fields that may be used at some future time. Such fields may be further qualified as *MBZ*, *RAZ*, *SBZ* or *IGN* (see definitions).

To preserve compatibility with future processors, software must observe the following constraints.

Software must not depend on the state of a reserved field, nor upon the ability of such fields to return to a previously-written state.

When a reserved field is not marked with one of the above qualifiers, software must not change the state of the reserved field; it must reload the field with the same values returned by a prior read.

REX

A legacy instruction prefix that specifies 64-bit operand size and provides access to additional registers.

RIP-relative addressing

Addressing relative to the 64-bit relative instruction pointer.

set

To write the value 1 to a bit or a range of bits. See *clear*.

SIMD

Single instruction, multiple data. See *vector*.

SSE

Streaming SIMD extensions instruction set. There are several versions, including SSE, SSE2. SSE3, SSE4.1, SSE4.2, and SSSE3. See *legacy instructions*.

sticky bit

A bit that is set or cleared by hardware and that remains in that state until explicitly changed by software.

TSS

Task-state segment.

underflow

The condition in which a floating-point number is smaller in magnitude than the smallest nonzero, positive or negative number that can be represented in the data-type format being used.

vector

(1) A set of integer or floating-point values, called *elements*, that are packed into a single operand. Most media instructions use vectors as operands. Also called *packed* or *SIMD* operands.

(2) An *interrupt descriptor table* index, used to access exception handlers. See *exception*.

virtual-8086 mode

A submode of *legacy mode*.

VEX prefix

Extended instruction identifier prefix, used by AVX, CLMUL, and FMA4 instructions.

VMCB

Virtual machine control block.

VMM

Virtual machine monitor.

word

Two bytes, or 16 bits.

x86

See *legacy x86*.

XOP prefix

Extended instruction identifier prefix, used by XOP instructions.

Notation

Chapter 1, “Introduction” describes notation relating specifically to instruction encoding.

1011b
  A binary value, in this example, a 4-bit value.

F0EAh
  A hexadecimal value, in this example a 2-byte value.

[1,2)
  A range that includes the left-most value (in this case, 1) but excludes the right-most value (in this case, 2).

[7:4]
  A bit range, from bit 7 to 4, inclusive. The high-order bit is shown first.

#GP(0)
  A general-protection exception (#GP) with error code of 0.

[CR0–CR4]
  A register range, from register CR0 through CR4, inclusive, with the low-order register first.

CR4.OXSAVE
  The OXSAVE bit of the CR4 register.

CR0.PE = 1
  The PE bit of the CR0 register has a value of 1.

DS:rSI
  The content of a memory location whose segment address is in the DS register and whose offset relative to that segment is in the rSI register.

EFER.LME = 0
  The LME bit of the EFER register has a value of 0.

FF /0
  FF is the first byte of an opcode, and a subopcode in the ModR/M byte has a value of 0.

Registers

In the following list of registers, mnemonics refer either to the register itself or to the register content:

[AH–DH]

The high 8-bit AH, BH, CH, and DH registers. See *[AL–DL]*.

[AL–DL]

The low 8-bit AL, BL, CL, and DL registers. See *[AH–DH]*.

[AL-r15B]

The low 8-bit AL, BL, CL, DL, SIL, DIL, BPL, SPL, and [r8B–r15B] registers, available in 64-bit mode.

BP

Base pointer register.

CRn

Control register number *n*.

CS

Code segment register.

[eAX–eSP]

The 16-bit AX, BX, CX, DX, DI, SI, BP, and SP registers or the 32-bit EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP registers. See *[rAX–rSP]*.

EFER

Extended features enable register.

eFLAGS

16-bit or 32-bit flags register. See *rFLAGS*.

EFLAGS

32-bit (extended) flags register.

eIP

16-bit or 32-bit instruction-pointer register. See *rIP*.

EIP

32-bit (extended) instruction-pointer register.

FLAGS

16-bit flags register.

GDTR

Global descriptor table register.

GPRs

General-purpose registers. For the 16-bit data size, these are AX, BX, CX, DX, DI, SI, BP, and SP. For the 32-bit data size, these are EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP. For the 64-bit data size, these include RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, and R8–R15.

IDTR

Interrupt descriptor table register.

IP

16-bit instruction-pointer register.

LDTR

Local descriptor table register.

MSR

Model-specific register.

[r8–r15]

The 8-bit R8B–R15B registers, or the 16-bit R8W–R15W registers, or the 32-bit R8D–R15D registers, or the 64-bit R8–R15 registers.

r[AX–rSP]

The 16-bit AX, BX, CX, DX, DI, SI, BP, and SP registers, or the 32-bit EAX, EBX, ECX, EDX, EDI, ESI, EBP, and ESP registers, or the 64-bit RAX, RBX, RCX, RDX, RDI, RSI, RBP, and RSP registers. Replace the placeholder *r* with nothing for 16-bit size, "E" for 32-bit size, or "R" for 64-bit size.

RAX

64-bit version of the EAX register.

RBP

64-bit version of the EBP register.

RBX

64-bit version of the EBX register.

RCX

64-bit version of the ECX register.

RDI

64-bit version of the EDI register.

RDX

64-bit version of the EDX register.

rFLAGS

16-bit, 32-bit, or 64-bit flags register. See *RFLAGS*.

RFLAGS

64-bit flags register. See *rFLAGS*.

rIP

16-bit, 32-bit, or 64-bit instruction-pointer register. See *RIP*.

RIP

64-bit instruction-pointer register.

RSI

64-bit version of the ESI register.

RSP

64-bit version of the ESP register.

SP

Stack pointer register.

SS

Stack segment register.

TPR

Task priority register (CR8).

TR

Task register.

Endian Order

The x86 and AMD64 architectures address memory using little-endian byte-ordering. Multibyte values are stored with the least-significant byte at the lowest byte address, and illustrated with their least significant byte at the right side. Strings are illustrated in reverse order, because the addresses of string bytes increase from right to left.

Related Documents

- Peter Abel, *IBM PC Assembly Language and Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- Rakesh Agarwal, *80x86 Architecture & Programming: Volume II*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- AMD, *AMD-K6™ MMX™ Enhanced Processor Multimedia Technology*, Sunnyvale, CA, 2000.
- AMD, *3DNow!™ Technology Manual*, Sunnyvale, CA, 2000.
- AMD, *AMD Extensions to the 3DNow!™ and MMX™ Instruction Sets*, Sunnyvale, CA, 2000.
- Don Anderson and Tom Shanley, *Pentium Processor System Architecture*, Addison-Wesley, New York, 1995.
- Nabajyoti Barkakati and Randall Hyde, *Microsoft Macro Assembler Bible*, Sams, Carmel, Indiana, 1992.
- Barry B. Brey, *8086/8088, 80286, 80386, and 80486 Assembly Language Programming*, Macmillan Publishing Co., New York, 1994.
- Barry B. Brey, *Programming the 80286, 80386, 80486, and Pentium Based Personal Computer*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
- Ralf Brown and Jim Kyle, *PC Interrupts,* Addison-Wesley, New York, 1994.
- Penn Brumm and Don Brumm, *80386/80486 Assembly Language Programming*, Windcrest McGraw-Hill, 1993.
- Geoff Chappell, *DOS Internals,* Addison-Wesley, New York, 1994.
- Chips and Technologies, Inc. *Super386 DX Programmer's Reference Manual*, Chips and Technologies, Inc., San Jose, 1992.
- John Crawford and Patrick Gelsinger, *Programming the 80386*, Sybex, San Francisco, 1987.
- Cyrix Corporation, *5x86 Processor BIOS Writer's Guide*, Cyrix Corporation, Richardson, TX, 1995.
- Cyrix Corporation, *M1 Processor Data Book*, Cyrix Corporation, Richardson, TX, 1996.
- Cyrix Corporation, *MX Processor MMX Extension Opcode Table*, Cyrix Corporation, Richardson, TX, 1996.
- Cyrix Corporation, *MX Processor Data Book*, Cyrix Corporation, Richardson, TX, 1997.
- Ray Duncan, *Extending DOS: A Programmer's Guide to Protected-Mode DOS*, Addison Wesley, NY, 1991.
- William B. Giles, *Assembly Language Programming for the Intel 80xxx Family*, Macmillan, New York, 1991.
- Frank van Gilluwe, *The Undocumented PC,* Addison-Wesley, New York, 1994.
- John L. Hennessy and David A. Patterson, *Computer Architecture*, Morgan Kaufmann Publishers, San Mateo, CA, 1996.
- Thom Hogan, *The Programmer's PC Sourcebook*, Microsoft Press, Redmond, WA, 1991.

- Hal Katircioglu, *Inside the 486, Pentium, and Pentium Pro*, Peer-to-Peer Communications, Menlo Park, CA, 1997.
- IBM Corporation, *486SLC Microprocessor Data Sheet*, IBM Corporation, Essex Junction, VT, 1993.
- IBM Corporation, *486SLC2 Microprocessor Data Sheet*, IBM Corporation, Essex Junction, VT, 1993.
- IBM Corporation, *80486DX2 Processor Floating Point Instructions*, IBM Corporation, Essex Junction, VT, 1995.
- IBM Corporation, *80486DX2 Processor BIOS Writer's Guide*, IBM Corporation, Essex Junction, VT, 1995.
- IBM Corporation, *Blue Lightning 486DX2 Data Book*, IBM Corporation, Essex Junction, VT, 1994.
- Institute of Electrical and Electronics Engineers, *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985.
- Institute of Electrical and Electronics Engineers, *IEEE Standard for Radix-Independent Floating-Point Arithmetic*, ANSI/IEEE Std 854-1987.
- Muhammad Ali Mazidi and Janice Gillispie Mazidi, *80X86 IBM PC and Compatible Computers*, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- Hans-Peter Messmer, *The Indispensable Pentium Book,* Addison-Wesley, New York, 1995.
- Karen Miller, *An Assembly Language Introduction to Computer Architecture: Using the Intel Pentium*, Oxford University Press, New York, 1999.
- Stephen Morse, Eric Isaacson, and Douglas Albert, *The 80386/387 Architecture*, John Wiley & Sons, New York, 1987.
- NexGen Inc.*, Nx586 Processor Data Book*, NexGen Inc., Milpitas, CA, 1993.
- NexGen Inc.*, Nx686 Processor Data Book*, NexGen Inc., Milpitas, CA, 1994.
- Bipin Patwardhan, *Introduction to the Streaming SIMD Extensions in the Pentium III*, www.x86.org/articles/sse_pt1/ simd1.htm, June, 2000.
- Peter Norton, Peter Aitken, and Richard Wilton, *PC Programmer's Bible,* Microsoft Press, Redmond, WA, 1993.
- *PharLap 386|ASM Reference Manual*, Pharlap, Cambridge MA, 1993.
- *PharLap TNT DOS-Extender Reference Manual*, Pharlap, Cambridge MA, 1995.
- Sen-Cuo Ro and Sheau-Chuen Her, *i386/i486 Advanced Programming*, Van Nostrand Reinhold, New York, 1993.
- Jeffrey P. Royer, *Introduction to Protected Mode Programming*, course materials for an onsite class, 1992.
- Tom Shanley, *Protected Mode System Architecture*, Addison Wesley, NY, 1996.
- SGS-Thomson Corporation, *80486DX Processor SMM Programming Manual*, SGS-Thomson Corporation, 1995.

- Walter A. Triebel, *The 80386DX Microprocessor*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- John Wharton, *The Complete x86*, MicroDesign Resources, Sebastopol, California, 1994.
- Web sites and newsgroups:
    - www.amd.com
    - news.comp.arch
    - news.comp.lang.asm.x86
    - news.intel.microprocessors
    - news.microsoft

1 Introduction

This chapter provides an overview of the legacy and extended 128-bit and 256-bit media instructions, with supplemental information about new capabilities. Chapter 2, “Instruction Reference” contains detailed descriptions of each instruction, organized in alphabetic order by mnemonic.

Processors capable of performing the same operation simultaneously on multiple data streams are classified as *single-instruction, multiple-data (SIMD)*. Media instructions utilize the SIMD capabilities of the AMD64 architecture. Most of the instructions perform simultaneous operations on sets of packed elements called *vectors*, although a subset operates on scalar values. There are instructions for both integer and floating-point operations.

Legacy instructions include members of the various sets of *Streaming SIMD (SSE)* instructions; extended instructions include the AVX, CLMUL, FMA4, and XOP instruction sets. When there are both legacy and extended forms of an instruction, the two forms are described together.

The instructions can be used in legacy mode or long (64-bit) mode. CPUID function 8000_0001h[LM] indicates the availability of long mode.

Compilation for execution in 64-bit mode offers the following advantages:

- Access to sixteen 128-bit XMM registers
- Access to sixteen 256-bit YMM registers
- Access to sixteen 64-bit general-purpose registers
- Access to the 64-bit virtual address space and the RIP-relative addressing mode

Hardware support for the various sets of media instructions is indicated by CPUID functions.
The CPUID functions that pertain to each instruction are shown in the instruction description.

1.1 Syntax and Notation

The descriptive synopsis of opcode syntax for legacy instructions follows the conventions described in *Volume 3: General Purpose and System Instructions*.

For further information, see:

- “128-Bit Media and Scientific Programming” in Volume 1.
- “Summary of Registers and Data Types” in Volume 3.
- “Notation” in Volume 3.
- “Instruction Prefixes” in Volume 3

The syntax of the extended instruction sets requires an expanded synopsis. The expanded synopsis includes a mnemonic summary and a summary of prefix fields. Figure 1-1 shows the descriptive synopsis of a typical XOP instruction. The synopses of other extended instructions have the same format, differing only in regard to the instruction set prefix.

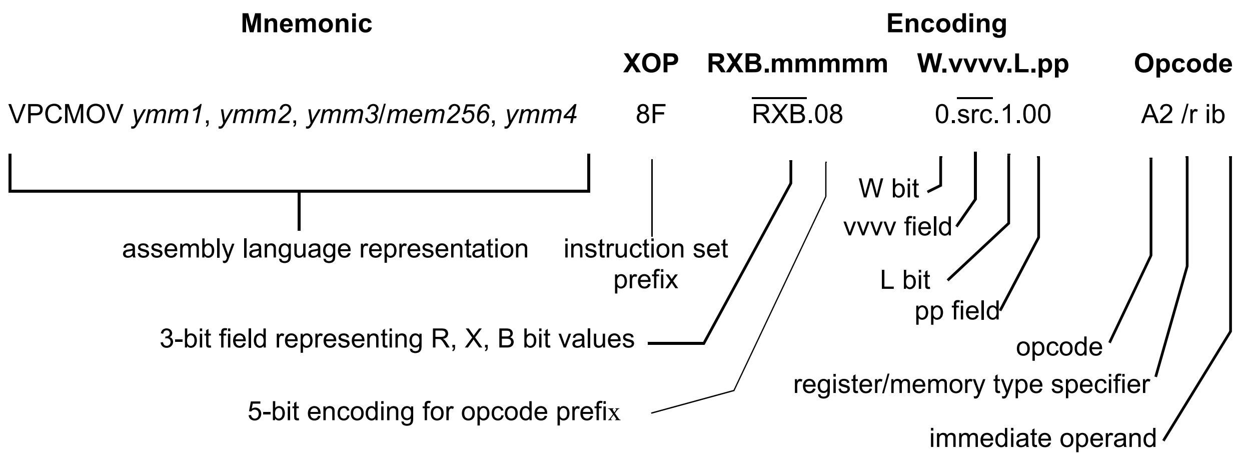

Mnemonic
Encoding
XOP
RXB.mmmmm
W.vvvv.L.pp
Opcode
VPCMOV ymm1, ymm2, ymm3/mem256, ymm4
8F
RXB.08
0.src.1.00
A2 /r ib
assembly language representation
instruction set prefix
3-bit field representing R, X, B bit values
5-bit encoding for opcode prefix
W bit
vvvv field
L bit
pp field
opcode
register/memory type specifier
immediate operand

Figure 1-1. Typical Descriptive Synopsis

1.2 Extended Instruction Format

Figure 1-2 shows the instruction element order of extended instructions. Each element is described in the following sections. The descriptions are overviews; reference is made to pertinent portions of the *AMD64 Architecture Programmer's Manual.*

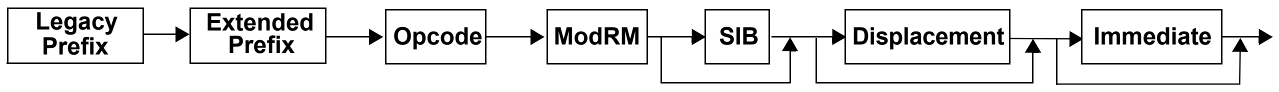


Figure 1-2. Instruction Byte Order

1.2.1 Legacy Prefixes

Optional legacy prefixes include operand-size override, address-size override, segment override, Lock and REP prefixes. For additional information, see section 1.2, "Instruction Prefixes" in the *AMD64 Architecture Programmer's Manual Volume 3: General Purpose and System Instructions*, order# 24594.

1.2.2 Three-Byte Extended Prefix

All extended instructions can be encoded using a three-byte prefix. XOP instructions use only the three-byte prefix, but VEX-encoded instructions that comply with the constraints described in Section 1.2.3, "Two-Byte Extended Prefix" can also utilize a two-byte prefix. Figure 1-3 shows the format of the three-byte prefix.

Byte 0	Byte 1				Byte 2			
7 0	7		5	4 0	7	6 3	2	1 0
Instruction set prefix	R	X	B	mmmmm	W	vvvv	L	pp

Prefix Byte	Bit	Mnemonic	Description
0	[7:0]	VEX, XOP	Value specific to the extended instruction set
1	[7]	R	Inverted one-bit extension of ModRM.reg field
	[6]	X	Inverted one-bit extension of SIB index field
	[5]	B	Inverted one-bit extension, ModRM r/m field or SIB base field
	[4:0]	mmmmm	Opcode map select
2	[7]	W	Default operand size override for a general purpose register to 64-bit size in 64-bit mode; operand configuration specifier for certain XMM/YMM-based operations.
	[6:3]	vvvv	Source or destination register selector, in ones' complement format.
	[2]	L	Vector length specifier
	[1:0]	pp	Implied 66, F2, or F3 opcode extension.

Figure 1-3. Three-Byte Extended Prefix

1.2.2.1 Prefix Byte 0

The value in this byte indicates the extended instruction type. AVX, CLMUL, and FMA4 instructions use the VEX prefix; XOP instructions use the XOP prefix.

- Byte 0 of the VEX prefix must be C4h for three-byte prefixes or C5h for two-byte prefixes.
- Byte 0 of the XOP prefix must be 8Fh, and all XOP instructions use a three-byte prefix.

1.2.2.2 Prefix Byte 1

Bit [7] — R

The bit-inverted equivalent of the REX.R bit. A one-bit extension of the ModRM.reg field in 64-bit mode, permitting access to 16 YMM/XMM and GPR registers. In 32-bit protected and compatibility modes, the value must be 1.

Bit [6]] — X

The bit-inverted equivalent of the REX.X bit. A one-bit extension of the SIB.index field in 64-bit mode, permitting access to 16 YMM/XMM and GPR registers. In 32-bit protected and compatibility modes, this value must be 1.

Bit [5] — B

The bit-inverted equivalent of the REX.B bit, available only in the 3-byte prefix format. A one-bit extension of either the ModRM.r/m field, to specify a GPR or XMM register, or of the SIB base field, to specify a GPR. This permits access to 16 GPR and 16 YMM/XMM registers. In 32-bit protected and compatibility modes, this bit is ignored.

Bits [4:0] — mmmmm

A five-bit field encoding an implied one- or two-byte opcode prefix, as shown in Table 1-3.

Table 1-1. VEX/XOP.mmmmm Encoding

Binary Value	Implied Prefix
00000	Reserved
00001	Implied 0Fh prefix
00010	Implied 0F38h prefix
00011	Implied 0F3Ah prefix
00100 - 11111	Reserved

The XOP.mmmmm field must have a value greater than or equal to 8; if the value is less than 8 these two bytes are interpreted as a form of the POP instruction rather than as an XOP prefix.

1.2.2.3 Prefix Byte 2

Bit [7] — W

Function is instruction-specific. The bit is often used to configure source operand order.

Bits [6:3] — vvvv

Encodes an XMM or YMM register in inverted ones' complement form, as shown in Table 1-2

Table 1-2. VEX/XOP.vvvv Encoding

Binary Value	Register	Binary Value	Register
0000	XMM15/YMM15	1000	XMM07/YMM07
0001	XMM14/YMM14	1001	XMM06/YMM06
0010	XMM13/YMM13	1010	XMM05/YMM05
0011	XMM12/YMM12	1011	XMM04/YMM04
0100	XMM11/YMM11	1100	XMM03/YMM03
0101	XMM10/YMM10	1101	XMM02/YMM02
0110	XMM09/YMM09	1110	XMM01/YMM01
0111	XMM08/YMM08	1111	XMM00/YMM00

Values 0000h to 0111h are not valid in 32-bit modes. The selected registers are typically first sources, but for the VPSLLDQ, VPSRLDQ, VPSRLW, VPSRLD, VPSRLQ, VPSRAW, VPSRAD, VPSLLW, VPSLLD, and VPSLLQ shift instructions, a destination is selected.

Bit [2] — L

L = 0 specifies 128-bit vector length (XMM registers/128-bit memory locations) or use of scalar operands. L=1 specifies 256-bit vector length (YMM registers/256-bit memory locations).

Bits [1:0] — pp

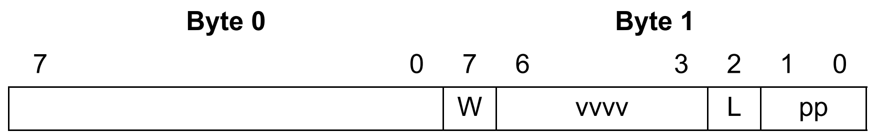
Specifies an implied 66h, F2h, or F3h opcode extension, as shown in Table 1-3. These prefixes are not allowed with extended instructions.

Table 1-3. VEX/XOP.pp Encoding

Binary Value	Implied Prefix
00	None
01	66h
10	F3h
11	F2h

1.2.3 Two-Byte Extended Prefix

All extended instructions can be encoded using the three-byte prefix, but certain VEX-encoded instructions can also utilize a compact, two-byte prefix. XOP instructions do not use the two-byte prefix. The format of the two-byte prefix is shown in Figure 1-3.



Prefix Byte	Bit	Mnemonic	Description
0	[7:0]	VEX	Value specific to the extended instruction set
1	[7]	R	Default operand size override for a general purpose register to 64-bit size in 64-bit mode; operand configuration specifier for certain XMM/YMM-based operations.
	[6:3]	vvvv	Source or destination register selector, in ones' complement format.
	[2]	L	Vector length specifier
	[1:0]	pp	Implied 66, F2, or F3 opcode extension.

Figure 1-4. Two-byte Extended Prefix

When the two-byte prefix is used, specific fields of the three-byte prefix are automatically replaced by predetermined values, as shown in Table 1-4.

Table 1-4. Fixed Two-byte Prefix Field Values

VEX Field	Value
X	1
B	1
W	0
mmmmm	00001

Because the replacement values are used, all two-byte forms can also be encoded as three-byte forms. Other field definitions within the two bytes are the same as for the three-byte prefix.

An instruction that satisfies the constraints can be expressed as an instruction with a two-byte prefix. In the two-byte form, the fixed value of the mmmmm field is 00001b, which decodes to an implied 0Fh leading the opcode byte; all extended instructions of that form can be expressed with a two-byte prefix, providing the other constraints are met. Instructions that use the other legal forms of mmmm, 0010h (0F 38h leading the opcode byte) and 00011 (0F 3Ah leading the opcode byte), cannot be expressed with a two-byte prefix. Note that these implied opcode prefixes are distinct from the implied opcode extensions defined by the pp field; any pp field value can be used.

1.2.4 Opcode Byte

Figure 1-5 shows the format of the opcode byte. For most instructions, operand element size (OES) is specified by the two least-significant opcode bits, as shown in Table 1-5.

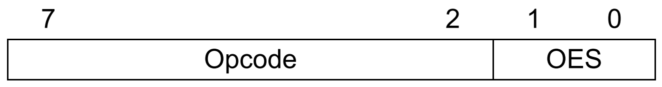


Figure 1-5. Opcode Byte Format

Table 1-5. Operand Element Size (OES)

Binary Value	Integer, Operation	Floating-Point, Operation
00	Byte	PS
01	Word	PD
10	Doubleword	SS
11	Quadword	SD

1.2.5 ModRM, SIB, and Displacement

The ModRM byte is used in certain instruction encodings to define a register or memory reference or to provide additional opcode bits with which to define the instruction's function. Figure 1-6 shows the format of the byte.

7 6	5 3	2 0
mod	reg	r/m

Figure 1-6. ModRM Byte Format

The following summarizes the field functions for extended instructions.

- ModRM.r/m generally specifies a memory operand, as determined by ModRM.mod, but for some instructions that do not address memory, it specifies a source or destination register operand.
- ModRM.reg generally encodes a source or destination register operand, but is sometimes treated as an opcode extension.
- ModRM.mod, the SIB byte, and the displacement specify the type of memory access and addressing mode.

In some instructions, the ModRM byte is followed by a scale-index-base (SIB) byte, which defines memory addressing for the complex-addressing modes described in "Effective Addresses" in Volume 1. The SIB byte has three fields (*scale, index*, and *base*) that define the scale factor, index-register number, and base-register number for complex addressing modes.

A *displacement*, or *offset,* is a signed value that is added to the base of a code segment for absolute addressing or to an instruction pointer for relative addressing. Displacement values can be one to four bytes in length. When a displacement is required, the displacement bytes follow the opcode, ModRM, or SIB byte in the instruction encoding.

1.2.6 Immediate Bytes

An *immediate* is a value, typically an operand, encoded directly into an instruction. Depending on the opcode and the operating mode, the size of an immediate operand can be 1, 2, 4, or 8 bytes. Legacy and extended media instructions typically use an immediate byte operand (*imm8)*.

The immediate byte is generally shown in the instruction synopsis as an "ib" suffix. For four-byte FMA4 instructions, the suffix "is4" is used to indicate the presence of the immediate byte used to select the fourth source operand. See Section 1.2.7.4, "Four-Operand Instructions" and "Immediate Operand Size" in Volume 1 for more information.

1.2.7 Instruction Format Examples

The following sections provide examples of two-, three-, and four-operand extended instructions. These instructions generally perform *nondestructive-source operations*, meaning a single register is not used as both a source and a destination, so source content is preserved. Most legacy instructions perform *destructive-source operations*, in which a single register is both source and destination, so source content is lost.

1.2.7.1 XMM Register Destinations

The following general properties apply to XMM/YMM register destination operands.

- For legacy instructions that use XMM registers as a destination: When a result is written to a destination XMM register, bits [255:128] of the corresponding YMM register are not affected.
- For extended instructions that use XMM registers as a destination: When a result is written to a destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

1.2.7.2 Two Operand Instructions

Two-operand instructions use ModRM-based operand assignment. For most instructions, the first operand is the destination, selected by the ModRM.reg field, and the second operand is either a register or a memory source, selected by the ModRM.r/m field.

VCVTDQ2PD is an example of a two-operand AVX instruction.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VCVTDQ2PD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.01	0.1111.0.10	E6 /r
VCVTDQ2PD *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.01	0.1111.1.10	E6 /r

The destination register is selected by ModRM.reg. The size of the destination register is determined by VEX.L. The source is either an XMM register or a memory location specified by ModRM.r/m Because this instruction converts packed doubleword integers to double-precision floating-point values, the source data size is smaller than the destination data size.

VEX.vvvv is not used and must be set to 1111b.

1.2.7.3 Three-Operand Instructions

These extended instructions have two source operands and a destination operand.

VPROTB is an example of a three-operand XOP instruction.

There are versions of the instruction for variable-count rotation and for fixed-count rotation.

VPROTB *dest, src, variable-count*

VPROTB *dest, src, fixed-count*

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPROTB *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\textit{src}}$.0.00	90 /r
VPROTB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\textit{src}}$.0.00	90 /r
VPROTB *xmm1*, *xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	90 /r ib

For both versions of the instruction, the destination (*dest*) operand is an XMM register specified by MODRM.reg.

The variable-count version of the instruction rotates each byte of the source as specified by the corresponding byte element *variable-count*.

Selection of *src* and *variable-count* is controlled by XOP.W.

- When XOP.W = 0, *src* is either an XMM register or a 128-bit memory location specified by MODRM.rm, and *variable-count* is an XMM register specified by XOP.vvvv.
- When XOP.W = 1, *src* is an XMM register specified by XOP.vvvv and *variable-count* is either an XMM register or a 128-bit memory location specified by MODRM.rm.

Table 1-6 summarizes the effect of the XOP.W bit on operand selection.

Table 1-6. Three-Operand Selection

XOP.W	*dest*	*src*	*variable-count*
0	ModRM.reg	ModRM.r/m	XOP.vvvv
1	ModRM.reg	XOP.vvvv	ModRM.r/m

The fixed-count version of the instruction rotates each byte of *src* as specified by the immediate byte operand *fixed-count*. For this version, *src* is either an XMM register or a 128-bit memory location specified by MODRM.r/m. Because XOP.vvvv is not used to specify the source register, it must be set to 1111b or execution of the instruction will cause an Invalid Opcode (#UD) exception.

1.2.7.4 Four-Operand Instructions

Some extended instructions have three source operands and a destination operand. This is accomplished by using the VEX/XOP.vvvv field, the ModRM.reg and ModRM.r/m fields, and bits [7:4] of an immediate byte to select the operands. The opcode suffix "is4" is used to identify the immediate byte, and the selected operands are shown in the synopsis.

VFMSUBPD is an example of an four-operand FMA4 instruction.

VFMSUBPD *dest, src1, src2, src3* *dest = src1* src2 - src3*

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src}$.0.01	6D /r is4
VFMSUBPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src}$.1.01	6D /r is4
VFMSUBPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src}$.0.01	6D /r is4
VFMSUBPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src}$.1.01	6D /r is4

The first operand, the destination (*dest*), is an XMM register or a YMM register (as determined by VEX.L) selected by MODRM.reg. The following three operands (*src1*, *src2*, *src3*) are sources.

The *src1* operand is an XMM or YMM register specified by VEX.vvvv.

VEX.W determines the configuration of the *src2* and *src3* operands.

- When VEX.W = 0, *src2* is either a register or a memory location specified by ModRM.r/m, and *src3* is a register specified by bits [7:4] of the immediate byte.
- When VEX.W = 1, *src2* is a register specified by bits [7:4] of the immediate byte and *src3* is either a register or a memory location specified by ModRM.r/m.

Table 1-6 summarizes the effect of the VEX.W bit on operand selection.

Table 1-7. Four-Operand Selection

VEX.W	*dest*	*src1*	*src2*	*src3*
0	ModRM.reg	VEX.vvvv	ModRM.r/m	is4[7:4]
1	ModRM.reg	VEX.vvvv	is4[7:4]	ModRM.r/m

1.3 XSAVE/XRSTOR Instructions

The XSAVE, XRSTOR, XGETBV, and XSETBV instructions and associated data structures extend the FXSAVE/FXRSTOR memory image used to manage processor states and provide additional functionality. These instructions do not obviate the FXSAVE/FXRSTOR instructions. For more information about FXSAVE/FXRSTOR, refer to the *AMD64 Architecture Programmer's Manual Volume 3: General Purpose and System Instructions*.

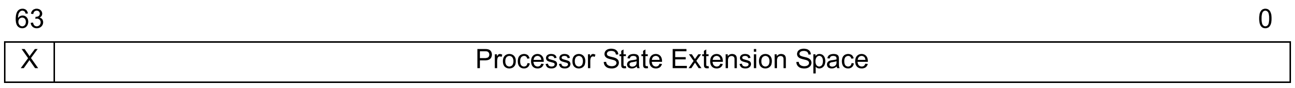
The CPUID instruction is used to identify features supported in processor hardware. Extended control registers are used to enable and disable the handling of processor states associated with supported hardware features and to communicate to an application whether an operating system supports a particular feature that has a processor state specific to it.

1.3.1 CPUID Enhancements

- CPUID Fn0000_00001_ECX[XSAVE] indicates that the processor supports XSAVE/XRSTOR instructions and at least one XCR.
- CPUID Fn0000_00001_ECX[OSXSAVE] indicates whether the operating system has enabled extensible state management and supports processor extended state management.
- CPUID leaf function 0DH enumerates the list of processor states (including legacy x87 FPU states, SSE states, and processor extended states), the offset, and the size of the save area for each processor extended state.

1.3.2 Extended Control Registers

Currently, the only defined extended control register (XCR) is XFEATURE_ENABLED_MASK (XCR0), shown in Figure 1-7. XCR0 specifies the processor states enabled on a particular device, including x87 floating point states, SIMD states, and extended states developed for the AMD64 architecture.



Bits	Mnemonic	Description
[63]	Reserved for XCR0 bit vector expansion	
[62:0]	Processor State Extension Space	

Figure 1-7. XFEATURE_ENABLED_MASK Register (XCR0)

Table 1-8 shows the processor state components currently supported by the AMD64 architecture.

Table 1-8. XCR0 Processor State Components

Bit	Meaning
0	When set, indicates XSAVE/XRSTOR support for x87 state management. This bit must be set
1	When set, indicates XSAVE/XRSTOR support for SSE state management. This bit must be set to enable AVX extensions.
2	When set, indicates XSAVE/XRSTOR support for YMM state management. This bit must be set to enable AVX extensions.
62	When set, indicates support for Lightweight Profiling (LWP) extensions are enabled and XSAVE/XRSTOR support LWP state management.

1.3.3 Extended Save Area

The XSAVE/XRSTOR save area extends the legacy 512-byte FXSAVE/FXRSTOR memory image to provide a compatible register state management environment as well as an upward migration path. The save area is architecturally defined to be extendable and enumerated by the sub-leaves of the CPUID.0DH leaf. Figure 1-9 shows the format of the XSAVE/XRSTOR area.

Table 1-9. Extended Save Area Format

Save Area	Offset (Byte)	Size (Bytes)
FPU/SSE Save Area	0	512
Header	512	64
Reserved, (Ext_Save_Area_2)	CPUID.(EAX=0DH, ECX=2):EBX	CPUID.(EAX=0DH, ECX=2):EAX
Reserved, (Ext_Save_Area_3)	CPUID.(EAX=0DH, ECX=3):EBX	CPUID.(EAX=0DH, ECX=3):EAX
Reserved, (Ext_Save_Area_4)	CPUID.(EAX=0DH, ECX=4):EBX	CPUID.(EAX=0DH, ECX=4):EAX
Reserved, (…)	…	…
Note: *Bytes [464:511] are available for software use. XRSTOR ignores bits [464:511] of an XSAVE image.*		

The register fields of the first 512 bytes of the XSAVE/XRSTOR area are the same as those of the FXSAVE/FXRSTOR area, but the 512-byte area is organized as x87 FPU states, MXCSR (including MXCSR_MASK), and XMM registers. The layout of the save area is fixed and may contain non-contiguous individual save areas because a processor does not support certain extended states or because system software does not support certain processor extended states. The save area is not compacted when features are not saved or are not supported by the processor or by system software.

1.3.4 Instruction Functions

CR4.OSXSAVE and XCR0 can be read at all privilege levels but written only at ring 0.

- XGETBV reads XCR0.
- XSETBV writes XCR0, ring 0 only.
- XRSTOR restores states specified by bitwise AND of a mask operand in EDX:EAX with XCR0.
- XSAVE saves states specified by bitwise AND of a mask operand in EDX:EAX with XCR0.

1.3.5 YMM States and Supported Operating Modes

Extended instructions operate on YMM states by means of extended (XOP/VEX) prefix encoding. When a processor supports YMM states, the states exist in all operating modes, but interfaces to access the YMM states may vary by mode. Processor support for extended prefix encoding is independent of processor support of YMM states.

Instructions that use extended prefix encoding are generally supported in long and protected modes, but are not supported in real or virtual 8086 modes, or when entering SMM mode. Bits [255:128] of the YMM register state are maintained across transitions into and out of these modes. The XSAVE/XRSTOR instructions function in all operating modes; XRSTOR can modify YMM register state in any operating mode, using state information from the XSAVE/XRSTOR area.

1.3.6 YMM State Management

Operating systems must use the XSAVE/XRSTOR instructions for YMM state management. The instructions also provide an interface to manage XMM/MXCSR states and x87 FPU states in conjunction with processor extended states.An operating system must enable YMM state management to support extended instructions. Attempting to execute an extended instruction without enabling YMM state management causes a #UD exception.

1.3.6.1 Enabling YMM State

To enable YMM state support, the operating system must perform the following steps.

- Verify support for XSAVE/XRSTOR instructions and XCR0
  by checking CPUID Fn0000_00001_ECX[XSAVE].
- Verify CPUID.(EAX = 0DH, ECX = 0):EAX.SSE[bit 1] = 1,
  because the lower 128-bits of an YMM register are aliased to an XMM register.
- Determine buffer size requirement for the XSAVE area.
- Set CR4.OSXSAVE to enable the use of XSETBV/XGETBV to write/read XCR0.
- Provide a mask in EDX:EAX that allows XSETBV to enable processor state components managed by XSAVE/XRSTOR instructions.
  - To enable x87 FPU, SSE, and YMM state management, the mask is EDX = 0H, EAX = 7H.
  - EDX:EAX[2:1] = 11b must be used to enable YMM state.
    Attempting to execute XSETBV with EDX:EAX[2:1] = 10b causes a #GP(0) exception.

1.3.7 Saving Processor State

The XSTATE header starts at byte offset 512 in the save area. XSTATE_BV is the first 64-bit field in the header. The order of bit vectors in XSTATE_BV matches the order of bit vectors in XCR0. The XSAVE instruction sets bits in the XSTATE_BV vector field when it writes the corresponding processor extended state to a save area in memory. XSAVE modifies only bits for processor states specified by bitwise AND of the XSAVE bit mask operand in EDX:EAX with XCR0. If software modifies the save area image of a particular processor state component directly, it must also set the corresponding bit of XSTATE_BV. If the bit is not set, directly modified state information in a save area image may be ignored by XRSTOR.

1.3.8 Restoring Processor State

When XRSTOR is executed, processor state components are updated only if the corresponding bits in the mask operand (EDX:EAX) and XCR0 are both set. For each updated component, when the corresponding bit in the XSTATE_BV field in the save area header is set, the component is loaded from the save area in memory. When the XSTATE_BV bit is cleared, the state is set to the hardware-specified initial values shown in Table 1-10.

Table 1-10. XRSTOR Hardware-Specified Initial Values

Component	Initial Value

x87	FCW = 037Fh
	FSW = 0000h
	FTW = FFFFh
	x87 Error Pointers = 0
	ST0 - ST7 = 0

XMM	XMM0 - XMM15 = 0, if 64-bit mode XMM0 - XMM7 = 0, if !64-bit mode
YMM_HI	YMM_HI0 -Y MM_HI15 = 0, if 64-bit mode YMM_HI0-YMM_HI7 = 0, if !64-bit mode
LWP	LWP disabled

1.3.9 MXCSR State Management

The MXCSR has no hardware-specified initial state; it is read from the save area in memory whenever either XMM or YMM_HI are updated.

1.3.10 Mode-Specific XSAVE/XRSTOR State Management

Some state is conditionally saved or updated, depending on processor state:

- The x87 error pointers are not saved or restored if the state saved or loaded from memory doesn't have a pending #MF.
- XMM8 - XMM15 are not saved or restored in !64-bit mode.
- YMM_HI8-YMM_HI15 are not saved or restored in !64-bit mode.

1.4 AES Instructions

This section provides an overview of AMD64 instructions that support AES software implementation.

The U.S. National Institute of Standards and Technology has adopted the Rijndael algorithm, a block cipher that processes 16-byte data blocks using a shared key of variable length, as the Advanced Encryption Standard (AES). The standard is defined in Federal Information Processing Standards Publication 197 (FIPS 197), *Specification for the Advanced Encryption Standard (AES)*. There are three versions of the algorithm, based on key widths of 16 (AES-128), 24 (AES-192), and 32 (AES-256) bytes.

The following AMD64 instructions support AES implementation:

- AESDEC/VAESDEC and AESDECLAST/VAESDECLAST
  Perform one round of AES decryption
- AESENC/VAESENC and AESENCLAST/VAESENCLAST
  Perform one round of AES encryption
- AESIMC/VAESIMC
  Perform the AES InvMixColumn transformation
- AESKEYGENASSIST/VAESKEYGENASSIST
  Assist AES round key generation
- PCLMULQDQ, VPCLMULQDQ
  Perform carry-less multiplication

See Chapter 2, “Instruction Reference” for detailed descriptions of the instructions.

1.4.1 Coding Conventions

This overview uses descriptive code that has the following basic characteristics.

- Syntax and notation based on the C language
- Four numerical data types:
  - bool: The numbers 0 and 1, the values of the Boolean constants false and true
  - nat: The infinite set of all natural numbers, including bool as a subtype
  - int: The infinite set of all integers, including nat as a subtype
  - rat: The infinite set of all rational numbers, including int as a subtype
- Standard logical and arithmetic operators
- Enumeration (enum) types, arrays, structures (struct), and union types
- Global and local variable and constant declarations, initializations, and assignments
- Standard control constructs (if, then, else, for, while, switch, break, and continue)
- Function subroutines
- Macro definitions (#define)

1.4.2 AES Data Structures

The AES instructions operate on 16-byte blocks of text called the *state*. Each block is represented as a 4×4 matrix of bytes which is assigned the Galois field matrix data type (GFMatrix). In the AMD64 implementation, the matrices are formatted as 16-byte vectors in XMM registers or 128-bit memory locations. This overview represents each matrix as a sequence of 16 bytes in little-endian format (least significant byte on the right and most significant byte on the left).

Figure 1-8 shows a state block in 4×4 matrix representation.

$$GFMatrix = \begin{vmatrix} X_{3,0} & X_{2,0} & X_{1,0} & X_{0,0} \\ X_{3,1} & X_{2,1} & X_{1,1} & X_{0,1} \\ X_{3,2} & X_{2,2} & X_{1,2} & X_{0,2} \\ X_{3,3} & X_{2,3} & X_{1,3} & X_{0,3} \end{vmatrix}$$

Figure 1-8. GFMatrix Representation of 16-byte Block

Figure 1-9 shows the AMD64 AES format, with the corresponding mapping of FIPS 197 AES "words" to operand bytes.

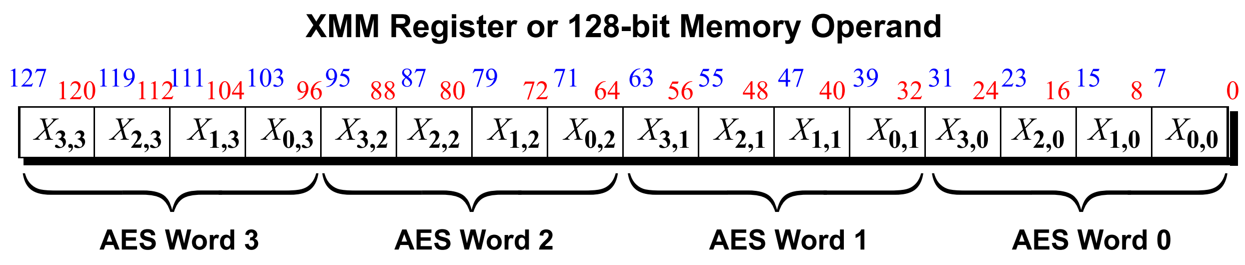

XMM Register or 128-bit Memory Operand
127 120 119 112 111 104 103 96 95 88 87 80 79 72 71 64 63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
$X_{3,3}$ $X_{2,3}$ $X_{1,3}$ $X_{0,3}$ $X_{3,2}$ $X_{2,2}$ $X_{1,2}$ $X_{0,2}$ $X_{3,1}$ $X_{2,1}$ $X_{1,1}$ $X_{0,1}$ $X_{3,0}$ $X_{2,0}$ $X_{1,0}$ $X_{0,0}$
AES Word 3
AES Word 2
AES Word 1
AES Word 0

Figure 1-9. GFMatrix to Operand Byte Mappings

1.4.3 Algebraic Preliminaries

AES operations are based on the Galois field $GF = GF(2^8)$, of order 256, constructed by adjoining a root of the irreducible polynomial

$$p(X) = X^8 + X^4 + X^3 + X + 1$$

to the field of two elements, $\mathbb{Z}_2$. Equivalently, GF is the quotient field $\mathbb{Z}_2[X]/p(X)$ and thus may be viewed as the set of all polynomials of degree less than 8 in $\mathbb{Z}_2[X]$ with the operations of addition and multiplication modulo $p(X)$. These operations may be implemented efficiently by exploiting the mapping from $\mathbb{Z}_2[X]$ to the natural numbers given by

$$a_n X^n + \ldots + a_1 X + a_0 \rightarrow 2^n a_n + \ldots + 2a_1 + a_0 \rightarrow a_n \ldots a_1 a_0 \mathrm{b}$$

For example:

$$1 \rightarrow 01\text{h}$$

$$X \rightarrow 02\text{h}$$

$$X^2 \rightarrow 04\text{h}$$

$$X^4 + X^3 + 1 \rightarrow 19\text{h}$$

$$p(X) \rightarrow 11\text{Bh}$$

Thus, each element of *GF* is identified with a unique byte. This overview uses the data type **GF256** as an alias of **nat**, to identify variables that are to be thought of as elements of *GF*.

The operations of addition and multiplication in *GF* are denoted by $\oplus$ and $\odot$, respectively. Since $\mathbb{Z}_2$ is of characteristic 2, addition is simply the “exclusive or” operation:

$$x \oplus y = x \wedge y$$

In particular, every element of *GF* is its own additive inverse.

Multiplication in *GF* may be computed as a sequence of additions and multiplications by 2. Note that this operation may be viewed as multiplication in $\mathbb{Z}_2[X]$ followed by a possible reduction modulo $p(X)$. Since 2 corresponds to the polynomial X and 11B corresponds to $p(X)$, for any $x \in GF$,

$$2 \odot \text{x} = \begin{cases} x << 1 & \text{if } x < 80\text{h} \\ (x << 1) \oplus 11\text{Bh} & \text{if } x \geq 80\text{h} \end{cases}$$

Now, if $y = b_7 \ldots b_1 b_0 b$, then

$$x \odot y = 2 \odot (\ldots(2 \odot (2 \odot (b_7 \odot x) \oplus b_6 \odot x) \oplus b_5 \odot x) \ldots b_0.$$

This computation is performed by the **GFMul( )** function.

1.4.3.1 Multiplication in the Field GF

The **GFMul( )** function operates on GF256 elements in SRC1 and SRC2 and returns a GF256 matrix in the destination.

```
GF256 GFMul(GF256 x, GF256 y) {
  nat sum = 0;
  for (int i=7; i>=0; i--) {
    // Multiply sum by 2. This amounts to a shift followed
    // by reduction mod 0x11B:
    sum <<= 1;
    if (sum > 0xFF) {sum = sum ^ 0x11B;}
    // Add y[i]*x:
    if (y[i]) {sum = sum ^ x;}
  }
  return sum;
}
```

Because the multiplicative group *GF** is of order 255, the inverse of an element x of *GF* may be computed by repeated multiplication as $x^{--1} = x^{254}$. A more efficient computation, however, is performed by the **GFInv( )** function as an application of Euclid's greatest common divisor algorithm. See Section 1.4.10, "Computation of GFInv with Euclidean Greatest Common Divisor" for an analysis of this computation and the **GFInv( )** function.

The AES algorithms operate on the vector space GF^4, of dimension 4 over *GF*, which is represented by the array type **GFWord**. FIPS 197 refers to an object of this type as a *word*. This overview uses the term *GF word* in order to avoid confusion with the AMD64 notion of a 16-bit word.

A **GFMatrix** is an array of four *GF* words, which are viewed as the rows of a 4×4 matrix over *GF.*

The field operation symbols $\oplus$ and $\odot$ are used to denote addition and multiplication of matrices over *GF* as well. The **GFMatrixMul( )** function computes the product A $\odot$ B of 4×4 matrices.

1.4.3.2 Multiplication of 4x4 Matrices Over GF

```
, GFMatrix GFMatrixMul(GFMatrix a, GFMatrix b) {
  GFMatrix c;
  for (nat i=0; i<4; i++) {
    for (nat j=0; j<4; j++) {
      c[i][j] = 0;
      for (nat k=0; k<4; k++) {
        c[i][j] = c[i][j] ^ GFMul(a[i][k], b[k][j]);
      }
    }
  }
  return c;
}
```

1.4.4 AES Operations

The AES encryption and decryption procedures may be specified as follows, in terms of a set of basic operations that are defined later in this section. See the alphabetic instruction reference for detailed descriptions of the instructions that are used to implement the procedures.

Call the **Encrypt** or **Decrypt** procedure, which pass the same expanded key to the functions

TextBlock Cipher(TextBlock in, ExpandedKey w, nat Nk)

and

TextBlock InvCipher(TextBlock in, ExpandedKey w, nat Nk)

In both cases, the input text is converted by

GFMatrix Text2Matrix(TextBlock A)

to a matrix, which becomes the initial state of the process. This state is transformed through the sequence of N_r + 1 rounds and ultimately converted back to a linear array by

TextBlock Matrix2Text(GFMatrix M).

In each round *i*, the round key K_i is extracted from the expanded key *w* and added to the state by

GFMatrix AddRoundKey(GFMatrix state, ExpandedKey w, nat round).

Note that **AddRoundKey** does not explicitly construct K_i , but operates directly on the bytes of *w*.

The rounds of **Cipher** are numbered $0,\ldots N_r$. Let X be the initial state an an execution, i.e., the input in matrix format, let S_i be the state produced by round *i*, and let $Y = S_{Nr}$ be the final state. Let Σ, R , and C denote the operations performed by **SubBytes**, **ShiftRows**, **MixColumns**, respectively. Then

The initial round is a simple addition:

$$S_0 = X \oplus K_0;$$

Each of the next N_r + 1 rounds is a composition of four operations:

$$S_i = \mathcal{C}(\mathcal{R}(\Sigma(S_{i-1}))) \oplus K_i \quad \text{for} \quad i = 1, \ldots, N_r - 1;$$

The **MixColumns** transformation is omitted from the final round:

$$Y = S_{N_r} = \mathcal{R}(\Sigma(S_{N_r-1})) \oplus K_{N_r}.$$

Composing these expressions yields

$$Y = \mathcal{R}(\Sigma(\mathcal{C}(\mathcal{R}(\Sigma(\cdots(\mathcal{C}(\mathcal{R}(\Sigma(X \oplus K_0))) \oplus K_1)\cdots))) \oplus K_{N_r-1})) \oplus K_{N_r}.$$

Note that the rounds of **InvCipher** are numbered in reverse order, $N_r,\ldots,0$. If Σ' and Y' are the initial and final states and S'_i is the state following round i , then

$$S'_{N_r} = X' \oplus K_{N_r};$$

$$S'_i = \mathcal{C}^{-1}(\Sigma^{-1}(\mathcal{R}^{-1}(S'_{i+1})) \oplus K_i) \quad \text{for} \quad i = N_r - 1, \ldots, 1;$$

$$Y' = \Sigma^{-1}(\mathcal{R}^{-1}(S'_1)) \oplus K_0.$$

Composing these expressions yields

$$Y' = \Sigma^{-1}(\mathcal{R}^{-1}(\mathcal{C}^{-1}(\Sigma^{-1}(\mathcal{R}^{-1}(\cdots(\mathcal{C}^{-1}(\Sigma^{-1}(\mathcal{R}^{-1}(X' \oplus K_{n_r}) \oplus K_{N_r-1}))\cdots)) \oplus K_1))) \oplus K_0.$$

In order to show that **InvCipher** is the inverse of **Cipher**, it is only necessary to combine these expanded expressions by replacing X' with Y and cancel inverse operations to yield $Y' = X$.

1.4.4.1 Sequence of Operations

- Use predefined **SBox** and **InvSBox** matrices or initialize the matrices using the **ComputeSBox** and **ComputeInvSBox** functions.
- Call the **Encrypt** or **Decrypt** procedure.
- For the **Encrypt** procedure:

1. Load the input **TextBlock** and **CipherKey**.
2. Expand the cipher key using the **KeyExpansion** function.
3. Call the **Cipher** function to perform the number of rounds determined by the cipher key length.
4. Perform round entry operations.
   a. Convert input text block to state matrix using the **Text2Matrix** function.
   b. Combine state and round key bytes by bitwise XOR using the **AddRoundKey** function.
5. Perform round iteration operations.
   a. Replace each state byte with another by non-linear substitution using the **SubBytes** function.
   b. Shift each row of the state cyclically using the **ShiftRows** function.
   c. Combine the four bytes in each column of the state using the **MixColumns** function.
   d. Perform **AddRoundKey**.
6. Perform round exit operations.
   a. Perform **SubBytes**.
   b. Perform **ShiftRows**.
   c. Perform **AddRoundKey**.
   d. Convert state matrix to output text block using the **Matrix2Text** function and return **TextBlock**.

- For the **Decrypt** procedure:

1. Load the input **TextBlock** and **CipherKey**.
2. Expand the cipher key using the **KeyExpansion** function.
3. Call the **InvCipher** function to perform the number of rounds determined by the cipher key length.
4. Perform round entry operations.
   a. Convert input text block to state matrix using the **Text2Matrix** function.
   b. Combine state and round key bytes by bitwise XOR using the **AddRoundKey** function.
5. Perform round iteration operations.
   a. Shift each row of the state cyclically using the **InvShiftRows** function.
   b. Replace each state byte with another by non-linear substitution using the **InvSubBytes** function.
   c. Perform **AddRoundKey**.
   d. Combine the four bytes in each column of the state using the **InvMixColumns** function.
6. Perform round exit operations.
   a. Perform **InvShiftRows**.
   b. Perform **InvSubBytes** (**InvSubWord**).
   c. Perform **AddRoundKey**.
   d. Convert state matrix to output text block using the **Matrix2Text** function and return **TextBlock**.

1.4.5 Initializing the Sbox and InvSBox Matrices

The AES makes use of a bijective mapping $\sigma : GF \rightarrow GF$, which is encoded, along with its inverse mapping, in the 16×16 arrays **SBox** (for encryption) and **InvSBox** (for decryption), as follows:

for all $x \in G$,

$$\sigma(x) = \mathrm{SBox}[x[7{:}4], x[3{:}0]]$$

and

$$\sigma^{-1}(x) = \mathrm{InvSBox}[x[7{:}4], x[3{:}0]]$$

While the FIPS 197 standard defines the contents of the **SBox[ ]** and **InvSbox [ ]** matrices, the matrices may also be initialized algebraically (and algorithmically) by means of the **ComputeSBox( )** and **ComputeInvSBox( )** functions, discussed below.

The bijective mappings for encryption and decryption are computed by the **SubByte( )** and **InvSubByte ( )** functions, respectively:

SubByte( ) computation:

```
GF256 SubByte(GF256 x) {
  return SBox[x[7:4]][x[3:0]];
}
```

InvSubByte ( ) computation:

```
GF256 InvSubByte(GF256 x) {
  return InvSBox[x[7:4]][x[3:0]];
}
```

1.4.5.1 Computation of SBox and InvSBox

Computation of SBox and InvSBox elements has a direct relationship to the cryptographic properties of the AES, but not to the algorithms that use the tables. Readers who prefer to view σ as a primitive operation may skip the remainder of this section.

The algorithmic definition of the bijective mapping σ is based on the consideration of *GF* as an 8-dimensional vector space over the subfield $\mathbb{Z}_2$. Let φ be a linear operator on this vector space and let $M = [a_{ij}]$ be the matrix representation of φ with respect to the ordered basis $\{1, 2, 4, 10, 20, 40, 80\}$. Then φ may be encoded concisely as an array of bytes *A* of dimension 8, each entry of which is the concatenation of the corresponding row of *M*:

$$A[i] = a_{i8}a_{i7}\ldots a_{i0}$$

This expression may be represented algorithmically by means of the **ApplyLinearOp( )** function, which applies a linear operator to an element of GF. The **ApplyLinear Op( )** function is used in the initialization of both the **sBox[]** and I**nvSBox[ ]** matrices.

```
// The following function takes the array A representing a linear operator phi and
// an element x of G and returns phi(x):

GF256 ApplyLinearOp(GF256 A[8], GF256 x) {
  GF256 result = 0;
  for (nat i=0; i<8; i++) {
    bool sum = 0;
    for (nat j=0; j<8; j++) {
      sum = sum ^ (A[i][j] & x[j]);
    }
    result[i] = sum;
  }
  return result;
}
```

The definition of σ involves the linear operator φ with matrix

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In this case,

$$A = \{F1, E3, C7, 8F, 1F, 3E, 7C, F8\}.$$

Initialization of SBox[ ]

The mapping $\sigma : G \rightarrow G$ is defined by

$$\sigma(x) = \varphi\,(x^{-1}) \oplus 63$$

This computation is performed by **ComputeSBox( )**.

ComputeSBox( )

```
GF256[16][16] ComputeSBox() {
  GF256 result[16][16];
  GF256 A[8] = {0xF1, 0xE3, 0xC7, 0x8F, 0x1F, 0x3E, 0x7C, 0xF8};
  for (nat i=0; i<16; i++) {
    for (nat j=0; j<16; j++) {
      GF256 x = (i << 4) | j;
      result[i][j] = ApplyLinearOp(A, GFInv(x)) ^ 0x63;
    }
  }
  return result;
}

const GF256 SBox[16][16] = ComputeSBox();
```

Table 1-11 shows the resulting **SBox[ ]**, as defined in FIPS 197.

Table 1-11. SBox Definition

		S[3:0]															
S[7:4]		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	a5
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

1.4.5.2 Initialization of InvSBox[ ]

A straightforward calculation confirms that the matrix M is nonsingular with inverse.

Thus, φ is invertible and φ^{-1} is encoded as the array

$$M^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$B = \{A4, 49, 92, 25, 4A, 94, 29, 52\}.$$

If $y = \sigma(x)$, then

$$\begin{aligned}(\varphi^{-1}((y) \oplus 5)^{-1} &= (\varphi^{-1}(y \oplus \varphi(5))^{-1} \\ &= (\varphi^{-1}(y \oplus 63))^{-1} \\ &= (\varphi^{-1}(\varphi(x^{-1}) \oplus 63 \oplus 63))^{-1} \\ &= (\varphi^{-1}(\varphi(x^{-1})))^{-1} \\ &= x,\end{aligned}$$

and σ is a permutation of GF with

$$\sigma^{-1}(y) = (\varphi^{-1}(y) \oplus 5)^{-1}$$

This computation is performed by **ComputeInvSBox( )**.

ComputeInvSBox( )

```
GF256[16][16] ComputeInvSBox() {
  GF256 result[16][16];
  GF256 B[8] = {0xA4, 0x49, 0x92, 0x25, 0x4A, 0x94, 0x29, 0x52};
  for (nat i=0; i<16; i++) {
    for (nat j=0; j<16; j++) {
      GF256 y = (i << 4) | j;
      result[i][j] = GFInv(ApplyLinearOp(B, y) ^ 0x5);
    }
  }
  return result;
}

const GF256 InvSBox[16][16] = ComputeInvSBox();
```

Table 1-12 shows the resulting **InvSBox[ ]**, as defined in the FIPS 197.

Table 1-12. InvSBox Definition

		S[3:0]															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
S[7:4]	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

1.4.6 Encryption and Decryption

The AMD64 architecture implements the AES algorithm by means of an iterative function called a *round* for both encryption and the inverse operation, decryption.

The top-level encryption and decryption procedures **Encrypt( )** and **Decrypt( )** set up the rounds and invoke the functions that perform them. Each of the procedures takes two 128-bit binary arguments:

- *input data* — a 16-byte block of text stored in a source 128-bit XMM register
- *cipher key* — a 16-, 24-, or 32-byte cipher key stored in either a second 128-bit XMM register or 128-bit memory location

1.4.6.1 The Encrypt( ) and Decrypt( ) Procedures

```
TextBlock Encrypt(TextBlock in, CipherKey key, nat Nk) {
  return Cipher(in, ExpandKey(key, Nk), Nk);
}

TextBlock Decrypt(TextBlock in, CipherKey key, nat Nk) {
  return InvCipher(in, ExpandKey(key, Nk), Nk);
}
```

The array types **TextBlock** and **CipherKey** are introduced to accommodate the text and key parameters. The 16-, 24-, or 32-byte cipher keys correspond to AES-128, AES-192, or AES-256 key sizes. The cipher key is logically partitioned into $N_k = 4$, 6, or 8 AES 32-bit words. N_k is passed as a parameter to determine the AES version to be executed, and the number of rounds to be performed.

Both the **Encrypt( )** and **Decrypt( )** procedures invoke the **ExpandKey( )** function to expand the cipher key for use in round key generation. When key expansion is complete, either the **Cipher( )** or **InvCipher( )** functions are invoked.

The **Cipher( )** and **InvCipher( )** functions are the key components of the encryption and decryption process. See Section 1.4.7, “The Cipher Function” and Section 1.4.8, “The InvCipher Function” for detailed information.

1.4.6.2 Round Sequences and Key Expansion

Encryption and decryption are performed in a sequence of rounds indexed by 0, …, N_r, where N_r is determined by the number N_k of *GF* words in the cipher key. A key matrix called a *round key* is generated for each round. The number of *GF* words required to form $N_r + 1$ round keys is equal to , $4(N_r + 1)$. Table 1-13 shows the relationship between cipher key length, round sequence length, and round key length.

Table 1-13. Cipher Key, Round Sequence, and Round Key Length

N_k	N_r	$4(N_r + 1)$
4	10	44
6	12	52
8	14	60

Expanded keys are generated from the cipher key by the **ExpandKey( )** function, where the array type **ExpandedKey** is defined to accommodate 60 words (the maximum required) corresponding to $N_k = 8$.

The ExpandKey( ) Function

```
ExpandedKey ExpandKey(CipherKey key, nat Nk) {
  assert((Nk == 4) || (Nk == 6) || (Nk == 8));
  nat Nr = Nk + 6;
  ExpandedKey w;

  // Copy key into first Nk rows of w:
  for (nat i=0; i<Nk; i++) {
    for (nat j=0; j<4; j++) {
      w[i][j] = key[4*i+j];
    }
  }
  // Write next row of w:
  for (nat i=Nk; i<4*(Nr+1); i++) {

    // Encode preceding row:
    GFWord tmp = w[i-1];
    if (mod(i, Nk) == 0) {
      tmp = SubWord(RotWord(tmp));
      tmp[0] = tmp[0] ^ RCON[i/Nk];
    }
    else if ((Nk == 8) && (mod(i, Nk) == 4)) {
      tmp = SubWord(tmp);
    }

    // XOR tmp with w[i-Nk]:
    for (nat j=0; j<4; j++) {
      w[i][j] = w[i-Nk][j] ^ tmp[j];
    }
  }
  return w;
}
```

ExpandKey( ) begins by copying the input cipher key into the first N_k *GF* words of the expanded key *w*. The remaining $4(N_r + 1) - N_k$ GF words are computed iteratively. For each $i \geq N_k$, $w[i]$ is derived from the two *GF* words $w[i - 1]$ and $w[i - N_k]$. In most cases, $w[i]$ is simply the sum $w[i - 1] \oplus w[i - N_k]$. There are two exceptions:

- If i is divisible by N_k, then before adding it to $w[i - N_k]$, $w[i - 1]$ is first rotated by one position to the left by **RotWord( )**, then transformed by the substitution **SubWord( )**, and an element of the array **RCON** is added to it.

  RCON[11] = {00h, 01h, 02h, 04h, 08h, 10h, 20h, 40h, 80h, 1Bh, 36h}

- In the case $N_k = 8$, if i is divisible by 4 but not 8, then $w[i - 1]$ is transformed by the substitution **SubWord( )**.

The i^{th} round keyK_i comprises the four *GF* words $w[4i]$, …, $w[4i + 3]$. More precisely, let W_i be the matrix

$$W= \{w[4i], w[4i + 1], w[4i + 2], w[4i + 3]\}$$

Then $K_i = W_i^t$, the transpose of W_i. Thus, the entries of the array *w* are the columns of the round keys.

1.4.7 The Cipher Function

This function performs encryption. It converts the input text to matrix form, generates the round key from the expanded key matrix, and iterates through the transforming functions the number of times determined by encryption key size to produce a 128-bit binary cipher matrix. As a final step, it converts the matrix to an output text block.

```
TextBlock Cipher(TextBlock in, ExpandedKey w, nat Nk) {
  assert((Nk == 4) || (Nk == 6) || (Nk == 8));
  nat Nr = Nk + 6;
  GFMatrix state = Text2Matrix(in);
  state = AddRoundKey(state, w, 0);
  for (nat round=1; round<Nr; round++) {
    state = SubBytes(state);
    state = ShiftRows(state);
    state = MixColumns(state);
    state = AddRoundKey(state, w, round);
  }
  state = SubBytes(state);
  state = ShiftRows(state);
  state = AddRoundKey(state, w, Nr);
  return Matrix2Text(state);
}
```

1.4.7.1 Text to Matrix Conversion

Prior to processing, the input text block must be converted to matrix form. The **Text2Matrix( )** function stores a **TextBlock** in a GFMatrix in column-major order as follows.

```
GFMatrix Text2Matrix(TextBlock A) {
  GFMatrix result;
  for (nat j=0; j<4; j++) {
    for (nat i=0; i<4; i++) {
      result[i][j] = A[4*j+i];
    }
  }
  return result;
}
```

1.4.7.2 Cipher Transformations

The Cipher function employs the following transformations.

SubBytes( ) — Applies a non-linear substitution table (SBox) to each byte of the state.

SubWord( ) — Uses a non-linear substitution table (SBox) to produce a four-byte AES output word from the four bytes of an AES input word.

ShiftRows( ) — Cyclically shifts the last three rows of the state by various offsets.

RotWord( ) — Rotates an AES (4-byte) word to the right.

MixColumns( ) — Mixes data in all the state columns independently to produce new columns.

AddRoundKey( ) — Extracts a 128-bit round key from the expanded key matrix and adds it to the 128-bit state using an XOR operation.

Inverses of **SubBytes( )**, **SubWord( )**, **ShiftRows( )** and **MixColumns( )** are used in decryption. See Section 1.4.8, “The InvCipher Function” for more information.

The SubBytes( ) Function

Performs a byte substitution operation using the invertible substitution table (**SBox**) to convert input text to an intermediate encryption state.

```
GFMatrix SubBytes(GFMatrix M) {
  GFMatrix result;
  for (nat i=0; i<4; i++) {
    result[i] = SubWord(M[i]);
  }
  return result;
}
```

The SubWord( ) Function

Applies **SubBytes** to each element of a vector or a matrix:

```
GFWord SubWord(GFWord x) {
  GFWord result;
  for (nat i=0; i<4; i++) {
    result[i] = SubByte(x[i]);
  }
  return result;
}
```

The ShiftRows( ) Function

Cyclically shifts the last three rows of the state by various offsets.

```
GFMatrix ShiftRows(GFMatrix M) {
  GFMatrix result;
  for (nat i=0; i<4; i++) {
    result[i] = RotateLeft(M[i], -i);
  }
  return result;
```

The RotWord( ) Function

Performs byte-wise cyclic permutation of a 32-bit AES word.

```
GFWord RotWord(GFWord x)
{ return RotateLeft(x, 1); }
```

The MixColumns( ) Function

Performs a byte-oriented column-by-column matrix multiplication

$M \rightarrow \mathrm{C} \odot M$, where C is the predefined fixed matrix

$$C = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

The function is implemented as follows:

```
GFMatrix MixColumns(GFMatrix M) {
  GFMatrix C = {
    {0x02,0x03,0x01,0x01},
    {0x01,0x02,0x03,0x01},
    {0x01,0x01,0x02,0x03},
    {0x03,0x01,0x01,0x02}
  };
  return GFMatrixMul(C, M);
}
```

The AddRoundKey( ) Function

Extracts the round key from the expanded key and adds it to the state using a bitwise XOR operation.

```
GFMatrix AddRoundKey(GFMatrix state, ExpandedKey w, nat round) {
  GFMatrix result = state;
  for (nat i=0; i<4; i++) {
    for (nat j=0; j<4; j++) {
      result[i][j] = result[i][j] ^ w[4*round+j][i];
    }
  }
  return result;
}
```

1.4.7.3 Matrix to Text Conversion

After processing, the output matrix must be converted to a text block. The **Matrix2Text( )** function converts a GFMatrix in column-major order to a **TextBlock** as follows.

```
TextBlock Matrix2Text(GFMatrix M) {
  TextBlock result;
  for (nat j=0; j<4; j++) {
    for (nat i=0; i<4; i++) {
      result[4*j+i] = M[i][j];
    }
  }
  return result;
}
```

1.4.8 The InvCipher Function

This function performs decryption. It iterates through the round function the number of times determined by encryption key size and produces a 128-bit block of text as output.

```
TextBlock InvCipher(TextBlock in, ExpandedKey w, nat Nk) {
  assert((Nk == 4) || (Nk == 6) || (Nk == 8));
  nat Nr = Nk + 6;
  GFMatrix state = Text2Matrix(in);
  state = AddRoundKey(state, w, Nr);
  for (nat round=Nr-1; round>0; round--) {
    state = InvShiftRows(state);
    state = InvSubBytes(state);
    state = AddRoundKey(state, w, round);
    state = InvMixColumns(state);
  }
  state = InvShiftRows(state);
  state = InvSubBytes(state);
  state = AddRoundKey(state, w, 0);
  return Matrix2Text(state);
}
```

1.4.8.1 Text to Matrix Conversion

Prior to processing, the input text block must be converted to matrix form. The **Text2Matrix( )** function stores a **TextBlock** in a GFMatrix in column-major order as follows.

```
GFMatrix Text2Matrix(TextBlock A) {
  GFMatrix result;
  for (nat j=0; j<4; j++) {
    for (nat i=0; i<4; i++) {
      result[i][j] = A[4*j+i];
    }
  }
  return result;
}
```

1.4.8.2 InvCypher Transformations

The following functions are used in decryption:

InvShiftRows( ) — The inverse of **ShiftRows( )**.

InvSubBytes( ) — The inverse of **SubBytes( )**.

InvSubWord( ) — The inverse of **SubWord( )**.

InvMixColumns( ) — The inverse of **MixColumns( )**.

AddRoundKey( ) — Is its own inverse.

Decryption is the inverse of encryption and is accomplished by means of the inverses of the, **SubBytes( )**, **SubWord( )**, **ShiftRows( )** and **MixColumns( )** transformations used in encryption.

SubWord( ), **SubBytes( )**, and **ShiftRows( )** are injective. This is also the case with **MixColumns( )**. A simple computation shows that *C* is invertible with

$$C^{-1} = \begin{bmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{bmatrix}$$

The InvShiftRows( ) Function

The inverse of **ShiftRows( )**.

```
GFMatrix InvShiftRows(GFMatrix M) {
  GFMatrix result;
  for (nat i=0; i<4; i++) {
    result[i] = RotateLeft(M[i], -i);
  }
  return result;
```

The InvSubBytes( ) Function

The inverse of *SubBytes*( ).

```
GFMatrix InvSubBytes(GFMatrix M) {
  GFMatrix result;
  for (nat i=0; i<4; i++) {
    result[i] = InvSubWord(M[i]);
  }
  return result;
}
```

The InvSubWord( ) Function

The inverse of **SubWord( )**, **InvSubBytes( )** applied to each element of a vector or a matrix.

```
GFWord InvSubWord(GFWord x) {
  GFWord result;
  for (nat i=0; i<4; i++) {
    result[i] = InvSubByte(x[i]);
  }
  return result;
}
```

The InvMixColumns( ) Function

The inverse of the **MixColumns( )** function. Multiplies by the inverse of the predefined fixed matrix, *C*, C^{-1}, as discussed previously.

```
GFMatrix InvMixColumns(GFMatrix M) {
  GFMatrix D = {
    {0x0e,0x0b,0x0d,0x09},
    {0x09,0x0e,0x0b,0x0d},
    {0x0d,0x09,0x0e,0x0b},
    {0x0b,0x0d,0x09,0x0e}
  };
  return GFMatrixMul(D, M);
}
```

The AddRoundKey( ) Function

Extracts the round key from the expanded key and adds it to the state using a bitwise XOR operation.

```
GFMatrix AddRoundKey(GFMatrix state, ExpandedKey w, nat round) {
  GFMatrix result = state;
  for (nat i=0; i<4; i++) {
    for (nat j=0; j<4; j++) {
      result[i][j] = result[i][j] ^ w[4*round+j][i];
    }
  }
  return result;
}
```

1.4.8.3 Matrix to Text Conversion

After processing, the output matrix must be converted to a text block. The **Matrix2Text( )** function converts a GFMatrix in column-major order to a **TextBlock** as follows.

```
TextBlock Matrix2Text(GFMatrix M) {
  TextBlock result;
  for (nat j=0; j<4; j++) {
    for (nat i=0; i<4; i++) {
      result[4*j+i] = M[i][j];
    }
  }
  return result;
}
```

1.4.9 An Alternative Decryption Procedure

This section outlines an alternative decrypting procedure,

TextBlock EqDecrypt(TextBlock in, CipherKey key, nat Nk):

```
TextBlock EqDecrypt(TextBlock in, CipherKey key, nat Nk) {
  return EqInvCipher(in, MixRoundKeys(ExpandKey(key, Nk), Nk), Nk);
}
```

The procedure is based on a variation of **InvCipher**,

TextBlock EqInvCipher(TextBlock in, ExpandedKey w, nat Nk):

```
TextBlock EqInvCipher(TextBlock in, ExpandedKey dw, nat Nk) {
  assert((Nk == 4) || (Nk == 6) || (Nk == 8));
  nat Nr = Nk + 6;
  GFMatrix state = Text2Matrix(in);
  state = AddRoundKey(state, dw, Nr);
  for (nat round=Nr-1; round>0; round--) {
    state = InvSubBytes(state);
    state = InvShiftRows(state);
    state = InvMixColumns(state);
    state = AddRoundKey(state, dw, round);
  }
  state = InvSubBytes(state);
  state = InvShiftRows(state);
  state = AddRoundKey(state, dw, 0);
  return Matrix2Text(state);
}
```

The variant structure more closely resembles that of **Cipher**. This requires a modification of the expanded key generated by **ExpandKey**,

ExpandedKey MixRoundKeys(ExpandedKey w, nat Nk):

```
ExpandedKey MixRoundKeys(ExpandedKey w, nat Nk) {
  assert((Nk == 4) || (Nk == 6) || (Nk == 8));
  nat Nr = Nk + 6;
  ExpandedKey result;
  GFMatrix roundKey;
  for (nat round=0; round<Nr+1; round++) {
    for (nat i=0; i<4; i++) {
      roundKey[i] = w[4*round+i];
    }
    if ((round > 0) && (round < Nr)) {
      roundKey = InvMixRows(roundKey);
    }
    for (nat i=0; i<4; i++) {
      result[4*round+i] = roundKey[i];
    }
  }
  return result;
}
```

The transformation **MixRoundKeys** leaves $K0$ and K_{Nr} unchanged, but for $i = 1,...,N_r - 1$, it replaces W_i with the matrix product $W_i \odot Q$, where

$$Q = \begin{bmatrix} E & 9 & D & B \\ B & E & 9 & D \\ D & B & E & 9 \\ 9 & D & B & E \end{bmatrix} = \begin{bmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{bmatrix}^t = (C^{-1})^t.$$

The effect of this is to replace K_i with

$$(W_i \odot Q)^t = Q^t \odot W_i^t = C^{-1} \odot K_i = \mathcal{C}^{-1}(K_i)$$

for $i = 1,...,\mathrm{N}r - 1$.

The equivalence of **EqDecrypt** and **Decrypt** follows from two properties of the basic operations:

$\mathcal{C}$ is a linear transformation and therefore, so is $\mathcal{C}^{-1;}$

Σ and R commute, and hence so do Σ^{-1} and R^{-1}, for if

$$S = \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix},$$

then

$$\Sigma(\mathcal{R}(S)) = \begin{bmatrix} \sigma(s_{00}) & \sigma(s_{01}) & \sigma(s_{02}) & \sigma(s_{03}) \\ \sigma(s_{11}) & \sigma(s_{12}) & \sigma(s_{13}) & \sigma(s_{10}) \\ \sigma(s_{22}) & \sigma(s_{23}) & \sigma(s_{20}) & \sigma(s_{21}) \\ \sigma(s_{33}) & \sigma(s_{30}) & \sigma(s_{31}) & \sigma(s_{32}) \end{bmatrix} = \mathcal{R}(\Sigma(S)).$$

Now let $X''$and Y''be the initial and final states of an execution of **EqDecrypt** and let S''_i be the state following round i. Suppose $X''= X'$. Appealing to the definitions of **EqDecrypt** and **EqInvCipher**, we have

$$S''_{N_r} = X'' \oplus K_{N_r} = X' \oplus K_{N_r} = S'_{N_r},$$

and for $i = N_r - 1,...,1$, by induction,

$$\begin{aligned} S''_i &= \mathcal{C}^{-1}(\Sigma^{-1}(\mathcal{R}^{-1}(S''_{i+1}))) \oplus \mathcal{C}^{-1}(K_{N_r}) \\ &= \mathcal{C}^{-1}(\Sigma^{-1}(\mathcal{R}^{-1}(S''_{i+1})) \oplus K_{N_r}) \\ &= \mathcal{C}^{-1}(\mathcal{R}^{-1}(\Sigma^{-1}(S''_{i+1})) \oplus K_{N_r}) \\ &= \mathcal{C}^{-1}(\mathcal{R}^{-1}(\Sigma^{-1}(S'_{i+1})) \oplus K_{N_r}) \\ &= S'_i. \end{aligned}$$

Finally,

$$\begin{aligned} Y'' = S''_0 &= \mathcal{R}^{-1}(\Sigma^{-1}(S''_1)) \oplus K_0 \\ &= \Sigma^{-1}(\mathcal{R}^{-1}(S''_1)) \oplus K_0 \\ &= \Sigma^{-1}(\mathcal{R}^{-1}(S'_1)) \oplus K_0 \\ &= S'_0 = Y'. \end{aligned}$$

1.4.10 Computation of GFInv with Euclidean Greatest Common Divisor

Note that the operations performed by **GFInv( )** are in the ring $\mathbb{Z}_2[X]$ rather than the quotient field *GF*.

The initial values of the variables x_1 and x_2 are the inputs x and 11b, the latter representing the polynomial $p(X)$. The variables a_1 and a_2 are initialized to 1 and 0.

On each iteration of the loop, a multiple of the lesser of x_1 and x_2 is added to the other. If $x_1 \le x_2$, then the values of x_2 and a_2 are adjusted as follows:

$$x_2 \rightarrow x_2 \oplus 2^s \odot x_1$$

$$a_2 \rightarrow a_2 \oplus 2^s \odot a_1$$

where s is the difference in the exponents (*i.e.*, degrees) of x_1 and x_2 . In the remaining case, x_1 and a_1 are similarly adjusted. This step is repeated until either $x_1 = 0$ or $x_2 = 0$.

We make the following observations:

- On each iteration, the value added to *xi* has the same exponent as x_i, and hence the sum has lesser exponent. Therefore, termination is guaranteed.
- Since $p(X)$ is irreducible and x is of smaller degree than $p(X)$, the initial values of x_1 and x_2 have no non-trivial common factor. This property is clearly preserved by each step.
- Initially,

$$x_1 \oplus a_1 \odot x = x \oplus x = 0$$

and

$$x_2 \oplus a_2 \odot x = 11\text{b} \oplus 0 = 11\text{b}$$

are both divisible by 11b. This property is also invariant, since, for example, the above assignments result in

$$x_2 \oplus a_2 \odot x \rightarrow (x_2 \oplus 2^s \odot x_1) \oplus (a_2 \oplus 2^s \odot a_1) \odot x = (x_2 \oplus a2 \odot \text{x}) \oplus 2^s \odot (x_1 \oplus a_1 \odot \text{x}).$$

Now suppose that the loop terminates with $x_2 = 0$. Then x_1 has no non-trivial factor and, hence, $x_1 = 1$. Thus, $1 \oplus a_1 \odot x$ is divisible by 11b. Since the final result y is derived by reducing a_1 modulo 11b, it follows that $1 \oplus y \odot x$ is also divisible by 11b and, hence, in the quotient field *GF*, $1 + y \odot x = 0$, which implies $y \odot x = 1$.

The computation of the multiplicative inverse utilizing Euclid's algorithm is as follows:

```
// Computation of multiplicative inverse based on Euclid's algorithm:

GF256 GFInv(GF256 x) {
  if (x == 0) {
    return 0;
  }
  // Initialization:
  nat x1 = x;
  nat x2 = 0x11B; // the irreducible polynomial p(X)
  nat a1 = 1;
  nat a2 = 0;
  nat shift; // difference in exponents
  while ((x1 != 0) && (x2!= 0)) {

  // Termination is guaranteed, since either x1 or x2 decreases on each iteration.
  // We have the following loop invariants, viewing natural numbers as elements of
  // the polynomial ring Z2[X]:
  // (1) x1 and x2 have no common divisor other than 1.
  // (2) x1 ^ GFMul(a1, x) and x2 ^ GFMul(a2, x) are both divisible by p(X).

    if (x1 <= x2) {
      shift = expo(x2) - expo(x1);
      x2 = x2 ^ (x1 << shift);
      a2 = a2 ^ (a1 << shift);
    }
    else {
      shift = expo(x1) - expo(x2);
      x1 = x1 ^ (x2 << shift);
      a1 = a1 ^ (a2 << shift);
    }
  }
  nat y;

  // Since either x1 or x2 is 0, it follows from (1) above that the other is 1.

  if (x1 == 1) { // x2 == 0
    y = a1;
  }
  else if (x2 == 1) { // x1 == 0
    y = a2;
  }
  else {
    assert(false);
  }

  // Now it follows from (2) that GFMul(y, x) ^ 1 is divisible by 0x11b.
  // We need only reduce y modulo 0x11b:

  nat e = expo(y);
  while (e >= 8) {
    y = y ^ (0x11B << (e - 8));
    e = expo(y);
  }
  return y;
}
```

1.5 String Compare Instructions

The (V)PCMPESTRI, (V)PCMPESTRM, (V)PCMPISTRI, and (V)PCMPISTRM instructions perform arithmetic comparisons of byte or word elements in two source operands using positive Boolean logic. All possible comparisons are performed, 64 for words, 256 for bytes. Individual comparison results are aggregated and processed to produce final results.

Instruction operation affects the arithmetic status flags (ZF, CF, SF, OF, AF, PF), but the flags are defined to provide additional information.

The instructions have a defined base function and additional functionality controlled by bit fields in an immediate byte operand. The base function determines whether the source strings have implicitly (I) or explicitly (E) defined lengths, and whether the result is an index (I) or a mask (M).

Some immediate operand functions are specific to a particular instruction and others pertain to two or more instructions. This description covers functions that are common to all of the instructions. Individual functional differences are covered in the specific instruction descriptions.

Bit fields of the immediate operand control the following functions:

Source data format — data element length, signed or unsigned

Aggregation — comparison type and intermediate result aggregation

Complementation — intermediate result processing

Output selection — type of processing performed to produce final result

1.5.1 Source Data Format

Bits [1:0] of the immediate byte operand determine source data format, as shown in Table 1-14.

Table 1-14. Source Data Format

Imm8[1:0]	Source Content
00b	16 packed unsigned bytes
01b	8 packed unsigned words
10b	16 packed signed bytes
11b	8 packed signed words

1.5.2 Aggregation

Bits [3:2] of the immediate byte operand determine comparison type and aggregation method, as shown in Table 1-15. Aggregation results are stored in *IntRes1*. See Section 1.5.5, "Valid/Invalid Override of Comparisons" for more information about the override function.

Table 1-15. Comparison and Aggregation Method

Imm8[3:2]	Comparison	Method
00b	Equal	Equal any
01b	Greater than or equal for even-indexed elements of *reg* and corresponding elements of *reg/mem*. Less than or equal for odd-indexed elements of *reg* and corresponding elements of *reg/mem*.	Ranges
10b	Equal	Equal each
11b	Equal	Equal ordered

1.5.2.1 Equal Any Aggregation Method

Finds characters in a set.

```
UpperBound = imm8[0] ? 7:15;
IntRes1 = 0;
For j = 0 to UpperBound, j++
For i = 0 to UpperBound, i++
IntRes1[j] OR= overrideIfDataInvalid(BoolRes[j,i])
```

1.5.2.2 Ranges Aggregation Method

Finds characters in ranges.

```
UpperBound = imm8[0]?7:15;
IntRes1 = 0;
For j = 0 to UpperBound, j++
For i = 0 to UpperBound, i+=2
IntRes1[j]  OR= (overrideIfDataInvalid(BoolRes[j,i])
            AND overrideIfDataInvalid(BoolRes[j,i+1]))
```

1.5.2.3 Equal Each Aggregation Method

Performs string compare.

```
UpperBound = imm8[0]?7:15;
IntRes1 = 0;
For i = 0 to UpperBound, i++
IntRes1[i] = overrideIfDataInvalid(BoolRes[i,i])
```

1.5.2.4 Equal Ordered Aggregation Method

Performs a substring search

```
UpperBound = imm8[0]?7:15;
IntRes1 = imm8[0] ? 0xFF : 0xFFFF
For j = 0 to UpperBound, j++
For i = 0 to UpperBound-j, k=j to UpperBound, k++, i++
IntRes1[j] AND= overrideIfDataInvalid(BoolRes[k,i])
```

1.5.3 Complementation

Bit [4] of the immediate operand determines whether a ones'-complement is performed on *IntRes1*; bit [5] of the immediate operand determines whether a complementation mask is used (see Table 1-16). The mask allows complementation only when an *IntRes1* bit corresponds to a valid reg/mem source element. The basic functional definition of each instruction determines the validity of elements.

Table 1-16. Complementation

Imm8[5:4]	Description
00b	IntRes2 = IntRes1
01b	IntRes2 = -1 XOR IntRes1
10b	IntRes2 = IntRes1
11b	IntRes2[i] = IntRes1[i] if reg/mem[i] invalid, else =~IntRes1[i]

After complementation, the data is stored in a second intermediate result, *IntRes2*.

1.5.4 Output Selection

For (V)PCMPESTRI and (V)PCMPISTRI, bit [6] of the immediate operand determines whether the index of the lowest set bit or the highest set bit of *IntRes2* is written to the destination, as shown in Table 1-17.

Table 1-17. Indexed Comparison Output Selection

Imm8[6]	Description
0b	Return the index of the least significant set bit in IntRes2.
1b	Return the index of the most significant set bit in IntRes2.

For (V)PCMPESTRM and (V)PCMPISTRM, bit [6] of the immediate operand determines whether the mask is a 16-bit or 8-bit mask or an expanded 128-bit byte/word mask, as shown in Table 1-18. Mask size is determined by *imm8[1]*. The mask is expanded by copying each bit of *IntRes2* to all bits of the element of the same index.

Table 1-18. Masked Comparison Output Selection

Imm8[6]	Description
0b	Return IntRes2 as the mask with zero extension to 128 bits.
1b	Return expanded IntRes2 mask.

1.5.5 Valid/Invalid Override of Comparisons

The string comparison instructions allow for occurrence of an end-of-string (EOS) within the source data. Source data elements that are determined to be past the EOS are considered invalid. Aggregation method determines how invalid data within a comparison pair are handled. In most cases, the comparison result for each element pair BoolRes[i,j] is forced true or false if one or more elements of the pair are invalid. Table 1-19 summarizes override operation.

Table 1-19. End-of-String Comparison Override

xmm1, **byte/word**	***xmm2/m128***, **byte/word**	***Imm8[3:2]*=00b, (equal any)**	***Imm8[3:2]*=01b, (ranges)**	***Imm8[3:2]*=10b, (equal each)**	***Imm8[3:2]*=11b, (equal ordered)**
Invalid	Invalid	Force false	Force false	Force true	Force true
Invalid	Valid	Force false	Force false	Force false	Force true
Valid	Invalid	Force false	Force false	Force false	Force false
Valid	Valid	Do not force	Do not force	Do not force	Do not force

Bit [7] of the immediate byte operand is reserved.

2 Instruction Reference

Instructions are listed by mnemonic, in alphabetic order. Each entry describes instruction function, syntax, opcodes, affected flags and exceptions related to the instruction.

Figure 2-1 shows the conventions used in the descriptions. Items that do not pertain to a particular instruction, such as a synopsis of the 256-bit form, may be omitted.

INST
VINST

Instruction
Mnemonic Expansion

Brief functional description

INST

Description of legacy version of instruction.

VINST

Description of extended version of instruction.

XMM Encoding

Description of 128-bit extended instruction.

YMM Encoding

Description of 256-bit extended instruction.

Information about CPUID functions related to the instruction set.

Synopsis diagrams for legacy and extended versions of the instruction.

Mnemonic	Opcode	Description
INST *xmm1*, *xmm2*/*mem128*	FF FF /r	Brief summary of legacy operation.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VINST *xmm1*, *xmm2*/*mem128, xmm3*	C4	$\overline{\text{RXB}}$.11	0.$\overline{\textit{src}}$.0.00	FF /r
VINST *ymm1*, *ymm2*/*mem256, ymm3*	C4	$\overline{\text{RXB}}$.11	0.$\overline{\textit{src}}$.0.00	FF /r

Related Instructions

Instructions that perform similar or related functions.

rFLAGS Affected

Rflags diagram.

MXCSR Flags Affected

MXCSR diagram.

Exceptions

Exception summary table.

Figure 2-1. Typical Instruction Description

ADDPD VADDPD — Add Packed Double-Precision Floating-Point

Adds each packed double-precision floating-point value of the first source operand to the corresponding value of the second source operand and writes the result of each addition into the corresponding quadword of the destination.

There are legacy and extended forms of the instruction:

ADDPD

Adds two pairs of values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VADDPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Adds two pairs of values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Adds four pairs of values.

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ADDPD is an SSE2 instruction and VADDPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ADDPD *xmm1, xmm2/mem128*	66 0F 58 /r	Adds two packed double-precision floating-point values in *xmm1* to corresponding values in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VADDPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	58 /r
VADDPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	58 /r

Related Instructions

(V)ADDPS, (V)ADDSD, (V)ADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ADDPS VADDPS — Add Packed Single-Precision Floating-Point

Adds each packed single-precision floating-point value of the first source operand to the corresponding value of the second source operand and writes the result of each addition into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

ADDPS

Adds four pairs of values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VADDPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Adds four pairs of values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Adds eight pairs of values.

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ADDPS is an SSE2 instruction and VADDPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ADDPS *xmm1*, *xmm2/mem128*	0F 58 /r	Adds four packed single-precision floating-point values in *xmm1* to corresponding values in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VADDPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	58 /r
VADDPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	58 /r

Related Instructions

(V)ADDPD, (V)ADDSD, (V)ADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ADDSD VADDSD — Add Scalar Double-Precision Floating-Point

Adds the double-precision floating-point value in the low-order quadword of the first source operand to the corresponding value in the low-order quadword of the second source operand and writes the result into the low-order quadword of the destination.

There are legacy and extended forms of the instruction:

ADDSD

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The first source register is also the destination register. Bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are not affected.

VADDSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The destination is a third XMM register. Bits [127:64] of the first source operand are copied to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

ADDSD is an SSE2 instruction and VADDSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ADDSD *xmm1*, *xmm2/mem64*	F2 0F 58 /r	Adds low-order double-precision floating-point values in *xmm1* to corresponding values in *xmm2* or *mem64*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VADDSD *xmm1*, *xmm2*, *xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.11	58 /r

Related Instructions

(V)ADDPD, (V)ADDPS, (V)ADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ADDSS VADDSS — Add Scalar Single-Precision Floating-Point

Adds the single-precision floating-point value in the low-order doubleword of the first source operand to the corresponding value in the low-order doubleword of the second source operand and writes the result into the low-order doubleword of the destination.

There are legacy and extended forms of the instruction:

ADDSS

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The first source register is also the destination. Bits [127:32] of the destination register and bits [255:128] of the corresponding YMM register are not affected.

VADDSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The destination is a third XMM register. Bits [127:32] of the first source register are copied to bits [127:32] of the of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

ADDSS is an SSE instruction and VADDSS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ADDSS *xmm1*, *xmm2*/*mem32*	F3 0F 58 /r	Adds a single-precision floating-point value in the low-order doubleword of *xmm1* to a corresponding value in *xmm2* or *mem32*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VADDSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	58 /r

Related Instructions

(V)ADDPD, (V)ADDPS, (V)ADDSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ADDSUBPD VADDSUBPD — Alternating Addition and Subtraction Packed Double-Precision Floating-Point

Adds the odd-numbered packed double-precision floating-point values of the first source operand to the corresponding values of the second source operand and writes the sum to the corresponding odd-numbered element of the destination; subtracts the even-numbered packed double-precision floating-point values of the second source operand from the corresponding values of the first source operand and writes the differences to the corresponding even-numbered element of the destination.

There are legacy and extended forms of the instruction:

ADDSUBPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VADDSUBPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ADDSUBPD is an SSE2 instruction and VADDSUBPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ADDSUBPD *xmm1*, *xmm2/mem128*	66 0F D0 /r	Adds a value in the upper 64 bits of *xmm1* to the corresponding value in *xmm2* and writes the result to the upper 64 bits of *xmm1*; subtracts the value in the lower 64 bits of *xmm1* from the corresponding value in *xmm2* and writes the result to the lower 64 bits of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VADDSUBPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	D0 /r
VADDSUBPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	D0 /r

Related Instructions

(V)ADDSUBPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ADDSUBPS VADDSUBPS — Alternating Addition and Subtraction Packed Single-Precision Floating Point

Adds the second and fourth single-precision floating-point values of the first source operand to the corresponding values of the second source operand and writes the sums to the second and fourth elements of the destination. Subtracts the first and third single-precision floating-point values of the second source operand from the corresponding values of the first source operand and writes the differences to the first and third elements of the destination.

There are legacy and extended forms of the instruction:

ADDSUBPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VADDSUBPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ADDSUBPS is an SSE instruction and VADDSUBPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ADDSUBPS *xmm1*, *xmm2/mem128*	F2 0F D0 /r	Adds the second and fourth packed single-precision values in *xmm2* or *mem128* to the corresponding values in *xmm1* and writes results to the corresponding positions of *xmm1*. Subtracts the first and third packed single-precision values in *xmm2* or *mem128* from the corresponding values in *xmm1* and writes results to the corresponding positions of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VADDSUBPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.11	D0 /r
VADDSUBPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.11	D0 /r

Related Instructions

(V)ADDSUBPD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

AESDEC VAESDEC — AES Decryption Round

Performs a single round of AES decryption. Transforms a state value specified by the first source operand using a round key value specified by the second source operand, and writes the result to the destination.

See Section 1.4, “AES Instructions” for more information about the operation of the AES instructions.

Decryption consists of 1, …, $N_r - 1$ iterations of sequences of operations called rounds, terminated by a unique final round, N_r. The AESDEC and VAESDEC instructions perform all the rounds except the last; the AESDECLAST and VAESDECLAST instructions perform the final round.

The 128-bit state and round key vectors are interpreted as 16-byte column-major entries in a 4-by-4 matrix of bytes.The transformed state is written to the destination in column-major order. For both instructions, the destination register is the same as the first source register.

There are legacy and extended forms of the instruction:

AESDEC

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VAESDEC

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

AESDEC is an AES instruction and VAESDEC is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AES] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
AESDEC *xmm1*, *xmm2/mem128*	66 0F 38 DE /r	Performs one decryption round on a state value in *xmm1* using the key value in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VAESDEC *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	DE /r

Related Instructions

(V)AESENC, (V)AESENCLAST, (V)AESIMC, (V)AESKEYGENASSIST

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

AESDECLAST VAESDECLAST — AES Last Decryption Round

Performs the final round of AES decryption. Completes transformation of a state value specified by the first source operand using a round key value specified by the second source operand, and writes the result to the destination.

See Section 1.4, "AES Instructions" for more information about the operation of the AES instructions.

Decryption consists of 1, …, $N_r - 1$ iterations of sequences of operations called rounds, terminated by a unique final round, N_r.The AESDEC and VAESDEC instructions perform all the rounds before the final round; the AESDECLAST and VAESDECLAST instructions perform the final round.

The 128-bit state and round key vectors are interpreted as 16-byte column-major entries in a 4-by-4 matrix of bytes.The transformed state is written to the destination in column-major order. For both instructions, the destination register is the same as the first source register

AESDECLAST

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VAESDECLAST

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

AESDECLAST is an AES instruction and VAESDECLAST is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AES] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
AESDECLAST *xmm1*, *xmm2/mem128*	66 0F 38 DF/r	Performs the last decryption round on a state value in *xmm1* using the key value in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VAESDECLAST *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	DF /r

Related Instructions

(V)AESENC, (V)AESENCLAST, (V)AESIMC, (V)AESKEYGENASSIST

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

AESENC VAESENC — AES Encryption Round

Performs a single round of AES encryption. Transforms a state value specified by the first source operand using a round key value specified by the second source operand, and writes the result to the destination.

See Section 1.4, “AES Instructions” for more information about the operation of the AES instructions.

Encryption consists of 1, …, N_r – 1 iterations of sequences of operations called rounds, terminated by a unique final round, N_r. The AESENC and VAESENC instructions perform all the rounds before the final round; the AESENCLAST and VAESENCLAST instructions perform the final round.

The 128-bit state and round key vectors are interpreted as 16-byte column-major entries in a 4-by-4 matrix of bytes.The transformed state is written to the destination in column-major order. For both instructions, the destination register is the same as the first source register

There are legacy and extended forms of the instruction:

AESENC

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VAESENC

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

AESENC is an AES instruction and VAESENC is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AES] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
AESENC *xmm1*, *xmm2/mem128*	66 0F 38 DC /r	Performs one encryption round on a state value in *xmm1* using the key value in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VAESENC *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	DC /r

Related Instructions

(V)AESDEC, (V)AESDECLAST, (V)AESIMC, (V)AESKEYGENASSIST

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

AESENCLAST VAESENCLAST — AES Last Encryption Round

Performs the final round of AES encryption. Completes transformation of a state value specified by the first source operand using a round key value specified by the second source operand, and writes the result to the destination.

See Section 1.4, “AES Instructions” for more information about the operation of the AES instructions.

Encryption consists of 1, …, N_r – 1 iterations of sequences of operations called rounds, terminated by a unique final round, N_r. The AESENC and VAESENC instructions perform all the rounds before the final round; the AESENCLAST and VAESENCLAST instructions perform the final round.

The 128-bit state and round key vectors are interpreted as 16-byte column-major entries in a 4-by-4 matrix of bytes.The transformed state is written to the destination in column-major order. For both instructions, the destination register is the same as the first source register.

AESENCLAST

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VAESENCLAST

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

AESENCLAST is an AES instruction and VAESENCLAST is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AES] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
AESENCLAST *xmm1*, *xmm2/mem128*	66 0F 38 DD /r	Performs the last encryption round on a state value in *xmm1* using the key value in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VAESENCLAST *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	DD /r

Related Instructions

(V)AESDEC, (V)AESDECLAST, (V)AESIMC, (V)AESKEYGENASSIST

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

AESIMC VAESIMC — AES InvMixColumn Transformation

Applies the AES *InvMixColumns*( ) transformation to expanded round keys in preparation for decryption. Transforms an expanded key specified by the second source operand and writes the result to a destination register.

See Section 1.4, “AES Instructions” for more information about the operation of the AES instructions.

The 128-bit round key vector is interpreted as 16-byte column-major entries in a 4-by-4 matrix of bytes.The transformed result is written to the destination in column-major order.

AESIMC and VAESIMC are not used to transform the first and last round key in a decryption sequence.

There are two forms of these instructions:

AESIMC

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VAESIMC

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

AESIMC is an AES instruction and VAESIMC is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AES] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
AESIMC *xmm1*, *xmm2/mem128*	66 0F 38 DB /r	Performs AES InvMixColumn transformation on a round key in the *xmm2* or *mem128* and stores the result in *xmm1*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VAESIMC *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	DB /r

Related Instructions

(V)AESDEC, (V)AESDECLAST, (V)AESENC, (V)AESENCLAST, (V)AESKEYGENASSIST

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

AESKEYGENASSIST VAESKEYGENASSIST — AES Assist Round Key Generation

Expands a round key for encryption. Transforms a 128-bit round key operand using an 8-bit round constant and writes the result to a destination register.

See Section 1.4, “AES Instructions” for more information about the operation of the AES instructions.

The round key is provided by the second source operand and the round constant is specified by an immediate operand. The 128-bit round key vector is interpreted as 16-byte column-major entries in a 4-by-4 matrix of bytes. The transformed result is written to the destination in column-major order.

There are legacy and extended forms of the instruction:

AESKEYGENASSIST

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VAESKEYGENASSIST

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

AESKEYGENASSIST is an AES instruction and VAESKEYGENASSIST is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AES] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
AESKEYGENASSIST *xmm1*, *xmm2/mem128, imm8*	66 0F 3A DF /r ib	Expands a round key in *xmm2* or *mem128* using an immediate round constant. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
AESKEYGENASSIST *xmm1*, *xmm2 /mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	DF /r ib

Related Instructions

(V)AESDEC, (V)AESDECLAST, (V)AESENC, (V)AESENCLAST,(V)AESIMC

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

ANDNPD AND NOT
VANDNPD Packed Double-Precision Floating-Point

Performs a bitwise AND of two packed double-precision floating-point values in the second source operand with the ones'-complement of the two corresponding packed double-precision floating-point values in the first source operand and writes the result into the destination.

There are legacy and extended forms of the instruction:

ANDNPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VANDNPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ANDNPD is an SSE2 instruction and VANDNPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ANDNPD *xmm1*, *xmm2/mem128*	66 0F 55 /r	Performs bitwise AND of two packed double-precision floating-point values in *xmm2* or *mem128* with the ones'-complement of two packed double-precision floating-point values in *xmm1*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VANDNPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	55 /r
VANDNPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	55 /r

Related Instructions

(V)ANDNPS, (V)ANDPD, (V)ANDPS, (V)ORPD, (V)ORPS, (V)XORPD, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception A — AVX exception S — SSE exception				

ANDNPS VANDNPS — AND NOT Packed Single-Precision Floating-Point

Performs a bitwise AND of four packed single-precision floating-point values in the second source operand with the ones'-complement of the four corresponding packed single-precision floating-point values in the first source operand, and writes the result in the destination.

There are legacy and extended forms of the instruction:

ANDNPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VANDNPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ANDNPS is an SSE instruction and VANDNPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ANDNPS *xmm1*, *xmm2/mem128*	0F 55 /r	Performs bitwise AND of four packed double-precision floating-point values in *xmm2* or *mem128* with the ones'-complement of four packed double-precision floating-point values in *xmm1*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VANDNPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	55 /r
VANDNPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	55 /r

Related Instructions

(V)ANDNPD, (V)ANDPD, (V)ANDPS, (V)ORPD, (V)ORPS, (V)XORPD, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

ANDPD VANDPD — AND Packed Double-Precision Floating-Point

Performs bitwise AND of two packed double-precision floating-point values in the first source operand with the corresponding two packed double-precision floating-point values in the second source operand and writes the results into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

ANDPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VANDPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ANDPD is an SSE2 instruction and VANDPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ANDPD *xmm1*, *xmm2/mem128*	66 0F 54 /r	Performs bitwise AND of two packed double-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VANDPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	54 /r
VANDPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	54 /r

Related Instructions

(V)ANDNPD, (V)ANDNPS, (V)ANDPS, (V)ORPD, (V)ORPS, (V)XORPD, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

ANDPS / VANDPS — AND Packed Single-Precision Floating-Point

Performs bitwise AND of the four packed single-precision floating-point values in the first source operand with the corresponding four packed single-precision floating-point values in the second source operand, and writes the result into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

ANDPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VANDPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ANDPS is an SSE instruction and VANDPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ANDPS *xmm1*, *xmm2/mem128*	0F 54 /r	Performs bitwise AND of four packed double-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VANDPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	54 /r
VANDPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	54 /r

Related Instructions

(V)ANDNPD, (V)ANDNPS, (V)ANDPD, (V)ORPD, (V)ORPS, (V)XORPD, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

BLENDPD VBLENDPD Blend Packed Double-Precision Floating-Point

Copies packed double-precision floating-point values from either of two sources to a destination, as specified by an 8-bit mask operand.

Each mask bit specifies a 64-bit element in a source location and a corresponding 64-bit element in the destination register. When a mask bit = 0, the specified element of the first source is copied to the corresponding position in the destination register. When a mask bit = 1, the specified element of the second source is copied to the corresponding position in the destination register.

There are legacy and extended forms of the instruction:

BLENDPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. Only mask bits [1:0] are used.

VBLENDPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared. Only mask bits [1:0] are used.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register. Only mask bits [3:0] are used.

BLENDPD is an SSE4.1 instruction and VBLENDPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
BLENDPD *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 0D /r ib	Copies values from *xmm1* or *xmm2*/*mem128* to *xmm1*, as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VBLENDPD *xmm1*, *xmm2*, *xmm3/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	0D /r ib
VBLENDPD *ymm1, ymm2, ymm3/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.1.01	0D /r ib

Related Instructions

(V)BLENDPS, (B)BLENDVPD, (V)BLENDVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

BLENDPS VBLENDPS — Blend Packed Single-Precision Floating-Point

Copies packed single-precision floating-point values from either of two sources to a destination, as specified by an 8-bit mask operand.

Each mask bit specifies a 32-bit element in a source location and a corresponding 32-bit element in the destination register. When a mask bit = 0, the specified element of the first source is copied to the corresponding position in the destination register. When a mask bit = 1, the specified element of the second source is copied to the corresponding position in the destination register.

There are legacy and extended forms of the instruction:

BLENDPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. Only mask bits [3:0] are used.

VBLENDPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.Only mask bits [3:0] are used.

YMM Encoding

The first operand is a YMM register and the second operand is either another YMM register or a 256-bit memory location. The destination is a third YMM register. Only mask bits [3:0] are used.

BLENDPS is an SSE4.1 instruction and VBLENDPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
BLENDPS *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 0C /r ib	Copies values from *xmm1* or *xmm2*/*mem128* to *xmm1*, as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VBLENDPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	0C /r ib
VBLENDPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.1.01	0C /r ib

Related Instructions

(V)BLENDPD, (V)BLENDVPD, (V)BLENDVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

BLENDVPD VBLENDVPD — Variable Blend Packed Double-Precision Floating-Point

Copies packed double-precision floating-point values from either of two sources to a destination, as specified by a mask operand.

Each mask bit specifies a 64-bit element of a source location and a corresponding 64-bit element of the destination. The position of a mask bit corresponds to the position of the most significant bit of a copied value. When a mask bit = 0, the specified element of the first source is copied to the corresponding position in the destination. When a mask bit = 1, the specified element of the second source is copied to the corresponding position in the destination.

There are legacy and extended forms of the instruction:

BLENDVPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. The mask is defined by bits 127 and 63 of the implicit register XMM0.

VBLENDVPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared. The mask is defined by bits 127 and 63 of a fourth XMM register.

YMM Encoding

The first operand is a YMM register and the second operand is either another YMM register or a 256-bit memory location. The destination is a third YMM register. The mask is defined by bits 255, 191, 127, and 63 of a fourth YMM register.

BLENDVPD is an SSE4.1 instruction and VBLENDVPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
BLENDVPD *xmm1*, *xmm2/mem128*	66 0F 38 15 /r	Copies values from *xmm1* or *xmm2/mem128* to *xmm1*, as specified by the MSB of corresponding elements of xmm0.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VBLENDVPD *xmm1*, *xmm2*, *xmm3/mem128*, *xmm4*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	4B /r
VBLENDVPD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.1.01	4B /r

Related Instructions

(V)BLENDPD, (V)BLENDPS, (V)BLENDVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

BLENDVPS VBLENDVPS — Variable Blend Packed Single-Precision Floating-Point

Copies packed single-precision floating-point values from either of two sources to a destination, as specified by a mask operand.

Each mask bit specifies a 32-bit element of a source location and a corresponding 32-bit element of the destination register. The position of a mask bits corresponds to the position of the most significant bit of a copied value. When a mask bit = 0, the specified element of the first source is copied to the corresponding position in the destination. When a mask bit = 1, the specified element of the second source is copied to the corresponding position in the destination.

There are legacy and extended forms of the instruction:

BLENDVPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. The mask is defined by bits 127, 95, 63, and 31 of the implicit register XMM0.

VBLENDVPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared. The mask is defined by bits 127, 95, 63, and 31 of a fourth XMM register.

YMM Encoding

The first operand is a YMM register and the second operand is either another YMM register or a 256-bit memory location. The destination is a third YMM register. The mask is defined by bits 255, 223, 191, 159, 127, 95, 63, and 31 of a fourth YMM register.

BLENDVPS is an SSE4.1 instruction and VBLENDVPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
BLENDVPS *xmm1*, *xmm2/mem128*	66 0F 38 14 /r	Copies packed single-precision floating-point values from *xmm1* or *xmm2/mem128* to *xmm1*, as specified by bits in xmm0.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VBLENDVPS *xmm1*, *xmm2*, *xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	4A /r
VBLENDVPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.1.01	4A /r

Related Instructions

(V)BLENDPD, (V)BLENDPS, (V)BLENDVPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CMPPD VCMPPD — Compare Packed Double-Precision Floating-Point

Compares each of the two packed double-precision floating-point values of the first source operand to the corresponding values of the second source operand and writes the result of each comparison to the corresponding 64-bit element of the destination. When a comparison is TRUE, all 64 bits of the destination element are set; when a comparison is FALSE, all 64 bits of the destination element are cleared. The type of comparison is specified by an immediate byte operand.

Signed comparisons return TRUE only when both operands are valid numbers and the numbers have the relation specified by the type of comparison operation. Ordered comparison returns TRUE when both operands are valid numbers, or FALSE when either operand is a NaN. Unordered comparison returns TRUE only when one or both operands are NaN and FALSE otherwise.

QNaN operands generate an Invalid Operation Exception (IE) only if the comparison type isn't Equal, Unequal, Ordered, or Unordered. SNaN operands always generate an IE.

There are legacy and extended forms of the instruction:

CMPPD

The first source operand is an XMM register and the second source operand is either another XMM register or a128-bit memory location.The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. Comparison type is specified by bits [2:0] of an immediate byte operand.

VCMPPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared. Comparison type is specified by bits [4:0] of an immediate byte operand.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination operand is a YMM register. Comparison type is specified by bits [4:0] of an immediate byte operand.

Immediate Operand Encoding

CMPPD uses bits [2:0] of the 8-bit immediate operand and VCMPPD uses bits [4:0] of the 8-bit immediate operand. Although VCMPPD supports 20h encoding values, the comparison types echo those of CMPPD on 4-bit boundaries. The following table shows the immediate operand value for CMPPD and each of the VCMPPD echoes.

Some comparison operations that are not directly supported by immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and executing the appropriate comparison of the swapped values. These additional comparison operations are shown with the directly supported comparison operations.

Immediate Operand Value	Compare Operation	Result If NaN Operand	QNaN Operand Causes Invalid Operation Exception
00h, 08h, 10h, 18h	Equal	FALSE	No
01h, 09h, 11h, 19h	Less than	FALSE	Yes
	Greater than (swapped operands)	FALSE	Yes
02h, 0Ah, 12h, 1Ah	Less than or equal	FALSE	Yes
	Greater than or equal (swapped operands)	FALSE	Yes
03h, 0Bh, 13h, 1Bh	Unordered	TRUE	No
04h, 0Ch, 14h, 1Ch	Not equal	TRUE	No
05h, 0Dh, 15h, 1Dh	Not less than	TRUE	Yes
	Not greater than (swapped operands)	TRUE	Yes
06h, 0Eh, 16h, 1Eh	Not less than or equal	TRUE	Yes
	Not greater than or equal (swapped operands)	TRUE	Yes
07h, 0Fh, 17h, 1Fh	Ordered	FALSE	No

The following alias mnemonics for (V)CMPPD with appropriate value of *imm8* are supported.

Mnemonic	Implied Value of *imm8*
(V)CMPEQPD	00h, 08h, 10h, 18h
(V)CMPLTPD	01h, 09h, 11h, 19h
(V)CMPLEPD	02h, 0Ah, 12h, 1Ah
(V)CMPUNORDPD	03h, 0Bh, 13h, 1Bh
(V)CMPNEQPD	04h, 0Ch, 14h, 1Ch
(V)CMPNLTPD	05h, 0Dh, 15h, 1Dh
(V)CMPNLEPD	06h, 0Eh, 16h, 1Eh
(V)CMPORDPD	07h, 0Fh, 17h, 1Fh

CMPPD is an SSE2 instruction and VCMPPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CMPPD *xmm1*, *xmm2/mem128*, *imm8*	66 0F C2 /r ib	Compares two pairs of values in *xmm1* to corresponding values in *xmm2* or *mem128*. Comparison type is determined by *imm8*. Writes comparison results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCMPPD *xmm1*, *xmm2*, *xmm3/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	C2 /r ib
VCMPPD *ymm1, ymm2, ymm3/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	C2 /r ib

Related Instructions

(V)CMPPS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)UCOMISD, (V)UCOMISS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CMPPS VCMPPS Compare Packed Single-Precision Floating-Point

Compares each of the four packed single-precision floating-point values of the first source operand to the corresponding values of the second source operand and writes the result of each comparison to the corresponding 32-bit element of the destination. When a comparison is TRUE, all 32 bits of the destination element are set; when a comparison is FALSE, all 32 bits of the destination element are cleared. The type of comparison is specified by an immediate byte operand.

Signed comparisons return TRUE only when both operands are valid numbers and the numbers have the relation specified by the type of comparison operation. Ordered comparison returns TRUE when both operands are valid numbers, or FALSE when either operand is a NaN. Unordered comparison returns TRUE only when one or both operands are NaN and FALSE otherwise.

QNaN operands generate an Invalid Operation Exception (IE) only if the comparison type isn't Equal, Unequal, Ordered, or Unordered. SNaN operands always generate an IE.

There are legacy and extended forms of the instruction:

CMPPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. Comparison type is specified by bits [2:0] of an immediate byte operand.

VCMPPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared. Comparison type is specified by bits [4:0] of an immediate byte operand.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination operand is a YMM register. Comparison type is specified by bits [4:0] of an immediate byte operand.

Immediate Operand Encoding

CMPPS uses bits [2:0] of the 8-bit immediate operand and VCMPPS uses bits [4:0] of the 8-bit immediate operand. Although VCMPPS supports 20h encoding values, the comparison types echo those of CMPPS on 4-bit boundaries. The following table shows the immediate operand value for CMPPS and each of the VCMPPDS echoes.

Some comparison operations that are not directly supported by immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and executing the appropriate comparison of the swapped values. These additional comparison operations are shown in with the directly supported comparison operations.

Immediate Operand Value	Compare Operation	Result If NaN Operand	QNaN Operand Causes Invalid Operation Exception
00h, 08h, 10h, 18h	Equal	FALSE	No
01h, 09h, 11h, 19h	Less than	FALSE	Yes
	Greater than (swapped operands)	FALSE	Yes
02h, 0Ah, 12h, 1Ah	Less than or equal	FALSE	Yes
	Greater than or equal (swapped operands)	FALSE	Yes
03h, 0Bh, 13h, 1Bh	Unordered	TRUE	No
04h, 0Ch, 14h, 1Ch	Not equal	TRUE	No
05h, 0Dh, 15h, 1Dh	Not less than	TRUE	Yes
	Not greater than (swapped operands)	TRUE	Yes
06h, 0Eh, 16h, 1Eh	Not less than or equal	TRUE	Yes
	Not greater than or equal (swapped operands)	TRUE	Yes
07h, 0Fh, 17h, 1Fh	Ordered	FALSE	No

The following alias mnemonics for (V)CMPPS with appropriate value of *imm8* are supported.

Mnemonic	Implied Value of *imm8*
(V)CMPEQPS	00h, 08h, 10h, 18h
(V)CMPLTPS	01h, 09h, 11h, 19h
(V)CMPLEPS	02h, 0Ah, 12h, 1Ah
(V)CMPUNORDPS	03h, 0Bh, 13h, 1Bh
(V)CMPNEQPS	04h, 0Ch, 14h, 1Ch
(V)CMPNLTPS	05h, 0Dh, 15h, 1Dh
(V)CMPNLEPS	06h, 0Eh, 16h, 1Eh
(V)CMPORDPS	07h, 0Fh, 17h, 1Fh

CMPPS is an SSE instruction and VCMPPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CMPPS *xmm1*, *xmm2/mem128*, *imm8*	0F C2 /r ib	Compares four pairs of values in *xmm1* to corresponding values in *xmm2* or *mem128*. Comparison type is determined by *imm8*. Writes comparison results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCMPPS *xmm1*, *xmm2*, *xmm3/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	C2 /r ib

Related Instructions

(V)CMPPD, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)UCOMISD, (V)UCOMISS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CMPSD VCMPSD — Compare Scalar Double-Precision Floating-Point

Compares a double-precision floating-point value in the low-order 64 bits of the first source operand with a double-precision floating-point value in the low-order 64 bits of the second source operand and writes the result to the low-order 64 bits of the destination. When a comparison is TRUE, all 64 bits of the destination element are set; when a comparison is FALSE, all 64 bits of the destination element are cleared. Comparison type is specified by an immediate byte operand.

Signed comparisons return TRUE only when both operands are valid numbers and the numbers have the relation specified by the type of comparison operation. Ordered comparison returns TRUE when both operands are valid numbers, or FALSE when either operand is a NaN. Unordered comparison returns TRUE only when one or both operands are NaN and FALSE otherwise.

QNaN operands generate an Invalid Operation Exception (IE) only when the comparison type is not Equal, Unequal, Ordered, or Unordered. SNaN operands always generate an IE.

There are legacy and extended forms of the instruction:

CMPSD

The first source operand is an XMM register. The second source operand is either another XMM register or a 64-bit memory location. The first source register is also the destination. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected. Comparison type is specified by bits [2:0] of an immediate byte operand.

This CMPSD instruction must not be confused with the same-mnemonic CMPSD (compare strings by doubleword) instruction in the general-purpose instruction set. Assemblers can distinguish the instructions by the number and type of operands.

VCMPSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 64-bit memory location. The destination is a third XMM register. Bits [127:64] of the destination are copied from bits [127:64] of the first source. Bits [255:128] of the YMM register that corresponds to the destination are cleared. Comparison type is specified by bits [4:0] of an immediate byte operand.

Immediate Operand Encoding

CMPSD uses bits [2:0] of the 8-bit immediate operand and VCMPSD uses bits [4:0] of the 8-bit immediate operand. Although VCMPSD supports 20h encoding values, the comparison types echo those of CMPSD on 4-bit boundaries. The following table shows the immediate operand value for CMPSD and each of the VCMPSD echoes.

Some comparison operations that are not directly supported by immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and executing the appropriate comparison of the swapped values. These additional comparison operations are shown with the directly supported comparison operations. When operands are swapped, the first source XMM register is overwritten by the result

Immediate Operand Value	Compare Operation	Result If NaN Operand	QNaN Operand Causes Invalid Operation Exception
00h, 08h, 10h, 18h	Equal	FALSE	No
01h, 09h, 11h, 19h	Less than	FALSE	Yes
	Greater than (swapped operands)	FALSE	Yes
02h, 0Ah, 12h, 1Ah	Less than or equal	FALSE	Yes
	Greater than or equal (swapped operands)	FALSE	Yes
03h, 0Bh, 13h, 1Bh	Unordered	TRUE	No
04h, 0Ch, 14h, 1Ch	Not equal	TRUE	No
05h, 0Dh, 15h, 1Dh	Not less than	TRUE	Yes
	Not greater than (swapped operands)	TRUE	Yes
06h, 0Eh, 16h, 1Eh	Not less than or equal	TRUE	Yes
	Not greater than or equal (swapped operands)	TRUE	Yes
07h, 0Fh, 17h, 1Fh	Ordered	FALSE	No

The following alias mnemonics for (V)CMPSD with appropriate value of *imm8* are supported.

Mnemonic	Implied Value of *imm8*
(V)CMPEQSD	00h, 08h, 10h, 18h
(V)CMPLTSD	01h, 09h, 11h, 19h
(V)CMPLESD	02h, 0Ah, 12h, 1Ah
(V)CMPUNORDSD	03h, 0Bh, 13h, 1Bh
(V)CMPNEQSD	04h, 0Ch, 14h, 1Ch
(V)CMPNLTSD	05h, 0Dh, 15h, 1Dh
(V)CMPNLESD	06h, 0Eh, 16h, 1Eh
(V)CMPORDSD	07h, 0Fh, 17h, 1Fh

CMPSD is an SSE2 instruction and VCMPSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CMPSD *xmm1*, *xmm2/mem64*, *imm8*	F2 0F C2 /r ib	Compares double-precision floating-point values in the low-order 64 bits of *xmm1* with corresponding values in *xmm2* or *mem64*. Comparison type is determined by *imm8*. Writes comparison results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCMPSD *xmm1*, *xmm2*, *xmm3/mem64*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	C2 /r ib

Related Instructions

(V)CMPPD, (V)CMPPS, (V)CMPSS, (V)COMISD, (V)COMISS, (V)UCOMISD, (V)UCOMISS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CMPSS VCMPSS — Compare Scalar Single-Precision Floating-Point

Compares a single-precision floating-point value in the low-order 32 bits of the first source operand with a single-precision floating-point value in the low-order 32 bits of the second source operand and writes the result to the low-order 32 bits of the destination. When a comparison is TRUE, all 32 bits of the destination element are set; when a comparison is FALSE, all 32 bits of the destination element are cleared. Comparison type is specified by an immediate byte operand.

Signed comparisons return TRUE only when both operands are valid numbers and the numbers have the relation specified by the type of comparison operation. Ordered comparison returns TRUE when both operands are valid numbers, or FALSE when either operand is a NaN. Unordered comparison returns TRUE only when one or both operands are NaN and FALSE otherwise.

QNaN operands generate an Invalid Operation Exception (IE) only if the comparison type isn't Equal, Unequal, Ordered, or Unordered. SNaN operands always generate an IE.

There are legacy and extended forms of the instruction:

CMPSS

The first source operand is an XMM register. The second source operand is either another XMM register or a 32-bit memory location. The first source register is also the destination. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected. Comparison type is specified by bits [2:0] of an immediate byte operand.

VCMPSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 32-bit memory location. The destination is a third XMM register. Bits [127:32] of the destination are copied from bits [127L32] of the first source. Bits [255:128] of the YMM register that corresponds to the destination are cleared. Comparison type is specified by bits [4:0] of an immediate byte operand.

Immediate Operand Encoding

CMPSS uses bits [2:0] of the 8-bit immediate operand and VCMPSS uses bits [4:0] of the 8-bit immediate operand. Although VCMPSS supports 20h encoding values, the comparison types echo those of CMPSS on 4-bit boundaries. The following table shows the immediate operand value for CMPSS and each of the VCMPSS echoes.

Some comparison operations that are not directly supported by immediate-byte encodings can be implemented by swapping the contents of the source and destination operands and executing the appropriate comparison of the swapped values. These additional comparison operations are shown below with the directly supported comparison operations. When operands are swapped, the first source XMM register is overwritten by the result.

Immediate Operand Value	Compare Operation	Result If NaN Operand	QNaN Operand Causes Invalid Operation Exception
00h, 08h, 10h, 18h	Equal	FALSE	No
01h, 09h, 11h, 19h	Less than	FALSE	Yes
	Greater than (swapped operands)	FALSE	Yes
02h, 0Ah, 12h, 1Ah	Less than or equal	FALSE	Yes
	Greater than or equal (swapped operands)	FALSE	Yes
03h, 0Bh, 13h, 1Bh	Unordered	TRUE	No
04h, 0Ch, 14h, 1Ch	Not equal	TRUE	No
05h, 0Dh, 15h, 1Dh	Not less than	TRUE	Yes
	Not greater than (swapped operands)	TRUE	Yes
06h, 0Eh, 16h, 1Eh	Not less than or equal	TRUE	Yes
	Not greater than or equal (swapped operands)	TRUE	Yes
07h, 0Fh, 17h, 1Fh	Ordered	FALSE	No

The following alias mnemonics for (V)CMPSS with appropriate value of *imm8* are supported.

Mnemonic	Implied Value of *imm8*
(V)CMPEQSS	00h, 08h, 10h, 18h
(V)CMPLTSS	01h, 09h, 11h, 19h
(V)CMPLESS	02h, 0Ah, 12h, 1Ah
(V)CMPUNORDSS	03h, 0Bh, 13h, 1Bh
(V)CMPNEQSS	04h, 0Ch, 14h, 1Ch
(V)CMPNLTSS	05h, 0Dh, 15h, 1Dh
(V)CMPNLESS	06h, 0Eh, 16h, 1Eh
(V)CMPORDSS	07h, 0Fh, 17h, 1Fh

CMPSS is an SSE instruction and VCMPSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CMPSS *xmm1*, *xmm2*/*mem32*, *imm8*	F3 0F C2 /r ib	Compares single-precision floating-point values in the low-order 32 bits of *xmm1* with corresponding values in *xmm2* or *mem32*. Comparison type is determined by *imm8*. Writes comparison results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCMPSS *xmm1*, *xmm2*, *xmm3/mem32*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	C2 /r ib

Related Instructions

(V)CMPPD, (V)CMPPS, (V)CMPSD, (V)COMISD, (V)COMISS, (V)UCOMISD, (V)UCOMISS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

COMISD VCOMISD — Compare Ordered Scalar Double-Precision Floating-Point

Compares a double-precision floating-point value in the low-order 64 bits of an operand with a double-precision floating-point value in the low-order 64 bits of another operand or a 64-bit memory location and sets rFLAGS.ZF, PF, and CF to show the result of the comparison:

Comparison	ZF	PF	CF
NaN input	1	1	1
operand 1 > operand 2	0	0	0
operand 1 < operand 2	0	0	1
operand 1 = operand 2	1	0	0

The result is unordered if one or both of the operand values is a NaN. The rFLAGS.OF, AF, and SF bits are cleared. If an #XF SIMD floating-point exception occurs the rFLAGS bits are not updated.

There are legacy and extended forms of the instruction:

COMISD

The first source operand is an XMM register and the second source operand is another XMM register or a 64-bit memory location.

VCOMISD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location.

COMISD is an SSE2 instruction and VCOMISD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
COMISD *xmm1*, *xmm2/mem64*	66 0F 2F /r	Compares double-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem64* and sets rFLAGS.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCOMISD *xmm1*, *xmm2* /*mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.01	2F /r

Related Instructions

(V)CMPPD, (V)CMPPS, (V)CMPSD, (V)CMPSS, (V)COMISS, (V)UCOMISD, (V)UCOMISS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	M	0	M	M
21	20	19	18	17	16	14	13 : 12	11	10	9	8	7	6	4	2	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected. Bits 31:22, 15, 5, 3, and 1 are reserved. For #XF, rFLAGS bits are not updated.*

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

COMISS Compare
VCOMISS Ordered Scalar Single-Precision Floating-Point

Compares a double-precision floating-point value in the low-order 32 bits of an operand with a double-precision floating-point value in the low-order 32 bits of another operand or a 32-bit memory location and sets rFLAGS.ZF, PF, and CF to show the result of the comparison:

Comparison	ZF	PF	CF
NaN input	1	1	1
operand 1 > operand 2	0	0	0
operand 1 < operand 2	0	0	1
operand 1 = operand 2	1	0	0

The result is unordered if one or both of the operand values is a NaN. The rFLAGS.OF, AF, and SF bits are cleared. If an #XF SIMD floating-point exception occurs the rFLAGS bits are not updated.

There are legacy and extended forms of the instruction:

COMISS

The first source operand is an XMM register and the second source operand is another XMM register or a 32-bit memory location.

VCOMISS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location.

COMISS is an SSE instruction and VCOMISS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
COMISS *xmm1*, *xmm2*/*mem32*	0F 2F /r	Compares single-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem32* and sets rFLAGS.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCOMISS *xmm1*, *xmm2* /*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.00	2F /r

Related Instructions

(V)CMPPD, (V)CMPPS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)UCOMISD, (V)UCOMISS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	M	0	M	M
21	20	19	18	17	16	14	13 : 12	11	10	9	8	7	6	4	2	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected. Bits 31:22, 15, 5, 3, and 1 are reserved. For #XF, rFLAGS bits are not updated.*

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTDQ2PD VCVTDQ2PD — Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point

Converts packed 32-bit signed integer values to packed double-precision floating-point values and writes the converted values to the destination.

There are legacy and extended forms of the instruction:

CVTDQ2PD

Converts two packed 32-bit signed integer values in the low-order 64 bits of an XMM register or in a 64-bit memory location to two packed double-precision floating-point values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTDQ2PD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts two packed 32-bit signed integer values in the low-order 64 bits of an XMM register or in a 64-bit memory location to two packed double-precision floating-point values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts four packed 32-bit signed integer values in the low-order 128 bits of a YMM register or a 256-bit memory location to four packed double-precision floating-point values and writes the converted values to a YMM register.

CVTDQ2PD is an SSE2 instruction and VCVTDQ2PD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTDQ2PD *xmm1*, *xmm2/mem64*	F3 0F E6 /r	Converts packed doubleword signed integers in *xmm2* or *mem64* to double-precision floating-point values in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTDQ2PD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	E6 /r
VCVTDQ2PD *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.10	E6 /r

Related Instructions

(V)CVTPD2DQ, (V)CVTPI2PD, (V)CVTSD2SI, (V)CVTSI2SD, (V)CVTTPD2DQ, (V)CVTTSD2SI

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTDQ2PS VCVTDQ2PS Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point

Converts packed 32-bit signed integer values to packed single-precision floating-point values and writes the converted values to the destination. When the result is an inexact value, it is rounded as specified by MXCSR.RC.

There are legacy and extended forms of the instruction:

CVTDQ2PS

Converts four packed 32-bit signed integer values in an XMM register or a 128-bit memory location to four packed single-precision floating-point values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTDQ2PS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts four packed 32-bit signed integer values in an XMM register or a 128-bit memory location to four packed double-precision floating-point values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts eight packed 32-bit signed integer values in a YMM register or a 256-bit memory location to eight packed double-precision floating-point values and writes the converted values to a YMM register.

The CVTDQ2PS is an SSE2 instruction and the VCVTDQ2PS instruction is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTDQ2PS *xmm1*, *xmm2/mem128*	0F 5B /r	Converts packed doubleword integer values in *xmm2* or *mem128* to packed single-precision floating-point values in *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTDQ2PS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	5B /r
VCVTDQ2PS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	5B /r

Related Instructions

(V)CVTPS2DQ, (V)CVTSI2SS, (V)CVTSS2SI, (V)CVTTPS2DQ, (V)CVTTSS2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTPD2DQ VCVTPD2DQ Convert Packed Double-Precision Floating-Point to Packed Doubleword Integer

Converts packed double-precision floating-point values to packed signed doubleword integers and writes the converted values to the destination.

When the result is an inexact value, it is rounded as specified by MXCSR.RC. When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTPD2DQ

Converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed signed doubleword integers and writes the converted values to the two low-order doublewords of the destination XMM register. Bits [127:64] of the destination are cleared. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTPD2DQ

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two signed doubleword values and writes the converted values to the lower two doubleword elements of the destination XMM register. Bits [127:64] of the destination are cleared. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts four packed double-precision floating-point values in a YMM register or a 256-bit memory location to four signed doubleword values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTPD2DQ is an SSE2 instruction and VCVTPD2DQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481)

Mnemonic	Opcode	Description
CVTPD2DQ *xmm1*, *xmm2/mem128*	F2 0F E6 /r	Converts two packed double-precision floating-point values in *xmm2* or *mem128* to packed doubleword integers in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTPD2DQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.11	E6 /r
VCVTPD2DQ *xmm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.11	E6 /r

Related Instructions

(V)CVTDQ2PD, (V)CVTPI2PD, (V)CVTSD2SI, (V)CVTSI2SD, (V)CVTTPD2DQ, (V)CVTTSD2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTPD2PS VCVTPD2PS Convert Packed Double-Precision Floating-Point to Packed Single-Precision Floating-Point

Converts packed double-precision floating-point values to packed single-precision floating-point values and writes the converted values to the low-order doubleword elements of the destination. When the result is an inexact value, it is rounded as specified by MXCSR.RC.

There are legacy and extended forms of the instruction:

CVTPD2PS

Converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed single-precision floating-point values and writes the converted values to an XMM register. Bits [127:64] of the destination are cleared. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTPD2PS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed single-precision floating-point values and writes the converted values to an XMM register. Bits [127:64] of the destination are cleared. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts four packed double-precision floating-point values in a YMM register or a 256-bit memory location to four packed single-precision floating-point values and writes the converted values to a YMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTPD2PS is an SSE2 instruction and VCVTPD2PS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTPD2PS *xmm1*, *xmm2/mem128*	66 0F 5A /r	Converts packed double-precision floating-point values in *xmm2* or *mem128* to packed single-precision floating-point values in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTPD2PS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	5A /r
VCVTPD2PS *xmm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	5A /r

Related Instructions

(V)CVTPS2PD, (V)CVTSD2SS, (V)CVTSS2SD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTPS2DQ VCVTPS2DQ — Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers

Converts packed single-precision floating-point values to packed signed doubleword integer values and writes the converted values to the destination.

When the result is an inexact value, it is rounded as specified by MXCSR.RC. When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTPS2DQ

Converts four packed single-precision floating-point values in an XMM register or a 128-bit memory location to four packed signed doubleword integer values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTPS2DQ

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts four packed single-precision floating-point values in an XMM register or a 128-bit memory location to four packed signed doubleword integer values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts eight packed single-precision floating-point values in a YMM register or a 256-bit memory location to eight packed signed doubleword integer values and writes the converted values to a YMM register.

CVTPS2DQ is an SSE2 instruction and VCVTPS2DQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTPS2DQ *xmm1*, *xmm2/mem128*	66 0F 5B /r	Converts four packed single-precision floating-point values in *xmm2* or *mem128* to four packed doubleword integers in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTPS2DQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	5B /r
VCVTPS2DQ *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	5B /r

Related Instructions

(V)CVTDQ2PS, (V)CVTSI2SS, (V)CVTSS2SI, (V)CVTTPS2DQ, (V)CVTTSS2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTPS2PD VCVTPS2PD — Convert Packed Single-Precision Floating-Point to Packed Double-Precision Floating-Point

Converts packed single-precision floating-point values to packed double-precision floating-point values and writes the converted values to the destination.

There are legacy and extended forms of the instruction:

CVTPS2PD

Converts two packed single-precision floating-point values in the two low order doubleword elements of an XMM register or a 64-bit memory location to two double-precision floating-point values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTPS2PD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts two packed single-precision floating-point values in the two low order doubleword elements of an XMM register or a 64-bit memory location to two double-precision floating-point values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts four packed single-precision floating-point values in a YMM register or a 128-bit memory location to four double-precision floating-point values and writes the converted values to a YMM register.

CVTPS2PD is an SSE2 instruction and VCVTPS2PD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTPS2PD *xmm1*, *xmm2/mem64*	0F 5A /r	Converts packed single-precision floating-point values in *xmm2* or *mem64* to packed double-precision floating-point values in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTPS2PD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	5A /r
VCVTPS2PD *ymm1*, *ymm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	5A /r

Related Instructions

(V)CVTPD2PS, (V)CVTSD2SS, (V)CVTSS2SD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTSD2SI VCVTSD2SI — Convert Scalar Double-Precision Floating-Point to Signed Doubleword or Quadword Integer

Converts a scalar double-precision floating-point value to a 32-bit or 64-bit signed integer value and writes the converted value to a general-purpose register.

When the result is an inexact value, it is rounded as specified by MXCSR.RC. When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$) or quadword value (-2^{63} to $+2^{63} - 1$), the instruction returns the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTSD2SI

- When REX.W = 0, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 32-bit signed integer and writes the converted value to a 32-bit general purpose register.
- When REX.W = 1, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 64-bit sign-extended integer and writes the converted value to a 64-bit general purpose register.

VCVTSD2SI

The extended form of the instruction has 128-bit encoding.

- When VEX.W = 0, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 32-bit signed integer and writes the converted value to a 32-bit general purpose register.
- When VEX.W = 1, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 64-bit sign-extended integer and writes the converted value to a 64-bit general purpose register.

CVTSD2SI is an SSE2 instruction and VCVTSD2SI is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTSD2SI *reg32*, *xmm1*/*mem64*	F2 (W0) 0F 2D /r	Converts a packed double-precision floating-point value in *xmm1* or *mem64* to a doubleword integer in *reg32*.
CVTSD2SI *reg64*, *xmm1*/*mem64*	F2 (W1) 0F 2D /r	Converts a packed double-precision floating-point value in *xmm1* or *mem64* to a quadword integer in *reg64*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTSD2SI *reg32*, *xmm2*/*mem64*	C4	$\overline{\text{RXB}}$.00001	0.1111.X.11	2D /r
VCVTSD2SI *reg64*, *xmm2*/*mem64*	C4	$\overline{\text{RXB}}$.00001	1.1111.X.11	2D /r

Related Instructions

(V)CVTDQ2PD, (V)CVTPD2DQ, (V)CVTPI2PD, (V)CVTSI2SD, (V)CVTTPD2DQ, (V)CVTTSD2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTSD2SS VCVTSD2SS Convert Scalar Double-Precision Floating-Point to Scalar Single-Precision Floating-Point

Converts a scalar double-precision floating-point value to a scalar single-precision floating-point value and writes the converted value to the low-order 32 bits of the destination. When the result is an inexact value, it is rounded as specified by MXCSR.RC.

There are legacy and extended forms of the instruction:

CVTSD2SS

Converts a scalar double-precision floating-point value in the low-order 64 bits of a source XMM register or a 64-bit memory location to a scalar single-precision floating-point value and writes the converted value to the low-order 32 bits of a destination XMM register. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTSD2SS

The extended form of the instruction has 128-bit encoding.

Converts a scalar double-precision floating-point value in the low-order 64 bits of a source XMM register or a 64-bit memory location to a scalar single-precision floating-point value and writes the converted value to the low-order 64 bits of a destination XMM register. Bits [127:32] of the destination are copied from the first source XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTSD2SS is an SSE2 instruction and VCVTSD2SS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTSD2SS *xmm1*, *xmm2/mem64*	F2 0F 5A /r	Converts a scalar double-precision floating-point value in *xmm2* or *mem64* to a scalar single-precision floating-point value in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTSD2SS *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	5A /r

Related Instructions

(V)CVTPD2PS, (V)CVTPS2PD, (V)CVTSS2SD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTSI2SD VCVTSI2SD Convert Signed Doubleword or Quadword Integer to Scalar Double-Precision Floating-Point

Converts a signed integer value to a double-precision floating-point value and writes the converted value to a destination register. When the result of the conversion is an inexact value, the value is rounded as specified by MXCSR.RC.

There are legacy and extended forms of the instruction:

CVTSI2SD

- When REX.W = 0, converts a signed doubleword integer value from a 32-bit source general-purpose register or a 32-bit memory location to a double-precision floating-point value and writes the converted value to the low-order 64 bits of an XMM register. Bits [127:64] of the destination XMM register and bits [255:128] of the corresponding YMM register are not affected.
- When REX.W = 1, converts a a signed quadword integer value from a 64-bit source general-purpose register or a 64-bit memory location to a 64-bit double-precision floating-point value and writes the converted value to the low-order 64 bits of an XMM register. Bits [127:64] of the destination XMM register and bits [255:128] of the corresponding YMM register are not affected.

VCVTSI2SD

The extended form of the instruction has 128-bit encoding.

- When VEX.W = 0, converts a signed doubleword integer value from a 32-bit source general-purpose register or a 32-bit memory location to a double-precision floating-point value and writes the converted value to the low-order 64 bits of the destination XMM register. Bits [127:64] of the first source XMM register are copied to the destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- When VEX.W = 1, converts a signed quadword integer value from a 64-bit source general-purpose register or a 64-bit memory location to a double-precision floating-point value and writes the converted value to the low-order 64 bits of the destination XMM register. Bits [127:64] of the first source XMM register are copied to the destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTSI2SD is an SSE2 instruction and VCVTSI2SD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTSI2SD *xmm1*, *reg32*/*mem32*	F2 (W0) 0F 2A /r	Converts a doubleword integer in *reg32* or *mem32* to a double-precision floating-point value in *xmm1*.
CVTSI2SD *xmm1*, *reg64*/*mem64*	F2 (W1) 0F 2A /r	Converts a quadword integer in *reg64* or *mem64* to a double-precision floating-point value in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTSI2SD *xmm1*, *xmm2*, *reg32*/*mem32*	C4	$\overline{\text{RXB}}$.00001	0.$\overline{\textit{src}}$.X.11	2A /r
VCVTSI2SD *ymm1*, *xmm2*, *reg64*/*mem64*	C4	$\overline{\text{RXB}}$.00001	1.$\overline{\textit{src}}$.X.11	2A /r

Related Instructions

(V)CVTDQ2PD, (V)CVTPD2DQ, (V)CVTPI2PD, (V)CVTSD2SI, (V)CVTTPD2DQ, (V)CVTTSD2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTSI2SS VCVTSI2SS Convert Signed Doubleword or Quadword Integer to Scalar Single-Precision Floating-Point

Converts a signed integer value to a single-precision floating-point value and writes the converted value to an XMM register. When the result of the conversion is an inexact value, the value is rounded as specified by MXCSR.RC.

There are legacy and extended forms of the instruction:

CVTSI2SS

- When REX.W = 0, converts a signed doubleword integer value from a 32-bit source general-purpose register or a 32-bit memory location to a single-precision floating-point value and writes the converted value to the low-order 32 bits of an XMM register. Bits [127:32] of the destination XMM register and bits [255:128] of the corresponding YMM register are not affected.
- When REX.W = 1, converts a a signed quadword integer value from a 64-bit source general-purpose register or a 64-bit memory location to a 64-bit double-precision floating-point value and writes the converted value to the low-order 64 bits of an XMM register. Bits [127:32] of the destination XMM register and bits [255:128] of the corresponding YMM register are not affected.

VCVTSI2SS

The extended form of the instruction has 128-bit encoding.

- When VEX.W = 0, converts a signed doubleword integer value from a 32-bit source general-purpose register or a 32-bit memory location to a double-precision floating-point value and writes the converted value to the low-order 32 bits of the destination XMM register. Bits [127:32] of the first source XMM register are copied to the destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- When VEX.W = 1, converts a signed quadword integer value from a 64-bit source general-purpose register or a 64-bit memory location to a double-precision floating-point value and writes the converted value to the low-order 32 bits of the destination XMM register. Bits [127:32] of the first source XMM register are copied to the destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTSI2SS is an SSE instruction and VCVTSI2SS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTSI2SS *xmm1*, *reg32*/*mem32*	F3 (W0) 0F 2A /r	Converts a doubleword integer in reg32 or *mem32* to a single-precision floating-point value in *xmm1*.
CVTSI2SS *xmm1*, *reg64*/*mem64*	F3 (W1) 0F 2A /r	Converts a quadword integer in reg64 or *mem64* to a single-precision floating-point value in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTSI2SS *xmm1*, *xmm2*, *reg32*/*mem32*	C4	$\overline{\text{RXB}}$.00001	0.$\overline{src}$.X.10	2A /r
VCVTSI2SS *xmm1*, *xmm2*, *reg64*/*mem64*	C4	$\overline{\text{RXB}}$.00001	1.$\overline{src}$.X.10	2A /r

Related Instructions

(V)CVTDQ2PS, (V)CVTPS2DQ, (V)CVTSS2SI, (V)CVTTPS2DQ, (V)CVTTSS2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTSS2SD Convert Scalar Single-Precision Floating-Point VCVTSS2SD to Scalar Double-Precision Floating-Point

Converts a scalar single-precision floating-point value to a scalar double-precision floating-point value and writes the converted value to the low-order 64 bits of the destination.

There are legacy and extended forms of the instruction:

CVTSS2SD

Converts a scalar single-precision floating-point value in the low-order 32 bits of a source XMM register or a 32-bit memory location to a scalar double-precision floating-point value and writes the converted value to the low-order 64 bits of a destination XMM register. Bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are not affected.

VCVTSS2SD

The extended form of the instruction has 128-bit encoding.

Converts a scalar single-precision floating-point value in the low-order 32 bits of a source XMM register or a 32-bit memory location to a scalar double-precision floating-point value and writes the converted value to the low-order 64 bits of a destination XMM register. Bits [127:64] of the destination are copied from a second source XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTSD2SD is an SSE2 instruction and VCVTSD2SD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTSS2SD *xmm1*, *xmm2*/*mem32*	F3 0F 5A /r	Converts a scalar single-precision floating-point value in *xmm2* or *mem32* to a scalar double-precision floating-point value in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTSS2SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.10	5A /r

Related Instructions

(V)CVTPD2PS, (V)CVTPS2PD, (V)CVTSD2SS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTSS2SI VCVTSS2SI — Convert Scalar Single-Precision Floating-Point to Signed Doubleword or Quadword Integer

Converts a single-precision floating-point value to a signed integer value and writes the converted value to a general-purpose register.

When the result of the conversion is an inexact value, the value is rounded as specified by MXCSR.RC. When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword ($–2^{31}$ to $+2^{31} – 1$) or quadword value ($–2^{63}$ to $+2^{63} – 1$), the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) is returned when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTSS2SI

- When REX.W = 0, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 32-bit signed integer value and writes the converted value to a 32-bit general-purpose register.
- When REX.W = 1, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 64-bit signed integer value and writes the converted value to a 64-bit general-purpose register.

VCVTSS2SI

The extended form of the instruction has 128-bit encoding.

- When VEX.W = 0, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 32-bit signed integer value and writes the converted value to a 32-bit general-purpose register.
- When VEX.W = 1, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 64-bit signed integer value and writes the converted value to a 64-bit general-purpose register.

CVTSS2SI is an SSE instruction and VCVTSS2SI is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTSS2SI *reg32, xmm1/mem32*	F3 (W0) 0F 2D /r	Converts a single-precision floating-point value in *xmm1* or *mem32* to a 32-bit integer value in reg32
CVTSS2SI *reg64, xmm1//mem64*	F3 (W1) 0F 2D /r	Converts a single-precision floating-point value in *xmm1* or *mem64* to a 64-bit integer value in reg64

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTSS2SI *reg32, xmm1/mem32*	C4	$\overline{\text{RXB}}$.00001	0.1111.X.10	2D /r
VCVTSS2SI *reg64, xmm1/mem64*	C4	$\overline{\text{RXB}}$.00001	1.1111.X.10	2D /r

Related Instructions

(V)CVTDQ2PS, (V)CVTPS2DQ, (V)CVTSI2SS, (V)CVTTPS2DQ, (V)CVTTSS2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTTPD2DQ VCVTTPD2DQ — Convert Packed Double-Precision Floating-Point to Packed Doubleword Integer, Truncated

Converts packed double-precision floating-point values to packed signed doubleword integer values and writes the converted values to the destination.

When the result is an inexact value, it is truncated (rounded toward zero). When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTTPD2DQ

Converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two packed signed doubleword integers and writes the converted values to the two low-order doublewords of the destination XMM register. Bits [127:64] of the destination are cleared. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VCVTTPD2DQ

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts two packed double-precision floating-point values in an XMM register or a 128-bit memory location to two signed doubleword values and writes the converted values to the lower two doubleword elements of the destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts four packed double-precision floating-point values in a YMM register or a 256-bit memory location to four signed doubleword integer values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

CVTTPD2DQ is an SSE2 instruction and VCVTTPD2DQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTTPD2DQ *xmm1*, *xmm2/mem128*	66 0F E6 /r	Converts two packed double-precision floating-point values in *xmm2* or *mem128* to packed doubleword integers in *xmm1*. Truncates inexact result.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTTPD2DQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	E6 /r
VCVTTPD2DQ *xmm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	E6 /r

Related Instructions

(V)CVTDQ2PD, (V)CVTPD2DQ, (V)CVTPI2PD, (V)CVTSD2SI, (V)CVTSI2SD, (V)CVTTSD2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

CVTTPS2DQ VCVTTPS2DQ Convert Packed Single-Precision Floating-Point to Packed Doubleword Integers, Truncated

Converts packed single-precision floating-point values to packed signed doubleword integer values and writes the converted values to the destination.

When the result of the conversion is an inexact value, the value is truncated (rounded toward zero). When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31}-1$), the instruction returns the 32-bit indefinite integer value (8000_0000h) when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTTPS2DQ

Converts four packed single-precision floating-point values in an XMM register or a 128-bit memory location to four packed signed doubleword integer values and writes the converted values to an XMM register. The high-order 128-bits of the corresponding YMM register are not affected.

VCVTTPS2DQ

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Converts four packed single-precision floating-point values in an XMM register or a 128-bit memory location to four packed signed doubleword integer values and writes the converted values to an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Converts eight packed single-precision floating-point values in a YMM register or a 256-bit memory location to eight packed signed doubleword integer values and writes the converted values to a YMM register.

CVTTPS2DQ is an SSE2 instruction and VCVTTPS2DQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTTPS2DQ *xmm1*, *xmm2/mem128*	F3 0F 5B /r	Converts four packed single-precision floating-point values in *xmm2* or *mem128* to four packed doubleword integers in *xmm1*. Truncates inexact result.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTTPS2DQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	5B /r
VCVTTPS2DQ *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.10	5B /r

Related Instructions

(V)CVTDQ2PS, (V)CVTPS2DQ, (V)CVTSI2SS, (V)CVTSS2SI, (V)CVTTSS2SI

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTTSD2SI VCVTTSD2SI Convert Scalar Double-Precision Floating-Point to Signed Double- or Quadword Integer, Truncated

Converts a scalar double-precision floating-point value to a signed integer value and writes the converted value to a general-purpose register.

When the result of the conversion is an inexact value, the value is truncated (rounded toward zero). When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$) or quadword value (-2^{63} to $+2^{63} - 1$), the instruction returns the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTTSD2SI

- When REX.W = 0, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 32-bit signed integer and writes the converted value to a 32-bit general purpose register.
- When REX.W = 1, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 64-bit sign-extended integer and writes the converted value to a 64-bit general purpose register.

VCVTTSD2SI

The extended form of the instruction has 128-bit encoding.

- When VEX.W = 0, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 32-bit signed integer and writes the converted value to a 32-bit general purpose register.
- When VEX.W = 1, converts a scalar double-precision floating-point value in the low-order 64 bits of an XMM register or a 64-bit memory location to a 64-bit sign-extended integer and writes the converted value to a 64-bit general purpose register.

CVTTSD2SI is an SSE2 instruction and VCVTTSD2SI is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTTSD2SI *reg32*, *xmm1*/*mem64*	F2 (W0) 0F 2C /r	Converts a packed double-precision floating-point value in *xmm1* or *mem64* to a doubleword integer in *reg32*. Truncates inexact result.
CVTTSD2SI *reg64*, *xmm1*/*mem64*	F2 (W1) 0F 2C /r	Converts a packed double-precision floating-point value in *xmm1* or *mem64* to a quadword integer in *reg64*.Truncates inexact result.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTTSD2SI *reg32*, *xmm2*/*mem64*	C4	$\overline{\text{RXB}}$.00001	0.1111.X.11	2C /r
VCVTTSD2SI *reg64*, *xmm2*/*mem64*	C4	$\overline{\text{RXB}}$.00001	1.1111.X.11	2C /r

Related Instructions

(V)CVTDQ2PD, (V)CVTPD2DQ, (V)CVTPI2PD, (V)CVTSD2SI, (V)CVTSI2SD, (V)CVTTPD2DQ

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

CVTTSS2SI VCVTTSS2SI Convert Scalar Single-Precision Floating-Point to Signed Double or Quadword Integer, Truncated

Converts a single-precision floating-point value to a signed integer value and writes the converted value to a general-purpose register.

When the result of the conversion is an inexact value, the value is truncated (rounded toward zero). When the floating-point value is a NaN, infinity, or the result of the conversion is larger than the maximum signed doubleword (-2^{31} to $+2^{31} - 1$) or quadword value (-2^{63} to $+2^{63} - 1$), the indefinite integer value (8000_0000h for 32-bit integers, 8000_0000_0000_0000h for 64-bit integers) is returned when the invalid-operation exception (IE) is masked.

There are legacy and extended forms of the instruction:

CVTTSS2SI

- When REX.W = 0, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 32-bit signed integer value and writes the converted value to a 32-bit general-purpose register. Bits [255:128] of the YMM register that corresponds to the source are not affected.
- When REX.W = 1, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 64-bit signed integer value and writes the converted value to a 64-bit general-purpose register. Bits [255:128] of the YMM register that corresponds to the source are not affected.

VCVTTSS2SI

The extended form of the instruction has 128-bit encoding.

- When VEX.W = 0, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 32-bit signed integer value and writes the converted value to a 32-bit general-purpose register. Bits [255:128] of the YMM register that corresponds to the source are cleared.
- When VEX.W = 1, converts a single-precision floating-point value in the low-order 32 bits of an XMM register or a 32-bit memory location to a 64-bit signed integer value and writes the converted value to a 64-bit general-purpose register. Bits [255:128] of the YMM register that corresponds to the source are cleared.

CVTTSS2SI is an SSE instruction and VCVTTSS2SI is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
CVTTSS2SI *reg32, xmm1/mem32*	F3 (W0) 0F 2C /r	Converts a single-precision floating-point value in *xmm1* or *mem32* to a 32-bit integer value in *reg32*. Truncates inexact result.
CVTTSS2SI *reg64, xmm1/mem64*	F3 (W1) 0F 2C /r	Converts a single-precision floating-point value in *xmm1* or *mem64* to a 64-bit integer value in *reg64*. Truncates inexact result.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VCVTTSS2SI *reg32, xmm1/mem32*	C4	$\overline{\text{RXB}}$.00001	0.1111.X.10	2C /r
VCVTTSS2SI *reg64, xmm1/mem64*	C4	$\overline{\text{RXB}}$.00001	1.1111.X.10	2C /r

Related Instructions

(V)CVTDQ2PS, (V)CVTPS2DQ, (V)CVTSI2SS, (V)CVTSS2SI, (V)CVTTPS2DQ

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

DIVPD VDIVPD — Divide Packed Double-Precision Floating-Point

Divides each of the packed double-precision floating-point values of the first source operand by the corresponding packed double-precision floating-point values of the second source operand and writes the quotients to the destination.

There are legacy and extended forms of the instruction:

DIVPD

Divides two packed double-precision floating-point values in the first source XMM register by the corresponding packed double-precision floating-point values in either a second source XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VDIVPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Divides two packed double-precision floating-point values in the first source XMM register by the corresponding packed double-precision floating-point values in either a second source XMM register or a 128-bit memory location and writes the two results a destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Divides four packed double-precision floating-point values in the first source YMM register by the corresponding packed double-precision floating-point values in either a second source YMM register or a 256-bit memory location and writes the two results a destination YMM register.

DIVPD is an SSE2 instruction and VDIVPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
DIVPD *xmm1*, *xmm2/mem128*	66 0F 5E /r	Divides packed double-precision floating-point values in *xmm1* by the packed double-precision floating-point values in xmm2 or *mem128*. Writes quotients to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VDIVPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	5E /r
VDIVPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.01	5E /r

Related Instructions

(V)DIVPS, (V)DIVSD, (V)DIVSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M	M	M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

DIVPS VDIVPS — Divide Packed Single-Precision Floating-Point

Divides each of the packed single-precision floating-point values of the first source operand by the corresponding packed single-precision floating-point values of the second source operand and writes the quotients to the destination.

There are legacy and extended forms of the instruction:

DIVPS

Divides four packed single-precision floating-point values in the first source XMM register by the corresponding packed single-precision floating-point values in either a second source XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VDIVPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Divides four packed single-precision floating-point values in the first source XMM register by the corresponding packed single-precision floating-point values in either a second source XMM register or a 128-bit memory location and writes two results to a third destination XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Divides eight packed single-precision floating-point values in the first source YMM register by the corresponding packed single-precision floating-point values in either a second source YMM register or a 256-bit memory location and writes the two results a destination YMM register.

DIVPS is an SSE instruction and VDIVPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
DIVPS *xmm1*, *xmm2/mem128*	0F 5E /r	Divides packed single-precision floating-point values in *xmm1* by the corresponding values in *xmm2* or *mem128*. Writes quotients to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VDIVPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	5E /r
VDIVPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.00	5E /r

Related Instructions

(V)DIVPD, (V)DIVSD, (V)DIVSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M	M	M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

DIVSD VDIVSD — Divide Scalar Double-Precision Floating-Point

Divides the double-precision floating-point value in the low-order quadword of the first source operand by the double-precision floating-point value in the low-order quadword of the second source operand and writes the quotient to the low-order quadword of the destination.

There are legacy and extended forms of the instruction:

DIVSD

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The first source register is also the destination register. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VDIVSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. Bits [127:64] of the first source operand are copied to bits [127:64] of the destination. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

DIVSD is an SSE2 instruction and VDIVSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
DIVSD *xmm1*, *xmm2/mem64*	F2 0F 5E /r	Divides the double-precision floating-point value in the low-order 64 bits of *xmm1*by the corresponding value in *xmm2* or *mem64*. Writes quotient to *xmm1*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VDIVSD *xmm1*, *xmm2*, *xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.11	5E /r

Related Instructions

(V)DIVPD, (V)DIVPS, (V)DIVSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M	M	M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

DIVSS VDIVSS Divide Scalar Single-Precision Floating-Point

Divides the single-precision floating-point value in the low-order doubleword of the first source operand by the single-precision floating-point value in the low-order doubleword of the second source operand and writes the quotient to the low-order doubleword of the destination.

There are legacy and extended forms of the instruction:

DIVSS

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The first source register is also the destination register. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VDIVSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The destination is a third XMM register. Bits [127:32] of the first source operand are copied to bits [127:32] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

DIVSS is an SSE instruction and VDIVSS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
DIVSS *xmm1*, *xmm2*/*mem32*	F3 0F 5E /r	Divides a single-precision floating-point value in the low-order doubleword of *xmm1* by a corresponding value in *xmm2* or *mem32*. Writes the quotient to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VDIVSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	5E /r

Related Instructions

(V)DIVPD, (V)DIVPS, (V)DIVSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M	M	M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

DPPD VDPPD — Dot Product Packed Double-Precision Floating-Point

Computes the dot-product of the input operands. An immediate operand specifies both the input values and the destination locations to which the products are written.

Selectively multiplies packed double-precision values in a source operand by the corresponding values in another source operand, writes the results to a temporary location, adds the results, writes the sum to a second temporary location and selectively writes the sum to a destination.

Mask bits [5:4] of an 8-bit immediate operand perform multiplicative selection. Bit 5 selects bits [127:64] of the source operands; bit 4 selects bits [63:0] of the source operands. When a mask bit = 1, the corresponding packed double-precision floating point values are multiplied and the product is written to the corresponding position of a 128-bit temporary location. When a mask bit = 0, the corresponding position of the temporary location is cleared.

After the two 64-bit values in the first temporary location are added and written to the 64-bit second temporary location, mask bits [1:0] of the same 8-bit immediate operand perform write selection. Bit 1 selects bits [127:64] of the destination; bit 0 selects bits [63:0] of the destination. When a mask bit = 1, the 64-bit value of the second temporary location is written to the corresponding position of the destination. When a mask bit = 0, the corresponding position of the destination is cleared.

When the operation produces a NaN, its value is determined as follows.

Source Operands (in either order)		**NaN Result**[1]
QNaN	Any non-NaN floating-point value (or single-operand instruction)	Value of QNaN
SNaN	Any non-NaN floating-point value (or single-operand instruction)	Value of SNaN, converted to a QNaN[2]
QNaN	QNaN	First operand
QNaN	SNaN	First operand (converted to QNaN if SNaN
SNaN	SNaN	First operand converted to a QNaN[2]
Note:	*1. A NaN result produced when the floating-point invalid-operation exception is masked.* *2. The conversion is done by changing the most-significant fraction bit to 1.*	

For each addition occurring in either the second or third step, for the purpose of NaN propagation, the addend of lower bit index is considered to be the first of the two operands. For example, when both multiplications produce NaNs, the one that corresponds to bits [64:0] is written to all indicated fields of the destination, regardless of how those NaNs were generated from the sources. When the high-order multiplication produces NaNs and the low-order multiplication produces infinities of opposite signs, the real indefinite QNaN (produced as the sum of the infinities) is written to the destination.

NaNs in source operands or in computational results result in at least one NaN in the destination. For the 256-bit version, NaNs are propagated within the two independent dot product operations only to their respective 128-bit results.

There are legacy and extended forms of the instruction:

DPPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VDPPD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

DPPD is an SSE4.1 instruction and VDPPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
DPPD *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 41 /r ib	Selectively multiplies packed double-precision floating-point values in *xmm2* or *mem128* by corresponding values in *xmm1*, adds interim products, selectively writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VDPPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	41 /r ib

Related Instructions

(V)DPPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*
Exceptions are determined separately for each add-multiply operation.
Unmasked exceptions do not affect the destination

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

DPPS VDPPS — Dot Product Packed Single-Precision Floating-Point

Computes the dot-product of the input operands. An immediate operand specifies both the input values and the destination locations to which the products are written.

Selectively multiplies packed single-precision values in a source operand by corresponding values in another source operand, writes results to a temporary location, adds pairs of results, writes the sums to additional temporary locations, and selectively writes a cumulative sum to a destination.

Mask bits [7:4] of an 8-bit immediate operand perform multiplicative selection. Each bit selects a 32-bit segment of the source operands; bit 7 selects bits [127:96], bit 6 selects bits [95:64], bit 5 selects bits [63:32], and bit 4 selects bits [31:0]. When a mask bit = 1, the corresponding packed single-precision floating point values are multiplied and the product is written to the corresponding position of a 128-bit temporary location. When a mask bit = 0, the corresponding position of the temporary location is cleared.

After multiplication, three pairs of 32-bit values are added and written to temporary locations.
Bits [63:32] and [31:0] of temporary location 1 are added and written to 32-bit temporary location 2; bits [127:96] and [95:64] of temporary location 1 are added and written to 32-bit temporary location 3; then the contents of temporary locations 2 and 3 are added and written to 32-bit temporary location 4.

After addition, mask bits [3:0] of the same 8-bit immediate operand perform write selection. Each bit selects a 32-bit segment of the source operands; bit 3 selects bits [127:96], bit 2 selects bits [95:64], bit 1 selects bits [63:32], and bit 0 selects bits [31:0] of the destination. When a mask bit = 1, the 64-bit value of the fourth temporary location is written to the corresponding position of the destination. When a mask bit = 0, the corresponding position of the destination is cleared.

For the 256-bit extended encoding, this process is performed on the upper and lower 128 bits of the affected YMM registers.

When the operation produces a NaN, its value is determined as follows.

Source Operands (in either order)		NaN Result[1]
QNaN	Any non-NaN floating-point value (or single-operand instruction)	Value of QNaN
SNaN	Any non-NaN floating-point value (or single-operand instruction)	Value of SNaN, converted to a QNaN[2]
QNaN	QNaN	First operand
QNaN	SNaN	First operand (converted to QNaN if SNaN
SNaN	SNaN	First operand converted to a QNaN[2]

Note: *1. A NaN result produced when the floating-point invalid-operation exception is masked.*
2. The conversion is done by changing the most-significant fraction bit to 1.

For each addition occurring in either the second or third step, for the purpose of NaN propagation, the addend of lower bit index is considered to be the first of the two operands. For example, when all four multiplications produce NaNs, the one that corresponds to bits [31:0] is written to all indicated fields of the destination, regardless of how those NaNs were generated from the sources. When the two highest-order multiplication produce NaNs and the two lowest-low-order multiplications produce infinities of opposite signs, the real indefinite QNaN (produced as the sum of the infinities) is written to the destination.

NaNs in source operands or in computational results result in at least one NaN in the destination. For the 256-bit version, NaNs are propagated within the two independent dot product operations only to their respective 128-bit results.There are legacy and extended forms of the instruction:

DPPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VDPPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

DPPS is an SSE4.1 instruction and VDPPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
DPPS *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 40 /r ib	Selectively multiplies packed single-precision floating-point values in *xmm2* or *mem128* by corresponding values in *xmm1*, adds interim products, selectively writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VDPPS *xmm1*, *xmm2*, *xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	40 /r ib
VDPPS *ymm1, ymm2, ymm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.1.01	40 /r ib

Related Instructions

(V)DPPD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected. Exceptions are determined separately for each add-multiply operation. Unmasked exceptions do not affect the destination*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

EXTRACTPS VEXTRACTPS — Extract Packed Single-Precision Floating-Point

Copies one of four packed single-precision floating-point values from a source XMM register to a general purpose register or a 32-bit memory location.

Bits [1:0] of an immediate byte operand specify the location of the 32-bit value that is copied. 00b corresponds to the low word of the source register and 11b corresponds to the high word of the source register. Bits [7:2] of the immediate operand are ignored.

There are legacy and extended forms of the instruction:

EXTRACTPS

The source operand is an XMM register. The destination can be a general purpose register or a 32-bit memory location. A 32-bit single-precision value extracted to a general purpose register is zero-extended to 64-bits.

VEXTRACTPS

The extended form of the instruction has 128-bit encoding.

XMM Encoding

The source operand is an XMM register. The destination can be a general purpose register or a 32-bit memory location.

EXTRACTPS is an SSE4.1 instruction and VEXTRACTPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
EXTRACTPS *reg32/mem32*, *xmm1* *imm8*	66 0F 3A 17 /r ib	Extract the single-precision floating-point element of *xmm1* specified by *imm8* to *reg32/mem32*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VEXTRACTPS *reg32/mem32*, *xmm1*, *imm8*	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	17 /r ib

Related Instructions

(V)INSERTPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

HADDPD VHADDPD — Horizontal Add Packed Double-Precision Floating-Point

Adds adjacent pairs of double-precision floating-point values in two source operands and writes the sums to a destination.

There are legacy and extended forms of the instruction:

HADDPD

Adds the packed double-precision values in bits [127:64] and bits [63:0] of the first source XMM register and writes the sum to bits [63:0] of the destination; adds the corresponding doublewords of the second source XMM register or a 128-bit memory location and writes the sum to bits [127:64] of the destination. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VHADDPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Adds the packed double-precision values in bits [127:64] and bits [63:0] of the first source XMM register and writes the sum to bits [63:0] of the destination XMM register; adds the corresponding doublewords of the second source XMM register or a 128-bit memory location and writes the sum to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Adds the packed double-precision values in bits [127:64] and bits [63:0] of the of the first source YMM register and writes the sum to bits [63:0] of the destination YMM register; adds the corresponding doublewords of the second source YMM register or a 256-bit memory location and writes the sum to bits [127:64] of the destination. Performs the same process for the upper 128 bits of the sources and destination.

HADDPD is an SSE3 instruction and VHADDPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
HADDPD *xmm1*, *xmm2/mem128*	66 0F 7C /r	Adds adjacent pairs of double-precision values in *xmm1* and *xmm2* or *mem128*. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VHADDPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	7C /r
VHADDPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	7C /r

Related Instructions

(V)HADDPS, (V)HSUBPD, (V)HSUBPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

HADDPS VHADDPS — Horizontal Add Packed Single-Precision

Adds adjacent pairs of single-precision floating-point values in two source operands and writes the sums to a destination.

There are legacy and extended forms of the instruction:

HADDPS

Adds the packed single-precision values in bits [63:32] and bits [31:0] of the first source XMM register and writes the sum to bits [31:0] of the destination; adds the packed single-precision values in bits [127:96] and bits [95:64] of the first source register and writes the sum to bits [63:32] of the destination. Adds the corresponding values in the second source XMM register or a 128-bit memory location and writes the sum to bits [95:64] and [127:96] of the destination. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VHADDPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Adds the packed single-precision values in bits [63:32] and bits [31:0] of the first source XMM register and writes the sum to bits [31:0] of the destination XMM register; adds the packed single-precision values in bits [127:96] and bits [95:64] of the first source register and writes the sum to bits [63:32] of the destination. Adds the corresponding values in the second source XMM register or a 128-bit memory location and writes the sum to bits [95:64] and [127:96] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Adds the packed single-precision values in bits [63:32] and bits [31:0] of the first source YMM register and writes the sum to bits [31:0] of the destination YMM register; adds the packed single-precision values in bits [127:96] and bits [95:64] of the first source register and writes the sum to bits [63:32] of the destination. Adds the corresponding values in the second source YMM register or a 256-bit memory location and writes the sums to bits [95:64] and [127:96] of the destination. Performs the same process for the upper 128 bits of the sources and destination.

HADDPS is an SSE3 instruction and VHADDPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
HADDPS *xmm1*, *xmm2/mem128*	F2 0F 7C /r	Adds adjacent pairs of single-precision values in *xmm1* and *xmm2* or *mem128*. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	**$\overline{\text{RXB}}$.mmmmm**	**W.$\overline{\text{vvvv}}$.L.pp**	**Opcode**
VHADDPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.11	7C /r
VHADDPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.11	7C /r

Related Instructions

(V)HADDPD, (V)HSUBPD, (V)HSUBPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

HSUBPD VHSUBPD — Horizontal Subtract Packed Double-Precision

Subtracts adjacent pairs of double-precision floating-point values in two source operands and writes the sums to a destination.

There are legacy and extended forms of the instruction:

HSUBPD

The first source register is also the destination.

Subtracts the packed double-precision value in bits [127:64] from the value in bits [63:0] of the first source XMM register and writes the difference to bits [63:0] of the destination; subtracts the corresponding values of the second source XMM register or a 128-bit memory location and writes the difference to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VHSUBPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Subtracts the packed double-precision values in bits [127:64] from the value in bits [63:0] of the first source XMM register and writes the difference to bits [63:0] of the destination XMM register; subtracts the corresponding values of the second source XMM register or a 128-bit memory location and writes the difference to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Subtracts the packed double-precision values in bits [127:64] from the value in bits [63:0] of the of the first source YMM register and writes the difference to bits [63:0] of the destination YMM register; subtracts the corresponding values of the second source YMM register or a 256-bit memory location and writes the difference to bits [127:64] of the destination. Performs the same process for the upper 128 bits of the sources and destination.

HSUBPD is an SSE3 instruction and VHSUBPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
HSUBPD *xmm1*, *xmm2/mem128*	66 0F 7D /r	Subtracts adjacent pairs of double-precision floating-point values in *xmm1* and *xmm2* or *mem128*. Writes the differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VHSUBPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	7D /r
VHSUBPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	7D /r

Related Instructions

(V)HSUBPS, (V)HADDPD, (V)HADDPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

HSUBPS VHSUBPS — Horizontal Subtract Packed Single

Subtracts adjacent pairs of single-precision floating-point values in two source operands and writes the differences to a destination.

There are legacy and extended forms of the instruction:

HSUBPS

Subtracts the packed single-precision values in bits [63:32] from the values in bits [31:0] of the first source XMM register and writes the difference to bits [31:0] of the destination; subtracts the packed single-precision values in bits [127:96] from the value in bits [95:64] of the first source register and writes the difference to bits [63:32] of the destination. Subtracts the corresponding values of the second source XMM register or a 128-bit memory location and writes the differences to bits [95:64] and [127:96] of the destination. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VHSUBPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Subtracts the packed single-precision values in bits [63:32] from the value in bits [31:0] of the first source XMM register and writes the difference to bits [31:0] of the destination XMM register; subtracts the packed single-precision values in bits [127:96] from the value bits [95:64] of the first source register and writes the sum to bits [63:32] of the destination. Subtracts the corresponding values of the second source XMM register or a 128-bit memory location and writes the differences to bits [95:64] and [127:96] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Subtracts the packed single-precision values in bits [63:32] from the value in bits [31:0] of the first source YMM register and writes the difference to bits [31:0] of the destination YMM register; subtracts the packed single-precision values in bits [127:96] from the value in bits [95:64] of the first source register and writes the difference to bits [63:32] of the destination. Subtracts the corresponding values of the second source YMM register or a 256-bit memory location and writes the differences to bits [95:64] and [127:96] of the destination. Performs the same process for the upper 128 bits of the sources and destination.

HSUBPS is an SSE3 instruction and VHSUBPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
HSUBPS *xmm1*, *xmm2/mem128*	F2 0F 7D /r	Subtracts adjacent pairs of values in *xmm1* and *xmm2* or *mem128*. Writes differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VHSUBPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.11	7D /r
VHSUBPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.11	7D /r

Related Instructions

(V)HSUBPD, (V)HADDPD, (V)HADDPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

INSERTPS VINSERTPS — Insert Packed Single-Precision Floating-Point

Copies a selected single-precision floating-point value from a source operand to a selected location in a destination register and optionally clears selected elements of the destination. The legacy and extended forms of the instruction treat the remaining elements of the destination in different ways.

Selections are specified by three fields of an immediate 8-bit operand:

7	6	5	4	3	2	1	0
COUNT_S		COUNT_D		ZMASK			

COUNT_S — The binary value of the field specifies a 32-bit element of a source register, counting upward from the low-order doubleword. COUNT_S is used only for register source; when the source is a memory operand, COUNT_S = 0.

COUNT_D — The binary value of the field specifies a 32-bit destination element, counting upward from the low-order doubleword.

ZMASK — Set a bit to clear a 32-bit element of the destination.

There are legacy and extended forms of the instruction:

INSERTPS

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

When the source operand is a register, the instruction copies the 32-bit element of the source specified by Count_S to the location in the destination specified by Count_D, and clears destination elements as specified by ZMask. Elements of the destination that are not cleared are not affected.

When the source operand is a memory location, the instruction copies a 32-bit value from memory, to the location in the destination specified by Count_D, and clears destination elements as specified by ZMask. Elements of the destination that are not cleared are not affected.

VINSERTPS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either another XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

When the second source operand is a register, the instruction copies the 32-bit element of the source specified by Count_S to the location in the destination specified by Count_D. The other elements of the destination are either copied from the first source operand or cleared as specified by ZMask.

When the second source operand is a memory location, the instruction copies a 32-bit value from the source to the location in the destination specified by Count_D. The other elements of the destination are either copied from the first source operand or cleared as specified by ZMask.

INSERTPS is an SSE4.1 instruction and VINSERTPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
INSERTPS *xmm1*, *xmm2/mem32, imm8*	66 0F 3A 21 /r ib	Insert a selected single-precision floating-point value from *xmm2* or from *mem32* at a selected location in *xmm1* and clear selected elements of *xmm1*. Selections specified by imm8.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VINSERTPS *xmm1*, *xmm2*, *xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	21 /r ib

Related Instructions

(V)EXTRACTPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

LDDQU VLDDQU — Load Unaligned Double Quadword

Loads unaligned double quadwords from a memory location to a destination register.

Like the (V)MOVUPD instructions, (V)LDDQU loads a 128-bit or 256-bit operand from an unaligned memory location. However, to improve performance when the memory operand is actually misaligned, (V)LDDQU may read an aligned 16 or 32 bytes to get the first part of the operand, and an aligned 16 or 32 bytes to get the second part of the operand. This behavior is implementation-specific, and (V)LDDQU may only read the exact 16 or 32 bytes needed for the memory operand. If the memory operand is in a memory range where reading extra bytes can cause performance or functional issues, use (V)MOVUPD instead of (V)LDDQU.

Memory operands that are not aligned on 16-byte or 32-byte boundaries do not cause general-protection exceptions.

There are legacy and extended forms of the instruction:

LDDQU

The source operand is an unaligned 128-bit memory location. The destination operand is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination register are not affected.

VLDDQU

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The source operand is an unaligned 128-bit memory location. The destination operand is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination register are cleared.

YMM Encoding

The source operand is an unaligned 256-bit memory location. The destination operand is a YMM register.

LDDQU is an SSE3 instruction and VLDDQU is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
LDDQU *xmm1*, *mem128*	F2 0F F0 /r	Loads a 128-bit value from an unaligned *mem128* to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VLDDQU *xmm1*, *mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.11	F0 /r
VLDDQU *ymm1*, *mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.11	F0 /r

Related Instructions

(V)MOVDQU

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

LDMXCSR VLDMXCSR — Load MXCSR Control/Status Register

Loads the MXCSR register with a 32-bit value from memory.

For both legacy LDMXCSR and extended VLDMXCSR forms of the instruction, the source operand is a 32-bit memory location and the destination operand is the MXCSR.

If an MXCSR load clears a SIMD floating-point exception mask bit and sets the corresponding exception flag bit, a SIMD floating-point exception is not generated immediately. An exception is generated only when the next instruction that operates on an XMM or YMM register operand and causes that particular SIMD floating-point exception to be reported executes.

A general protection exception occurs if the instruction attempts to load non-zero values into reserved MXCSR bits. Software can use MXCSR_MASK to determine which bits are reserved. For details, see “128-Bit, 64-Bit, and x87 Programming” in Volume 2.

The MXCSR register is described in “Registers” in Volume 1.

LDMXCSR is an SSE instruction and VLDMXCSR is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
LDMXCSR *mem32*	0F AE /2	Loads MXCSR register with 32-bit value from memory.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VLDMXCSR *mem32*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	AE /2

Related Instructions

(V)STMXCSR

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Null data segment used to reference memory.
	S	S	X	Attempt to load non-zero values into reserved MXCSR bits
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MASKMOVDQU VMASKMOVDQU — Masked Move Double Quadword Unaligned

Moves bytes from the first source operand to a memory location specified by the DS:rDI register. Bytes are selected by mask bits in the second source operand. The memory location may be unaligned.

The mask consists of the most significant bit of each byte of the second source register.
When a mask bit = 1, the corresponding byte of the first source register is written to the destination; when a mask bit = 0, the corresponding byte is not written.

An all-zero mask value results in the following behavior:

- No data is written to memory.
- Code and data breakpoints are not guaranteed to be signaled in all implementations.
- Exceptions associated with memory addressing and page faults are not guaranteed to be signaled in all implementations.

The instruction implicitly uses weakly-ordered, write-combining buffering for the data, as described in "Buffering and Combining Memory Writes" in Volume 2. For data that is shared by multiple processors, this instruction should be used together with a fence instruction in order to ensure data coherency (see "Cache and TLB Management" in Volume 2).

There are legacy and extended forms of the instruction:

MASKMOVDQU

The first source operand is an XMM register and the second source operand is another XMM register. The destination is a 128-bit memory location.

VMASKMOVDQU

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is another XMM register. The destination is a 128-bit memory location.

MASKMOVDQU is an SSE2 instruction and VMASKMOVDQU is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MASKMOVDQU *xmm1*, *xmm2*	66 0F F7 /r	Move bytes selected by a mask value in *xmm2* from *xmm1* to the memory location specified by DS:rDI.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMASKMOVDQU *xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	F7 /r

Related Instructions

(V)MASKMOVPD, (V)MASKMOVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MAXPD VMAXPD — Maximum Packed Double-Precision Floating-Point

Compares each packed double-precision floating-point value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MAXPD

Compares two pairs of packed double-precision floating-point values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMAXPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Compares two pairs of packed double-precision floating-point values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Compares four pairs of packed double-precision floating-point values.

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

MAXPD is an SSE2 instruction and VMAXPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MAXPD *xmm1*, *xmm2/mem128*	66 0F 5F /r	Compares two pairs of packed double-precision values in *xmm1* and *xmm2* or *mem128* and writes the greater value to the corresponding position in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMAXPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	5F /r
VMAXPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.01	5F /r

Related Instructions

(V)MAXPS, (V)MAXSD, (V)MAXSS, (V)MINPD, (V)MINPS, (V)MINSD, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MAXPS VMAXPS — Maximum Packed Single-Precision Floating-Point

Compares each packed single-precision floating-point value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MAXPS

Compares four pairs of packed single-precision floating-point values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMAXPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Compares four pairs of packed single-precision floating-point values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Compares eight pairs of packed single-precision floating-point values.

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

MAXPS is an SSE instruction and VMAXPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MAXPS *xmm1*, *xmm2/mem128*	0F 5F /r	Compares four pairs of packed single-precision values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMAXPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	5F /r
VMAXPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	5F /r

Related Instructions

(V)MAXPD, (V)MAXSD, (V)MAXSS, (V)MINPD, (V)MINPS, (V)MINSD, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

MAXSD VMAXSD — Maximum Scalar Double-Precision Floating-Point

Compares the scalar double-precision floating-point value in the low-order 64 bits of the first source operand to a corresponding value in the second source operand and writes the numerically greater value into the low-order 64 bits of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MAXSD

The first source operand is an XMM register. The second source operand is either another XMM register or a 64-bit memory location. The first source register is also the destination. When the second source is a 64-bit memory location, the upper 64 bits of the first source register are copied to the destination. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMAXSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either another XMM register or a 64-bit memory location. The destination is an XMM register. When the second source is a 64-bit memory location, the upper 64 bits of the first source register are copied to the destination. Bits [127:64] of the destination are copied from bits [127:64] of the first source. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MAXSD is an SSE2 instruction and VMAXSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MAXSD *xmm1*, *xmm2*/*mem64*	F2 0F 5F /r	Compares a pair of scalar double-precision values in the low-order 64 bits of *xmm1* and *xmm2* or *mem64* and writes the greater value to the low-order 64 bits of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMAXSD *xmm1*, *xmm2*, *xmm3*/*mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	5F /r

Related Instructions

(V)MAXPD, (V)MAXPS, (V)MAXSS, (V)MINPD, (V)MINPS, (V)MINSD, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MAXSS VMAXSS — Maximum Scalar Single-Precision Floating-Point

Compares the scalar single-precision floating-point value in the low-order 32 bits of the first source operand to a corresponding value in the second source operand and writes the numerically greater value into the low-order 32 bits of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MAXSS

The first source operand is an XMM register. The second source operand is either another XMM register or a 32-bit memory location. The first source register is also the destination. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMAXSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either another XMM register or a 32-bit memory location. The destination is an XMM register. Bits [127:32] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MAXSS is an SSE instruction and VMAXSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MAXSS *xmm1*, *xmm2*/*mem32*	F3 0F 5F /r	Compares a pair of scalar single-precision values in the low-order 32 bits of *xmm1* and *xmm2* or *mem32* and writes the greater value to the low-order 32 bits of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMAXSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	5F /r

Related Instructions

(V)MAXPD, (V)MAXPS, (V)MAXSD, (V)MINPD, (V)MINPS, (V)MINSD, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MINPD VMINPD Minimum Packed Double-Precision Floating-Point

Compares each packed double-precision floating-point value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MINPD

Compares two pairs of packed double-precision floating-point values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMINPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Compares two pairs of packed double-precision floating-point values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Compares four pairs of packed double-precision floating-point values.

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

MINPD is an SSE2 instruction and VMINPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MINPD *xmm1*, *xmm2/mem128*	66 0F 5D /r	Compares two pairs of packed double-precision values in *xmm1* and *xmm2* or *mem128* and writes the lesser value to the corresponding position in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMINPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	5D /r
VMINPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.01	5D /r

Related Instructions

(V)MAXPD, (V)MAXPS, (V)MAXSD, (V)MAXSS, (V)MINPS, (V)MINSD, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MINPS VMINPS — Minimum Packed Single-Precision Floating-Point

Compares each packed single-precision floating-point value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MINPS

Compares four pairs of packed single-precision floating-point values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMINPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Compares four pairs of packed single-precision floating-point values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Compares eight pairs of packed single-precision floating-point values.

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

MINPS is an SSE instruction and VMINPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MINPS *xmm1*, *xmm2/mem128*	0F 5D /r	Compares four pairs of packed single-precision values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMINPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	5D /r
VMINPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	5D /r

Related Instructions

(V)MAXPD, (V)MAXPS, (V)MAXSD, (V)MAXSS, (V)MINPD, (V)MINSD, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

MINSD VMINSD — Minimum Scalar Double-Precision Floating-Point

Compares the scalar double-precision floating-point value in the low-order 64 bits of the first source operand to a corresponding value in the second source operand and writes the numerically lesser value into the low-order 64 bits of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MINSD

The first source operand is an XMM register. The second source operand is either another XMM register or a 64-bit memory location. The first source register is also the destination. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMINSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either another XMM register or a 64-bit memory location. The destination is an XMM register. Bits [127:64] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MINSD is an SSE2 instruction and VMINSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MINSD *xmm1*, *xmm2*/*mem64*	F2 0F 5D /r	Compares a pair of scalar double-precision values in the low-order 64 bits of *xmm1* and *xmm2* or *mem64* and writes the lesser value to the low-order 64 bits of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMINSD *xmm1*, *xmm2*, *xmm3*/*mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	5D /r

Related Instructions

(V)MAXPD, (V)MAXPS, (V)MAXSD, (V)MAXSS, (V)MINPD, (V)MINPS, (V)MINSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MINSS VMINSS — Minimum Scalar Single-Precision Floating-Point

Compares the scalar single-precision floating-point value in the low-order 32 bits of the first source operand to a corresponding value in the second source operand and writes the numerically lesser value into the low-order 32 bits of the destination.

If both source operands are equal to zero, the value of the second source operand is returned. If either operand is a NaN (SNaN or QNaN), and invalid-operation exceptions are masked, the second source operand is written to the destination.

There are legacy and extended forms of the instruction:

MINSS

The first source operand is an XMM register. The second source operand is either another XMM register or a 32-bit memory location. The first source register is also the destination. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMINSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either another XMM register or a 32-bit memory location. The destination is an XMM register. Bits [127:32] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MINSS is an SSE instruction and VMINSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MINSS *xmm1*, *xmm2*/*mem32*	F3 0F 5D /r	Compares a pair of scalar single-precision values in the low-order 32 bits of *xmm1* and *xmm2* or *mem32* and writes the lesser value to the low-order 32 bits of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMINSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	5D /r

Related Instructions

(V)MAXPD, (V)MAXPS, (V)MAXSD, (V)MAXSS, (V)MINPD, (V)MINPS, (V)MINSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

MOVAPD VMOVAPD — Move Aligned Packed Double-Precision Floating-Point

Moves packed double-precision floating-point values. Values can be moved from a register or memory location to a register; or from a register to a register or memory location.

A memory operand that is not aligned causes a general-protection exception.

There are legacy and extended forms of the instruction:

MOVAPD

Moves two double-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVAPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves two double-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves four double-precision floating-point values. There are encodings for each type of move.

- The source operand is either a YMM register or a 256-bit memory location. The destination operand is a YMM register.
- The source operand is a YMM register. The destination operand is either a YMM register or a 256-bit memory location.

MOVAPD is an SSE2 instruction and VMOVAPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVAPD *xmm1*, *xmm2/mem128*	66 0F 28 /r	Moves two packed double-precision floating-point values from *xmm2* or *mem128* to *xmm1*.
MOVAPD *xmm1/mem128*, *xmm2*	66 0F 29 /r	Moves two packed double-precision floating-point values from *xmm1* or *mem128* to *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVAPD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	28 /r
VMOVAPD *xmm1/mem128*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	29 /r
VMOVAPD *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	28 /r
VMOVAPD *ymm1/mem256*, *ymm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	29 /r

Related Instructions

(V)MOVHPD, (V)MOVLPD, (V)MOVMSKPD, (V)MOVSD, (V)MOVUPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVAPS VMOVAPS — Move Aligned Packed Single-Precision Floating-Point

Moves packed single-precision floating-point values. Values can be moved from a register or memory location to a register; or from a register to a register or memory location.

A memory operand that is not aligned causes a general-protection exception.

There are legacy and extended forms of the instruction:

MOVAPS

Moves four single-precision floating-point values.

There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVAPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves four single-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves eight single-precision floating-point values. There are encodings for each type of move.

- The source operand is either a YMM register or a 256-bit memory location. The destination operand is a YMM register.
- The source operand is a YMM register. The destination operand is either a YMM register or a 256-bit memory location.

MOVAPS is an SSE instruction and VMOVAPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVAPS *xmm1*, *xmm2/mem128*	0F 28 /r	Moves four packed single-precision floating-point values from *xmm2* or *mem128* to *xmm1*.
MOVAPS *xmm1/mem128*, *xmm2*	0F 29 /r	Moves four packed single-precision floating-point values from *xmm1* or *mem128* to *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVAPS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	28 /r
VMOVAPS *xmm1/mem128*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	29 /r
VMOVAPS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	28 /r
VMOVAPS *ymm1/mem256*, *ymm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	29 /r

Related Instructions

(V)MOVHLPS, (V)MOVHPS, (V)MOVLHPS, (V)MOVLPS, (V)MOVMSKPS, (V)MOVSS, (V)MOVUPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVD VMOVD — Move Doubleword or Quadword

Moves 32-bit and 64-bit values. A value can be moved from a general-purpose register or memory location to the corresponding low-order bits of an XMM register, with zero-extension to 128 bits; or from the low-order bits of an XMM register to a general-purpose register or memory location.

The quadword form of this instruction is distinct from the differently-encoded (V)MOVQ instruction.

There are legacy and extended forms of the instruction:

MOVD

There are two encodings for 32-bit moves, characterized by REX.W = 0.

- The source operand is either a 32-bit general-purpose register or a 32-bit memory location. The destination is an XMM register. The 32-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either a 32-bit general-purpose register or a 32-bit memory location.

There are two encodings for 64-bit moves, characterized by REX.W = 0.

- The source operand is either a 64-bit general-purpose register or a 64-bit memory location. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either a 64-bit general-purpose register or a 64-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVD

The extended form of the instruction has 128-bit encoding.

There are two encodings for 32-bit moves, characterized by VEX.W = 0.

- The source operand is either a 32-bit general-purpose register or a 32-bit memory location. The destination is an XMM register. The 32-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either a 32-bit general-purpose register or a 32-bit memory location.

There are two encodings for 64-bit moves, characterized by VEX.W = 1.

- The source operand is either a 64-bit general-purpose register or a 64-bit memory location. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either a 64-bit general-purpose register or a 64-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVD is an SSE2 instruction and VMOVD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVD *xmm*, *reg32*/*mem32*	66 (W0) 0F 6E /r	Move a 32-bit value from *reg32*/*mem32* to *xmm*.
MOVQ *xmm*, *reg64*/*mem64*	66 (W1) 0F 6E /r	Move a 64-bit value from *reg64*/*mem64* to *xmm*.
MOVD *reg32*/*mem32*, *xmm*	66 (W0) 0F 7E /r	Move a 32-bit value from *xmm* to *reg32*/*mem32*
MOVQ *reg64*/*mem64*, *xmm*	66 (W1) 0F 7E /r	Move a 64-bit value from *xmm* to *reg64*/*mem64*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVD *xmm*, *reg32*/*mem32*	C4	$\overline{\text{RXB}}$.00001	0.1111.0.01	6E /r
VMOVQ *xmm*, *reg64*/*mem64*	C4	$\overline{\text{RXB}}$.00001	1.1111.0.01	6E /r
VMOVD r*eg32*/*mem32*, *xmm*	C4	$\overline{\text{RXB}}$.00001	0.1111.1.01	7E /r
VMOVQ r*eg64*/*mem64*, *xmm*	C4	$\overline{\text{RXB}}$.00001	1.1111.1.01	7E /r

Related Instructions

(V)MOVDQA, (V)MOVDQU, (V)MOVQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVDDUP VMOVDDUP — Move and Duplicate Double-Precision Floating-Point

Moves and duplicates double-precision floating-point values.

There are legacy and extended forms of the instruction:

MOVDDUP

Moves and duplicates one quadword value.

The source operand is either the low 64 bits of an XMM register or the address of the least-significant byte of 64 bits of data in memory. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVDDUP

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves and duplicates one quadword value.

The source operand is either the low 64 bits of an XMM register or the address of the least-significant byte of 64 bits of data in memory. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves and duplicates two even-indexed quadword values.

The source operand is either a YMM register or the address of the least-significant byte of 256 bits of data in memory. The destination is another YMM register.Bits [63:0] of the source are written to bits [127:64] and [63:0] of the destination; bits [191:128] of the source are written to bits [255:192] and [191:128] of the destination.

MOVDDUP is an SSE3 instruction and VMOVDDUP is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVDDUP *xmm1*, *xmm2*/*mem64*	F2 0F 12 /r	Moves two copies of the low 64 bits of *xmm2* or *mem64* to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
MOVDDUP *xmm1*, *xmm2*/*mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.11	12 /r
MOVDDUP *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.11	12 /r

Related Instructions

(V)MOVSHDUP, (V)MOVSLDUP

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVDQA VMOVDQA — Move Aligned Double Quadword

Moves aligned packed integer values. Values can be moved from a register or a memory location to another register, or from a register to another register or a memory location.

A memory operand that is not aligned causes a general-protection exception.

There are legacy and extended forms of the instruction:

MOVDQA

Moves two aligned quadwords (128-bit move). There are two encodings.

- The source operand is an XMM register. The destination is either another XMM register or a 128-bit memory location.
- The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

The extended form of the instruction has both 128-bit and 256-bit encoding.

VMOVDQA

XMM Encoding

Moves two aligned quadwords (128-bit move). There are two encodings.

- The source operand is an XMM register. The destination is either another XMM register or a 128-bit memory location.
- The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves four aligned quadwords (256-bit move). There are two encodings.

- The source operand is a YMM register. The destination is either another YMM register or a 256-bit memory location.
- The source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

MOVDQA is an SSE2 instruction and VMOVDQA is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVDQA *xmm1*, *xmm2/mem128*	66 0F 6F /r	Moves aligned packed integer values from *xmm2* or*mem128* to *xmm1*.
MOVDQA *xmm1/mem128*, *xmm2*	66 0F 7F /r	Moves aligned packed integer values from *xmm1* or *mem128* to *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVDQA *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	6F /r
VMOVDQA *xmm1/mem128*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	6F /r
VMOVDQA *ymm1*, *xmm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	7F /r
VMOVDQA *ymm1/mem256*, *ymm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	7F /r

Related Instructions

(V)MOVD, (V)MOVDQU, (V)MOVQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVDQU VMOVDQU — Move Unaligned Double Quadword

Moves unaligned packed integer values. Values can be moved from a register or a memory location to another register, or from a register to another register or a memory location.

There are legacy and extended forms of the instruction:

MOVDQU

Moves two unaligned quadwords (128-bit move). There are two encodings.

- The source operand is an XMM register. The destination is either another XMM register or a 128-bit memory location.
- The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

The extended form of the instruction has both 128-bit and 256-bit encoding.

VMOVDQU

XMM Encoding

Moves two unaligned quadwords (128-bit move). There are two encodings.

- The source operand is an XMM register. The destination is either another XMM register or a 128-bit memory location.
- The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves four unaligned quadwords (256-bit move). There are two encodings.

- The source operand is a YMM register. The destination is either another YMM register or a 256-bit memory location.
- The source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

MOVDQU is an SSE2 instruction and VMOVDQU is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVDQU *xmm1*, *xmm2/mem128*	F3 0F 6F /r	Moves unaligned packed integer values from *xmm2* or *mem128* to *xmm1*.
MOVDQU *xmm1/mem128*, *xmm2*	F3 0F 7F /r	Moves unaligned packed integer values from *xmm1* or *mem128* to *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVDQU *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	6F /r
VMOVDQU *xmm1/mem128*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	6F /r
VMOVDQU *ymm1*, *xmm2*/*mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.10	7F /r
VMOVDQU *ymm1*/*mem256*, *ymm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.10	7F /r

Related Instructions

(V)MOVD, (V)MOVDQA, (V)MOVQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVHLPS Move High to Low
VMOVHLPS Packed Single-Precision Floating-Point

Moves two packed single-precision floating-point values from the high quadword of an XMM register to the low quadword of another XMM register.

There are legacy and extended forms of the instruction:

MOVHLPS

The source operand is bits [127:64] of an XMM register. The destination is bits [63:0] of another XMM register. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVHLPS

The extended form of the instruction has 128-bit encoding.

The source operands are bits [127:64] of two XMM registers. The destination is a third XMM register. Bits [127:64] of the first source are moved to bits [127:64] of the destination; bits [127:64] of the second source are moved to bits [63:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVHLPS is an SSE instruction and VMOVHLPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVHLPS *xmm1*, *xmm2*	0F 12 /r	Moves two packed single-precision floating-point values from *xmm2[127:64]* to *xmm1[63:0]*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVHLPS *xmm1*, *xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	12 /r

Related Instructions

(V)MOVAPS, (V)MOVHPS, (V)MOVLHPS, (V)MOVLPS, (V)MOVMSKPS, (V)MOVSS, (V)MOVUPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVHPD VMOVHPD — Move High Packed Double-Precision Floating-Point

Moves a packed double-precision floating-point value. Values can be moved from a 64-bit memory location to the high-order quadword of an XMM register, or from the high-order quadword of an XMM register to a 64-bit memory location.

There are legacy and extended forms of the instruction:

MOVHPD

There are two encodings.

- The source operand is a 64-bit memory location. The destination is bits [127:64] of an XMM register.
- The source operand is bits [127:64] of an XMM register. The destination is a 64-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVHPD

The extended form of the instruction has two 128-bit encodings.

- There are two source operands. The first source is an XMM register. The second source is a 64-bit memory location. The destination is an XMM register. Bits [63:0] of the source register are written to bits [63:0] of the destination; bits [63:0] of the source memory location are written to bits [127:64] of the destination.
- The source operand is bits [127:64] of an XMM register. The destination is a 64-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVHPD is an SSE2 instruction and VMOVHPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVHPD *xmm1*, *mem64*	66 0F 16 /r	Moves a packed double-precision floating-point value from *mem64* to *xmm1[127:64].*
MOVHPD *mem64*, *xmm1*	66 0F 17 /r	Moves a packed double-precision floating-point value from *xmm1[127:64]* to *mem64*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVHPD *xmm1*, *xmm2*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	16 /r
VMOVHPD *mem64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	17 /r

Related Instructions

(V)MOVAPD, (V)MOVLPD, (V)MOVMSKPD, (V)MOVSD, (V)MOVUPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVHPS VMOVHPS — Move High Packed Single-Precision Floating-Point

Moves two packed single-precision floating-point value. Values can be moved from a 64-bit memory location to the high-order quadword of an XMM register, or from the high-order quadword of an XMM register to a 64-bit memory location.

There are legacy and extended forms of the instruction:

MOVHPS

There are two encodings.

- The source operand is a 64-bit memory location. The destination is bits [127:64] of an XMM register.
- The source operand is bits [127:64] of an XMM register. The destination is a 64-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVHPS

The extended form of the instruction has two 128-bit encodings.

- There are two source operands. The first source is an XMM register. The second source is a 64-bit memory location. The destination is an XMM register. Bits [63:0] of the source register are written to bits [63:0] of the destination; bits [63:0] of the source memory location are written to bits [127:64] of the destination.
- The source operand is bits [127:64] of an XMM register. The destination is a 64-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVHPS is an SSE instruction and VMOVHPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVHPS *xmm1*, *mem64*	0F 16 /r	Moves two packed double-precision floating-point value from *mem64* to *xmm1[127:64].*
MOVHPS *mem64*, *xmm1*	0F 17 /r	Moves two packed double-precision floating-point value from *xmm1[127:64]* to *mem64*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVHPS *xmm1*, *xmm2*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	16 /r
VMOVHPS *mem64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	17 /r

Related Instructions

(V)MOVAPS, (V)MOVHLPS, (V)MOVLHPS, (V)MOVLPS, (V)MOVMSKPS, (V)MOVSS, (V)MOVUPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination encoding only).
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVLHPS VMOVHLPS — Move Low to High Packed Single-Precision Floating-Point

Moves two packed single-precision floating-point values from the low quadword of an XMM register to the high quadword of another XMM register.

There are legacy and extended forms of the instruction:

MOVLHPS

The source operand is bits [63:0] of an XMM register. The destination is bits [127:64] of another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVLHPS

The extended form of the instruction has 128-bit encoding.

The source operands are bits [63:0] of two XMM registers. The destination is a third XMM register. Bits [63:0] of the first source are moved to bits [63:0] of the destination; bits [63:0] of the second source are moved to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVLHPS is an SSE instruction and VMOVLHPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVLHPS *xmm1*, *xmm2*	0F 16 /r	Moves two packed single-precision floating-point values from *xmm2[63:0]* to *xmm1[127:64]*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVLHPS *xmm1*, *xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	16 /r

(V)MOVAPS, (V)MOVHLPS, (V)MOVHPS, (V)MOVLPS, (V)MOVMSKPS, (V)MOVSS, (V)MOVUPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination encoding only).
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVLPD VMOVLPD — Move Low Packed Double-Precision Floating-Point

Moves a packed double-precision floating-point value. Values can be moved from a 64-bit memory location to the low-order quadword of an XMM register, or from the low-order quadword of an XMM register to a 64-bit memory location.

There are legacy and extended forms of the instruction:

MOVLPD

There are two encodings.

- The source operand is a 64-bit memory location. The destination is bits [63:0] of an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.
- The source operand is bits [63:0] of an XMM register. The destination is a 64-bit memory location. VMOVLPD

The extended form of the instruction has two 128-bit encodings.

- There are two source operands. The first source is an XMM register. The second source is a 64-bit memory location. The destination is an XMM register. Bits [127:64] of the source register are written to bits [127:64] of the destination; bits [63:0] of the source memory location are written to bits [63:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- The source operand is bits [63:0] of an XMM register. The destination is a 64-bit memory location.

MOVLPD is an SSE2 instruction and VMOVLPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVLPD *xmm1*, *mem64*	66 0F 12 /r	Moves a packed double-precision floating-point value from *mem64* to *xmm1[63:0].*
MOVHPD *mem64*, *xmm1*	66 0F 13 /r	Moves a packed double-precision floating-point value from *xmm1[63:0]* to *mem64*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVLPD *xmm1*, *xmm2*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	12 /r
VMOVLPD *mem64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	13 /r

Related Instructions

(V)MOVAPD, (V)MOVHPD, (V)MOVMSKPD, (V)MOVSD, (V)MOVUPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination encoding only).
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception A — AVX exception S — SSE exception				

MOVLPS VMOVLPS — Move Low Packed Single-Precision Floating-Point

Moves two packed single-precision floating-point values. Values can be moved from a 64-bit memory location to the low-order quadword of an XMM register, or from the low-order quadword of an XMM register to a 64-bit memory location.

There are legacy and extended forms of the instruction:

MOVLPS

There are two encodings.

- The source operand is a 64-bit memory location. The destination is bits [63:0] of an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.
- The source operand is bits [63:0] of an XMM register. The destination is a 64-bit memory location.

VMOVLPS

The extended form of the instruction has two 128-bit encodings.

- There are two source operands. The first source is an XMM register. The second source is a 64-bit memory location. The destination is an XMM register. Bits [127:64] of the source register are written to bits [127:64] of the destination; bits [63:0] of the source memory location are written to bits [63:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- The source operand is bits [63:0] of an XMM register. The destination is a 64-bit memory location.

MOVLPS is an SSE instruction and VMOVLPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVLPS *xmm1*, *mem64*	0F 12 /r	Moves two packed single-precision floating-point value from *mem64* to *xmm1[63:0].*
MOVLPS *mem64*, *xmm1*	0F 13 /r	Moves two packed single-precision floating-point value from *xmm1[63:0]* to *mem64*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VMOVLPS *xmm1*, *xmm2*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	12 /r
VMOVLPS *mem64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	13 /r

Related Instructions

(V)MOVAPS, (V)MOVHLPS, (V)MOVHPS, (V)MOVLHPS, (V)MOVMSKPS, (V)MOVSS, (V)MOVUPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination encoding only).
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVMSKPD VMOVMSKPD — Extract Sign Mask Packed Double-Precision Floating-Point

Extracts the sign bits of packed double-precision floating-point values from an XMM register, zero-extends the value, and writes it to the low-order bits of a general-purpose register.

There are legacy and extended forms of the instruction:

MOVMSKPD

Extracts two mask bits.

The source operand is an XMM register. The destination can be either a 64-bit or a 32-bit general purpose register. Writes the extracted bits to positions [1:0] of the destination and clears the remaining bits. Bits [255:128] of the YMM register that corresponds to the source are not affected.

MOVMSKPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Extracts two mask bits.

The source operand is an XMM register. The destination can be either a 64-bit or a 32-bit general purpose register. Writes the extracted bits to positions [1:0] of the destination and clears the remaining bits. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Extracts four mask bits.

The source operand is a YMM register. The destination can be either a 64-bit or a 32-bit general purpose register. Writes the extracted bits to positions [3:0] of the destination and clears the remaining bits.

MOVMSKPD is an SSE2 instruction and VMOVMSKPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVMSKPD *reg*, *xmm*	66 0F 50 /r	Move zero-extended sign bits of packed double-precision values from *xmm* to a general-purpose register.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVMSKPD *reg*, *xmm*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	50 /r
VMOVMSKPD *reg*, *ymm*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	50 /r

Related Instructions

(V)MOVMSKPS, (V)PMOVMSKB

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVMSKPS VMOVMSKPS — Extract Sign Mask Packed Single-Precision Floating-Point

Extracts the sign bits of packed single-precision floating-point values from an XMM register, zero-extends the value, and writes it to the low-order bits of a general-purpose register.

There are legacy and extended forms of the instruction:

MOVMSKPS

Extracts four mask bits.

The source operand is an XMM register. The destination can be either a 64-bit or a 32-bit general purpose register. Writes the extracted bits to positions [3:0] of the destination and clears the remaining bits.

MOVMSKPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Extracts four mask bits.

The source operand is an XMM register. The destination can be either a 64-bit or a 32-bit general purpose register. Writes the extracted bits to positions [3:0] of the destination and clears the remaining bits.

YMM Encoding

Extracts eight mask bits.

The source operand is a YMM register. The destination can be either a 64-bit or a 32-bit general purpose register. Writes the extracted bits to positions [7:0] of the destination and clears the remaining bits.

MOVMSKPS is an SSE instruction and VMOVMSKPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVMSKPS *reg*, *xmm*	0F 50 /r	Move zero-extended sign bits of packed single-precision values from *xmm* to a general-purpose register.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVMSKPS *reg*, *xmm*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	50 /r
VMOVMSKPS *reg*, *ymm*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	50 /r

Related Instructions

(V)MOVMSKPD, (V)PMOVMSKB

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVNTDQ VMOVNTDQ — Move Non-Temporal Double Quadword

Moves double quadword values from a register to a memory location.

Indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining (WC) memory write, which minimizes cache pollution. The method of minimization depends on the hardware implementation of the instruction. For further information, see “Memory Optimization” in Volume 1.

The instruction is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE or MFENCE instruction to force strong memory ordering of MOVNTDQ with respect to other stores.

There are legacy and extended forms of the instruction:

MOVNTDQ

Moves one 128-bit value.

The source operand is an XMM register. The destination is a 128-bit memory location.

MOVNTDQ

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves one 128-bit value.

The source operand is an XMM register. The destination is a 128-bit memory location.

YMM Encoding

Moves two 128-bit values.

The source operand is a YMM register. The destination is a 256-bit memory location.

MOVNTDQ is an SSE2 instruction and VMOVNTDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVNTDQ *mem128*, *xmm*	66 0F E7 /r	Moves a 128-bit value from *xmm* to *mem128*, minimizing cache pollution.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVNTDQ *mem128, xmm*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	E7 /r
VMOVNTDQ *mem256, ymm*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	E7 /r

Related Instructions

(V)MOVNTDQA, (V)MOVNTPD, (V)MOVNTPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVNTDQA VMOVNTDQA — Move Non-Temporal Double Quadword Aligned

Moves aligned double quadword values from a register to a memory location.

Indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining (WC) memory write, which minimizes cache pollution. The method of minimization depends on the hardware implementation of the instruction. For further information, see "Memory Optimization" in Volume 1.

The instruction is weakly-ordered with respect to other instructions that operate on memory. Software should use an MFENCE instruction to force strong memory ordering of MOVNTDQ with respect to other stores.

There are legacy and extended forms of the instruction:

MOVNTDQA

Moves one 128-bit value.

The source operand is an XMM register. The destination is a 128-bit memory location.

MOVNTDQ

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves one 128-bit value.

The source operand is an XMM register. The destination is a 128-bit memory location.

MOVNTDQA is an SSE4.1 instruction and VMOVNTDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVNTDQA *mem128*, *xmm*	66 0F 38 2A /r	Moves an aligned 128-bit value from *xmm* to *mem128*, minimizing cache pollution.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVNTDQA *mem128, xmm*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	2A /r

Related Instructions

(V)MOVNTDQ, (V)MOVNTPD, (V)MOVNTPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L field = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

MOVNTPD VMOVNTPD — Move Non-Temporal Packed Double-Precision Floating-Point

Moves packed double-precision floating-point values from a register to a memory location.

Indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining (WC) memory write, which minimizes cache pollution. The method of minimization depends on the hardware implementation of the instruction. For further information, see "Memory Optimization" in Volume 1.

The instruction is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE or MFENCE instruction to force strong memory ordering of MOVNTDQ with respect to other stores.

There are legacy and extended forms of the instruction:

MOVNTPD

Moves two values.

The source operand is an XMM register. The destination is a 128-bit memory location.

MOVNTPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves two values.

The source operand is an XMM register. The destination is a 128-bit memory location.

YMM Encoding

Moves four values.

The source operand is a YMM register. The destination is a 256-bit memory location.

MOVNTPD is an SSE2 instruction and VMOVNTPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVNTPD *mem128*, *xmm*	66 0F 2B /r	Moves two packed double-precision floating-point values from *xmm* to *mem128*, minimizing cache pollution.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVNTPD *mem128, xmm*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	2B /r
VMOVNTPD *mem256, ymm*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	2B /r

Related Instructions

MOVNTDQ, MOVNTI, MOVNTPS, MOVNTQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVNTPS VMOVNTPS — Move Non-Temporal PackedSingle-Precision Floating-Point

Moves packed single-precision floating-point values from a register to a memory location.

Indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the store as a write-combining (WC) memory write, which minimizes cache pollution. The method of minimization depends on the hardware implementation of the instruction. For further information, see "Memory Optimization" in Volume 1.

The instruction is weakly-ordered with respect to other instructions that operate on memory. Software should use an SFENCE or MFENCE instruction to force strong memory ordering of MOVNTDQ with respect to other stores.

There are legacy and extended forms of the instruction:

MOVNTPS

Moves four values.

The source operand is an XMM register. The destination is a 128-bit memory location.

MOVNTPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves four values.

The source operand is an XMM register. The destination is a 128-bit memory location.

YMM Encoding

Moves eight values.

The source operand is a YMM register. The destination is a 256-bit memory location.

MOVNTPS is an SSE instruction and VMOVNTPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVNTPS *mem128*, *xmm*	0F 2B /r	Moves four packed double-precision floating-point values from *xmm* to *mem128*, minimizing cache pollution.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVNTPS *mem128, xmm*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	2B /r
VMOVNTPS *mem256, ymm*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	2B /r

Related Instructions

(V)MOVNTDQ, (V)MOVNTDQA, (V)MOVNTPD, (V)MOVNTQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVQ VMOVQ — Move Quadword

Moves 64-bit values. The source is either the low-order quadword of an XMM register or a 64-bit memory location. The destination is either the low-order quadword of an XMM register or a 64-bit memory location. When the destination is a register, the 64-bit value is zero-extended to 128 bits.

There are legacy and extended forms of the instruction:

MOVQ

There are two encodings.

- The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either an XMM register or a 64-bit memory location. When the destination is a register, the 64-bit value is zero-extended to 128 bits.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVQ

The extended form of the instruction has three 128-bit encodings.

- The source operand is an XMM register. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is a 64-bit memory location. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either an XMM register or a 64-bit memory location. When the destination is a register, the 64-bit value is zero-extended to 128 bits.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVQ is an SSE2 instruction and VMOVQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVQ *xmm1, xmm2/mem64*	F3 0F 7E /r	Move a zero-extended 64-bit value from *xmm2* or *mem64* to *xmm1*.
MOVQ *xmm1/mem64*, *xmm2*	66 0F D6 /r	Move a 64-bit value from *xmm2* to *xmm1* or *mem64*. Zero-extends for register destination.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVQ *xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	7E /r
VMOVQ *xmm1*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	7E /r
VMOVQ *xmm1/mem64*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	D6 /r

Related Instructions

(V)MOVD, (V)MOVDQA, (V)MOVDQU

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVSD VMOVSD — Move Scalar Double-Precision Floating-Point

Moves scalar double-precision floating point values. The source is either a low-order quadword of an XMM register or a 64-bit memory location. The destination is either a low-order quadword of an XMM register or a 64-bit memory location.

There are legacy and extended forms of the instruction:

MOVSD

There are three encodings.

- The source operand is an XMM register. The destination is an XMM register. Bits [127:64] of the destination are not affected.
- The source operand is a 64-bit memory location. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either an XMM register or a 64-bit memory location. When the destination is a register, the 64-bit value is zero-extended to 128 bits.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVSD

The extended form of the instruction has four 128-bit encodings. Two of the encodings are functionally equivalent.

- The source operand is a 64-bit memory location. The destination is an XMM register. The 64-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is a 64-bit memory location.
- Two functionally-equivalent encodings:
  There are two source XMM registers. The destination is an XMM register. Bits [127:64] of the first source register are copied to bits [127:64] of the destination; the 64-bit value in bits [63:0] of the second source register is written to bits [63:0] of the destination.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVSD is an SSE2 instruction and VMOVSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

This instruction must not be confused with the MOVSD (move string doubleword) instruction of the general-purpose instruction set. Assemblers can distinguish the instructions by the number and type of operands.

Mnemonic	Opcode	Description
MOVSD *xmm1, xmm2*	F2 0F 10 /r	Moves a zero-extended 64-bit value from *xmm2* to *xmm1*.
MOVSD *xmm1, mem64*	F2 0F 10 /r	Moves a zero-extended 64-bit value from *mem64* to *xmm1*.
MOVSD *xmm2*/*mem64*, *xmm1*	F2 0F 11 /r	Moves a 64-bit value from *xmm1* to *xmm2* or *mem64*. Zero-extends for register destination.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVSD *xmm1*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.11	10 /r
VMOVSD *mem64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.11	11 /r
VMOVSD *xmm, xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.11	10 /r
VMOVSD *xmm, xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.11	11 /r

Related Instructions

(V)MOVAPD, (V)MOVHPD, (V)MOVLPD, (V)MOVMSKPD, (V)MOVUPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination enoding only).
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVSHDUP VMOVSHDUP — Move High and Duplicate Single-Precision

Moves and duplicates odd-indexed single-precision floating-point values.

There are legacy and extended forms of the instruction:

MOVSHDUP

Moves and duplicates two odd-indexed single-precision floating-point values.

The source operand is an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [127:96] of the source are duplicated and written to bits [127:96] and [95:64] of the destination. Bits [63:32] of the source are duplicated and written to bits [63:32] and [31:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVSHDUP

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves and duplicates two odd-indexed single-precision floating-point values.

The source operand is an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [127:96] of the source are duplicated and written to bits [127:96] and [95:64] of the destination. Bits [63:32] of the source are duplicated and written to bits [63:32] and [31:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves and duplicates four odd-indexed single-precision floating-point values.

The source operand is a YMM register or a 256-bit memory location. The destination is a YMM register. Bits [255:224] of the source are duplicated and written to bits [255:224] and [223:192] of the destination. Bits [191:160] of the source are duplicated and written to bits [191:160] and [159:128] of the destination. Bits [127:96] of the source are duplicated and written to bits [127:96] and [95:64] of the destination. Bits [63:32] of the source are duplicated and written to bits [63:32] and [31:0] of the destination.

MOVSHDUP is an SSE3 instruction and VMOVSHDUP is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVSHDUP *xmm1*, *xmm2*/*mem128*	F3 0F 16 /r	Moves and duplicates two odd-indexed single-precision floating-point values in *xmm2* or *mem128*. Writes to *xmm1*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VMOVSHDUP *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	16 /r
VMOVSHDUP *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.10	16 /r

Related Instructions

(V)MOVDDUP, (V)MOVSLDUP

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVSLDUP VMOVSLDUP — Move Low and Duplicate Single-Precision

Moves and duplicates even-indexed single-precision floating-point values.

There are legacy and extended forms of the instruction:

MOVSLDUP

Moves and duplicates two even-indexed single-precision floating-point values.

The source operand is an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [95:64] of the source are duplicated and written to bits [127:96] and [95:64] of the destination. Bits [31:0] of the source are duplicated and written to bits [63:32] and [31:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVSLDUP

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves and duplicates two even-indexed single-precision floating-point values.

The source operand is an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [95:64] of the source are duplicated and written to bits [127:96] and [95:64] of the destination. Bits [31:0] of the source are duplicated and written to bits [63:32] and [31:0] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves and duplicates four even-indexed single-precision floating-point values.

The source operand is a YMM register or a 256-bit memory location. The destination is a YMM register. Bits [223:192] of the source are duplicated and written to bits [255:224] and [223:192] of the destination. Bits [159:128] of the source are duplicated and written to bits [191:160] and [159:128] of the destination. Bits [95:64] of the source are duplicated and written to bits [127:96] and [95:64] of the destination. Bits [31:0] of the source are duplicated and written to bits [63:32] and [31:0] of the destination.

MOVSLDUP is an SSE3 instruction and VMOVSLDUP is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVSLDUP *xmm1*, *xmm2*/*mem128*	F3 0F 12 /r	Moves and duplicates two even-indexed single-precision floating-point values in *xmm2* or *mem128*. Writes to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVSLDUP *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	12 /r
VMOVSLDUP *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.10	12 /r

Related Instructions

(V)MOVDDUP, (V)MOVSHDUP

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVSS VMOVSS — Move Scalar Single-Precision Floating-Point

Moves scalar single-precision floating point values. The source is either a low-order doubleword of an XMM register or a 32-bit memory location. The destination is either a low-order doubleword of an XMM register or a 32-bit memory location.

There are legacy and extended forms of the instruction:

MOVSS

There are three encodings.

- The source operand is an XMM register. The destination is an XMM register. Bits [127:32] of the destination are not affected.
- The source operand is a 32-bit memory location. The destination is an XMM register. The 32-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is either an XMM register or a 32-bit memory location. When the destination is a register, the 32-bit value is zero-extended to 128 bits.

Bits [255:128] of the YMM register that corresponds to the source are not affected.

VMOVSS

The extended form of the instruction has four 128-bit encodings. Two of the encodings are functionally equivalent.

- The source operand is a 32-bit memory location. The destination is an XMM register. The 32-bit value is zero-extended to 128 bits.
- The source operand is an XMM register. The destination is a 32-bit memory location.
- Two functionally-equivalent encodings:
  There are two source XMM registers. The destination is an XMM register. Bits [127:64] of the first source register are copied to bits [127:64] of the destination; the 32-bit value in bits [31:0] of the second source register is written to bits [31:0] of the destination.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MOVSS is an SSE instruction and VMOVSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVSS *xmm1, xmm2*	F3 0F 10 /r	Moves a 32-bit value from *xmm2* to *xmm1*.
MOVSS *xmm1, mem32*	F3 0F 10 /r	Moves a zero-extended 32-bit value from *mem32* to *xmm1*.
MOVSS *xmm2/mem32*, *xmm1*	F3 0F 11 /r	Moves a 32-bit value from *xmm1* to *xmm2* or *mem32.* Zero-extended for register destination.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVSS *xmm1*, *mem32*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	10 /r
VMOVSS *mem32*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	11 /r
VMOVSS *xmm, xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.10	10 /r
VMOVSS *xmm, xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.10	11 /r

Related Instructions

(V)MOVAPS, (V)MOVHLPS, (V)MOVHPS, (V)MOVLHPS, (V)MOVLPS, (V)MOVMSKPS, (V)MOVUPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination enoding only).
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVUPD VMOVUPD — Move Unaligned Packed Double-Precision Floating-Point

Moves packed double-precision floating-point values. Values can be moved from a register or memory location to a register; or from a register to a register or memory location.

A memory operand that is not aligned does not cause a general-protection exception.

There are legacy and extended forms of the instruction:

MOVUPD

Moves two double-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVUPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves two double-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves four double-precision floating-point values. There are encodings for each type of move.

- The source operand is either a YMM register or a 256-bit memory location. The destination operand is a YMM register.
- The source operand is a YMM register. The destination operand is either a YMM register or a 256-bit memory location.

MOVUPD is an SSE2 instruction and VMOVUPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVUPD *xmm1*, *xmm2/mem128*	66 0F 10 /r	Moves two packed double-precision floating-point values from *xmm2* or *mem128* to *xmm1*.
MOVUPD *xmm1/mem128*, *xmm2*	66 0F 11 /r	Moves two packed double-precision floating-point values from *xmm1* or *mem128* to *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVUPD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	10 /r
VMOVUPD *xmm1/mem128*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	11 /r
VMOVUPD *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	10 /r
VMOVUPD *ymm1/mem256*, *ymm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	11 /r

Related Instructions

(V)MOVAPD, (V)MOVHPD, (V)MOVLPD, (V)MOVMSKPD, (V)MOVSD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MOVUPS VMOVUPS — Move Unaligned Packed Single-Precision Floating-Point

Moves packed single-precision floating-point values. Values can be moved from a register or memory location to a register; or from a register to a register or memory location.

A memory operand that is not aligned does not cause a general-protection exception.

There are legacy and extended forms of the instruction:

MOVUPS

Moves four single-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMOVUPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Moves four single-precision floating-point values. There are encodings for each type of move.

- The source operand is either an XMM register or a 128-bit memory location. The destination operand is an XMM register.
- The source operand is an XMM register. The destination operand is either an XMM register or a 128-bit memory location.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Moves eight single-precision floating-point values. There are encodings for each type of move.

- The source operand is either a YMM register or a 256-bit memory location. The destination operand is a YMM register.
- The source operand is a YMM register. The destination operand is either a YMM register or a 256-bit memory location.

MOVUPS is an SSE instruction and VMOVUPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MOVUPS *xmm1*, *xmm2/mem128*	0F 10 /r	Moves four packed single-precision floating-point values from *xmm2* or unaligned *mem128* to *xmm1*.
MOVUPS *xmm1/mem128*, *xmm2*	0F 11 /r	Moves four packed single-precision floating-point values from *xmm1* or unaligned *mem128* to *xmm2*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVUPS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	10 /r
VMOVUPS *xmm1/mem128*, *xmm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	11 /r
VMOVUPS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	10 /r
VMOVUPS *ymm1/mem256*, *ymm2*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	11 /r

Related Instructions

(V)MOVAPS, (V)MOVHLPS, (V)MOVHPS, (V)MOVLHPS, (V)MOVLPS, (V)MOVMSKPS, (V)MOVSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MPSADBW VMPSADBW Multiple Sum of Absolute Differences

Calculates the sum of absolute differences of each member of four sequential groups of four unsigned byte integers in the first source register and each of four unsigned byte integers in a second source register, and writes the 16-bit integer sums to the destination.

Bit fields in an 8-bit immediate operand are used to calculate offsets that select sequences of bytes in the two source registers. The binary value of each bit field is multiplied by 32 to produce a 32-bit offset. Bit 2 of the immediate operand determines the offset for the first source register; 11 bytes beginning at the offset position are used. Bits [1:0] of the immediate operand determine the offset for the second source register; four bytes beginning at the offset position are used.

The selected bytes are repositioned in the source registers. Bytes [10:0] of the first source occupy bits [87:0] of the first source register; bytes [3:0] of the second source occupy bits [31:0] of the second source register.

Operation is iterative and repeats eight times. Each repetition increments the starting byte position in the first source by one and calculates the sum of differences with the four integers of the second source. Results are written to eight consecutive words in the destination, starting with the low word. In the first iteration, bytes [0:4] of the second source are subtracted from bytes [0:4] of the first source and the sum of the differences is written to bits [15:0] of the destination; in the second iteration, bytes [0:4] of the second source are subtracted from bytes [1:5] of the first source and the sum of the differences is written to bits [31:16] of the destination. The process continues until bytes [0:4] of the second source are subtracted from bytes [7:10] of the first source and the sum of the differences is written to bits [127:112] of the destination.

There are legacy and extended forms of the instruction:

MPSADBW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMPSADBW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MPSADBW is an SSE4.1 instruction and VMPSADBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MPSADBW *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 42 /r ib	Sums absolute difference of groups of four 8-bit integer in *xmm1 and xmm2 or mem128.* Writes results to *xmm1*. Starting source offsets are determined by *imm8* bit fields.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VMPSADBW *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	42 /r

Related Instructions

(V)PSADBW, (V)PABSB, (V)PABSD, (V)PABSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MULPD VMULPD — Multiply Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source operand by the corresponding packed double-precision floating-point value of the second source operand and writes the product of each multiplication into the corresponding quadword of the destination.

There are legacy and extended forms of the instruction:

MULPD

Multiplies two double-precision floating-point values in the first source XMM register by the corresponding double precision floating-point values in either a second XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMULPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Multiplies two double-precision floating-point values in the first source XMM register by the corresponding double-precision floating-point values in either a second source XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Multiplies four double-precision floating-point values in the first source YMM register by the corresponding double precision floating-point values in either a second source YMM register or a 256-bit memory location. The destination is a third YMM register.

MULPD is an SSE2 instruction and VMULPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MULPD *xmm1, xmm2/mem128*	66 0F 59 /r	Multiplies two packed double-precision floating-point values in *xmm1* by corresponding values in *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMULPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	59 /r
VMULPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	59 /r

Related Instructions

(V)MULPS, (V)MULSD, (V)MULSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MULPS VMULPS — Multiply Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source operand by the corresponding packed single-precision floating-point value of the second source operand and writes the product of each multiplication into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

MULPS

Multiplies four single-precision floating-point values in the first source XMM register by the corresponding single-precision floating-point values of either a second source XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VMULPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Multiplies four single-precision floating-point values in the first source XMM register by the corresponding single-precision floating-point values of either a second source XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Multiplies eight single-precision floating-point values in the first source YMM register by the corresponding single-precision floating-point values of either a second source YMM register or a 256-bit memory location. Writes the results to a third YMM register.

MULPS is an SSE2 instruction and VMULPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MULPS *xmm1*, *xmm2/mem128*	0F 59 /r	Multiplies four packed single-precision floating-point values in *xmm1* by corresponding values in *xmm2* or *mem128*. Writes the products to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMULPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	59 /r
VMULPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	59 /r

Related Instructions

(V)MULPD, (V)MULSD, (V)MULSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

MULSD VMULSD — Multiply Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value in the low-order quadword of the first source operand by the double-precision floating-point value in the low-order quadword of the second source operand and writes the product into the low-order quadword of the destination.

There are legacy and extended forms of the instruction:

MULSD

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The first source register is also the destination register. Bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are not affected.

VMULSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The destination is a third XMM register. Bits [127:64] of the first source operand are copied to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MULSD is an SSE2 instruction and VMULSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MULSD *xmm1*, *xmm2/mem64*	F2 0F 59 /r	Multiplies low-order double-precision floating-point values in *xmm1* by corresponding values in *xmm2* or *mem64*. Writes the products to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMULSD *xmm1*, *xmm2*, *xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.11	59 /r

Related Instructions

(V)MULPD, (V)MULPS, (V)MULSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

MULSS VMULSS Multiply Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value in the low-order doubleword of the first source operand by the single-precision floating-point value in the low-order doubleword of the second source operand and writes the product into the low-order doubleword of the destination.

There are legacy and extended forms of the instruction:

MULSS

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The first source register is also the destination. Bits [127:32] of the destination register and bits [255:128] of the corresponding YMM register are not affected.

VMULSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The destination is a third XMM register. Bits [127:32] of the first source register are copied to bits [127:32] of the of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

MULSS is an SSE instruction and VMULSS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
MULSS *xmm1*, *xmm2*/*mem32*	F3 0F 59 /r	Multiplies a single-precision floating-point value in the low-order doubleword of *xmm1* by a corresponding value in *xmm2* or *mem32*. Writes the product to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMULSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.10	59 /r

Related Instructions

(V)MULPD, (V)MULPS, (V)MULSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ORPD VORPD — OR Packed Double-Precision Floating-Point

Performs bitwise OR of two packed double-precision floating-point values in the first source operand with the corresponding two packed double-precision floating-point values in the second source operand and writes the results into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

ORPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VORPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ORPD is an SSE2 instruction and VORPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ORPD *xmm1*, *xmm2/mem128*	66 0F 56 /r	Performs bitwise OR of two packed double-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VORPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	56 /r
VORPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.01	56 /r

Related Instructions

(V)ANDNPS, (V)ANDPD, (V)ANDPS, (V)ORPS, (V)XORPD, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ORPS OR
VORPS Packed Single-Precision Floating-Point

Performs bitwise OR of the four packed single-precision floating-point values in the first source operand with the corresponding four packed single-precision floating-point values in the second source operand, and writes the result into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

ORPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VORPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

ORPS is an SSE instruction and VORPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ORPS *xmm1*, *xmm2/mem128*	0F 56 /r	Performs bitwise OR of four packed double-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VORPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	56 /r
VORPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	56 /r

Related Instructions

(V)ANDNPD, (V)ANDNPS, (V)ANDPD, (V)ANDPS, (V)ORPD, (V)XORPD, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PABSB VPABSB — Packed Absolute Value Signed Byte

Computes the absolute value of 16 packed 8-bit signed integers in the source operand and writes 8-bit unsigned results to the destination.

There are legacy and extended forms of the instruction:

PABSB

The source operand is an XMM register or a 128-bit memory location. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPABSB

The extended form of the instruction has 128-bit encoding.

The source operand is an XMM register or a 128-bit memory location. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PABSB is an SSSE3 instruction and VPABSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PABSB *xmm1, xmm2/mem128*	0F 38 1C /r	Computes the absolute value of each packed 8-bit signed integer value in *xmm2/mem128* and writes the 8-bit unsigned results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPABSB *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	1C /r

Related Instructions

(V)PABSW, (V)PABSD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PABSD VPABSD — Packed Absolute Value Signed Doubleword

Computes the absolute value of two packed 32-bit signed integers in the source operand and writes 32-bit unsigned results to the destination.

There are legacy and extended forms of the instruction:

PABSD

The source operand is an XMM register or a 128-bit memory location. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPABSD

The extended form of the instruction has 128-bit encoding.

The source operand is an XMM register or a 128-bit memory location. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PABSD is an SSSE3 instruction and VPABSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PABSD *xmm1, xmm2/mem128*	0F 38 1E /r	Computes the absolute value of each packed 32-bit signed integer value in *xmm2/mem128* and writes the 32-bit unsigned results to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPABSD *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	1E /r

Related Instructions

(V)PABSB, (V)PABSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PABSW VPABSW — Packed Absolute Value Signed Word

Computes the absolute values of four packed 16-bit signed integers in the source operand and writes 16-bit unsigned results to the destination.

There are legacy and extended forms of the instruction:

PABSW

The source operand is an XMM register or a 128-bit memory location. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPABSW

The extended form of the instruction has 128-bit encoding.

The source operand is an XMM register or a 128-bit memory location. The destination is another XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PABSW is an SSSE3 instruction and VPABSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PABSW *xmm1, xmm2/mem128*	0F 38 1D /r	Computes the absolute value of each packed 16-bit signed integer value in *xmm2/mem128* and writes the 16-bit unsigned results to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPABSW *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	1D /r

Related Instructions

(V)PABSB, (V)PABSD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PACKSSDW VPACKSSDW — Pack with Signed Saturation Doubleword to Word

Converts four 32-bit signed integers from the first source operand and four 32-bit signed integers from the second source operand into eight 16-bit signed integers and packs the results into the destination.

Positive source value greater than 7FFFh are saturated to 7FFFh; negative source values less than 8000h are saturated to 8000h.

Converted values from the first source operand are packed into the low-order words of the destination; converted values from the second source operand are packed into the high-order words of the destination.

There are legacy and extended forms of the instruction:

PACKSSDW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPACKSSDW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PACKSSDW is an SSE2 instruction and VPACKSSDW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PACKSSDW *xmm1*, *xmm2/mem128*	66 0F 6B /r	Converts 32-bit signed integers in *xmm1* and *xmm2* or *mem128* into 16-bit signed integers with saturation. Writes packed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPACKSSDW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	0.$\overline{\text{src1}}$.0.01	6B /r

Related Instructions

(V)PACKSSWB, (V)PACKUSDW, (V)PACKUSWB

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PACKSSWB VPACKSSWB — Pack with Signed Saturation Word to Byte

Converts eight 16-bit signed integers from the first source operand and eight 16-bit signed integers from the second source operand into sixteen 8-bit signed integers and packs the results into the destination.

Positive source values greater than 7Fh are saturated to 7Fh; negative source values less than 80h are saturated to 80h.

Converted values from the first source operand are packed into the low-order bytes of the destination; converted values from the second source operand are packed into the high-order bytes of the destination.

There are legacy and extended forms of the instruction:

PACKSSWB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPACKSSWB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PACKSSWB is an SSE2 instruction and VPACKSSWB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PACKSSWB *xmm1*, *xmm2/mem128*	66 0F 63 /r	Converts 16-bit signed integers in *xmm1* and *xmm2* or *mem128* into 8-bit signed integers with saturation. Writes packed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPACKSSWB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	63 /r

Related Instructions

(V)PACKSSDW, (V)PACKUSDW, (V)PACKUSWB

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PACKUSDW VPACKUSDW — Pack with Unsigned Saturation Doubleword to Word

Converts four 32-bit signed integers from the first source operand and four 32-bit signed integers from the second source operand into eight 16-bit unsigned integers and packs the results into the destination.

Source values greater than FFFFh are saturated to FFFFh; source values less than 0000h are saturated to 0000h.

Packs converted values from the first source operand into the low-order words of the destination; packs converted values from the second source operand into the high-order words of the destination.

There are legacy and extended forms of the instruction:

PACKUSDW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPACKUSDW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PACKUSDW is an SSE4.1 instruction and VPACKUSDW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PACKUSDW *xmm1*, *xmm2/mem128*	66 0F 38 2B /r	Converts 32-bit signed integers in *xmm1* and *xmm2* or *mem128* into 16-bit unsigned integers with saturation. Writes packed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPACKUSDW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	2B /r

Related Instructions

(V)PACKSSDW, (V)PACKSSWB, (V)PACKUSWB

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PACKUSWB VPACKUSWB — Pack with Unsigned Saturation Word to Byte

Converts eight 16-bit signed integers from the first source operand and eight 16-bit signed integers from the second source operand into sixteen 8-bit unsigned integers and packs the results into the destination.

When a source value is greater than 7Fh it is saturated to FFh; when source value is less than 00h, it is saturated to 00h.

Packs converted values from the first source operand into the low-order bytes of the destination; packs converted values from the second source operand into the high-order bytes of the destination.

There are legacy and extended forms of the instruction:

PACKUSWB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPACKUSWB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PACKUSWB is an SSE2 instruction and VPACKUSWB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PACKUSWB *xmm1*, *xmm2/mem128*	66 0F 67 /r	Converts 16-bit signed integers in *xmm1* and *xmm2* or *mem128* into 8-bit signed integers with saturation. Writes packed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPACKUSWB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	67 /r

Related Instructions

(V)PACKSSDW, (V)PACKSSWB, (V)PACKUSDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PADDB VPADDB — Packed Add Bytes

Adds 16 packed 8-bit integer values in the first source operand to corresponding values in the second source operand and writes the integer sums to the corresponding bytes of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PADDB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDB is an SSE2 instruction and VPADDB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDB *xmm1*, *xmm2/mem128*	66 0F FC /r	Adds packed byte integer values in *xmm1* and *xmm2* or *mem128* Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	FC /r

Related Instructions

(V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDD VPADDD — Packed Add Doublewords

Adds four packed 32-bit integer value in the first source operand to corresponding values in the second source operand and writes integer sums to the corresponding doublewords of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 32 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PADDD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDD is an SSE2 instruction and VPADDD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDD *xmm1*, *xmm2/mem128*	66 0F FE /r	Adds packed doubleword integer values in *xmm1* and *xmm2* or *mem128* Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	FE /r

Related Instructions

(V)PADDB, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDQ VPADDQ — Packed Add Quadwords

Adds two packed 64-bit integer values in the first source operand to corresponding values in the second source operand and writes the integer sums to the corresponding quadwords of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 64 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PADDQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDQ is an SSE2 instruction and VPADDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDQ *xmm1*, *xmm2/mem128*	66 0F D4 /r	Adds packed quadword integer values in *xmm1* and *xmm2* or *mem128* Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDQ *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	D4 /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDSB VPADDSB — Packed Add with Signed Saturation Bytes

Adds 16 packed 8-bit signed integer values in the first source operand to the corresponding values in the second source operand and writes the signed integer sums to corresponding bytes of the destination.

Positive sums greater than 7Fh are saturated to FFh; negative sums less than 80h are saturated to 80h.

There are legacy and extended forms of the instruction:

PADDSB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDSB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDSB is an SSE2 instruction and VPADDSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDSB *xmm1*, *xmm2/mem128*	66 0F EC /r	Adds packed signed 8-bit integer values in *xmm1* and *xmm2* or *mem128* with signed saturation. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDSB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	EC /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDSW VPADDSW — Packed Add with Signed Saturation Words

Adds eight packed 16-bit signed integer value in the first source operand to the corresponding values in the second source operand and writes the signed integer sums to the corresponding words of the destination.

Positive sums greater than 7FFFh are saturated to 7FFFh; negative sums less than 8000h are saturated to 8000h.

There are legacy and extended forms of the instruction:

PADDSW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDSW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDSW is an SSE2 instruction and VPADDSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDSW *xmm1*, *xmm2/mem128*	66 0F ED /r	Adds packed signed 16-bit integer values in *xmm1* and *xmm2* or *mem128* with signed saturation. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDSW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	ED /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDUSB, (V)PADDUSW, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDUSB VPADDUSB — Packed Add with Unsigned Saturation Bytes

Adds 16 packed 8-bit unsigned integer values in the first source operand to the corresponding values in the second source operand and writes the unsigned integer sums to the corresponding bytes of the destination.

Sums greater than 7Fh are saturated to 7Fh; Sums less than 00h are saturated to 00h.

There are legacy and extended forms of the instruction:

PADDUSB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDUSB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDUSB is an SSE2 instruction and VPADDUSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDUSB *xmm1*, *xmm2/mem128*	66 0F DC /r	Adds packed unsigned 8-bit integer values in *xmm1* and *xmm2* or *mem128* with unsigned saturation. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDUSB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	DC /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSW, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDUSW VPADDUSW — Packed Add with Unsigned Saturation Words

Adds eight packed 16-bit unsigned integer value in the first source operand to the corresponding values in the second source operand and writes the unsigned integer sums to the corresponding words of the destination.

Sums greater than FFFFh are saturated to FFFFh; sums less than 0000h are saturated to 0000h.

There are legacy and extended forms of the instruction:

PADDUSW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDUSW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDUSW is an SSE2 instruction and VPADDUSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDUSW *xmm1*, *xmm2/mem128*	66 0F DD /r	Adds packed unsigned 16-bit integer values in *xmm1* and *xmm2* or *mem128* with unsigned saturation. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDUSW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	DD /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PADDW VPADDW — Packed Add Words

Adds eight packed 16-bit integer value in the first source operand to the corresponding values in the second source operand and writes the integer sums to the corresponding word of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 16 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PADDW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPADDW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PADDW is an SSE2 instruction and VPADDW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PADDW *xmm1*, *xmm2/mem128*	66 0F FD /r	Adds packed 16-bit integer values in *xmm1* and *xmm2* or *mem128*. Writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPADDW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	FD /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PALIGNR VPALIGNR — Packed Align Right

Concatenates [source1:source2] in a temporary 256-bit location and right-shifts the concatenated value the number of bytes specified by the unsigned immediate operand. Writes the shifted result to the destination.

The binary value of the immediate operand determineS the byte shift value. For byte shifts greater than 16 bytes, the upper bytes of the destination are zero-filled; when the byte shift is greater than 32 bytes, the destination is zeroed.

There are two forms of the instruction.

PALIGNR

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPALIGNR

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PALIGNR is an SSSE3 instruction and VPALIGNR is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PALIGNR *xmm1, xmm2/mem128, imm8*	66 0F 3A 0F /r ib	Right-shifts *xmm1:xmm2/mem128 imm8* bytes. Writes shifted result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPALIGNR *xmm1, xmm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	0F /r ib

Related Instructions

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PAND VPAND Packed AND

Performs a bitwise AND of the packed values in the first and second source operands and writes the result to the destination.

There are legacy and extended forms of the instruction:

PAND

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPAND

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PAND is an SSE2 instruction and VPAND is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PAND *xmm1*, *xmm2/mem128*	66 0F DB /r	Performs bitwise AND of values in *xmm1* and *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPAND *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	DB /r

Related Instructions

(V)PANDN, (V)POR, (V)PXOR

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PANDN VPANDN — Packed AND NOT

Generates the ones' complement of the value in the first source operand and performs a bitwise AND of the complement and the value in the second source operand. Writes the result to the destination.

There are legacy and extended forms of the instruction:

PANDN

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPANDN

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PANDN is an SSE2 instruction and VPANDN is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PANDN *xmm1*, *xmm2/mem128*	66 0F DF /r	Generates ones' complement of *xmm1*, then performs bitwise AND with value in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPANDN *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	DF /r

Related Instructions

(V)PAND, (V)POR, (V)PXOR

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PAVGB VPAVGB — Packed Average Unsigned Bytes

Computes the rounded averages of 16 packed unsigned 8-bit integer values in the first source operand and the corresponding values of the second source operand. Writes each average to the corresponding byte of the destination.

An average is computed by adding pairs of operands, adding 1 to a 9-bit temporary sum, and right-shifting the temporary sum by one bit position.

There are legacy and extended forms of the instruction:

PAVGB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPAVGB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PAVGB is an SSE2 instruction and VPAVGB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PAVGB *xmm1*, *xmm2/mem128*	66 0F E0 /r	Averages pairs of packed 8-bit unsigned integer values in *xmm1* and *xmm2* or *mem128*. Writes the averages to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPAVGB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	E0 /r

Related Instructions

PAVGW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PAVGW VPAVGW — Packed Average Unsigned Words

Computes the rounded average of packed unsigned 16-bit integer values in the first source operand and the corresponding values of the second source operand. Writes each average to the corresponding word of the destination.

An average is computed by adding pairs of operands, adding 1 to a 17-bit temporary sum, and right-shifting the temporary sum by one bit position.

There are legacy and extended forms of the instruction:

PAVGW

The first source operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The destination is the same XMM register as the first source operand; the upper 128-bits of the corresponding YMM register are not affected.

VPAVGW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PAVGW is an SSE2 instruction and VPAVGW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PAVGW *xmm1*, *xmm2/mem128*	66 0F E3 /r	Averages pairs of packed 16-bit unsigned integer values in *xmm1* and *xmm2* or *mem128*. Writes the averages to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPAVGW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	E3 /r

Related Instructions

(V)PAVGB

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PBLENDVB VPBLENDVB — Variable Blend Packed Bytes

Copies packed bytes from either of two sources to a destination, as specified by a mask operand.

The mask is defined by the msb of each byte of the mask operand. The position of a mask bit corresponds to the position of the most significant bit of a copied value.

- When a mask bit = 0, the specified element of the first source is copied to the corresponding position in the destination.
- When a mask bit = 1, the specified element of the second source is copied to the corresponding position in the destination.

There are legacy and extended forms of the instruction:

PBLENDVB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected. The mask operand is the implicit register XMM0.

VPBLENDVB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared. The mask operand is a fourth XMM register selected byte bits [7:4] of an immediate byte.

PBLENDVB is an SSE4.1 instruction and VPBLENDVB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PBLENDVB *xmm1*, *xmm2/mem128*	66 0F 38 10 /r	Selects byte values from *xmm1* or *xmm2/mem128*, depending on the value of corresponding mask bits in XMM0. Writes the selected values to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPBLENDVB *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	4C /r is4

Related Instructions

(V)BLENDVPD, (V)BLENDVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PBLENDW VPBLENDW — Blend Packed Words

Copies packed words from either of two sources to a destination, as specified by an immediate 8-bit mask operand.

Each mask bit corresponds to a source word value, in ascending order. Mask bit [0] corresponds to source bits [15:0], mask bit [7] corresponds to source bits [127:112].

- When a mask bit = 0, the specified element of the first source is copied to the corresponding position in the destination.
- When a mask bit = 1, the specified element of the second source is copied to the corresponding position in the destination.

There are legacy and extended forms of the instruction:

PBLENDW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPBLENDW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PBLENDW is an SSE4.1 instruction and VPBLENDW is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers (see the CPUID Specification, order# 25481).

Mnemonic	Opcode	Description
PBLENDW *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 0E /r ib	Selects word values from *xmm1* or *xmm2/mem128*, as specified by *imm8*. Writes the selected values to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPBLENDW *xmm1, xmm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	0E /r /ib

Related Instructions

(V)BLENDPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCLMULQDQ VPCLMULQDQ — Carry-less Multiply Quadwords

Performs a carry-less multiplication of a selected quadword element of the first source operand by a selected quadword element of the second source operand and writes the product to the destination.

Carry-less multiplication, also known as *binary polynomial multiplication*, is the mathematical operation of computing the product of two operands without generating or propagating carries. It is an essential component of cryptographic processing, and typically requires a large number of cycles.

The instruction provides an efficient means of performing the operation and is particularly useful in implementing the Galois counter mode used in the Advanced Encryption Standard (AES). See Section 1.4, “AES Instructions” for additional information.

Bits 4 and 0 of an 8-bit immediate byte operand specify which quadword of each source operand to multiply, as follows.

Mnemonic	Imm[0]	Imm[4]	Quadword Operands Selected
(V)PCLMULLQLQDQ	0	0	SRC1[63:0], SRC2[63:0]
(V)PCLMULHQLQDQ	1	0	SRC1[127:64], SRC2[63:0]
(V)PCLMULLQHQDQ	0	1	SRC1[63:0], SRC2[127:64]
(V)PCLMULHQHQDQ	1	1	SRC1[127:64], SRC2[127:64]

Alias mnemonics are provided for the various immediate byte combinations.

There are legacy and extended forms of the instruction:

PCLMULQDQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCLMULQDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCLMULQDQ is a CLMUL instruction and VPCLMULQDQ is both a CLMUL instruction and an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[PCLMULQDQ] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCLMULQDQ *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 44 /r ib	Performs carry-less multiplication of a selected quadword element of *xmm1* by a selected quadword element of *xmm2* or *mem128.* Elements are selected by bits 4 and 0 of *imm8.* Writes the product to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCLMULQDQ *xmm1*, *xmm2*, *xmm3/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	44 /r ib

Related Instructions

(V)PMULDQ, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception A — AVX exception S — SSE exception				

PCMPEQB VPCMPEQB — Packed Compare Equal Bytes

Compares 16 packed byte values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding byte of the destination.

When values are equal, the result is FFh; when values are not equal, the result is 00h.

There are legacy and extended forms of the instruction:

PCMPEQB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPEQB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPEQB is an SSE2 instruction and VPCMPEQB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPEQB *xmm1*, *xmm2/mem128*	66 0F 74 /r	Compares packed bytes in *xmm1* to packed bytes in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPEQB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	74 /r

Related Instructions

(V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPEQD VPCMPEQD — Packed Compare Equal Doublewords

Compares four packed doubleword values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding doubleword of the destination.

When values are equal, the result is FFFFFFFFh; when values are not equal, the result is 00000000h.

There are legacy and extended forms of the instruction:

PCMPEQD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPEQD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPEQD is an SSE2 instruction and VPCMPEQD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPEQD *xmm1*, *xmm2/mem128*	66 0F 76 /r	Compares packed doublewords in *xmm1* to packed doublewords in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPEQD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	76 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPEQQ VPCMPEQQ — Packed Compare Equal Quadwords

Compares two packed quadword values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding quadword of the destination.

When values are equal, the result is FFFFFFFFFFFFFFFFh; when values are not equal, the result is 0000000000000000h.

There are legacy and extended forms of the instruction:

PCMPEQQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPEQQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPEQQ is an SSE4.1 instruction and VPCMPEQQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPEQQ *xmm1*, *xmm2/mem128*	66 0F 38 29 /r	Compares packed quadwords in *xmm1* to packed quadwords in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPEQQ *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	29 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPEQW VPCMPEQW — Packed Compare Equal Words

Compares four packed word values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding word of the destination.

When values are equal, the result is FFFFh; when values are not equal, the result is 0000h.

There are legacy and extended forms of the instruction:

PCMPEQW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPEQW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPEQW is an SSE2 instruction and VPCMPEQW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPEQW *xmm1*, *xmm2/mem128*	66 0F 75 /r	Compares packed words in *xmm1* to packed words in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPEQW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	75 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPESTRI VPCMPESTRI — Packed Compare Explicit Length Strings Return Index

Compares character string data in the first and second source operands. Aggregation and comparison operations are carried out as determined by values of the fields of the immediate operand. Writes an index to the ECX register.

See Section 1.5, “String Compare Instructions” for information about immediate byte definition, compare encoding and control functions.

Each string has two associated data items: The actual characters in the two source operands, and length data in two implicit registers.

The absolute value of the data in the EAX/RAX register represents the length of the character string in the first source operand; the absolute value of the data in the EDX/RDX register represents the length of the character string in the second source operand.

The Data Size bit of the immediate operand specifies length parameters. When Data Size = 0, length values represent bytes, and saturate to 16 if greater than 16 or less than –16; when Data Size = 1, length values represent words, and saturate to 8 if greater than 8 or less than –8;

The values of the Mode and Byte fields of the immediate operand determine how the aggregation and comparison operations are carried out.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if intermediate result is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	1 if the absolute value of EDX is less than 16 (8); otherwise cleared.
SF	Set if the absolute value of EAX is less than 16 (8); otherwise cleared.
OF	Equal to the value of InterimResult2[0].

There are legacy and extended forms of the instruction:

PCMPESTRI

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A result index is written to the ECX register.

VPCMPESTRI

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A result index is written to the ECX register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPESTRI is an SSE4.2 instruction and VPCMPESTRI is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE42] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPESTRI *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 61 /r ib	Compares packed string data in *xmm1* and *xmm2* or *mem128*. Writes a result index to the ECX register.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPESTRI *xmm1, xmm2/mem128,* imm8	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	61 /r ib

Related Instructions

(V)PCMPESTRM, (V)PCMPISTRI, (V)PCMPISTRM

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								M				M	M	M	M	M
21	20	19	18	17	16	14	13 : 12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag that is set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PCMPESTRM VPCMPESTRM — Packed Compare Explicit Length Strings Return Mask

Compares character string data in the first and second source operands. Aggregation and comparison operations are carried out as determined by values of the fields of the immediate operand. Writes a mask value to the implicit XMM0 register.

See Section 1.5, "String Compare Instructions" for information about immediate byte definition, compare encoding and control functions.

Each string has two associated data items: The actual characters in the two source operands, and length data in two implicit registers.

The absolute value of the data in the EAX/RAX register represents the length of the character string in the first source operand; the absolute value of the data in the EDX/RDX register represents the length of the character string in the second source operand.

The Data Size bit of the immediate operand specifies length parameters. When Data Size = 0, length values represent bytes, and saturate to 16 if greater than 16 or less than –16; when Data Size = 1, length values represent words, and saturate to 8 if greater than 8 or less than –8;

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if intermediate result is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	1 if the absolute value of EDX is less than 16 (8); otherwise cleared.
SF	Set if the absolute value of EAX is less than 16 (8); otherwise cleared.
OF	Equal to the value of InterimResult2[0].

There are legacy and extended forms of the instruction:

PCMPESTRM

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A mask result is written to the XMM0 register.

VPCMPESTRM

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A mask result is written to the XMM0 register.

PCMPESTRM is an SSE4.2 instruction and VPCMPESTRM is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE42] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PCMPESTRM*xmm1*, *xmm2/mem128, imm8*	66 0F 3A 60 /r ib	Compares packed string data in *xmm1* and *xmm2* or *mem128*. Writes a mask value to the XMM0 register.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCMPESTRM *xmm1, xmm2/mem128,* imm8	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	60 /r ib

Related Instructions

(V)PCMPESTRI, (V)PCMPISTRI, (V)PCMPISTRM

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								M				M	M	M	M	M
21	20	19	18	17	16	14	13 : 12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag set or cleared to 0 is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	**Mode**			**Cause of Exception**
	Real	**Virt**	**Prot**	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PCMPGTB VPCMPGTB — Packed Compare Greater Than Signed Bytes

Compares 16 packed signed byte values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding byte of the destination.

When a value in the first operand is greater than a value in the second source operand, the result is FFh; when a value in the first operand is less than or equal to a value in the second operand, the result is 00h.

There are legacy and extended forms of the instruction:

PCMPGTB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPGTB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPGTB is an SSE2 instruction and VPCMPGTB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPGTB *xmm1*, *xmm2/mem128*	66 0F 66 /r	Compares packed bytes in *xmm1* to packed bytes in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPGTB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	66 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTD, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPGTD VPCMPGTD — Packed Compare Greater Than Signed Doublewords

Compares four packed signed doubleword values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding doubleword of the destination.

When a value in the first operand is greater than a value in the second operand, the result is FFFFFFFFh; when a value in the first operand is less than or equal to a value in the second operand, the result is 00000000h.

There are legacy and extended forms of the instruction:

PCMPGTD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPGTD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPGTD is an SSE2 instruction and VPCMPGTD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPGTD *xmm1*, *xmm2/mem128*	66 0F 66 /r	Compares packed bytes in *xmm1* to packed bytes in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPGTD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	66 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPGTQ VPCMPGTQ — Packed Compare Greater Than Signed Quadwords

Compares two packed signed quadword values in the first source operand to corresponding values in the second source operand and writes a comparison result to the corresponding quadword of the destination.

When a value in the first operand is greater than a value in the second operand, the result is FFFFFFFFFFFFFFFFh; when a value in the first operand is less than or equal to a value in the second operand, the result is 0000000000000000h.

There are legacy and extended forms of the instruction:

PCMPGTQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPGTQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPGTQ is an SSE4.2 instruction and VPCMPGTQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE42] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPGTQ *xmm1*, *xmm2/mem128*	66 0F 38 37 /r	Compares packed bytes in *xmm1* to packed bytes in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPGTQ *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	37 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPGTW VPCMPGTW Packed Compare Greater Than Signed Words

Compares two packed signed word values in the first operand to corresponding values in the second source operand and writes a comparison result to the corresponding word of the destination.

When a value in the first operand is greater than a value in the second operand, the result is FFFFh; when a value in the first operand is less than or equal to a value in the second operand, the result is 0000h.

There are legacy and extended forms of the instruction:

PCMPGTW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPCMPGTW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PCMPGTW is an SSE2 instruction and VPCMPGTW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPGTW *xmm1*, *xmm2/mem128*	66 0F 65 /r	Compares packed bytes in *xmm1* to packed bytes in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPGTW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	65 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PCMPISTRI VPCMPISTRI — Packed Compare Implicit Length Strings Return Index

Compares character string data in the first and second source operands. Aggregation and comparison operations are carried out as determined by values of the fields of the immediate operand. Writes an index to the ECX register.

See Section 1.5, “String Compare Instructions” for information about immediate byte definition, compare encoding and control functions.

Each input byte or word is augmented with a valid/invalid tag. A byte or word is considered valid when it has an index lower than that of the least significant null byte/word (the least significant null byte/word is also invalid).

Aggregation and comparison operations are performed and an index is returned in ECX. If no bits are set in InterimResult2[0], ECX is set to 16 for word strings or 8 for byte strings.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if intermediate result is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	1 if the absolute value of EDX is less than 16 (8); otherwise cleared.
SF	Set if the absolute value of EAX is less than 16 (8); otherwise cleared.
OF	Equal to the value of InterimResult2[0].

There are legacy and extended forms of the instruction:

PCMPISTRI

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A result index is written to the ECX register.

VPCMPISTRI

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A result index is written to the ECX register.

PCMPISTRI is an SSE4.2 instruction and VPCMPISTRI is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE42] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPISTRI *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 63 /r ib	Compares packed string data in *xmm1* and *xmm2* or *mem128*. Writes a result index to the ECX register.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPISTRI *xmm1, xmm2/mem128,* imm8	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	63 /r ib

Related Instructions

(V)PCMPESTRI, (V)PCMPESTRM, (V)PCMPISTRM

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								M				M	M	M	M	M
21	20	19	18	17	16	14	13 : 12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag that is set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PCMPISTRM VPCMPISTRM — Packed Compare Implicit Length Strings Return Mask

Compares character string data in the first and second source operands. Aggregation and comparison operations are carried out as determined by values of the fields of the immediate operand. Writes a result or mask value to the XMM0 register.

See Section 1.5, “String Compare Instructions” for information about immediate byte definition, compare encoding and control functions.

Each input byte or word is augmented with a valid/invalid tag. A byte or word is considered valid when it has an index lower than that of the least significant null byte/word (the least significant null byte/word is also invalid).

Aggregation and comparison operations are performed. Depending on the values of fields in the immediate byte, InterimResult2 is either zero-extended to 128 bits or expanded into a byte/word-mask, and then written to XMM0.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if intermediate result is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	1 if the absolute value of EDX is less than 16 (8); otherwise cleared.
SF	Set if the absolute value of EAX is less than 16 (8); otherwise cleared.
OF	Equal to the value of InterimResult2[0].

There are legacy and extended forms of the instruction:

PCMPISTRM

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A mask result is written to the XMM0 register.

VPCMPISTRM

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. A mask result is written to the XMM0 register.

PCMPISTRM is an SSE4.2 instruction and VPCMPISTRM is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE42] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PCMPISTRM *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 62 /r ib	Compares packed string data in *xmm1* and *xmm2* or *mem128*. Writes a result or mask to the XMM0 register.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMPISTRM *xmm1, xmm2/mem128,* imm8	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	62 /r ib

Related Instructions

(V)PCMPESTRI, (V)PCMPESTRM, (V)PCMPISTRI

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								M				M	M	M	M	M
21	20	19	18	17	16	14	13 : 12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag that is set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PEXTRB VPEXTRB — Extract Packed Byte

Extracts a byte from a source register and writes it to an 8-bit memory location or to the low-order byte of a general-purpose register, with zero-extension to 32 or 64 bits. Bits [3:0] of an immediate byte operand select the byte to be extracted:

Value of imm8 [3:0]	Source Bits Extracted
0000	[7:0]
0001	[15:8]
0010	[23:16]
0011	[31:24]
0100	[39:32]
0101	[47:40]
0110	[55:48]
0111	[63:56]
1000	[71:64]
1001	[79:72]
1010	[87:80]
1011	[95:88]
1100	[103:96]
1101	[111:104]
1110	[119:112]
1111	[127:120]

There are legacy and extended forms of the instruction:

PEXTRB

The source operand is an XMM register and the destination is either an 8-bit memory location or the low-order byte of a general-purpose register. When the destination is a general-purpose register, the extracted byte is zero-extended to 32 or 64 bits.

VPEXTRB

The extended form of the instruction has 128-bit encoding.

The source operand is an XMM register and the destination is either an 8-bit memory location or the low-order byte of a general-purpose register. When the destination is a general-purpose register, the extracted byte is zero-extended to 32 or 64 bits.

PEXTRB is an SSE4.1 instruction and VPEXTRB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PEXTRB *reg/m8*, *xmm*, *imm8*	66 0F 3A 14 /r ib	Extracts an 8-bit value specified by *imm8* from *xmm* and writes it to *m8* or the low-order byte of a general-purpose register, with zero-extension.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPEXTRB *reg/mem8*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	14 /r ib

Related Instructions

(V)PEXTRD, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PEXTRD VPEXTRD — Extract Packed Doubleword

Extracts a doubleword from a source register and writes it to an 32-bit memory location or a 32-bit general-purpose register. Bits [1:0] of an immediate byte operand select the doubleword to be extracted:

Value of imm8 [1:0]	Source Bits Extracted
00	[31:0]
01	[63:32]
10	[95:64]
11	[127:96]

There are legacy and extended forms of the instruction:

PEXTRD

The encoding is the same as PEXTRQ, with REX.W = 0.

The source operand is an XMM register and the destination is either an 32-bit memory location or a 32-bit general-purpose register.

VPEXTRD

The extended form of the instruction has 128-bit encoding.

The encoding is the same as VPEXTRQ, with VEX.W = 0.

The source operand is an XMM register and the destination is either an 32-bit memory location or a 32-bit general-purpose register.

PEXTRD is an SSE4.1 instruction and VPEXTRD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PEXTRD *reg32/mem32*, *xmm*, *imm8*	66 (W0) 0F 3A 16 /r ib	Extracts a 32-bit value specified by *imm8* from *xmm* and writes it to *mem32* or *reg32*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPEXTRD *reg32/mem32*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.0.01	16 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PEXTRQ VPEXTRQ — Extract Packed Quadword

Extracts a quadword from a source register and writes it to an 64-bit memory location or to a 64-bit general-purpose register. Bit [0] of an immediate byte operand selects the quadword to be extracted:

Value of imm8 [0]	Source Bits Extracted
0	[63:0]
1	[127:64]

There are legacy and extended forms of the instruction:

PEXTRQ

The encoding is the same as PEXTRD, with REX.W = 1.

The source operand is an XMM register and the destination is either an 64-bit memory location or a 64-bit general-purpose register.

VPEXTRQ

The extended form of the instruction has 128-bit encoding.

The encoding is the same as VPEXTRD, with VEX.W = 1.

The source operand is an XMM register and the destination is either an 64-bit memory location or a 64-bit general-purpose register.

PEXTRQ is an SSE4.1 instruction and VPEXTRQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PEXTRQ *reg64/mem64*, *xmm*, *imm8*	66 (W1) 0F 3A 16 /r ib	Extracts a 64-bit value specified by *imm8* from *xmm* and writes it to *mem64* or *reg64*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPEXTRQ *reg64/mem64*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	1.1111.0.01	16 /r ib

(V)PEXTRB, (V)PEXTRD, (V)PEXTRW, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PEXTRW VPEXTRW — Extract Packed Word

Extracts a word from a source register and writes it to a 16-bit memory location or to the low-order word of a general-purpose register, with zero-extension to 32 or 64 bits. Bits [3:0] of an immediate byte operand select the word to be extracted:

Value of imm8 [2:0]	Source Bits Extracted
000	[15:0]
001	[31:16]
010	[47:32
011	[63:48]
100	[79:64]
101	[95:80]
110	[111:96]
111	[127:112]

There are legacy and extended forms of the instruction:

PEXTRW

The legacy form of the instruction has SSE2 and SSE4.1 encodings.

The source operand is an XMM register and the destination is the low-order word of a general-purpose register. The extracted word is zero-extended to 32 or 64 bits.

The source operand is an XMM register and the destination is either an 16-bit memory location or the low-order word of a general-purpose register. When the destination is a general-purpose register, the extracted word is zero-extended to 32 or 64 bits.

VPEXTRW

The extended form of the instruction has two 128-bit encodings that correspond to the two legacy encodings.

The source operand is an XMM register and the destination is the low-order word of a general-purpose register. The extracted word is zero-extended to 32 or 64 bits.

The source operand is an XMM register and the destination is either an 16-bit memory location or the low-order word of a general-purpose register. When the destination is a general-purpose register, the extracted word is zero-extended to 32 or 64 bits.

PEXTRW is either an SSE2 or an SSE4.1 instruction. VPEXTRW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2], Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PEXTRW *reg*, *xmm*, *imm8*	66 0F C5 /r ib	Extracts a 16-bit value specified by *imm8* from *xmm* and writes it to the low-order byte of a general-purpose register, with zero-extension.
PEXTRW *reg/m16*, *xmm*, *imm8*	66 0F 3A 15 /r ib	Extracts a 16-bit value specified by *imm8* from *xmm* and writes it to *m16* or the low-order byte of a general-purpose register, with zero-extension.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPEXTRW *reg*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	C5 /r ib
VPEXTRW *reg/mem16*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	15 /r ib

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHADDD VPHADDD — Packed Horizontal Add Doubleword

Adds adjacent pairs of 32-bit signed integers in two source operands and packs the sums into a destination. If a sum overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set) and only the low-order 32 bits of the sum are written in the destination.

There are legacy and extended forms of the instruction:

PHADDD

The first source register is also the destination register.

Adds the 32-bit signed integer values in bits [63:32] and bits [31:0] of the first source XMM register and packs the sum into bits [31:0] of the destination; adds the 32-bit signed integer values in bits [127:96] and bits [95:64] of the first source register and packs the sum into bits [63:32] of the destination. Adds the corresponding values in the second source XMM register or a 128-bit memory location and packs the sums into bits [95:64] and [127:96] of the destination. Bits [255:128] of the YMM register that corresponds to the destination not affected.

VPHADDD

The extended form of the instruction has 128-bit encoding.

Adds the 32-bit signed integer values in bits [63:32] and bits [31:0] of the first source XMM register and packs the sum into bits [31:0] of the destination XMM register; adds the 32-bit signed integer values in bits [127:96] and bits [95:64] of the first source register and packs the sum into bits [63:32] of the destination. Adds the corresponding values in the second source XMM register or a 128-bit memory location and packs the sums into bits [95:64] and [127:96] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHADDD is an SSSE3 instruction and VPHADDD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHADDD *xmm1*, *xmm2/mem128*	66 0F 38 02 /r	Adds adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*. Writes packed sums to *xmm1*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHADDD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	02 /r

Related Instructions

(V)PHADDW, (V)PHADDSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHADDSW VPHADDSW — Packed Horizontal Add with Saturation Word

Adds adjacent pairs of 16-bit signed integers in two source operands, with saturation, and packs the sums into a destination.

Positive sums greater than 7FFFh are saturated to 7FFFh; negative sums less than 8000h are saturated to 8000h.

There are legacy and extended forms of the instruction:

PHADDSW

The first source register is also the destination.

Adds four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with bits [31:16] and [15:0] and packs each saturated 16-bit sum into the low quadword of the destination sequentially, starting with bits [15:0]. Adds the corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs each saturated 16-bit sum into the high quadword of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPHADDSW

The extended form of the instruction has 128-bit encoding.

Adds four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with bits [31:16] and [15:0] and packs each saturated 16-bit sum into the low quadword of the destination sequentially, starting with bits [15:0]. Adds the corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs each saturated 16-bit sum into the high quadword of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHADDSW is an SSSE3 instruction and VPHADDSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHADDSW *xmm1*, *xmm2/mem128*	66 0F 38 03 /r	Adds adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*, with saturation. Writes packed sums to *xmm1*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHADDSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	03 /r

Related Instructions

(V)PHADDD, (V)PHADDW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHADDW VPHADDW — Packed Horizontal Add Word

Adds adjacent pairs of 16-bit signed integers in two source operands and packs the sums into a destination. If a sum overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set) and only the low-order 32 bits of the sum are written in the destination.

There are legacy and extended forms of the instruction:

PHADDW

The first source register is also the destination.

Adds four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with bits [31:16] and [15:0] and packs each 16-bit sum into the low quadword of the destination sequentially, starting with bits [15:0]. Adds the corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs each 16-bit sum into the high quadword of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPHADDW

The extended form of the instruction has 128-bit encoding.

Adds four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with bits [31:16] and [15:0] and packs each 16-bit sum into the low quadword of the destination sequentially, starting with bits [15:0]. Adds the corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs each 16-bit sum into the high quadword of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHADDSW is an SSSE3 instruction and VPHADDSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHADDW *xmm1*, *xmm2/mem128*	66 0F 38 01 /r	Adds adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*. Writes packed sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	01 /r

Related Instructions

(V)PHADDD, (V)PHADDSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHMINPOSUW VPHMINPOSUW — Horizontal Minimum and Position

Finds the minimum unsigned 16-bit value in the source operand and copies it to the low order word element of the destination. Writes the source position index of the value to bits [18:16] of the destination and clears bits[127:19] of the destination.

There are legacy and extended forms of the instruction:

PHMINPOSUW

The source operand is an XMM register or 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPHMINPOSUW

The extended form of the instruction has 128-bit encoding.

The source operand is an XMM register or 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHMINPOSUW is an SSE4.1 instruction and VPHMINPOSUW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHMINPOSUW *xmm1*, *xmm2/mem128*	66 0F 38 41 /r	Finds the minimum unsigned word element in *xmm2* or *mem128*, copies it to *xmm1[15:0]*; writes its position index to *xmm1[18:16]*, and clears *xmm1[127:19]*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHMINPOSUW *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	41 /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUD, (V)PMINUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHSUBD VPHSUBD — Packed Horizontal Subtract Doubleword

Subtracts adjacent pairs of 32-bit signed integers in two source operands and packs the differences into a destination. The higher-order doubleword of each pair is subtracted from the lower-order doubleword.

There are legacy and extended forms of the instruction:

PHSUBD

The first source register is also the destination.

Subtracts the 32-bit signed integer value in bits [63:32] of the first source XMM register from the value in bits [31:0] of the first source XMM register and packs the difference into bits [31:0] of the destination; subtracts the 32-bit signed integer value in bits [127:96] from the value in bits [95:64] and packs the difference into bits [63:32] of the destination. Subtracts the corresponding values in the second source XMM register or a 128-bit memory location and packs the differences into bits [95:64] and [127:96] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPHSUBD

The extended form of the instruction has 128-bit encoding.

Subtracts the 32-bit signed integer value in bits [63:32] of the first source XMM register from the value in bits [31:0] of the first source XMM register and packs the difference into bits [31:0] of the destination XMM register; subtracts the 32-bit signed integer values in bits [127:96] from the value in bits [95:64] and packs the difference into bits [63:32] of the destination. Subtracts the corresponding values in the second source XMM register or a 128-bit memory location and packs the differences into bits [95:64] and [127:96] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHSUBD is an SSSE3 instruction and VPHSUBD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHSUBD *xmm1*, *xmm2/mem128*	66 0F 38 06 /r	Adds adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*. Writes packed sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	06 /r

Related Instructions

(V)PHSUBW, (V)PHSUBSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHSUBSW VPHSUBSW — Packed Horizontal Subtract with Saturation Word

Subtracts adjacent pairs of 16-bit signed integers in two source operands, with saturation, and packs the differences into a destination. The higher-order word of each pair is subtracted from the lower-order word.

Positive differences greater than 7FFFh are saturated to 7FFFh; negative differences less than 8000h are saturated to 8000h.

There are legacy and extended forms of the instruction:

PHSUBSW

The first source register is also the destination.

Subtracts four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with the value in bits [15:0] minus the value in bits [31:16], and packs four saturated 16-bit differences into bits [63:0] of the destination, starting with bits [15:0]. Subtracts the four corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs four saturated 16-bit differences into bits [127:64] of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

VPHSUBSW

The extended form of the instruction has 128-bit encoding.

Subtracts four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with the value in bits [15:0] minus the value in bits [31:16], and packs four saturated 16-bit differences into bits [63:0] of the destination XMM register, starting with bits [15:0]. Subtracts the four corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs four saturated 16-bit differences into bits [127:64] of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHSUBSW is an SSSE3 instruction and VPHSUBSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHSUBSW *xmm1*, *xmm2/mem128*	66 0F 38 07 /r	Subtracts adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*, with saturation. Writes packed differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	07 /r

Related Instructions

(V)PHSUBD, (V)PHSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PHSUBW VPHSUBW — Packed Horizontal Subtract Word

Subtracts adjacent pairs of 16-bit signed integers in two source operands and packs the differences into a destination. The higher-order word of each pair is subtracted from the lower-order word.

There are legacy and extended forms of the instruction:

PHSUBW

The first source register is also the destination register.

Subtracts four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with the value in bits [15:0] minus the value in bits [31:16], and packs four 16-bit differences into bits [63:0] of the destination, starting with bits [15:0]. Subtracts the four corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs four 16-bit differences into bits [127:64] of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

VPHSUBW

The extended form of the instruction has 128-bit encoding.

Subtracts four adjacent pairs of 16-bit signed integer values in the first source XMM register, starting with the value in bits [15:0] minus the value in bits [31:16], and packs four 16-bit differences into bits [63:0] of the destination XMM register, starting with bits [15:0]. Subtracts the four corresponding adjacent pairs of values in the second source XMM register or a 128-bit memory location and packs four 16-bit differences into bits [127:64] of the destination, starting with bits [79:64]. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PHSUBW is an SSSE3 instruction and VPHSUBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PHSUBW *xmm1*, *xmm2/mem128*	66 0F 38 05 /r	Adds adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*. Writes packed sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	05 /r

Related Instructions

(V)PHSUBD, (V)PHSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PINSRB VPINSRB — Packed Insert Byte

Inserts a byte from an 8-bit memory location or the low-order byte of a 32-bit general-purpose register into a destination register. Bits [3:0] of an immediate byte operand select the location where the byte is to be inserted:

Value of imm8 [3:0]	Insertion Location
0000	[7:0]
0001	[15:8]
0010	[23:16]
0011	[31:24]
0100	[39:32]
0101	[47:40]
0110	[55:48]
0111	[63:56]
1000	[71:64]
1001	[79:72]
1010	[87:80]
1011	[95:88]
1100	[103:96]
1101	[111:104]
1110	[119:112]
1111	[127:120]

There are legacy and extended forms of the instruction:

PINSRB

The source operand is either an 8-bit memory location or the low-order byte of a 32-bit general-purpose register and the destination an XMM register. The other bytes of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPINSRB

The extended form of the instruction has 128-bit encoding.

There are two source operands. The first source operand is either an 8-bit memory location or the low-order byte of a 32-bit general-purpose register and the second source operand is an XMM register. The destination is a second XMM register. All the bytes of the second source other than the byte that corresponds to the location of the inserted byte are copied to the destination. Bits [255:128] of the YMM register that corresponds to destination are cleared.

PINSRB is an SSE4.1 instruction and VPINSRB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PINSRB *xmm*, *reg32*/*mem8*, *imm8*	66 0F 3A 20 /r ib	Inserts an 8-bit value selected by *imm8* from the low-order byte of *reg32* or from *mem8* into *xmm*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPINSRB *xmm*, *reg/mem8*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	X.1111.0.01	20 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRD, (V)PINSRQ, (V)PINSRW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PINSRD VPINSRD — Packed Insert Doubleword

Inserts a doubleword from a 32-bit memory location or a 32-bit general-purpose register into a destination register. Bits [1:0] of an immediate byte operand select the location where the doubleword is to be inserted:

Value of imm8 [1:0]	Insertion Location
00	[31:0]
01	[63:32]
10	[95:64]
11	[127:96]

There are legacy and extended forms of the instruction:

PINSRD

The encoding is the same as PINSRQ, with REX.W = 0.

The source operand is either a 32-bit memory location or a 32-bit general-purpose register and the destination an XMM register. The other doublewords of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPINSRD

The extended form of the instruction has 128-bit encoding.

The encoding is the same as VPINSRQ, with VEX.W = 0.

There are two source operands. The first source operand is either a 32-bit memory location or a 32-bit general-purpose register and the second source operand is an XMM register. The destination is a second XMM register. All the doublewords of the second source other than the doubleword that corresponds to the location of the inserted doubleword are copied to the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PINSRD is an SSE4.1 instruction and VPINSRD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PINSRD *xmm*, *reg32*/*mem32*, *imm8*	66 (W0) 0F 3A 22 /r ib	Inserts a 32-bit value selected by *imm8* from *reg32* or *mem32* into *xmm*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPINSRD *xmm*, *reg32/mem32*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.0.01	22 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRB, (V)PINSRQ, (V)PINSRW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PINSRQ VPINSRQ — Packed Insert Quadword

Inserts a quadword from a 64-bit memory location or a 64-bit general-purpose register into a destination register. Bit [0] of an immediate byte operand selects the location where the doubleword is to be inserted:

Value of imm8 [0]	Insertion Location
0	[63:0]
1	[127:64]

There are legacy and extended forms of the instruction:

PINSRQ

The encoding is the same as PINSRD, with REX.W = 1.

The source operand is either a 64-bit memory location or a 64-bit general-purpose register and the destination an XMM register. The other quadwords of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPINSRQ

The extended form of the instruction has 128-bit encoding.

The encoding is the same as VPINSRD, with VEX.W = 1.

There are two source operands. The first source operand is either a 64-bit memory location or a 64-bit general-purpose register and the second source operand is an XMM register. The destination is a second XMM register. All the quadwords of the second source other than the quadword that corresponds to the location of the inserted quadword are copied to the destination. Bits [255:128] of the YMM register that corresponds to the destination XMM registers are cleared.

PINSRQ is an SSE4.1 instruction and VPINSRQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PINSRQ *xmm*, *reg64/mem64*, *imm8*	66 (W1) 0F 3A 22 /r ib	Inserts a 64-bit value selected by *imm8* from *reg64* or *mem64* into *xmm*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPINSRQ *xmm*, *reg64/mem64*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00011	1.1111.0.01	22 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRB, (V)PINSRD, (V)PINSRW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PINSRW VPINSRW — Packed Insert Word

Inserts a word from a 16-bit memory location or the low-order word of a 32-bit general-purpose register into a destination register. Bits [2:0] of an immediate byte operand select the location where the byte is to be inserted:

Value of imm8 [2:0]	Insertion Location
000	[15:0]
001	[31:16]
010	[47:32
011	[63:48]
100	[79:64]
101	[95:80]
110	[111:96]
111	[127:112]

There are legacy and extended forms of the instruction:

PINSRW

The source operand is either a 16-bit memory location or the low-order word of a 32-bit general-purpose register and the destination an XMM register. The other words of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPINSRW

The extended form of the instruction has 128-bit encoding.

There are two source operands. The first source operand is either a 16-bit memory location or the low-order word of a 32-bit general-purpose register and the second source operand is an XMM register. The destination is a second XMM register. All the words of the second source other than the word that corresponds to the location of the inserted word are copied to the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PINSRW is an SSE instruction and VPINSRW is an AVX instruction. Support for these instructions is indicated by CPUID CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PINSRW *xmm*, *reg32*/*mem16*, *imm8*	66 0F C4 /r ib	Inserts a 16-bit value selected by *imm8* from the low-order word of *reg32* or from *mem16* into *xmm*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPINSRW *xmm*, *reg/mem8*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	C4 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRB, (V)PINSRD, (V)PINSRQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMADDUBSW VPMADDUBSW — Packed Multiply and Add Unsigned Byte to Signed Word

Multiplies and adds eight sets of two packed 8-bit unsigned values from the first source register and two packed 8-bit signed values from the second source register, with signed saturation; writes eight 16-bit sums to the destination.

Source registers 1 and 2 consist of bytes [a0 a1 a2 ...] and [b0 b1 b2 ...] and the destination register consists of words [w0 w1 w2 ...]. Operation is summarized as follows.

- The product of the values in bits [7:0] of the source registers (a0b0) is added to the product of the values in bits [15:8] of the source registers (a1b1). The saturated sum w0 = (a0b0 + a1b1) is written to bits [15:0] of the destination.
- The product of the values in bits [23:16] of the source registers (a2b2) is added to the product of the values in bits [31:24] of the source registers (a3b3). The saturated sum w1 = (a2b2 + a3b3) is written to bits [31:16] of the destination.
- The product of the values in bits [39:32] of the source registers (a4b4) is added to the product of the values in bits [47:40] of the source registers (a5b5). The saturated sum w2 = (a4b4 + a5b5) is written to bits [47:32] of the destination.
- The product of the values in bits [55:48] of the source registers (a6b6) is added to the product of the values in bits [63:56] of the source registers (a7b7). The saturated sum w3 = (a6b6 + a7b7) is written to bits [63:48] of the destination.
- The product of the values in bits [71:64] of the source registers (a8b8) is added to the product of the values in bits [79:72] of the source registers (a9b9). The saturated sum w4 = (a8b8 + a9b9) is written to bits [79:64] of the destination.
- The product of the values in bits [87:80] of the source registers (a10b10) is added to the product of the values in bits [95:88] of the source registers (a11b11). The saturated sum w5 = (a10b10 + a11b11) is written to bits [95:80] of the destination.
- The product of the values in bits [103:96] of the source registers (a12b12) is added to the product of the values in bits [111:104] of the source registers (a13b13). The saturated sum w6 = (a12b12 + a13b13) is written to bits [111:96] of the destination.
- The product of the values in bits [119:112] of the source registers (a14b14) is added to the product of the values in bits [127:120] of the source registers (a15b15). The saturated sum w7 = (a14b14 + a15b15) is written to bits [127:112] of the destination.

There are legacy and extended forms of the instruction:

PMADDUBSW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMADDUBSW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMADDUBSW is an SSSE3 instruction and VPMADDUBSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMADDUBSW *xmm1*, *xmm2/mem128*	66 0F 38 04 /r	Multiplies packed 8-bit unsigned values in *xmm1* and packed 8-bit signed values *xmm2* or *mem128*, adds the products, and writes saturated sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMADDUBSW *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	04 /r

Related Instructions

(V)PMADDWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMADDWD VPMADDWD — Packed Multiply and Add Word to Doubleword

Multiplies and adds four sets of four packed 16-bit signed values from two source registers; writes four 32-bit sums to the destination.

Source registers 1 and 2 consist of words [a0 a1 a2 ...] and [b0 b1 b2 ...] and the destination register consists of doublewords [w0 w1 w2 ...]. Operation is summarized as follows.

- The product of the values in bits [15:0] of the source registers (a0b0) is added to the product of the values in bits [31:16] of the source registers (a1b1). The sum d0 = (a0b0 + a1b1) is written to bits [31:0] of the destination.
- The product of the values in bits [47:33] of the source registers (a2b2) is added to the product of the values in bits [63:48] of the source registers (a3b3). The sum d1 = (a2b2 + a3b3) is written to bits [63:32] of the destination.
- The product of the values in bits [79:64] of the source registers (a4b4) is added to the product of the values in bits [95:80] of the source registers (a5b5). The sum d2 = (a4b4 + a5b5) is written to bits [95:64] of the destination.
- The product of the values in bits [111:96] of the source registers (a6b6) is added to the product of the values in bits [127:112] of the source registers (a7b7). The sum d3 = (a6b6 + a7b7) is written to bits [127:96] of the destination.

When all four of the signed 16-bit source operands in a set have the value 8000h, the 32-bit overflow wraps around to 8000_0000h. There are no other overflow cases.

There are legacy and extended forms of the instruction:

PMADDWD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMADDWD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMADDWD is an SSE2 instruction and VPMADDWD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMADDWD *xmm1*, *xmm2/mem128*	66 0F F5 /r	Multiplies packed 16-bit signed values in *xmm1* and *xmm2* or *mem128*, adds the products, and writes the sums to *xmm1*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMADDWD *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	F5 /r

Related Instructions

(V)PMADDUBSW, (V)PMULHUW, (V)PMULHW, (V)PMULLW, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMAXSB VPMAXSB — Packed Maximum Signed Bytes

Compares each packed 8-bit signed integer value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMAXSB

Compares16 pairs of 8-bit signed integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMAXSB

The extended form of the instruction has 128-bit encoding.

Compares 16 pairs of 8-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMAXSB is an SSE4.1 instruction and VPMAXSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMAXSB *xmm1*, *xmm2/mem128*	66 0F 38 3C /r	Compares 16 pairs of packed 8-bit values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMAXSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	3C /r

Related Instructions

(V)PMAXSD, (V)PMAXSW, (V)PMAXUB, (V)PMAXUD, (V)PMAXUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMAXSD VPMAXSD — Packed Maximum Signed Doublewords

Compares each packed 32-bit signed integer value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMAXSD

Compares four pairs of packed 32-bit signed integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMAXSD

The extended form of the instruction has 128-bit encoding.

Compares four pairs of packed 32-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMAXSD is an SSE4.1 instruction and VPMAXSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMAXSD *xmm1*, *xmm2/mem128*	66 0F 38 3D /r	Compares four pairs of packed 32-bit values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMAXSD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	3D /r

Related Instructions

(V)PMAXSB, (V)PMAXSW, (V)PMAXUB, (V)PMAXUD, (V)PMAXUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMAXSW VPMAXSW — Packed Maximum Signed Words

Compares each packed 16-bit signed integer value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMAXSW

Compares eight pairs of packed 16-bit signed integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMAXSW

The extended form of the instruction has 128-bit encoding.

Compares eight pairs of packed 16-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMAXSW is an SSE2 instruction and VPMAXSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMAXSW *xmm1*, *xmm2/mem128*	66 0F EE /r	Compares eight pairs of packed 16-bit values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMAXSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	EE /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXUB, (V)PMAXUD, (V)PMAXUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMAXUB VPMAXUB — Packed Maximum Unsigned Bytes

Compares each packed 8-bit unsigned integer value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMAXUB

Compares 16 pairs of 8-bit unsigned integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMAXUB

The extended form of the instruction has 128-bit encoding.

Compares 16 pairs of 8-bit unsigned integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMAXUB is an SSE2 instruction and VPMAXUB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PMAXUB *xmm1*, *xmm2/mem128*	66 0F DE /r	Compares 16 pairs of packed unsigned 8-bit values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMAXUB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	DE /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXSW, (V)PMAXUD, (V)PMAXUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMAXUD VPMAXUD — Packed Maximum Unsigned Doublewords

Compares each packed 32-bit unsigned integer value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMAXUD

Compares four pairs of packed 32-bit unsigned integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMAXUD

The extended form of the instruction has 128-bit encoding.

Compares four pairs of packed 32-bit unsigned integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMAXUD is an SSE4.1 instruction and VPMAXUD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMAXUD *xmm1*, *xmm2/mem128*	66 0F 38 3F /r	Compares four pairs of packed unsigned 32-bit values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMAXUD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	3F /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXSW, (V)PMAXUB, (V)PMAXUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMAXUW VPMAXUW — Packed Maximum Unsigned Words

Compares each packed 16-bit unsigned integer value of the first source operand to the corresponding value of the second source operand and writes the numerically greater value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMAXUW

Compares eight pairs of packed 16-bit unsigned integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMAXUW

The extended form of the instruction has 128-bit encoding.

Compares eight pairs of packed 16-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMAXUW is an SSE4.1 instruction and VPMAXUW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMAXUW *xmm1*, *xmm2/mem128*	66 0F 38 3E /r	Compares eight pairs of packed unsigned 16-bit values in *xmm1* and *xmm2* or *mem128* and writes the greater values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMAXUW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	3E /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXSW, (V)PMAXUB, (V)PMAXUD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMINSB VPMINSB — Packed Minimum Signed Bytes

Compares each packed 8-bit signed integer value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMINSB

Compares 16 pairs of 8-bit signed integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMINSB

The extended form of the instruction has 128-bit encoding.

Compares 16 pairs of 8-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMINSB is an SSE4.1 instruction and VPMINSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMINSB *xmm1*, *xmm2/mem128*	66 0F 38 38 /r	Compares 16 pairs of packed 8-bit values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMINSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	38 /r

Related Instructions

(V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUD, (V)PMINUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMINSD VPMINSD — Packed Minimum Signed Doublewords

Compares each packed 32-bit signed integer value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMINSD

Compares four pairs of packed 32-bit signed integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMINSD

The extended form of the instruction has 128-bit encoding.

Compares four pairs of packed 32-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMINSD is an SSE4.1 instruction and VPMINSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMINSD *xmm1*, *xmm2/mem128*	66 0F 38 39 /r	Compares four pairs of packed 32-bit values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMINSD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	39 /r

Related Instructions

(V)PMINSB, (V)PMINSW, (V)PMINUB, (V)PMINUD, (V)PMINUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMINSW VPMINSW — Packed Minimum Signed Words

Compares each packed 16-bit signed integer value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMINSW

Compares eight pairs of packed 16-bit signed integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMINSW

The extended form of the instruction has 128-bit encoding.

Compares eight pairs of packed 16-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMINSW is an SSE2 instruction and VPMINSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMINSW *xmm1*, *xmm2/mem128*	66 0F EA /r	Compares eight pairs of packed 16-bit values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMINSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	EA /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINUB, (V)PMINUD, (V)PMINUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMINUB VPMINUB — Packed Minimum Unsigned Bytes

Compares each packed 8-bit unsigned integer value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMINUB

Compares 16 pairs of 8-bit unsigned integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMINUB

The extended form of the instruction has 128-bit encoding.

Compares 16 pairs of 8-bit unsigned integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMINUB is an SSE2 instruction and VPMINUB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMINUB *xmm1*, *xmm2/mem128*	66 0F DA /r	Compares 16 pairs of packed unsigned 8-bit values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMINUB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	DA /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUD, (V)PMINUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMINUD VPMINUD — Packed Minimum Unsigned Doublewords

Compares each packed 32-bit unsigned integer value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMINUD

Compares four pairs of packed 32-bit unsigned integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMINUD

The extended form of the instruction has 128-bit encoding.

Compares four pairs of packed 32-bit unsigned integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMINUD is an SSE4.1 instruction and VPMINUD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMINUD *xmm1*, *xmm2/mem128*	66 0F 38 3B /r	Compares four pairs of packed unsigned 32-bit values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMINUD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	3B /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMINUW VPMINUW — Packed Minimum Unsigned Words

Compares each packed 16-bit unsigned integer value of the first source operand to the corresponding value of the second source operand and writes the numerically lesser value into the corresponding location of the destination.

There are legacy and extended forms of the instruction:

PMINUW

Compares eight pairs of packed 16-bit unsigned integer values.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source operand is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMINUW

The extended form of the instruction has 128-bit encoding.

Compares eight pairs of packed 16-bit signed integer values.

The first source operand is an XMM register and the second source operand is either another XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMINUW is an SSE4.1 instruction and VPMINUW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMINUW *xmm1*, *xmm2/mem128*	66 0F 38 3A /r	Compares eight pairs of packed unsigned 16-bit values in *xmm1* and *xmm2* or *mem128* and writes the lesser values to the corresponding positions in *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMINUW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	3A /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVMSKB VPMOVMSKB — Packed Move Mask Byte

Copies the values of the most-significant bits of each byte element of the source operand to create a 16-bit mask value, zero-extends the value, and writes it to the destination.

There are legacy and extended forms of the instruction:

PMOVMSKB

The source operand is an XMM register. The destination is a 32-bit general purpose register. The mask is zero-extended to fill the destination register, the mask occupies bits [15:0].

VPMOVMSKB

The source operand is an XMM register. The destination is a 64-bit general purpose register. The mask is zero-extended to fill the destination register, the mask occupies bits [15:0]. VEX.W is ignored.

PMOVMSKB is an SSE2 instruction and VPMOVMSKB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVMSKB *reg32*, *xmm1*	66 0F D7 /r	Moves a zero-extended mask consisting of the most-significant bit of each byte in *xmm1* to a 32-bit general-purpose register.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VMOVMSKB *reg64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	D7 /r

Related Instructions

(V)MOVMSKPD, (V)MOVMSKPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv field ! = 1111b.
			A	VEX.L field = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVSXBD VPMOVSXBD — Packed Move with Sign-Extension Byte to Doubleword

Sign-extends each of four packed 8-bit signed integers, in either the four low bytes of a source register or a 32-bit memory location, to 32 bits and writes four packed doubleword signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVSXBD

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVSXBD

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSXBD is an SSE4.1 instruction and VPMOVSXBD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVSXBD *xmm1*, *xmm2/mem32*	66 0F 38 21 /r	Sign-extends four packed signed 8-bit integers in the four low bytes of *xmm2* or *mem32* and writes four packed signed 32-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVSXBD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	21 /r

Related Instructions

(V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWD, (V)PMOVSXW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVSXBQ VPMOVSXBQ — Packed Move with Sign Extension Byte to Quadword

Sign-extends each of two packed 8-bit signed integers, in either the two low bytes of a source register or a 16-bit memory location, to 64 bits and writes two packed quadword signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVSXBQ

The source operand is either an XMM register or a 16-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVSXBQ

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSXBQ is an SSE4.1 instruction and VPMOVSXBQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVSXBQ *xmm1, xmm2/mem16*	66 0F 38 22 /r	Sign-extends two packed signed 8-bit integers in the two low bytes of *xmm2* or *mem16* and writes two packed signed 64-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVSXBQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	22 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWD, (V)PMOVSXW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVSXBW VPMOVSXBW — Packed Move with Sign Extension Byte to Word

Sign-extends each of eight packed 8-bit signed integers, in either the eight low bytes of a source register or a 64-bit memory location, to 16 bits and writes eight packed word signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVSXBW

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVSXBW

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSXBW is an SSE4.1 instruction and VPMOVSXBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVSXBW *xmm1*, *xmm2/mem128*	66 0F 38 20 /r	Sign-extends eight packed signed 8-bit integers in the eight low bytes of *xmm2* or *mem128* and writes eight packed signed 16-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVSXBW *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	20 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXDQ, (V)PMOVSXWD, (V)PMOVSXW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVSXDQ VPMOVSXDQ — Packed Move with Sign-Extension Doubleword to Quadword

Sign-extends each of two packed 32-bit signed integers, in either the two low doublewords of a source register or a 64-bit memory location, to 64 bits and writes two packed quadword signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVSXDQ

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVSXDQ

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSXDQ is an SSE4.1 instruction and VPMOVSXDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVSXDQ *xmm1*, *xmm2/mem64*	66 0F 38 25 /r	Sign-extends two packed signed 32-bit integers in the two low doublewords of *xmm2* or *mem64* and writes two packed signed 64-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVSXDQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	25 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXWD, (V)PMOVSXWQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVSXWD VPMOVSXWD — Packed Move with Sign-Extension Word to Doubleword

Sign-extends each of four packed 16-bit signed integers, in either the four low words of a source register or a 64-bit memory location, to 32 bits and writes four packed doubleword signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVSXWD

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVSXWD

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSXWD is an SSE4.1 instruction and VPMOVSXWD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVSXWD *xmm1*, *xmm2/mem64*	66 0F 38 23 /r	Sign-extends four packed signed 16-bit integers in the four low words of *xmm2* or *mem64* and writes four packed signed 32-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMOVSXWD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	23 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVSXWQ VPMOVSXWQ — Packed Move with Sign-Extension Word to Quadword

Sign-extends each of two packed 16-bit signed integers, in either the two low words of a source register or a 32-bit memory location, to 64 bits and writes two packed quadword signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVSXWQ

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVSXWQ

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSXWQ is an SSE4.1 instruction and VPMOVSXWQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVSXWQ *xmm1*, *xmm2/mem32*	66 0F 38 24 /r	Sign-extends two packed signed 16-bit integers in the two low words of *xmm2* or *mem32* and writes two packed signed 64-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVSXWQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	24 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVZXBD VPMOVZXBD — Packed Move with Zero-Extension Byte to Doubleword

Zero-extends each of four packed 8-bit unsigned integers, in either the four low bytes of a source register or a 32-bit memory location, to 32 bits and writes four packed doubleword positive-signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVZXBD

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVZXBD

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVZXBD is an SSE4.1 instruction and VPMOVZXBD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the CPUID Specification, order# 25481).

Mnemonic	Opcode	Description
PMOVZXBD *xmm1*, *xmm2/mem32*	66 0F 38 31 /r	Zero-extends four packed unsigned 8-bit integers in the four low bytes of *xmm2* or *mem32* and writes four packed positive-signed 32-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVZXBD *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	31 /r

Related Instructions

(V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWD, (V)PMOVZXW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVZXBQ VPMOVZXBQ — Packed Move Byte to Quadword with Zero-Extension

Zero-extends each of two packed 8-bit unsigned integers, in either the two low bytes of a source register or a 16-bit memory location, to 64 bits and writes two packed quadword positive-signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVZXBQ

The source operand is either an XMM register or a 16-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVZXBQ

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVZXBQ is an SSE4.1 instruction and VPMOVZXBQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVZXBQ *xmm1, xmm2/mem16*	66 0F 38 32 /r	Zero-extends two packed unsigned 8-bit integers in the two low bytes of *xmm2* or *mem16* and writes two packed positive-signed 64-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVZXBQ *xmm1, xmm2/mem16*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	32 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWD, (V)PMOVZXW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVZXBW VPMOVZXBW Packed Move Byte to Word with Zero-Extension

Zero-extends each of eight packed 8-bit unsigned integers, in either the eight low bytes of a source register or a 64-bit memory location, to 16 bits and writes eight packed word positive-signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVZXBW

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVZXBW

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVZXBW is an SSE4.1 instruction and VPMOVZXBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVZXBW *xmm1*, *xmm2/mem128*	66 0F 38 30 /r	Zero-extends eight packed unsigned 8-bit integers in the eight low bytes of *xmm2* or *mem128* and writes eight packed positive-signed 16-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVZXBW *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	30 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXDQ, (V)PMOVZXWD, (V)PMOVZXW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVZXDQ VPMOVZXDQ — Packed Move with Zero-Extension Doubleword to Quadword

Zero-extends each of two packed 32-bit unsigned integers, in either the two low doublewords of a source register or a 64-bit memory location, to 64 bits and writes two packed quadword positive-signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVZXDQ

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVZXDQ

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

The PMOVZXDQ is an SSE4.1 instruction and VPMOVZXDQ is an AVX instruction.Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVZXDQ *xmm1*, *xmm2/mem64*	66 0F 38 35 /r	Zero-extends two packed unsigned 32-bit integers in the two low doublewords of *xmm2* or *mem64* and writes two packed positive-signed 64-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVZXDQ *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	35 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXWD, (V)PMOVZXWQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVZXWD VPMOVZXWD — Packed Move Word to Doubleword with Zero-Extension

Zero-extends each of four packed 16-bit unsigned integers, in either the four low words of a source register or a 64-bit memory location, to 32 bits and writes four packed doubleword positive-signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVZXWD

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVZXWD

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVZXWD is an SSE4.1 instruction and VPMOVZXWD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVZXWD *xmm1*, *xmm2/mem64*	66 0F 38 33 /r	Zero-extends four packed unsigned 16-bit integers in the four low words of *xmm2* or *mem64* and writes four packed positive-signed 32-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVZXWD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	33 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMOVZXWQ VPMOVZXWQ — Packed Move with Zero-Extension Word to Quadword

Zero-extends each of two packed 16-bit unsigned integers, in either the two low words of a source register or a 32-bit memory location, to 64 bits and writes two packed quadword positive-signed integers to the destination.

There are legacy and extended forms of the instruction:

PMOVZXWQ

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMOVZXWQ

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMOVSZWQ is an SSE4.1 instruction and VPMOVZXWQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMOVZXWQ *xmm1*, *xmm2/mem32*	66 0F 38 34 /r	Zero-extends two packed unsigned 16-bit integers in the two low words of *xmm2* or *mem32* and writes two packed positive-signed 64-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMOVZXWQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	34 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMULDQ VPMULDQ — Packed Multiply Signed Doubleword to Quadword

Multiplies the packed 32-bit signed integer in bits [31:0] of the first source operand by the corresponding value of the second source operand and writes the packed 64-bit signed integer product to bits [63:0] of the destination; multiplies the packed 32-bit signed integer in bits [95:64] of the first source operand by the corresponding value of the second source operand and writes the packed 64-bit signed integer product to bits [127:64] of the destination.

When the source is a memory location, all 128 bits are fetched, but only the first and third doublewords are used in the computation.

There are legacy and extended forms of the instruction:

PMULDQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULDQ is an SSE4.1 instruction and VPMULDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULDQ *xmm1*, *xmm2/mem128*	66 0F 38 28 /r	Multiplies two packed 32-bit signed integers in *xmm1[31:0]* and *xmm1[95:64]* by the corresponding values in *xmm2* or *mem128*. Writes packed 64-bit signed integer products to *xmm1[63:0]* and *xmm1[127:64]*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULDQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	28 /r

Related Instructions

(V)PMULLD, (V)PMULHW, (V)PMULHUW,(V)PMULUDQ, (V)PMULLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMULHRSW VPMULHRSW — Packed Multiply High with Round and Scale Words

Multiplies each packed 16-bit signed value in the first source operand by the corresponding value in the second source operand, truncates the 32-bit product to the 18 most significant bits by right-shifting, then rounds the truncated value by adding 1 to its least-significant bit. Writes bits [16:1] of the sum to the corresponding word of the destination.

There are legacy and extended forms of the instruction:

PMULHRSW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULHRSW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULHRSW instruction is an SSSE3 instruction and VPMULHRSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULHRSW *xmm1*, *xmm2*/*mem128*	66 0F 38 0B /r	Multiplies each packed 16-bit signed value in *xmm1* by the corresponding value in *xmm2* or *mem128*, truncates product to 18 bits, rounds by adding 1. Writes bits [16:1] of the sum to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULHRSW *xmm1*, *xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	0B /r

Related Instructions

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMULHUW VPMULHUW — Packed Multiply High Unsigned Word

Multiplies each packed 16-bit unsigned value in the first source operand by the corresponding value in the second source operand; writes the high-order 16 bits of each 32-bit product to the corresponding word of the destination.

There are legacy and extended forms of the instruction:

PMULHUW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULHUW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULHUW is an SSE2 instruction and VPMULHUW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULHUW *xmm1*, *xmm2/mem128*	66 0F E4 /r	Multiplies packed 16-bit unsigned values in *xmm1* by the corresponding values in *xmm2* or *mem128.* Writes bits [31:16] of each product to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULHUW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	E4 /r

Related Instructions

(V)PMULDQ, (V)PMULHW, (V)PMULLD, (V)PMULLW, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception A — AVX exception S — SSE exception				

PMULHW VPMULHW — Packed Multiply High Signed Word

Multiplies each packed 16-bit signed value in the first source operand by the corresponding value in the second source operand; writes the high-order 16 bits of each 32-bit product to the corresponding word of the destination.

There are legacy and extended forms of the instruction:

PMULHW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULHW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULHW is an SSE2 instruction and VPMULHW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULHW *xmm1*, *xmm2/mem128*	66 0F E5 /r	Multiplies packed 16-bit signed values in *xmm1* by the corresponding values in *xmm2* or *mem128.* Writes bits [31:16] of each product to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULHW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	E5 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULLD, (V)PMULLW, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMULLD VPMULLD — Packed Multiply and Store Low Signed Doubleword

Multiplies four packed 32-bit signed integers in the first source operand by the corresponding values in the second source operand and writes bits [31:0] of each 64-bit product to the corresponding 32-bit element of the destination.

There are legacy and extended forms of the instruction:

PMULLD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULLD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULLD is an SSE4.1 instruction and VPMULLD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULLD *xmm1*, *xmm2/mem128*	66 0F 38 40 /r	Multiplies four packed 32-bit signed integers in *xmm1* by corresponding values in *xmm2* or *m128.* Writes bits [31:0] of each 64-bit product to the corresponding 32-bit element of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULLD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	40 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULHW, (V)PMULLW, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMULLW VPMULLW — Packed Multiply Low Signed Word

Multiplies eight packed 16-bit signed integers in the first source operand by the corresponding values in the second source operand and writes bits [15:0] of each 32-bit product to the corresponding 16-bit element of the destination.

There are legacy and extended forms of the instruction:

PMULLW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULLW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULLW is an SSE2 instruction and VPMULLW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULLW *xmm1*, *xmm2/mem128*	66 0F D5 /r	Multiplies eight packed 16-bit signed integers in *xmm1* by corresponding values in *xmm2* or *m128.* Writes bits [15:0] of each 32-bit product to the corresponding 16-bit element of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULLW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	D5 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULHW, (V)PMULLD, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PMULUDQ VPMULUDQ — Packed Multiply Unsigned Doubleword to Quadword

Multiplies the packed 32-bit unsigned integer in bits [31:0] of the first source operand by the corresponding value of the second source operand and writes the packed 64-bit unsigned integer product to bits [63:0] of the destination; multiplies the packed 32-bit unsigned integer in bits [95:64] of the first source operand by the corresponding value of the second source operand and writes the packed 64-bit unsigned integer product to bits [127:64] of the destination.

When the source is a memory location, all 128 bits are fetched, but only the first and third doublewords are used in the computation.

There are legacy and extended forms of the instruction:

PMULUDQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPMULUDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PMULUDQ is an SSE2 instruction and VPMULUDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PMULUDQ *xmm1*, *xmm2/mem128*	66 0F F4 /r	Multiplies two packed 32-bit unsigned integers in *xmm1[31:0]* and *xmm1[95:64]* by the corresponding values in *xmm2* or *mem128*. Writes packed 64-bit unsigned integer products to *xmm1[63:0]* and *xmm1[127:64]*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMULUDQ *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	F4 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULHW, (V)PMULLD, (V)PMULLW, (V)PMULUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

POR VPOR — Packed OR

Performs a bitwise OR of the first and second source operands and writes the result to the destination. When one or both of a pair of corresponding bits in the first and second operands are set, the corresponding bit of the destination is set; when neither source bit is set, the destination bit is cleared.

There are legacy and extended forms of the instruction:

POR

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPOR

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

POR is an SSE2 instruction and VPOR is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
POR *xmm1*, *xmm2*/*mem128*	66 0F EB /r	Performs bitwise OR of values in *xmm1* and *xmm2* or *mem128*. Writes results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPOR *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	EB /r

Related Instructions

(V)PAND, (V)PANDN, (V)PXOR

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSADBW VPSADBW — Packed Sum of Absolute Differences Bytes to Words

Subtracts the 16 packed 8-bit unsigned integers in the second source operand from the corresponding values in the first source operand and computes the absolute difference for each subtraction, then computes two unsigned 16-bit integer sums, one for the eight differences that correspond to the upper eight source bytes, and one for the differences that correspond to the lower eight source bytes. Writes the sums to the destination.

The unsigned 16-bit integer sum of the differences of the eight bytes in bits [127:64] of the source operands is written to bits [15:0] of the destination; bits [63:16] are cleared.

The unsigned 16-bit integer sum of the differences of the eight bytes in bits [63:0] of the source operands is written to bits [79:64] of the destination; bits [127:80] are cleared.

There are legacy and extended forms of the instruction:

PSADBW

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSADBW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSADBW is an SSE2 instruction and VPSADBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSADBW *xmm1*, *xmm2*/*mem128*	66 0F F6 /r	Compute the sum of the absolute differences of two sets of packed 8-bit unsigned integer values in *xmm1* and *xmm2* or *mem128*. Writes 16-bit unsigned integer sums to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSADBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	F6 /r

Related Instructions

(V)MPSADBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSHUFB VPSHUFB — Packed Shuffle Byte

Copies bytes from the first source operand to the destination or clears bytes in the destination, as specified by control bytes in the second source operand.

The control bytes occupy positions in the source operand that correspond to positions in the destination. Each control byte has the following fields.

7	6 — 4	3 — 0
FRZ	Reserved	SRC_Index

Bits	Description
[7]	Set the bit to clear the corresponding byte of the destination. Clear the bit to copy the selected source byte to the corresponding byte of the destination.
[6:4]	Reserved
[3:0]	Binary value selects the source byte.

There are legacy and extended forms of the instruction:

PSHUFB

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSHUFB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSHUFB is an SSSE3 instruction and VPSHUFB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSHUFB *xmm1*, *xmm2*/*mem128*	66 0F 38 00 /r	Moves bytes in *xmm1* as specified by control bytes in *xmm2* or *mem128*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHUFB *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\textit{src}}$.0.01	00 /r

Related Instructions

(V)PSHUFD, (V)PSHUFW, (V)PSHUHW, (V)PSHUFLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSHUFD VPSHUFD — Packed Shuffle Doublewords

Copies packed doubleword values from a source to a doubleword in the destination, as specified by bit fields of an immediate byte operand. A source doubleword can be copied more than once.

Source doublewords are selected by two-bit fields in the immediate-byte operand. Each bit field corresponds to a destination doubleword, as shown:

Destination Doubleword	Immediate-Byte Bit Field	Value of Bit Field	Source Doubleword
[31:0]	[1:0]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[63:32]	[3:2]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[95:64]	[5:4]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[127:96]	[7:6]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]

There are legacy and extended forms of the instruction:

PSHUFD

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSHUFD

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSHUFD is an SSE2 instruction and VPSHUFD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSHUFD *xmm1*, *xmm2*/*mem128*, *imm8*	66 0F 70 /r ib	Moves packed 32-bit values from *xmm2* or *mem128* to *xmm1*, as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHUFD *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	70 /r ib

Related Instructions

(V)PSHUFHW, (V)PSHUFLW, (V)PSHUFW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSHUFHW VPSHUFHW — Packed Shuffle High Words

Copies packed word values from the high quadword of a source to a word in the high quadword of the destination, as specified by bit fields of an immediate byte operand. A source word can be copied more than once. The low-order quadword of the source is copied to the low-order quadword of the destination.

Source words are selected by two-bit fields in the immediate-byte operand. Each bit field corresponds to a destination word, as shown:

Destination Word	Immediate-Byte Bit Field	Value of Bit Field	Source Word
[79:64]	[1:0]	00	[79:64]
		01	[95:80]
		10	[111:96]
		11	[127:112]
[95:80]	[3:2]	00	[79:64]
		01	[95:80]
		10	[111:96]
		11	[127:112]
[111:96]	[5:4]	00	[79:64]
		01	[95:80]
		10	[111:96]
		11	[127:112]
[127:112]	[7:6]	00	[79:64]
		01	[95:80]
		10	[111:96]
		11	[127:112]

There are legacy and extended forms of the instruction:

PSHUFHW

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSHUFHW

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSHUFHW is an SSE2 instruction and VPSHUFHW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSHUFHW *xmm1*, *xmm2*/*mem128*, *imm8*	F3 0F 70 /r ib	Copies packed 16-bit values from the high-order quadword of *xmm2* or *mem128* to the high-order quadword of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHUFHW *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	70 /r ib

Related Instructions

(V)PSHUFD, (V)PSHUFLW, (V)PSHUFW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSHUFLW VPSHUFLW — Packed Shuffle Low Words

Copies packed word values from the low quadword of a source to a word in the low quadword of the destination, as specified by bit fields of an immediate byte operand. A source word can be copied more than once. The high-order quadword of the source is copied to the high-order quadword of the destination.

Source words are selected by two-bit fields in the immediate-byte operand. Each bit field corresponds to a destination word, as shown:

Destination Word	Immediate-Byte Bit Field	Value of Bit Field	Source Word
[15:0]	[1:0]	00	[15:0]
		01	[31:16]
		10	[47:32]
		11	[63:48]
[31:16]	[3:2]	00	[15:0]
		01	[31:16]
		10	[47:32]
		11	[63:48]
[47:32]	[5:4]	00	[15:0]
		01	[31:16]
		10	[47:32]
		11	[63:48]
[63:48]	[7:6]	00	[15:0]
		01	[31:16]
		10	[47:32]
		11	[63:48]

There are legacy and extended forms of the instruction:

PSHUFLW

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSHUFLW

The extended form of the instruction has 128-bit encoding.

The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSHUFLW is an SSE2 instruction and VPSHUFLW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSHUFLW *xmm1*, *xmm2*/*mem128*, *imm8*	F2 0F 70 /r ib	Copies packed 16-bit values from the low-order quadword of *xmm2* or *mem128* to the low-order quadword of *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHUFLW *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.11	70 /r ib

Related Instructions

(V)PSHUFD, (V)PSHUFHW, (V)PSHUFW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSIGNB VPSIGNB — Packed Sign Byte

For each packed signed byte in the first source operand, evaluate the corresponding byte of the second source operand and perform one of the following operations.

- When a byte of the second source is negative, write the two's-complement of the corresponding byte of the first source to the destination.
- When a byte of the second source is positive, copy the corresponding byte of the first source to the destination.
- When a byte of the second source is zero, clear the corresponding byte of the destination.

There are legacy and extended forms of the instruction:

PSIGNB

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSIGNB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSIGNB is an SSSE3 instruction and VPSIGNB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSIGNB *xmm1*, *xmm2*/*mem128*	66 0F 38 08 /r	Perform operation based on evaluation of each packed 8-bit signed integer value in *xmm2* or *mem128*. Write 8-bit signed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSIGNB *xmm1*, xmm2*, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{\mathit{src}}$.0.01	08 /r

Related Instructions

(V)PSIGNW, (V)PSIGND

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSIGND VPSIGND — Packed Sign Doubleword

For each packed signed doubleword in the first source operand, evaluate the corresponding doubleword of the second source operand and perform one of the following operations.

- When a doubleword of the second source is negative, write the two's-complement of the corresponding doubleword of the first source to the destination.
- When a doubleword of the second source is positive, copy the corresponding doubleword of the first source to the destination.
- When a doubleword of the second source is zero, clear the corresponding doubleword of the destination.

There are legacy and extended forms of the instruction:

PSIGND

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSIGND

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSIGND is an SSSE3 instruction and VPSIGND is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSIGND *xmm1*, *xmm2*/*mem128*	66 0F 38 0A /r	Perform operation based on evaluation of each packed 32-bit signed integer value in *xmm2* or *mem128*. Write 32-bit signed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSIGND *xmm1*, xmm2*, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	0A /r

Related Instructions

(V)PSIGNB, (V)PSIGNW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSIGNW VPSIGNW — Packed Sign Word

For each packed signed word in the first source operand, evaluate the corresponding word of the second source operand and perform one of the following operations.

- When a word of the second source is negative, write the two's-complement of the corresponding word of the first source to the destination.
- When a word of the second source is positive, copy the corresponding word of the first source to the destination.
- When a word of the second source is zero, clear the corresponding word of the destination.

There are legacy and extended forms of the instruction:

PSIGNW

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSIGNW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSIGNW is an SSSE3 instruction and VPSIGNW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSSE3] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSIGNW *xmm1*, *xmm2*/*mem128*	66 0F 38 09 /r	Perform operation based on evaluation of each packed 16-bit signed integer value in *xmm2* or *mem128*. Write 16-bit signed results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSIGNW *xmm1*, xmm2*, xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.$\overline{src}$.0.01	09 /r

Related Instructions

(V)PSIGNB, (V)PSIGND

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSLLD VPSLLD — Packed Shift Left Logical Doublewords

Left-shifts each packed 32-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands.Only bits [63:0] of the source register or memory location are used to generate the shift count.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 31, the destination is cleared.

There are legacy and extended forms of the instruction:

PSLLD

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSLLD

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSLLD is an SSE2 instruction and VPSLLD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSLLD *xmm1*, *xmm2*/*mem128*	66 0F F2 /r	Left-shifts packed doublewords in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSLLD *xmm*, *imm8*	66 0F 72 /6 ib	Left-shifts packed doublewords in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSLLD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\mathit{src}}$.0.01	F2 /r
VPSLLD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	72 /6 ib

Related Instructions

(V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSLLDQ VPSLLDQ — Packed Shift Left Logical Double Quadword

Left-shifts the double quadword value in an XMM register the number of bytes specified by an immediate byte operand and writes the shifted values to the destination.

Low-order bytes emptied by shifting are cleared. When the shift value is greater than 15, the destination is cleared.

There are legacy and extended forms of the instruction:

PSLLDQ

The source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSLLDQ

The source operand is an XMM register. The destination is an XMM register specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSLLDQ is an SSE2 instruction and VPSLLDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSLLDQ *xmm*, *imm8*	66 0F 73 /7 ib	Left-shifts double quadword value in *xmm1* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSLLDQ *xmm1*, xmm2, *imm8*	C4	$\overline{\text{RXB}}$.00001	0.$\overline{\text{dest}}$.0.01	73 /7 ib

Related Instructions

(V)PSLLD, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSLLQ VPSLLQ — Packed Shift Left Logical Quadwords

Left-shifts each packed 64-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands.Only bits [63:0] of the source register or memory location are used to generate the shift count.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 64, the destination is cleared.

There are legacy and extended forms of the instruction:

PSLLQ

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSLLQ

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSLLQ is an SSE2 instruction and VPSLLQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSLLQ *xmm1*, *xmm2*/*mem128*	66 0F F3 /r	Left-shifts packed quadwords in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSLLQ *xmm*, *imm8*	66 0F 73 /6 ib	Left-shifts packed quadwords in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSLLQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	F3 /r
VPSLLQ *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	73 /6 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSLLW VPSLLW — Packed Shift Left Logical Words

Left-shifts each packed 16-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands.Only bits [63:0] of the source register or memory location are used to generate the shift count.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 64, the destination is cleared.

There are legacy and extended forms of the instruction:

PSLLW

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSLLW

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSLLW is an SSE2 instruction and VPSLLW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSLLW *xmm1*, *xmm2*/*mem128*	66 0F F1 /r	Left-shifts packed words in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSLLW *xmm*, *imm8*	66 0F 71 /6 ib	Left-shifts packed words in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSLLW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	F1 /r
VPSLLW *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	71 /6 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSRAD VPSRAD — Packed Shift Right Arithmetic Doublewords

Right-shifts each packed 32-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands.Only bits [63:0] of the source register or memory location are used to generate the shift count.

High-order bits emptied by shifting are filled with the sign bit of the initial value. When the shift value is greater than 31, each doubleword of the destination is filled with the sign bit of its initial value.

There are legacy and extended forms of the instruction:

PSRAD

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSRAD

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSRAD is an SSE2 instruction and VPSRAD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSRAD *xmm1*, *xmm2/mem128*	66 0F E2 /r	Right-shifts packed doublewords in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSRAD *xmm*, *imm8*	66 0F 72 /4 ib	Right-shifts packed doublewords in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSRAD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	F2 /r
VPSRAD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	72 /4 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW

T

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSRAW VPSRAW — Packed Shift Right Arithmetic Words

Right-shifts each packed 16-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands.Only bits [63:0] of the source register or memory location are used to generate the shift count.

High-order bits emptied by shifting are filled with the sign bit of the initial value. When the shift value is greater than 31, each doubleword of the destination is filled with the sign bit of its initial value.

There are legacy and extended forms of the instruction:

PSRAW

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSRAW

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSRAW is an SSE2 instruction and VPSRAW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSRAW *xmm1*, *xmm2*/*mem128*	66 0F E1 /r	Right-shifts packed words in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSRAW *xmm*, *imm8*	66 0F 71 /4 ib	Right-shifts packed words in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSRAW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	E1 /r
VPSRAW *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	71 /4 ib

Related Instructions

PSLLD, PSLLDQ, PSLLQ, PSLLW, PSRAD, PSRLD, PSRLDQ, PSRLQ, PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PSRLD VPSRLD — Packed Shift Right Logical Doublewords

Right-shifts each packed 32-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands.Only bits [63:0] of the source register or memory location are used to generate the shift count.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 31, the destination is cleared.

There are legacy and extended forms of the instruction:

PSRLD

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSRLD

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSRLD is an SSE2 instruction and VPSRLD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PSRLD *xmm1*, *xmm2/mem128*	66 0F D2 /r	Right-shifts packed doublewords in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSRLD *xmm*, *imm8*	66 0F 72 /2 ib	Right-shifts packed doublewords in *xmm* as specified by *imm8*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSRLD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	D2 /r
VPSRLD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	72 /2 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSRLDQ VPSRLDQ — Packed Shift Right Logical Double Quadword

Right-shifts the double quadword value in an XMM register the number of bytes specified by an immediate byte operand and writes the shifted values to the destination.

High-order bytes emptied by shifting are cleared. When the shift value is greater than 15, the destination is cleared.

There are legacy and extended forms of the instruction:

PSRLDQ

The source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSRLDQ

The source operand is an XMM register. The destination is an XMM register specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSRLDQ is an SSE2 instruction and VPSRLDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSRLDQ *xmm*, *imm8*	66 0F 73 /3 ib	Right-shifts double quadword value in *xmm1* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSRLDQ *xmm1*, xmm2, *imm8*	C4	$\overline{\text{RXB}}$.00001	0.$\overline{\text{dest}}$.0.01	73 /3 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSRLQ VPSRLQ — Packed Shift Right Logical Quadwords

Right-shifts each packed 64-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands. Only bits [63:0] of the source register or memory location are used to generate the shift count.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 31, the destination is cleared.

There are legacy and extended forms of the instruction:

PSRLQ

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSRLQ

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSRLQ is an SSE2 instruction and VPSRLQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSRLQ *xmm1*, *xmm2*/*mem128*	66 0F D3 /r	Right-shifts packed quadwords in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSRLQ *xmm*, *imm8*	66 0F 73 /2 ib	Right-shifts packed quadwords in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSRLQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	D3 /r
VPSRLQ *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	73 /2 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSRLW VPSRLW — Packed Shift Right Logical Words

Right-shifts each packed 16-bit value in the source operand as specified by a shift-count operand and writes the shifted values to the destination.

The shift-count operand can be an immediate byte, a source register, or a memory location. There are different encodings for immediate operands and register/memory operands. Only bits [63:0] of the source register or memory location are used to generate the shift count.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 31, the destination is cleared.

There are legacy and extended forms of the instruction:

PSRLW

There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSRLW

The extended form of the instruction has 128-bit encoding. There are two forms of the instruction, based on the type of count operand.

The first source operand is an XMM register. The shift count is specified by either a second XMM register or a 128-bit memory location, or by an immediate 8-bit operand. The destination is an XMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSRLW is an SSE2 instruction and VPSRLW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSRLW *xmm1*, *xmm2*/*mem128*	66 0F D1 /r	Right-shifts packed words in *xmm1* as specified by *xmm2[63:0]* or *mem128[63:0]*.
PSRLW *xmm*, *imm8*	66 0F 71 /2 ib	Right-shifts packed words in *xmm* as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSRLQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	D1 /r
VPSRLQ *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{dest}}$.0.01	71 /2 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBB VPSUBB — Packed Subtract Bytes

Subtracts 16 packed 8-bit integer values in the second source operand from the corresponding values in the first source operand and writes the integer differences to the corresponding bytes of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PSUBB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSUBB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBB is an SSE2 instruction and VPSUBB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBB *xmm1*, *xmm2*/*mem128*	66 0F F8 /r	Subtracts 8-bit signed integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	F8 /r

Related Instructions

(V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBD VPSUBD — Packed Subtract Doublewords

Subtracts four packed 32-bit integer values in the second source operand from the corresponding values in the first source operand and writes the integer differences to the corresponding doubleword of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PSUBD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSUBD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

The PSUBD is an SSE2 instruction and VPSUBD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBD *xmm1*, *xmm2*/*mem128*	66 0F FA /r	Subtracts packed 32-bit integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{src}}$.0.01	FA /r

Related Instructions

(V)PSUBB, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBQ VPSUBQ — Packed Subtract Quadword

Subtracts two packed 64-bit integer values in the second source operand from the corresponding values in the first source operand and writes the differences to the corresponding quadword of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PSUBQ

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSUBQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBQ is an SSE2 instruction and VPSUBQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBQ *xmm1*, *xmm2/mem128*	66 0F FB /r	Subtracts packed 64-bit integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	FB /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBSB VPSUBSB Packed Subtract Signed With Saturation Bytes

Subtracts 16 packed 8-bit signed integer value in the second source operand from the corresponding values in the first source operand and writes the signed integer differences to the corresponding byte of the destination.

For each packed value in the destination, if the value is larger than the largest signed 8-bit integer, it is saturated to 7Fh, and if the value is smaller than the smallest signed 8-bit integer, it is saturated to 80h.

There are legacy and extended forms of the instruction:

PSUBSB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSUBSB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBSB is an SSE2 instruction and VPSUBSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBSB *xmm1*, *xmm2*/*mem128*	66 0F E8 /r	Subtracts packed 8-bit signed integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	E8 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBSW VPSUBSW — Packed Subtract Signed With Saturation Words

Subtracts eight packed 16-bit signed integer values in the second source operand from the corresponding values in the first source operand and writes the signed integer differences to the corresponding word of the destination.

Positive differences greater than 7FFFh are saturated to 7FFFh; negative differences less than 8000h are saturated to 8000h.

There are legacy and extended forms of the instruction:

PSUBSW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSUBSW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBSW is an SSE2 instruction and VPSUBSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBSW *xmm1*, *xmm2*/*mem128*	66 0F E9 /r	Subtracts packed 16-bit signed integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	E9 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBUSB VPSUBUSB — Packed Subtract Unsigned With Saturation Bytes

Subtracts 16 packed 8-bit unsigned integer value in the second source operand from the corresponding values in the first source operand and writes the unsigned integer difference to the corresponding byte of the destination.

Differences greater than 7Fh are saturated to 7Fh; differences less than 00h are saturated to 00h.

There are legacy and extended forms of the instruction:

PSUBUSB

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSUBUSB

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBUSB is an SSE2 instruction and VPSUBUSB is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBUSB *xmm1*, *xmm2*/*mem128*	66 0F D8 /r	Subtracts packed byte unsigned integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBUSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	D8 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSW, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBUSW VPSUBUSW Packed Subtract Unsigned With Saturation Words

Subtracts eight packed 16-bit unsigned integer value in the second source operand from the corresponding values in the first source operand and writes the unsigned integer differences to the corresponding word of the destination.

Differences greater than FFFFh are saturated to FFFFh; differences less than 0000h are saturated to 0000h.

There are legacy and extended forms of the instruction:

PSUBUSW

The first source operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSUBUSW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBUSW is an SSE2 instruction and VPSUBUSW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PSUBUSW *xmm1*, *xmm2*/*mem128*	66 0F D9 /r	Subtracts packed 16-bit unsigned integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSUBUSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	D9 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PSUBW VPSUBW — Packed Subtract Words

Subtracts eight packed 16-bit integer values in the second source operand from the corresponding values in the first source operand and writes the integer differences to the corresponding word of the destination.

This instruction operates on both signed and unsigned integers. When a result overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set), and only the low-order 8 bits of each result are written to the destination.

There are legacy and extended forms of the instruction:

PSUBW

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPSUBW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PSUBW is an SSE2 instruction and VPSUBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PSUBW *xmm1*, *xmm2/mem128*	66 0F F9 /r	Subtracts packed 16-bit integer values in *xmm2* or *mem128* from corresponding values in *xmm1*. Writes the differences to *xmm1*

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSUBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	F9 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PTEST VPTEST — Packed Bit Test

First, performs a bitwise AND of the first source operand with the second source operand.
Sets rFLAGS.ZF when all bit operations = 0; else, clears ZF.

Second. performs a bitwise AND of the second source operand with the logical complement (NOT) of the first source operand. Sets rFLAGS.CF when all bit operations = 0; else, clears CF.

Neither source operand is modified.

There are legacy and extended forms of the instruction:

PTEST

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location.

VPTEST

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location.

YMM Encoding

The first source operand is a YMM register. The second source operand is a YMM register or 256-bit memory location.

PTEST is an SSE4.1 instruction and VPTEST is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PTEST *xmm1*, *xmm2/mem128*	66 0F 38 17 /r	Set ZF if bitwise AND of *xmm2/m128* with *xmm1* = 0; else, clear ZF. Set CF if bitwise AND of *xmm2/m128* with NOT*xmm1* = 0; else, clear CF.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPTEST *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.0.01	17 /r
VPTEST *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00010	X.1111.1.01	17 /r

Related Instructions

VTESTPD, VTESTPS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	M	0	0	M
21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3 and 1 are reserved. A flag set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

PUNPCKHBW VPUNPCKHBW — Unpack and Interleave High Bytes

Unpacks the high-order bytes of the first and second source operands and interleaves the 16 values into the destination. The low-order bytes of the source operands are ignored.

Bytes are interleaved in ascending order from the lsb of the sources and the destination. Bits [71:64] of the first source are written to bits [7:0] of the destination; bits [71:64] of the second source are written to bits [15:8] of the destination and so on, ending with bits [127:120] of the second source in bits [127:120] of the destination

When the second source operand is all 0s, the destination effectively contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKHBW

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKHBW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKHBW is an SSE2 instruction and VPUNPCKHBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKHBW *xmm1*, *xmm2*/*mem128*	66 0F 68 /r	Unpacks and interleaves the high-order bytes of *xmm1* and *xmm2* or *mem128*. Writes the bytes to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKHBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	68 /r

Related Instructions

(V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKHDQ VPUNPCKHDQ — Unpack and Interleave High Doublewords

Unpacks the high-order doublewords of the first and second source operands and interleaves the four values into the destination. The low-order doublewords of the source operands are ignored.

Doublewords are interleaved in ascending order from the lsb of the sources and the destination. Bits [95:64] of the first source are written to bits [31:0] of the destination; bits [95:64] of the second source are written to bits [63:32] of the destination and so on, ending with bits [127:96] of the second source in bits [127:96] of the destination

When the second source operand is all 0s, the destination effectively contains the doublewords from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKHDQ

The first source operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKHDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKHDQ is an SSE2 instruction and VPUNPCKHDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKHDQ *xmm1*, *xmm2*/*mem128*	66 0F 6A /r	Unpacks and interleaves the high-order doublewords of *xmm1* and *xmm2* or *mem128*. Writes the doublewords to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKHDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	6A /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKHQDQ VPUNPCKHQDQ — Unpack and Interleave High Quadwords

Unpacks the high-order quadwords of the first and second source operands and interleaves the two values into the destination. The low-order bytes of the source operands are ignored.

Quadwords are interleaved in ascending order from the lsb of the sources and the destination. Bits [127:64] of the first source are written to bits [63:0] of the destination; bits [127:64] of the second source are written to bits [127:64] of the destination.

When the second source operand is all 0s, the destination effectively contains the quadword from the first source operand zero-extended to 128 bits. This operation is useful for expanding unsigned 64-bit values to unsigned 128-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKHQDQ

The first source operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKHQDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKHQDQ is an SSE2 instruction and VPUNPCKHQDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKHQDQ *xmm1*, *xmm2*/*mem128*	66 0F 6D /r	Unpacks and interleaves the high-order quadwords of *xmm1* and *xmm2* or *mem128*. Writes the bytes to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKHQDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	6D /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKHWD VPUNPCKHWD — Unpack and Interleave High Words

Unpacks the high-order words of the first and second source operands and interleaves the eight values into the destination. The low-order words of the source operands are ignored.

Words are interleaved in ascending order from the lsb of the sources and the destination. Bits [79:64] of the first source are written to bits [15:0] of the destination; bits [79:64] of the second source are written to bits [31:16] of the destination and so on, ending with bits [127:112] of the second source in bits [127:112] of the destination

When the second source operand is all 0s, the destination effectively contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKHWD

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKHWD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKHWD is an SSE2 instruction and VPUNPCKHWD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKHWD *xmm1*, *xmm2*/*mem128*	66 0F 69 /r	Unpacks and interleaves the high-order words of *xmm1* and *xmm2* or *mem128*. Writes the words to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKHWD *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	69 /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKLBW VPUNPCKLBW — Unpack and Interleave Low Bytes

Unpacks the low-order bytes of the first and second source operands and interleaves the 16 values into the destination. The high-order bytes of the source operands are ignored.

Bytes are interleaved in ascending order from the lsb of the sources and the destination. Bits [7:0] of the first source are written to bits [7:0] of the destination; bits [7:0] of the second source are written to bits [15:8] of the destination and so on, ending with bits [63:56] of the second source in bits [127:120] of the destination

When the second source operand is all 0s, the destination effectively contains the bytes from the first source operand zero-extended to 16 bits. This operation is useful for expanding unsigned 8-bit values to unsigned 16-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKLBW

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKLBW

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKLBW is an SSE2 instruction and VPUNPCKLBW is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKLBW *xmm1*, *xmm2*/*mem128*	66 0F 60 /r	Unpacks and interleaves the low-order bytes of *xmm1* and *xmm2* or *mem128*. Writes the bytes to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKLBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	69 /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKLDQ VPUNPCKLDQ — Unpack and Interleave Low Doublewords

Unpacks the low-order doublewords of the first and second source operands and interleaves the four values into the destination. The high-order doublewords of the source operands are ignored.

Doublewords are interleaved in ascending order from the lsb of the sources and the destination. Bits [31:0] of the first source are written to bits [31:0] of the destination; bits [31:0] of the second source are written to bits [63:32] of the destination and so on, ending with bits [63:32] of the second source in bits [127:96] of the destination

When the second source operand is all 0s, the destination effectively contains the doublewords from the first source operand zero-extended to 64 bits. This operation is useful for expanding unsigned 32-bit values to unsigned 64-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKLDQ

The first source operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKLDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKLDQ is an SSE2 instruction and VPUNPCKLDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKLDQ *xmm1*, *xmm2*/*mem128*	66 0F 62 /r	Unpacks and interleaves the low-order doublewords of *xmm1* and *xmm2* or *mem128*. Writes the doublewords to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKLDQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	62 /r

Related Instructions

(V)PUNPCKHW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLQDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKLQDQ VPUNPCKLQDQ — Unpack and Interleave Low Quadwords

Unpacks the low-order quadwords of the first and second source operands and interleaves the two values into the destination. The high-order bytes of the source operands are ignored.

Quadwords are interleaved in ascending order from the lsb of the sources and the destination. Bits [63:0] of the first source are written to bits [63:0] of the destination; bits [63:0] of the second source are written to bits [127:64] of the destination.

When the second source operand is all 0s, the destination effectively contains the quadword from the first source operand zero-extended to 128 bits. This operation is useful for expanding unsigned 64-bit values to unsigned 128-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKLQDQ

The first source operand is an XMM register and the second source operand is another XMM register or 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPUNPCKLQDQ

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKLQDQ is an SSE2 instruction and VPUNPCKLQDQ is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
PUNPCKLQDQ *xmm1*, *xmm2*/*mem128*	66 0F 6C /r	Unpacks and interleaves the low-order quadwords of *xmm1* and *xmm2* or *mem128*. Writes the bytes to *xmm1*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPUNPCKLQDQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	6C /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PUNPCKLWD VPUNPCKLWD — Unpack and Interleave Low Words

Unpacks the low-order words of the first and second source operands and interleaves the eight values into the destination. The high-order words of the source operands are ignored.

Words are interleaved in ascending order from the lsb of the sources and the destination. Bits [15:0] of the first source are written to bits [15:0] of the destination; bits [15:0] of the second source are written to bits [31:16] of the destination and so on, ending with bits [63:48] of the second source in bits [127:112] of the destination

When the second source operand is all 0s, the destination effectively contains the words from the first source operand zero-extended to 32 bits. This operation is useful for expanding unsigned 16-bit values to unsigned 32-bit operands for subsequent processing that requires higher precision.

There are legacy and extended forms of the instruction:

PUNPCKLWD

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source operand is also the destination register. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

PUNPCKLWD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PUNPCKLWD is an SSE2 instruction and VPUNPCKLWD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PUNPCKLWD *xmm1*, *xmm2*/*mem128*	66 0F 61 /r	Unpacks and interleaves the low-order words of *xmm1* and *xmm2* or *mem128*. Writes the words to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPUNPCKLLWD *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	61 /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

PXOR VPXOR — Packed Exclusive OR

Performs a bitwise XOR of the first and second source operands and writes the result to the destination. When either of a pair of corresponding bits in the first and second operands are set, the corresponding bit of the destination is set; when both source bits are set or when both source bits are not set, the destination bit is cleared.

There are legacy and extended forms of the instruction:

PXOR

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source XMM register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VPXOR

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

PXOR is an SSE2 instruction and VPXOR is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
PXOR *xmm1*, *xmm2*/*mem128*	66 0F EF /r	Performs bitwise XOR of values in *xmm1* and *xmm2* or *mem128*. Writes the result to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPXOR *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	EF /r

Related Instructions

(V)PAND, (V)PANDN, (V)POR

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

RCPPS VRCPPS — Reciprocal Packed Single-Precision Floating-Point

Computes the approximate reciprocal of each packed single-precision floating-point value in the source operand and writes the results to the corresponding doubleword of the destination. MXCSR.RC as no effect on the result.

The maximum error is less than or equal to 1.5 * 2^{-12} times the true reciprocal. A source value that is ±zero or denormal returns an infinity of the source value sign. Results that underflow are changed to signed zero. For both SNaN and QNaN source operands, a QNaN is returned.

There are legacy and extended forms of the instruction:

RCPPS

Computes four reciprocals. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VRCPPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Computes four reciprocals. The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Computes eight reciprocals. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

RCPPS is an SSE2 instruction and VRCPPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
RCPPS *xmm1*, *xmm2*/*mem128*	0F 53 /r	Computes reciprocals of packed single-precision floating-point values in *xmm1* or *mem128*. Writes result to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VRCPPS *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	53 /r
VRCPPS *ymm1*, *ymm2*/*mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	53 /r

Related Instructions

(V)RCPSS, (V)RSQRTPS, (V)RSQRTSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

RCPSS VRCPSS — Reciprocal Scalar Single-Precision Floating-Point

Computes the approximate reciprocal of the scalar single-precision floating-point value in a source operand and writes the results to the low-order doubleword of the destination. MXCSR.RC as no effect on the result.

The maximum error is less than or equal to $1.5 * 2^{-12}$ times the true reciprocal. A source value that is ±zero or denormal returns an infinity of the source value sign. Results that underflow are changed to signed zero. For both SNaN and QNaN source operands, a QNaN is returned.

There are legacy and extended forms of the instruction:

RCPSS

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VRCPSS

The extended form of the instruction has both 128-bit encoding.

XMM Encoding

The first source operand and the destination are XMM registers. The second source operand is either an XMM register or a 32-bit memory location. Bits [31:0] of the destination contain the reciprocal; bits [127:32] of the destination are copied from the first source register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

RCPSS is an SSE instruction and VRCPSS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
RCPSS *xmm1*, *xmm2/mem32*	F3 0F 53 /r	Computes reciprocal of scalar single-precision floating-point value in *xmm1* or *mem32*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VRCPSS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	53 /r

Related Instructions

(V)RCPPS, (V)RSQRTPS, (V)RSQRTSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

ROUNDPD VROUNDPD — Round Packed Double-Precision Floating-Point

Rounds two or four double-precision floating-point values as specified by an immediate byte operand. Source values are rounded to integral values and written to the destination as double-precision floating-point values.

SNaN source values are converted to QNaN. When DAZ =1, denormals are converted to zero before rounding.

The immediate byte operand is defined as follows.

7 4	3	2	1 0
Reserved	P	O	RC

Bits	Mnemonic	Description
[7:4]	—	Reserved
[3]	P	Precision Exception
[2]	O	Rounding Control Source
[1:0]	RC	Rounding Control

Precision exception definitions:

Value	Description
0	Normal PE exception
1	PE field is not updated. No precision exception is taken when unmasked.

Rounding control source definitions:

Value	Description
0	MXCSR:RC
1	Use RC field value.

Rounding control definition:

Value	Description
00	Nearest
01	Downward (toward negative infinity)
10	Upward (toward positive infinity)
11	Truncated

There are legacy and extended forms of the instruction:

ROUNDPD

Rounds two source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. There is a third 8-bit immediate operand. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VROUNDPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Rounds two source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. There is a third 8-bit immediate operand. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Rounds four source values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. There is a third 8-bit immediate operand. The destination is a third YMM register.

ROUNDPD is an SSE4.1 instruction and VROUNDPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ROUNDPD *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 09 /r ib	Rounds double-precision floating-point values in *xmm2 or mem128.* Writes rounded double-precision values to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VROUNDPD *xmm1*, *xmm2/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	09 /r ib
VROUNDPD *ymm1*, *xmm2/mem256, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.1.01	09 /r ib

Related Instructions

(V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

ROUNDPS VROUNDPS — Round Packed Single-Precision Floating-Point

Rounds four or eight single-precision floating-point values as specified by an immediate byte operand. Source values are rounded to integral values and written to the destination as single-precision floating-point values.

SNaN source values are converted to QNaN. When DAZ =1, denormals are converted to zero before rounding.

The immediate byte operand is defined as follows.

7 4	3	2	1 0
Reserved	P	O	RC

Bits	Mnemonic	Description
[7:4]	—	Reserved
[3]	P	Precision Exception
[2]	O	Rounding Control Source
[1:0]	RC	Rounding Control

Precision exception definitions:

Value	Description
0	Normal PE exception
1	PE field is not updated. No precision exception is taken when unmasked.

Rounding control source definitions:

Value	Description
0	MXCSR:RC
1	Use RC field value.

Rounding control definition:

Value	Description
00	Nearest
01	Downward (toward negative infinity)
10	Upward (toward positive infinity)
11	Truncated

There are legacy and extended forms of the instruction:

ROUNDPS

Rounds four source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. There is a third 8-bit immediate operand. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VROUNDPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Rounds four source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. There is a third 8-bit immediate operand. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Rounds eight source values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. There is a third 8-bit immediate operand. The destination is a third YMM register.

ROUNDPS is an SSE4.1 instruction and VROUNDPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ROUNDPS *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 08 /r ib	Rounds single-precision floating-point values in *xmm2 or mem128.* Writes rounded single-precision values to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VROUNDPS *xmm1*, *xmm2/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.0.01	08 /r ib
VROUNDPS *ymm1*, *xmm2/mem256, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.1.01	08 /r ib

Related Instructions

(V)ROUNDPD, (V)ROUNDSD, (V)ROUNDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

ROUNDSD VROUNDSD — Round Scalar Double-Precision

Rounds a scalar double-precision floating-point value as specified by an immediate byte operand. Source values are rounded to integral values and written to the destination as double-precision floating-point values.

SNaN source values are converted to QNaN. When DAZ =1, denormals are converted to zero before rounding.

The immediate byte operand is defined as follows.

7 4	3	2	1 0
Reserved	P	O	RC

Bits	Mnemonic	Description
[7:4]	—	Reserved
[3]	P	Precision Exception
[2]	O	Rounding Control Source
[1:0]	RC	Rounding Control

Precision exception definitions:

Value	Description
0	Normal PE exception
1	PE field is not updated. No precision exception is taken when unmasked.

Rounding control source definitions:

Value	Description
0	MXCSR:RC
1	Use RC field value.

Rounding control definition:

Value	Description
00	Nearest
01	Downward (toward negative infinity)
10	Upward (toward positive infinity)
11	Truncated

There are legacy and extended forms of the instruction:

ROUNDSD

The source operand is either an XMM register or a 64-bit memory location. When the source is an XMM register, the value to be rounded must be in the low doubleword. The destination is an XMM register. There is a third 8-bit immediate operand. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to destination XMM register are not affected.

VROUNDSD

The extended form of the instruction has 128-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 64-bit memory location. When the second source is an XMM register, the value to be rounded must be in the low doubleword. The destination is a third XMM register. There is a fourth 8-bit immediate operand. Bits [127:64] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

ROUNDSD is an SSE4.1 instruction and VROUNDSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ROUNDSD *xmm1*, *xmm2/mem64, imm8*	66 0F 3A 0B /r ib	Rounds a double-precision floating-point value in *xmm2[63:0]* or *mem64.* Writes a rounded double-precision value to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VROUNDSD *xmm1*, *xmm2*, *xmm3/mem64, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.X.01	0B /r ib

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					*M*
17	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

ROUNDSS VROUNDSS — Round Scalar Single-Precision

Rounds a scalar single-precision floating-point value as specified by an immediate byte operand. Source values are rounded to integral values and written to the destination as single-precision floating-point values.

SNaN source values are converted to QNaN. When DAZ =1, denormals are converted to zero before rounding.

The immediate byte operand is defined as follows.

7 4	3	2	1 0
Reserved	P	O	RC

Bits	Mnemonic	Description
[7:4]	—	Reserved
[3]	P	Precision Exception
[2]	O	Rounding Control Source
[1:0]	RC	Rounding Control

Precision exception definitions:

Value	Description
0	Normal PE exception
1	PE field is not updated. No precision exception is taken when unmasked.

Rounding control source definitions:

Value	Description
0	MXCSR:RC
1	Use RC field value.

Rounding control definition:

Value	Description
00	Nearest
01	Downward (toward negative infinity)
10	Upward (toward positive infinity)
11	Truncated

There are legacy and extended forms of the instruction:

ROUNDSS

The source operand is either an XMM register or a 32-bit memory location. When the source is an XMM register, the value to be rounded must be in the low word. The destination is an XMM register. There is a third 8-bit immediate operand. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to destination XMM register are not affected.

VROUNDSS

The extended form of the instruction has 128-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 32-bit memory location. When the second source is an XMM register, the value to be rounded must be in the low word. The destination is a third XMM register. There is a fourth 8-bit immediate operand. Bits [127:32] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

ROUNDSS is an SSE4.1 instruction and VROUNDSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[SSE41] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
ROUNDSS *xmm1*, *xmm2*/*mem64, imm8*	66 0F 3A 0A /r ib	Rounds a single-precision floating-point value in *xmm2[63:0]* or *mem64.* Writes a rounded single-precision value to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VROUNDSS *xmm1*, *xmm2*, *xmm3*/*mem64, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{\textit{src}}$.X.01	0A /r ib

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M					*M*
17	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

RSQRTPS VRSQRTPS — Reciprocal Square Root Packed Single-Precision Floating-Point

Computes the approximate reciprocal of the square root of each packed single-precision floating-point value in the source operand and writes the results to the corresponding doublewords of the destination. MXCSR.RC has no effect on the result.

The maximum error is less than or equal to $1.5 * 2^{-12}$ times the true reciprocal square root. A source value that is ±zero or denormal returns an infinity of the source value sign. Negative source values other than –zero and –denormal return a QNaN floating-point indefinite value. For both SNaN and QNaN source operands, a QNaN is returned.

There are legacy and extended forms of the instruction:

RSQRTPS

Computes four values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VRSQRTPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Computes four values The source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Computes eight values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

RSQRTPS is an SSE instruction and VRSQRTPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
RSQRTPS *xmm1*, *xmm2*/*mem128*	0F 52 /r	Computes reciprocals of square roots of packed single-precision floating-point values in *xmm1* or *mem128*. Writes result to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VRSQRTPS *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	52 /r
VRSQRTPS *ymm1*, *ymm2*/*mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	52 /r

Related Instructions

(V)RSQRTSS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSD, (V)SQRTSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

RSQRTSS VRSQRTSS — Reciprocal Square Root Scalar Single-Precision Floating-Point

Computes the approximate reciprocal of the square root of the scalar single-precision floating-point value in a source operand and writes the result to the low-order doubleword of the destination. MXCSR.RC as no effect on the result.

The maximum error is less than or equal to $1.5 * 2^{-12}$ times the true reciprocal square root. A source value that is ±zero or denormal returns an infinity of the source value’s sign. Negative source values other than –zero and –denormal return a QNaN floating-point indefinite value. For both SNaN and QNaN source operands, a QNaN is returned.

RSQRTSS

The source operand is either an XMM register or a 32-bit memory location. The destination is an XMM register. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VRSQRTSS

The extended form of the instruction has both 128-bit encoding.

XMM Encoding

The first source operand and the destination are XMM registers. The second source operand is either an XMM register or a 32-bit memory location. Bits [31:0] of the destination contain the reciprocal; bits [127:32] of the destination are copied from the first source register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

RSQRTSS is an SSE instruction and VSQRTSS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
RSQRTSS *xmm1*, *xmm2/mem32*	F3 0F 52 /r	Computes reciprocal of square root of a scalar single-precision floating-point value in *xmm1* or *mem32*. Writes result to *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VRSQRTSS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.10	52 /r

Related Instructions

(V)RSQRTPS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSD, (V)SQRTSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

SHUFPD VSHUFPD — Shuffle Packed Double-Precision Floating-Point

Copies packed double-precision floating-point values from either of two sources to quadwords in the destination, as specified by bit fields of an immediate byte operand.

Each bit corresponds to a quadword destination. The 128-bit legacy and extended versions of the instruction use bits [1:0]; the 256-bit extended version uses bits [3:0], as shown.

Destination Quadword	Immediate-Byte Bit Field	Value of Bit Field	Source 1 Bits Copied	Source 2 Bits Copied
Used by 128-bit encoding and 256-bit encoding				
[63:0]	[0]	0	[63:0]	—
		1	[127:64]	—
[127:64]	[1]	0	—	[63:0]
		1	—	]127:64]
Used only by 256-bit encoding				
[191:128]	[2]	0	[191:128]	—
		1	[255:192]	—
[255:192]	[3]	0	—	[191:128]
		1	—	[255:192]

There are legacy and extended forms of the instruction:

SHUFPD

Shuffles four source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. There is a third 8-bit immediate operand. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSHUFPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Shuffles four source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. There is a fourth 8-bit immediate operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Shuffles eight source values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register. There is a fourth 8-bit immediate operand.

SHUFPD is an SSE2 instruction and VSHUFPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SHUFPD *xmm1*, *xmm2*/*mem128*, *imm8*	66 0F C6 /r ib	Shuffles packed double-precision floating-point values in *xmm1* and *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSHUFPD *xmm1*, *xmm2, xmm3*/*mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	C6 /r
VSHUFPD *ymm1*, *ymm2, ymm3*/*mem256*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	C6 /r

Related Instructions

(V)SHUFPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

SHUFPS VSHUFPS — Shuffle Packed Single-Precision Floating-Point

Copies packed single-precision floating-point values from either of two sources to doublewords in the destination, as specified by bit fields of an immediate byte operand.

Each bit field corresponds to a doubleword destination. The 128-bit legacy and extended versions of the instruction use a single 128-bit destination; the 256-bit extended version performs duplicate operations on bits [127:0] and bits [255:128] of the source and destination.

Destination Doubleword	Immediate-Byte Bit Field	Value of Bit Field	Source 1 Bits Copied	Source 2 Bits Copied
[31:0]	[1:0]	00	[31:0]	—
		01	[63:32]	—
		10	[95:64]	—
		11	[127:96]	—
[63:32]	[3:2]	00	[31:0]	—
		01	[63:32]	—
		10	[95:64]	—
		11	[127:96]	—
[95:64]	[5:4]	00	—	[31:0]
		01	—	[63:32]
		10	—	[95:64]
		11	—	[127:96]
[127:96]	[7:6]	00	—	[31:0]
		01	—	[63:32]
		10	—	[95:64]
		11	—	[127:96]
Upper 128 bits of 256-bit source and destination used by 256-bit encoding				
[159:128]	[1:0]	00	[159:128]	—
		01	[191:160]	—
		10	[223:192]	—
		11	[255:224]	—
[191:160]	[3:2]	00	[159:128]	—
		01	[191:160]	—
		10	[223:192]	—
		11	[255:224]	—
[223:192]	[5:4]	00	—	[159:128]
		01	—	[191:160]
		10	—	[223:192]
		11	—	[255:224]
[255:224]	[7:6]	00	—	[159:128]
		01	—	[191:160]
		10	—	[223:192]
		11	—	[255:224]

There are legacy and extended forms of the instruction:

SHUFPS

Shuffles eight source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. There is a third 8-bit immediate operand. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSHUFPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Shuffles eight source values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. There is a fourth 8-bit immediate operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Shuffles 16 source values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register. There is a fourth 8-bit immediate operand.

SHUFPS is an SSE instruction and VSHUFPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SHUFPS *xmm1*, *xmm2/mem128*, *imm8*	0F C6 /r ib	Shuffles packed single-precision floating-point values in *xmm1* and *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSHUFPS *xmm1*, *xmm2, xmm3/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	C6 /r
VSHUFPS *ymm1*, *ymm2, ymm3/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	C6 /r

Related Instructions

(V)SHUFPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

SQRTPD VSQRTPD — Square Root Packed Double-Precision Floating-Point

Computes the square root of each packed double-precision floating-point value in a source operand and writes the result to the corresponding quadword of the destination.

Performing the square root of +infinity returns +infinity.

There are legacy and extended forms of the instruction:

SQRTPD

Computes two values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSQRTPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Computes two values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Computes four values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

SQRTPD is an SSE2 instruction and VSQRTPD is an AVX instruction.Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SQRTPD *xmm1*, *xmm2/mem128*	66 0F 51 /r	Computes square roots of packed double-precision floating-point values in *xmm1* or *mem128*. Writes the results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSQRTPD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.01	51 /r
VSQRTPD *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.01	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPS, (V)SQRTSD, (V)SQRTSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M				M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

SQRTPS VSQRTPS — Square Root Packed Single-Precision Floating-Point

Computes the square root of each packed single-precision floating-point value in a source operand and writes the result to the corresponding doubleword of the destination.

Performing the square root of +infinity returns +infinity.

There are legacy and extended forms of the instruction:

SQRTPS

Computes four values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSQRTPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Computes four values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Computes eight values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

SQRTPS is an SSE instruction and VSQRTPS is an AVX instruction.Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SQRTPS *xmm1*, *xmm2*/*mem128*	0F 51 /r	Computes square roots of packed single-precision floating-point values in *xmm1* or *mem128*. Writes the results to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSQRTPS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	51 /r
VSQRTPS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPD, (V)SQRTSD, (V)SQRTSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M				M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

SQRTSD / VSQRTSD — Square Root Scalar Double-Precision Floating-Point

Computes the square root of a double-precision floating-point value and writes the result to the low doubleword of the destination.

Performing the square root of +infinity returns +infinity.

There are legacy and extended forms of the instruction:

SQRTSD

The source operand is either an XMM register or a 64-bit memory location. When the source is an XMM register, the source value must be in the low doubleword. The destination is an XMM register. Bits [127:64] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to destination XMM register are not affected.

VSQRTSD

The extended form of the instruction has 128-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 64-bit memory location. When the second source is an XMM register, the source value must be in the low doubleword. The destination is a third XMM register. Bits [127:64] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

SQRTSD is an SSE2 instruction and VSQRTSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
SQRTSD *xmm1*, *xmm2*/*mem64*	F2 0F 51 /r	Computes the square root of a double-precision floating-point value in *xmm1* or *mem64*. Writes the result to *xmm1*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VSQRTSD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M				M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

SQRTSS / VSQRTSS — Square Root Scalar Single-Precision Floating-Point

Computes the square root of a single-precision floating-point value and writes the result to the low word of the destination.

Performing the square root of +infinity returns +infinity.

There are legacy and extended forms of the instruction:

SQRTSS

The source operand is either an XMM register or a 32-bit memory location. When the source is an XMM register, the source value must be in the low word. The destination is an XMM register. Bits [127:32] of the destination are not affected. Bits [255:128] of the YMM register that corresponds to destination XMM register are not affected.

VSQRTSS

The extended form of the instruction has 128-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 32-bit memory location. When the second source is an XMM register, the source value must be in the low word. The destination is a third XMM register. Bits [127:32] of the destination are copied from the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

SQRTSS is an SSE instruction and VSQRTSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SQRTSS *xmm1*, *xmm2*/*mem32*	F3 0F 51 /r	Computes square root of a single-precision floating-point value in *xmm1* or *mem32*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSQRTSS *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M				M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

STMXCSR VSTMXCSR — Store MXCSR

Saves the content of the MXCSR extended control/status register to a 32-bit memory location. Reserved bits are stored as zeroes. The MXCSR is described in “Registers” in Volume 1.

For both legacy STMXCSR and extended VSTMXCSR forms of the instruction, the source operand is the MXCSR and the destination is a 32-bit memory location.

STMXCSR is an SSE instruction and VSTMXCSR is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
STMXCSR *mem32*	0F AE /3	Stores content of MXCSR in *mem32*.

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VSTMXCSR *mem32*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	AE /3

Related Instructions

(V)LDMXCSR

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
	S	S	S	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

SUBPD VSUBPD — Subtract Packed Double-Precision Floating-Point

Subtracts each packed double-precision floating-point value of the second source operand from the corresponding value of the first source operand and writes the difference to the corresponding quadword of the destination.

There are legacy and extended forms of the instruction:

SUBPD

Subtracts two pairs of values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSUBPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Subtracts two pairs of values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Subtracts four pairs of values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

SUBPD is an SSE2 instruction and VSUBPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SUBPD *xmm1*, *xmm2/mem128*	66 0F 5C /r	Subtracts packed double-precision floating-point values in *xmm2* or *mem128* from corresponding values of *xmm1*. Writes differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSUBPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	5C /r
VSUBPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	5C /r

Related Instructions

(V)SUBPS, (V)SUBSD, (V)SUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

SUBPS VSUBPS — Subtract Packed Single-Precision Floating-Point

Subtracts each packed single-precision floating-point value of the second source operand from the corresponding value of the first source operand and writes the difference to the corresponding quadword of the destination.

There are legacy and extended forms of the instruction:

SUBPS

Subtracts four pairs of values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VSUBPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Subtracts four pairs of values. The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Subtracts eight pairs of values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

SUBPS is an SSE instruction and VSUBPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SUBPS *xmm1*, *xmm2*/*mem128*	0F 5C /r	Subtracts packed single-precision floating-point values in *xmm2* or *mem128* from corresponding values of *xmm1*. Writes differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSUBPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{src}}$.0.00	5C /r
VSUBPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\text{src}}$.1.00	5C /r

Related Instructions

(V)SUBPD, (V)SUBSD, (V)SUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

SUBSD VSUBSD — Subtract Scalar Double-Precision Floating-Point

Subtracts the double-precision floating-point value in the low-order quadword of the second source operand from the corresponding value in the first source operand and writes the result to the low-order quadword of the destination

SUBSD

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The first source register is also the destination register. Bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are not affected.

VSUBSD

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 64-bit memory location. The destination is a third XMM register. Bits [127:64] of the first source operand are copied to bits [127:64] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

SUBSD is an SSE2 instruction and VSUBSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SUBSD *xmm1*, *xmm2/mem64*	F2 0F 5C /r	Subtracts low-order double-precision floating-point value in *xmm2* or *mem64* from the corresponding value of *xmm1*. Writes the difference to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSUBSD *xmm1*, *xmm2*, *xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	5C /r

Related Instructions

(V)SUBPD, (V)SUBPS, (V)SUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

SUBSS VSUBSS — Subtract Scalar Single-Precision Floating-Point

Subtracts the single-precision floating-point value in the low-order word of the second source operand from the corresponding value in the first source operand and writes the result to the low-order word of the destination

SUBSS

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The first source register is also the destination register. Bits [127:32] of the destination and bits [255:128] of the corresponding YMM register are not affected.

VSUBSS

The extended form of the instruction has 128-bit encoding.

The first source operand is an XMM register and the second source operand is either an XMM register or a 32-bit memory location. The destination is a third XMM register. Bits [127:32] of the first source operand are copied to bits [127:32] of the destination. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

SUBSS is an SSE2 instruction and VSUBSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
SUBSS *xmm1*, *xmm2*/*mem32*	F3 0F 5C /r	Subtracts a low-order single-precision floating-point value in *xmm2* or *mem32* from the corresponding value of *xmm1*. Writes the difference to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VSUBSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.X.10	5C /r

Related Instructions

(V)SUBPD, (V)SUBPS, (V)SUBSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *M indicates a flag that may be modified (set or cleared). Blanks indicate flags that are not affected.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

UCOMISD VUCOMISD — Unordered Compare Scalar Double-Precision Floating-Point

Performs an unordered comparison of a double-precision floating-point value in the low-order 64 bits of an XMM register with a double-precision floating-point value in the low-order 64 bits of another XMM register or a 64-bit memory location.

The ZF, PF, and CF bits in the rFLAGS register reflect the result of the compare as follows.

Result of Compare	ZF	PF	CF
Unordered	1	1	1
Greater Than	0	0	0
Less Than	0	0	1
Equal	1	0	0

The OF, AF, and SF bits in rFLAGS are cleared. If the instruction causes an unmasked SIMD floating-point exception (#XF), the rFLAGS bits are not updated.

The result is unordered when one or both of the operand values is a NaN. UCOMISD signals a SIMD floating-point invalid operation exception (#I) only when a source operand is an SNaN.

The legacy and extended forms of the instruction operate in the same way.

UCOMISD is an SSE2 instruction and VUCOMISD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
UCOMISD *xmm1*, *xmm2/mem64*	66 0F 2E /r	Compares scalar double-precision floating-point values in *xmm1* and *xmm2* or *mem64*. Sets rFLAGS.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VUCOMISD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.X.01	2E /r

Related Instructions

(V)CMPPD, (V)CMPPS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)UCOMISS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	M	0	M	M
21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag set or cleared is M (modified). Unaffected flags are blank.*
Note: *If the instruction causes an unmasked SIMD floating-point exception (#XF), the rFLAGS bits are not updated.*

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

UCOMISS VUCOMISS — Unordered Compare Scalar Single-Precision Floating-Point

Performs an unordered comparison of a single-precision floating-point value in the low-order 32 bits of an XMM register with a double-precision floating-point value in the low-order 32 bits of another XMM register or a 32-bit memory location.

The ZF, PF, and CF bits in the rFLAGS register reflect the result of the compare as follows.

Result of Compare	ZF	PF	CF
Unordered	1	1	1
Greater Than	0	0	0
Less Than	0	0	1
Equal	1	0	0

The OF, AF, and SF bits in rFLAGS are cleared. If the instruction causes an unmasked SIMD floating-point exception (#XF), the rFLAGS bits are not updated.

The result is unordered when one or both of the operand values is a NaN. UCOMISD signals a SIMD floating-point invalid operation exception (#I) only when a source operand is an SNaN.

The legacy and extended forms of the instruction operate in the same way.

UCOMISS is an SSE instruction and VUCOMISS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
UCOMISS *xmm1*, *xmm2/mem32*	0F 2E /r	Compares scalar double-precision floating-point values in *xmm1* and *xmm2* or *mem64*. Sets rFLAGS.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VUCOMISS *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.00001	X.1111.X.00	2E /r

Related Instructions

(V)CMPPD, (V)CMPPS, (V)CMPSD, (V)CMPSS, (V)COMISD, (V)COMISS, (V)UCOMISD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				0	M	0	M	M
21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3, and 1 are reserved. A flag set or cleared is M (modified). Unaffected flags are blank.*
Note: *If the instruction causes an unmasked SIMD floating-point exception (#XF), the rFLAGS bits are not updated.*

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
															M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

UNPCKHPD VUNPCKHPD — Unpack High Double-Precision Floating-Point

Unpacks the high-order double-precision floating-point values of the first and second source operands and interleaves the values into the destination. Bits [63:0] of the source operands are ignored.

Values are interleaved in ascending order from the lsb of the sources and the destination. Bits [127:64] of the first source are written to bits [63:0] of the destination; bits [127:64] of the second source are written to bits [127:64] of the destination.For the 256-bit encoding, the process is repeated for bits [255:192] of the sources and bits [255:128] of the destination.

There are legacy and extended forms of the instruction:

UNPCKHPD

Interleaves one pair of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VUNPCKHPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Interleaves one pair of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Interleaves two pairs of values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

UNPCKHPD is an SSE2 instruction and VUNPCKHPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
UNPCKHPD *xmm1*, *xmm2/mem128*	66 0F 15 /r	Unpacks the high-order double-precision floating-point values in *xmm1* and *xmm2* or *mem128* and interleaves them into *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VUNPCKHPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	15 /r
VUNPCKHPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.01	15 /r

Related Instructions

(V)UNPCKHPS, (V)UNPCKLPD, (V)UNPCKLPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

UNPCKHPS VUNPCKHPS — Unpack High Single-Precision Floating-Point

Unpacks the high-order single-precision floating-point values of the first and second source operands and interleaves the values into the destination. Bits [63:0] of the source operands are ignored.

Values are interleaved in ascending order from the lsb of the sources and the destination. Bits [95:64] of the first source are written to bits [31:0] of the destination; bits [95:64] of the second source are written to bits [63:32] of the destination and so on, ending with bits [127:96] of the second source in bits [127:96] of the destination. For the 256-bit encoding, the process continues for bits [255:192] of the sources and bits [255:128] of the destination.

There are legacy and extended forms of the instruction:

UNPCKHPS

Interleaves two pairs of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VUNPCKHPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Interleaves two pairs of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Interleaves four pairs of values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

UNPCKHPS is an SSE instruction and VUNPCKHPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
UNPCKHPS *xmm1*, *xmm2/mem128*	0F 15 /r	Unpacks the high-order single-precision floating-point values in *xmm1* and *xmm2* or *mem128* and interleaves them into *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VUNPCKHPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	15 /r
VUNPCKHPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	15 /r

Related Instructions

(V)UNPCKHPD, (V)UNPCKLPD, (V)UNPCKLPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

UNPCKLPD VUNPCKLPD — Unpack Low Double-Precision Floating-Point

Unpacks the low-order double-precision floating-point values of the first and second source operands and interleaves the values into the destination. Bits [127:64] of the source operands are ignored.

Values are interleaved in ascending order from the lsb of the sources and the destination. Bits [63:0] of the first source are written to bits [63:0] of the destination; bits [63:0] of the second source are written to bits [127:64] of the destination. For the 256-bit encoding, the process is repeated for bits [191:128] of the sources and bits [255:128] of the destination.

There are legacy and extended forms of the instruction:

UNPCKLPD

Interleaves one pair of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VUNPCKLPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Interleaves one pair of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Interleaves two pairs of values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

UNPCKLPD is an SSE2 instruction and VUNPCKLPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
UNPCKLPD *xmm1*, *xmm2/mem128*	66 0F 14 /r	Unpacks the low-order double-precision floating-point values in *xmm1* and *xmm2* or *mem128* and interleaves them into *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VUNPCKLPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.01	14 /r
VUNPCKLPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.01	14 /r

Related Instructions

(V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

UNPCKLPS VUNPCKLPS — Unpack Low Single-Precision Floating-Point

Unpacks the low-order single-precision floating-point values of the first and second source operands and interleaves the values into the destination. Bits [127:64] of the source operands are ignored.

Values are interleaved in ascending order from the lsb of the sources and the destination. Bits [31:0] of the first source are written to bits [31:0] of the destination; bits [31:0] of the second source are written to bits [63:32] of the destination and so on, ending with bits [63:32] of the second source in bits [127:96] of the destination. For the 256-bit encoding, the process continues for bits [191:128] of the sources and bits [255:128] of the destination.

There are legacy and extended forms of the instruction:

UNPCKLPS

Interleaves two pairs of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VUNPCKLPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Interleaves two pairs of values. The first source operand is an XMM register and the second source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

Interleaves four pairs of values. The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

UNPCKLPS is an SSE instruction and VUNPCKLPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
UNPCKLPS *xmm1*, *xmm2/mem128*	0F 14 /r	Unpacks the high-order single-precision floating-point values in *xmm1* and *xmm2* or *mem128* and interleaves them into *xmm1*

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VUNPCKLPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.0.00	14 /r
VUNPCKLPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src}}$.1.00	14 /r

Related Instructions

(V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

VBROADCASTF128 Load With Broadcast From 128-bit Memory Location

Loads double-precision floating-point data from a 128-bit memory location and writes it to the two 128-bit elements of a YMM register

This extended-form instruction has 256-bit encoding.

The source operand is a128-bit memory location. The destination is a YMM register.

VBROADCASTF128 is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VBROADCASTF128 *ymm1*, *mem128*	C4	$\overline{\text{RXB}}$.00010	X.1111.1.01	1A /r

Related Instructions

VBROADCASTSD, VBROADCASTSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VBROADCASTSD Load With Broadcast From 64-Bit Memory Location

Loads a double-precision floating-point value from a 64-bit memory location and writes it to the four 64-bit elements of a YMM register

This extended-form instruction has 256-bit encoding.

The source operand is a 64-bit memory location. The destination is a YMM register.

VBROADCASTSD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VBROADCASTSD *ymm1*, *mem64*	C4	$\overline{\text{RXB}}$.00010	0.1111.1.01	19 /r

Related Instructions

VBROADCASTF128, VBROADCASTSS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VBROADCASTSS Load With Broadcast From 32-Bit Memory Location

Loads a single-precision floating-point value from a 32-bit memory location and writes it to 32-bit elements of an XMM or YMM register

This extended-form instruction has both 128-bit and 256-bit encoding.

XMM Encoding

Broadcasts to eight 32-bit elements.

The source operand is a 32-bit memory location. The destination is an XMM register.

YMM Encoding

Broadcasts to sixteen 32-bit elements.

The source operand is a 32-bit memory location. The destination is a YMM register.

VBROADCASTSS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VBROADCASTSS *xmm1*, *mem32*	C4	$\overline{\text{RXB}}$.00010	0.1111.0.01	18 /r
VBROADCASTSS *ymm1*, *mem32*	C4	$\overline{\text{RXB}}$.00010	0.1111.1.01	18 /r

Related Instructions

VBROADCASTF128, VBROADCASTSD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VEXTRACTF128 Extract Packed Values from 128-bit Memory Location

Extracts 128-bit packed-value data from a YMM register as specified by an immediate byte operand, and writes it to either an XMM register or a 128-bit memory location.

Only bit [0] of the immediate operand is used. Operation is as follows.

- When imm8[0] = 0, copy bits [127:0] of the source to the destination.
- When imm8[0] = 1, copy bits [255:128] of the source to the destination.

This extended-form instruction has 256-bit encoding.

The source operand is a YMM register and the destination is either an XMM register or a 128-bit memory location. There is a third immediate byte operand.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VEXTRACT128 *xmm/mem128, ymm, imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.1.01	19 /r ib

Related Instructions

VBROADCASTF128, VINSERTF128

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VFMADDPD Multiply and Add Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source by the corresponding value of the second source, adds each product to the corresponding value of the third source and writes the rounded results to the destination.

There are four operands: VFMADDPD *dest, src1, src2, src3* *dest = (src1* src2) + src3*

The 128-bit version multiplies each of two double-precision values in the first source XMM register by the corresponding double-precision value in the second source. It then adds each intermediate product to the corresponding double-precision value in the third source and places the result in the destination XMM register.

The 256-bit version multiplies each of four double-precision values in the first source YMM register by the corresponding double-precision value in the second source. It then adds each product to the corresponding double-precision value in the third source and places the results in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	69 /r /is4
VFMADDPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	69 /r /is4
VFMADDPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	69 /r /is4
VFMADDPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.1.01	69 /r /is4

Related Instructions

VFMADDPS, VFMADDSD, VFMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMADDPS Multiply and Add Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source by the corresponding value of the second source, adds each product to the corresponding value of the third source and writes the rounded results to the destination.

There are four operands: VFMADDPS *dest, src1, src2, src3* *dest = src1* src2 + src3*

The 128-bit version multiplies each of four single-precision values in the first source XMM register by the corresponding single-precision value in the second source. It then adds each product to the corresponding single-precision value in the third source and places the results in the destination XMM register.

The 256-bit version multiplies each of eight single-precision values in the first source YMM register by the corresponding double-precision value in the second source. It then adds each product to the corresponding double-precision value in the third source and places the results in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	68 /r /is4
VFMADDPS *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	68 /r /is4
VFMADDPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	68 /r /is4
VFMADDPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	68 /r /is4

Related Instructions

VFMADDPD, VFMADDSD, VFMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMADDSD Multiply and Add Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value of the low-order quadword of the first source by the corresponding value of the low-order quadword of the second source, adds the product to the corresponding value of the low-order quadword of the third source, and writes the result to the low-order quadword of the destination.

There are four operands: VFMADDSD *dest, src1, src2, src3* *dest = src1* src2 + src3*

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a 64-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a a register and the third source is a register or a 64-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are cleared.

The intermediate product is not rounded; the infinitely precise product is used in the addition. The result of the addition is rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMADDSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.X.01	6B /r /is4
VFMADDSD *xmm1*, *xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.X.01	6B /r /is4

Related Instructions

VFMADDPD, VFMADDPS, VFMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMADDSS Multiply and Add Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value of the low-order doubleword of the first source by the corresponding value of the second source, adds the product to the corresponding value of the third source, and writes the result to the low-order doubleword of the destination.

There are four operands: VFMADDSS *dest, src1, src2, src3* *dest = src1* src2 + src3*

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a 32-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a a register and the third source is either a register or a 32-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, bits [127:32] of the destination and bits [255:128] of the corresponding YMM register are cleared.

The intermediate product is not rounded; the infinitely precise product is used in the addition. The result of the addition is rounded, as specified byMXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMADDSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.X.01	6A /r /is4
VFMADDSS *xmm1*, *xmm2*, *xmm3, xmm4/mem32*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.X.01	6A /r /is4

Related Instructions

VFMADDPD, VFMADDPS, VFMADDSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMADDSUBPD Multiply with Alternating Add/Subtract Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source by the corresponding value of the second source. Adds each odd-numbered double-precision floating-point value of the third source to the corresponding infinite-precision intermediate product; subtracts each even-numbered double-precision floating-point value of the third source from the corresponding product. Writes the results to the destination.

The 128-bit version multiplies each of two double-precision floating-point values in the first source by the corresponding value in the second source. The low-order double-precision floating-point value in the third source is subtracted from the corresponding infinite-precision product and the high-order double-precision floating-point value in the third source is added to the corresponding product. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of four double-precision floating-point values in first source by the corresponding double-precision value in the second source. The even-numbered double-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the odd-numbered double-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is an XMM register or a YMM register, as determined by VEX.L.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When writing to an XMM destination, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the final addition and subtraction operation(s). The results of the addition and subtraction operations are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMADDSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	5D /r /is4
VFMADDSUBPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	5D /r /is4
VFMADDSUBPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	5D /r /is4
VFMADDSUBPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.1.01	5D /r /is4

Related Instructions

VFMADDSUBPD, VFMSUBADDPD, VFMSUBADDPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMADDSUBPS Multiply with Alternating Add/Subtract Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source by the corresponding value of the second source. Adds each odd-numbered single-precision floating-point value of the third source to the corresponding infinite-precision intermediate product; subtracts each even-numbered single-precision floating-point value of the third source from the corresponding product. Writes the results to the destination.

The 128-bit version multiplies each of four single-precision floating-point values in first source by the corresponding single-precision value in the second source. The even-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the odd-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of eight single-precision floating-point values in first source by the corresponding single-precision value in the second source. The even-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the odd-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is either an XMM register or a YMM register, as determined by VEX.L.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When writing to an XMM destination, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise intermediate products are used in the addition and subtraction operations. The results of the addition and subtraction operations are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	5C /r /is4
VFMADDSUBPS *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	5C /r /is4
VFMADDSUBPS *xmm1*, *xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	5C /r /is4
VFMADDSUBPS *ymm1*, *ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	5C /r /is4

Related Instructions

VFMADDSUBPD, VFMSUBADDPD, VFMSUBADDPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMSUBADDPD Multiply with Alternating Subtract/Add Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source by the corresponding value of the second source. Adds each even-numbered double-precision floating-point value of the third source to the corresponding infinite-precision intermediate product; subtracts each odd-numbered double-precision floating-point value of the third source from the corresponding product. Writes the results to the destination.

The 128-bit version multiplies each of two double-precision floating-point values in the first source by the corresponding value in the second source. The high-order double-precision floating-point value in the third source is subtracted from the corresponding infinite-precision product and the low-order double-precision floating-point value in the third source is added to the corresponding product. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of four double-precision floating-point values in first source by the corresponding double-precision value in the second source. The odd-numbered double-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the even-numbered double-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is either an XMM register or a YMM register, as determined by VEX.L.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source operand is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When writing to an XMM destination, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the two infinitely precise intermediate products are used in the addition. The results of the addition and subtraction operations are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMSUBADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	5F /r /is4
VFMSUBADDPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	5F /r /is4
VFMSUBADDPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	5F /r /is4
VFMSUBADDPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.1.01	5F /r /is4

Related Instructions

VFMADDSUBPD, VFMADDSUBPS, VFMSUBADDPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMSUBADDPS Multiply with Alternating Subtract/Add Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source by the corresponding value of the second source. Adds each even-numbered single-precision floating-point value of the third source to the corresponding infinite-precision intermediate product; subtracts each odd-numbered single-precision floating-point value of the third source from the corresponding product. Writes the results to the destination.

The 128-bit version multiplies each of four single-precision floating-point values in first source by the corresponding single-precision value in the second source. The odd-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the even-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of eight single-precision floating-point values in first source by the corresponding single-precision value in the second source. The odd-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the even-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is either an XMM register or a YMM register, as determined by VEX.L.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When writing to an XMM destination, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the additions and subtracts are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	5E /r /is4
VFMSUBADDPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	5E /r /is4
VFMSUBADDPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	5E /r /is4
VFMSUBADDPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.1.01	5E /r /is4

Related Instructions

VFMADDSUBPD, VFMADDSUBPS, VFMSUBADDPD, VFMSUBADDPS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.

F — FMA4 exception

VFMSUBPD Multiply and Subtract Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source by the corresponding value of the second source, subtracts the corresponding values of the third source from the intermediate products of the multiplication, and writes results to the destination.

There are four operands: VFMSUBPD *dest, src1, src2, src3* *dest = src1* src2 - src3*

The 128-bit version multiplies two packed double-precision floating-point values in the first source, by their corresponding packed double-precision floating point values in the second source, producing two intermediate products. The two double precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination XMM register.

The 256-bit version multiplies four packed double-precision floating-point values in the first source by their corresponding packed double-precision floating point values in the second source, producing four intermediate products. The four double-precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When writing to an XMM destination, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the two infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	6D /r /is4
VFMSUBPD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	6D /r /is4
VFMSUBPD *xmm1*, *xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	6D /r /is4
VFMSUBPD *ymm1*, *ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.1.01	6D /r /is4

Related Instructions

VFMSUBPS, VFMSUBSD, VFMSUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMSUBPS Multiply and Subtract Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source by the corresponding value of the second source, subtracts the corresponding values of the third source from the products, and writes four results to the destination.

There are four operands: VFMSUBPS *dest, src1, src2, src3* *dest = src1* src2 - src3*

The 128-bit version multiplies four packed single-precision floating-point values in the first source by their corresponding packed single-precision floating point values in the second source, producing four intermediate products. The four single-precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination XMM register.

The 256-bit version multiplies eight packed single-precision floating-point values in the first source by their corresponding packed single-precision floating point values in the second source, producing eight intermediate products. The eight single-precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When writing to an XMM destination, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the two infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	6C /r /is4
VFMSUBPS *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	6C /r /is4
VFMSUBPS *xmm1*, *xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	6C /r /is4
VFMSUBPS *ymm1*, *ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	6C /r /is4

Related Instructions

VFMSUBPD, VFMSUBSD, VFMSUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMSUBSD Multiply and Subtract Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value of the low-order quadword of the first source by the corresponding value of the low-order quadword of the second source, subtracts the corresponding value of the low-order quadword of the third source from the intermediate product, and writes results to the low-order quadword of the destination.

There are four operands: VFMSUBSD *dest, src1, src2, src3* *dest = src1* src2 - src3*

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or 64-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is a register or 64-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are cleared.

The intermediate product is not rounded; the infinitely precise product is used in the subtraction. The result of the subtraction is rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMSUBSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	6F /r /is4
VFMSUBSD *xmm1, xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	6F /r /is4

Related Instructions

VFMSUBPD, VFMSUBPS, VFMSUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFMSUBSS Multiply and Subtract Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value of the low-order doubleword of the first source by the corresponding value of the low-order doubleword of the second source, subtracts the corresponding value of the low-order doubleword of the third source from the product, and writes results to the low-order doubleword of the destination.

There are four operands: VFMSUBSS *dest, src1, src2, src3* *dest = src1* src2 - src3*

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or 32-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is a register or 32-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, bits [127:32] of the XMM register and bits [255:128] of the corresponding YMM register are cleared.

The intermediate product is not rounded; the infinitely precise product is used in the subtraction. The result of the subtraction is rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFMSUBSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	6E /r /is4
VFMSUBSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	6E /r /is4

Related Instructions

VFMSUBPD, VFMSUBPS, VFMSUBSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMADDPD Negative Multiply and Add Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source by the corresponding value of the second source, negates the products, adds them to the corresponding values of the third source, and writes results to the destination.

There are four operands: VFNMADDPD *dest, src1, src2, src3* *dest = – (src1* src2) + src3*

The 128-bit version multiplies the two double-precision values in the first source XMM register by the corresponding double-precision values in the second source, which can be either an XMM register or a 128-bit memory location. It then negates each product and adds it to the corresponding double-precision value in the third source. The results are then placed in the destination XMM register.

The 256-bit version multiplies the four double-precision values in the first source YMM register by the four double-precision values in the second source, which can be either a YMM register or a 256-bit memory location. It then negates each product and adds it to the corresponding double-precision value in the third source. The results are then placed in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFNMADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	79 /r /is4
VFNMADDPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	79 /r /is4
VFNMADDPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	79 /r /is4
VFNMADDPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	79 /r /is4

Related Instructions

VFNMADDPS, VFNMADDSD, VFNMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.

F — FMA4 exception

VFNMADDPS Negative Multiply and Add Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source by the corresponding value of the second source, negates the products, adds them to the corresponding values of the third source, and writes results to the destination.

There are four operands: VFNMADDPS *dest, src1, src2, src3* *dest = – (src1* src2) + src3*

The 128-bit version multiplies the four single-precision values in the first source XMM register by the corresponding single-precision values in the second source, which can be either an XMM register or a 128-bit memory location. It then negates each product and adds it to the corresponding single-precision value in the third source. The results are then placed in the destination XMM register.

The 256-bit version multiplies the eight single-precision values in the first source YMM register by the eight single-precision values in the second source, which can be either a YMM register or a 256-bit memory location. It then negates each product and adds it to the corresponding single-precision value in the third source. The result is then placed in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFNMADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	78 /r /is4
VFNMADDPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	78 /r /is4
VFNMADDPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	78 /r /is4
VFNMADDPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	78 /r /is4

Related Instructions

VFNMADDPD, VFNMADDSD, VFNMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMADDSD Negative Multiply and Add Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value of the low-order quadword of the first source by the corresponding value of the low-order quadword of the second source, negates the product, adds it to the corresponding value of the low-order quadword of the third source, and writes the result to the low-order quadword of the destination.

There are four operands: VFNMADDSD *dest, src1, src2, src3* *dest = – (src1* src2) + src)*

The first source is an XMM register specified by VEX.vvvv.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or 64-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is a register or 64-bit memory location. The destination is an XMM register. When the result is written to the destination, bits [127:64] of the XMM register and bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMADDSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.X.01	7B /r /is4
VFNMADDSD *xmm1, xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.X.01	7B /r /is4

Related Instructions

VFNMADDPD, VFNMADDPS, VFNMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMADDSS Negative Multiply and Add Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value of the low-order doubleword of the first source by the corresponding value of the low-order doubleword of the second source, negates the product, adds it to the corresponding value of the low-order doubleword of the third source, and writes results to the low-order doubleword of the destination.

There are four operands: VFNMADDSS *dest, src1, src2, src3* *dest = - (src1* src2) + src3*

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or 32-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is a register or 32-bit memory location.

The destination is an XMM register. When the result is written to the destination, bits [127:32] of the XMM register and bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMADDSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.X.01	7A /r /is4
VFNMADDSS *xmm1*, *xmm2*, *xmm3, xmm4/mem32*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.X.01	7A /r /is4

Related Instructions

VFNMADDPD, VFNMADDPS, VFNMADDSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMSUBPD Negative Multiply and Subtract Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value of the first source by the corresponding value of the second source, subtracts the corresponding value of the third source from the negated interim products, and writes results to the destination.

There are four operands: VFNMSUBPD *dest, src1, src2, src3* *dest = – (src1* src2) - src3*

The 128-bit version multiplies each of two double-precision values in the first source XMM register by the corresponding double-precision value in the second source, which can be either an XMM register or a 128-bit memory location. It then subtracts the corresponding double-precision value in the third source from the negated interim product. The results are then placed in the destination XMM register.

The 256-bit version multiplies each of four double-precision values in the first source YMM register by the corresponding double-precision value in the second source, which can be either a YMM register or a 256-bit memory location. It then subtracts the corresponding double-precision value in the third source from the negated interim product. The results are then placed in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFNMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	7D /r /is4
VFNMSUBPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	7D /r /is4
VFNMSUBPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	7D /r /is4
VFNMSUBPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	7D /r /is4

Related Instructions

VFNMSUBPS, VFNMSUBSD, VFNMSUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMSUBPS Negative Multiply and Subtract Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value of the first source by the corresponding value of the second source, subtracts the corresponding values of the third source from the negated products, and writes results to the destination.

There are four operands: VFNMSUBPS *dest, src1, src2, src3* *dest = – (src1* src2) – src3*

The 128-bit version multiplies each of four single-precision values in the first source XMM register by the corresponding single-precision value in the second source, which can be either an XMM register or a 128-bit memory location. It then subtracts the corresponding single-precision value in the third source from the negated interim product. The results are then placed in the destination XMM register.

The 256-bit version multiplies each of eight single-precision values in the first source YMM register by the corresponding single-precision value in the second source, which can be either a YMM register or a 256-bit memory location. It then subtracts the corresponding single-precision value in the third source from the negated interim product. The results are then placed in the destination YMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a memory location.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	7C /r /is4
VFNMSUBPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	7C /r /is4
VFNMSUBPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	7C /r /is4
VFNMSUBPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	7C /r /is4

Related Instructions

VFNMSUBPD, VFNMSUBSD, VFNMSUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMSUBSD Negative Multiply and Subtract Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value of the low-order quadword of the first source by the corresponding value of the low-order quadword of the second source, subtracts the corresponding value of the low-order quadword of the third source from the negated interim product, and writes results to the low-order quadword of the destination.

There are four operands: VFNMSUBSD *dest, src1, src2, src3* *dest = – (src1* src2) – src3*

The first source is an XMM register.

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a 64-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a 64-bit memory location.

The destination is an XMM register specified by VEX.vvvv. Bits [127:64] of the destination XMM register and bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFNMSUBSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.X.01	7F /r /is4
VFNMSUBSD *xmm1*, *xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.X.01	7F /r /is4

Related Instructions

VFNMSUBPD, VFNMSUBPS, VFNMSUBSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

VFNMSUBSS Negative Multiply and Subtract Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value of the low-order doubleword of the first source by the corresponding value of the low-order doubleword of the second source, subtracts the corresponding value of the low-order doubleword of the third source from the negated product, and writes results to the low-order doubleword of the destination.

There are four operands: VFNMSUBSS *dest, src1, src2, src3* *dest = - (src1* src2) - src3*

VEX.W determines operand configuration.

- When VEX.W = 0, the second source is either a register or a 32-bit memory location and the third source is a register.
- When VEX.W = 1, the second source is a register and the third source is either a register or a 32-bit memory location.

The destination is an XMM register specified by VEX.vvvv. Bits[127:32] of the destination XMM register and bits [255:128] of the corresponding YMM register are cleared.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by MXCSR.RC.

This is an FMA4 instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[FMA4] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFNMSUBSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.X.01	7E /r /is4
VFNMSUBSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.X.01	7E /r /is4

Related Instructions

VFNMSUBPD, VFNMSUBPS, VFNMSUBSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M	M		M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.

F — FMA4 exception

VFRCZPD Extract Fraction Packed Double-Precision Floating-Point

Extracts the fractional portion of each double-precision floating-point value of either a source register or a memory location and writes the resulting values to the corresponding elements of the destination. The fractional results are precise.

- When XOP.L = 0, the source is either an XMM register or a 128-bit memory location.
- When XOP.L = 1, the source is a YMM register or 256-bit memory location.

When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

Exception conditions are the same as for other arithmetic instructions, except with respect to the sign of a zero result. A zero is returned in the following cases:

- When the operand is a zero.
- When the operand is a normal integer.
- When the operand is a denormal value and is coerced to zero by MXCSR.DAZ.
- When the operand is a denormal value that is not coerced to zero by MXCSR.DAZ.

In the first three cases, when MXCSR.RC = 01b (round toward $-\infty$) the sign of the zero result is negative, and is otherwise positive.

In the fourth case, the operand is its own fractional part, which results in underflow, and the result is forced to zero by MXCSR.FZ; the result has the same sign as the operand.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VFRCZPD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	81 /r
VFRCZPD *ymm1*, *ymm2*/*mem256*	8F	$\overline{\text{RXB}}$.01001	0.1111.1.00	81 /r

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS, VFRCZPS, VFRCZSS, VFRCZSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M			M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			X	A source operand was an SNaN value.
			X	Undefined operation.
Denormalized operand, DE			X	A source operand was a denormal value.
Underflow, UE			X	Rounded result too small to fit into the format of the destination operand.
Precision, PE			X	A result could not be represented exactly in the destination format.
X — XOP exception				

VFRCZPS Extract Fraction Packed Single-Precision Floating-Point

Extracts the fractional portion of each single-precision floating-point value of either a source register or a memory location and writes the resulting values to the corresponding elements of the destination. The fractional results are exact.

- When XOP.L = 0, the source is either an XMM register or a 128-bit memory location.
- When XOP.L = 1, the source is a YMM register or 256-bit memory location.

When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

Exception conditions are the same as for other arithmetic instructions, except with respect to the sign of a zero result. A zero is returned in the following cases:

- When the operand is a zero.
- When the operand is a normal integer.
- When the operand is a denormal value and is coerced to zero by MXCSR.DAZ.
- When the operand is a denormal value that is not coerced to zero by MXCSR.DAZ.

In the first three cases, when MXCSR.RC = 01b (round toward $-\infty$) the sign of the zero result is negative, and is otherwise positive.

In the fourth case, the operand is its own fractional part, which results in underflow, and the result is forced to zero by MXCSR.FZ; the result has the same sign as the operand.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFRCZPS *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	80 /r
VFRCZPS *ymm1*, *ymm2*/*mem256*	8F	$\overline{\text{RXB}}$.01001	0.1111.1.00	80 /r

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS, VFRCZPD, VFRCZSS, VFRCZSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M			M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*																

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			X	A source operand was an SNaN value.
			X	Undefined operation.
Denormalized operand, DE			X	A source operand was a denormal value.
Underflow, UE			X	Rounded result too small to fit into the format of the destination operand.
Precision, PE			X	A result could not be represented exactly in the destination format.
X — XOP exception				

VFRCZSD Extract Fraction Scalar Double-Precision Floating-Point

Extracts the fractional portion of the double-precision floating-point value of either the low-order quadword of an XMM register or a 64-bit memory location and writes the result to the low-order quadword of the destination XMM register. The fractional results are precise.

When the result is written to the destination XMM register, bits [127:64] of the destination and bits [255:128] of the corresponding YMM register are cleared.

Exception conditions are the same as for other arithmetic instructions, except with respect to the sign of a zero result. A zero is returned in the following cases:

- When the operand is a zero.
- When the operand is a normal integer.
- When the operand is a denormal value and is coerced to zero by MXCSR.DAZ.
- When the operand is a denormal value that is not coerced to zero by MXCSR.DAZ.

In the first three cases, when MXCSR.RC = 01b (round toward $-\infty$) the sign of the zero result is negative, and is otherwise positive.

In the fourth case, the operand is its own fractional part, which results in underflow, and the result is forced to zero by MXCSR.FZ; the result has the same sign as the operand.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFRCZSD *xmm1*, *xmm2/mem64*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	83 /r

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS, VFRCZPS, VFRCZPD, VFRCZSS

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M			M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			X	A source operand was an SNaN value.
			X	Undefined operation.
Denormalized operand, DE			X	A source operand was a denormal value.
Underflow, UE			X	Rounded result too small to fit into the format of the destination operand.
Precision, PE			X	A result could not be represented exactly in the destination format.
X — XOP exception				

VFRCZSS Extract Fraction Scalar Single-Precision Floating Point

Extracts the fractional portion of the single-precision floating-point value of the low-order doubleword of an XMM register or 32-bit memory location and writes the result in the low-order doubleword of the destination XMM register. The fractional results are precise.

When the result is written to the destination XMM register, bits [127:32] of the destination and bits [255:128] of the corresponding YMM register are cleared.

Exception conditions are the same as for other arithmetic instructions, except with respect to the sign of a zero result. A zero is returned in the following cases:

- When the operand is a zero.
- When the operand is a normal integer.
- When the operand is a denormal value and is coerced to zero by MXCSR.DAZ.
- When the operand is a denormal value that is not coerced to zero by MXCSR.DAZ.

In the first three cases, when MXCSR.RC = 01b (round toward $-\infty$) the sign of the zero result is negative, and is otherwise positive.

In the fourth case, the operand is its own fractional part, which results in underflow, and the result is forced to zero by MXCSR.FZ; the result has the same sign as the operand.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFRCZSS *xmm1*, *xmm2/mem32*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	82 /r

Related Instructions

ROUNDPD, ROUNDPS, ROUNDSD, ROUNDSS, VFRCZPS, VFRCZPD, VFRCZSD

MXCSR Flags Affected

MM	FZ	RC		PM	UM	OM	ZM	DM	IM	DAZ	PE	UE	OE	ZE	DE	IE
											M	M			M	M
17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: *A flag that may be set or cleared is M (modified). Unaffected flags are blank.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			X	A source operand was an SNaN value.
			X	Undefined operation.
Denormalized operand, DE			X	A source operand was a denormal value.
Underflow, UE			X	Rounded result too small to fit into the format of the destination operand.
Precision, PE			X	A result could not be represented exactly in the destination format.
X — XOP exception				

VINSERTF128 Insert Packed Values 128-bit

Combines 128 bits of data from a YMM register with 128-bit packed-value data from an XMM register or a 128-bit memory location, as specified by an immediate byte operand, and writes the combined data to the destination.

Only bit [0] of the immediate operand is used. Operation is a follows.

- When imm8[0] = 0, copy bits [255:128] of the first source to bits [255:128] of the destination and copy bits [127:0] of the second source to bits [127:0] of the destination.
- When imm8[0] = 1, copy bits [127:0] of the first source to bits [127:0] of the destination and copy bits [127:0] of the second source to bits [255:128] of the destination.

This extended-form instruction has 256-bit encoding.

The first source operand is a YMM register. The second source operand is either an XMM register or a 128-bit memory location.The destination is a YMM register. There is a third immediate byte operand.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VINSERTF128 *ymm1, ymm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	18 /r ib

Related Instructions

VBROADCASTF128, VEXTRACTF128

Exceptions

Exception	Mode			Cause of Exception
	Real	**Virt**	**Prot**	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VMASKMOVPD Masked Move Packed Double-Precision

Moves packed double-precision data elements from a source element to a destination element, as specified by mask bits in a source operand. There are load and store versions of the instruction.

For loads, the data elements are in a source memory location; for stores the data elements are in a source register. The mask bits are the msb of the corresponding data element of a source register.

- For loads, when a mask bit = 1, the corresponding data element is copied from the source to the same element of the destination; when a mask bit = 0, the corresponding element of the destination is cleared.
- For stores, when a mask bit = 1, the corresponding data element is copied from the source to the same element of the destination; when a mask bit = 0, the corresponding element of the destination is not affected.

XMM Encoding

There are load and store encodings.

- For loads, there are two 64-bit source data elements in a 128-bit memory location, the mask operand is an XMM register, and the destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- For stores, there are two 64-bit source data elements in an XMM register, the mask operand is another XMM register, and the destination is a 128-bit memory location.

YMM Encoding

There are load and store encodings.

- For loads, there are four 64-bit source data elements in a 256-bit memory location, the mask operand is a YMM register, and the destination is a YMM register.
- For stores, there are four 64-bit source data elements in a YMM register, the mask operand is another YMM register, and the destination is a 128-bit memory location.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
Loads:				
VMASKMOVPD *xmm1*, *xmm2*, *mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.0.01	2D /r
VMASKMOVPD *ymm1*, *ymm2*, *mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.1.01	2D /r
Stores:				
VMASKMOVPD *mem128, xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.0.01	2F /r
VMASKMOVPD *mem256, ymm1*, *ymm2*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.1.01	2F /r

Related Instructions

VMASKMOVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
	S	S	X	Write to a read-only data segment.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VMASKMOVPS Masked Move Packed Single-Precision

Moves packed single-precision data elements from a source element to a destination element, as specified by mask bits in a source operand. There are load and store versions of the instruction.

For loads, the data elements are in a source memory location; for stores the data elements are in a source register. The mask bits are the msb of the corresponding data element of a source register.

- For loads, when a mask bit = 1, the corresponding data element is copied from the source to the same element of the destination; when a mask bit = 0, the corresponding element of the destination is cleared.
- For stores, when a mask bit = 1, the corresponding data element is copied from the source to the same element of the destination; when a mask bit = 0, the corresponding element of the destination is not affected.

XMM Encoding

There are load and store encodings.

- For loads, there are four 32-bit source data elements in a 128-bit memory location, the mask operand is an XMM register, and the destination is an XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- For stores, there are four 32-bit source data elements in an XMM register, the mask operand is another XMM register, and the destination is a 128-bit memory location.

YMM Encoding

There are load and store encodings.

- For loads, there are eight 32-bit source data elements in a 256-bit memory location, the mask operand is a YMM register, and the destination is a YMM register.
- For stores, there are eight 32-bit source data elements in a YMM register, the mask operand is another YMM register, and the destination is a 128-bit memory location.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
Loads:				
VMASKMOVPS *xmm1*, *xmm2*, *mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.0.01	2C /r
VMASKMOVPS *ymm1*, *ymm2*, *mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.1.01	2C /r
Stores:				
VMASKMOVPS *mem128, xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.0.01	2E /r
VMASKMOVPS *mem256, ymm1*, *ymm2*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.1.01	2E /r

Related Instructions

VMASKMOVPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
	S	S	X	Write to a read-only data segment.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VPCMOV — Vector Conditional Move

Moves bits of either the first source or the second source to the corresponding positions in the destination, depending on the value of the corresponding bit of a third source.

When a bit of the third source = 1, the corresponding bit of the first source is moved to the destination; when a bit of the third source = 0, the corresponding bit of the second source is moved to the destination.

This instruction directly implements the C-language ternary "?" operation on each source bit.

Arbitrary bit-granular predicates can be constructed by any number of methods, or loaded as constants from memory. This instruction may use the results of any SSE instructions as the predicate in the selector. VPCMPEQB (VPCMPGTB), VPCMPEQW (VPCMPGTW), VPCMPEQD (VPCMPGTD) and VPCMPEQQ (VPCMPGTQ) compare bytes, words, doublewords, quadwords and integers, respectively, and set the predicate in the destination to masks of 1s and 0s accordingly. VCMPPS (VCMPSS) and VCMPPD (VCMPSD) compare word and doubleword floating-point source values, respectively, and provide the predicate for the floating-point instructions.

There are four operands: VPCMOV *dest, src1, src2, src3.*

The first source (*src1*) is an XMM or YMM register specified by XOP.vvvv.

XOP.W and bits [7:4] of an immediate byte (*imm8*) configure *src2* and *src3*:

- When XOP.W = 0, *src2* is either a register or a memory location specified by MODRM.rm and *src3* is a register specified by *imm8[7:4]*.
- When XOP.W = 1, s*rc2* is a register specified by *imm8[7:4]* and *src3* is either a register or a memory location specified by MODRM.rm.

The destination (*dest*) is either an XMM or a YMM register, as determined by XOP.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCMOV *xmm1*, *xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	A2 /r ib
VPCMOV *ymm1*, *ymm2, ymm3/mem256, ymm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.1.00	A2 /r ib
VPCMOV *xmm1*, *xmm2, xmm3, xmm4/mem128*	8F	$\overline{\text{RXB}}$.01000	1.$\overline{src}$.0.00	A2 /r ib
VPCMOV *ymm1*, *ymm2, ymm3*, *ymm4/mem256*	8F	$\overline{\text{RXB}}$.01000	1.$\overline{src}$.1.00	A2 /r ib

Related Instructions

VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMUW, VCMPPD, VCMPPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMB Compare Vector Signed Bytes

Compares corresponding packed signed bytes in the first and second sources and writes the result of each comparison in the corresponding byte of the destination. The result of each comparison is an 8-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMB *dest, src1, src2, imm8*

The destination (*dest*) is an XMM registers specified by MODRM.reg. When the comparison results are written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of the immediate-byte operand (*imm8*). Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	**Comparison**	**Mnemonic**
000	Less Than	VPCOMLTB
001	Less Than or Equal	VPCOMLEB
010	Greater Than	VPCOMGTB
011	Greater Than or Equal	VPCOMGEB
100	Equal	VPCOMEQB
101	Not Equal	VPCOMNEQB
110	False	VPCOMFALSEB
111	True	VPCOMTRUEB

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCOMB *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	CC /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMW, VPCOMD, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMD Compare Vector Signed Doublewords

Compares corresponding packed signed doublewords in the first and second sources and writes the result of each comparison to the corresponding doubleword of the destination. The result of each comparison is a 32-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMD *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the results of the comparisons are written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8*). Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	**Comparison**	**Mnemonic**
000	Less Than	VPCOMLTD
001	Less Than or Equal	VPCOMLED
010	Greater Than	VPCOMGTD
011	Greater Than or Equal	VPCOMGED
100	Equal	VPCOMEQD
101	Not Equal	VPCOMNEQD
110	False	VPCOMFALSED
111	True	VPCOMTRUED

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCOMD xmm1, xmm2, xmm3/mem128, imm8	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	CE /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMQ Compare Vector Signed Quadwords

Compares corresponding packed signed quadwords in the first and second sources and writes the result of each comparison to the corresponding quadword of the destination. The result of each comparison is a 64-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMQ *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the result is written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8)*. Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	Comparison	Mnemonic
000	Less Than	VPCOMLTQ
001	Less Than or Equal	VPCOMLEQ
010	Greater Than	VPCOMGTQ
011	Greater Than or Equal	VPCOMGEQ
100	Equal	VPCOMEQQ
101	Not Equal	VPCOMNEQQ
110	False	VPCOMFALSEQ
111	True	VPCOMTRUEQ

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMQ *xmm1*, *xmm2, xmm3*/*mem128, imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	CF /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMUB Compare Vector Unsigned Bytes

Compares corresponding packed unsigned bytes in the first and second sources and writes the result of each comparison to the corresponding byte of the destination. The result of each comparison is an 8-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMUB *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the result is written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8)*. Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	**Comparison**	**Mnemonic**
000	Less Than	VPCOMLTUB
001	Less Than or Equal	VPCOMLEUB
010	Greater Than	VPCOMGTUB
011	Greater Than or Equal	VPCOMGEUB
100	Equal	VPCOMEQUB
101	Not Equal	VPCOMNEQUB
110	False	VPCOMFALSEUB
111	True	VPCOMTRUEUB

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCOMUB *xmm1*, *xmm2*, *xmm3*/*mem128*, *imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	6C /r ib

Related Instructions

VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMUD Compare Vector Unsigned Doublewords

Compares corresponding packed unsigned doublewords in the first and second sources and writes the result of each comparison to the corresponding doubleword of the destination. The result of each comparison is a 32-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMUD *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the results are written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8)*. Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	Comparison	Mnemonic
000	Less Than	VPCOMLTUD
001	Less Than or Equal	VPCOMLEUD
010	Greater Than	VPCOMGTUD
011	Greater Than or Equal	VPCOMGEUD
100	Equal	VPCOMEQUD
101	Not Equal	VPCOMNEQUD
110	False	VPCOMFALSEUD
111	True	VPCOMTRUEUD

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMUD *xmm1*, *xmm2*, *xmm3*/*mem128*, *imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	6E /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMUQ Compare Vector Unsigned Quadwords

Compares corresponding packed unsigned quadwords in the first and second sources and writes the result of each comparison to the corresponding quadword of the destination. The result of each comparison is a 64-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMUQ *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the results are written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8)*. Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	**Comparison**	**Mnemonic**
000	Less Than	VPCOMLTUQ
001	Less Than or Equal	VPCOMLEUQ
010	Greater Than	VPCOMGTUQ
011	Greater Than or Equal	VPCOMGEUQ
100	Equal	VPCOMEQUQ
101	Not Equal	VPCOMNEQUQ
110	False	VPCOMFALSEUQ
111	True	VPCOMTRUEUQ

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCOMUQ *xmm1*, *xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	6F /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMUW Compare Vector Unsigned Words

Compares corresponding packed unsigned words in the first and second sources and writes the result of each comparison to the corresponding word of the destination. The result of each comparison is a 16-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMUW *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the results are written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (src1) is an XMM register specified by the XOP.vvvv field and the second source (src2) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8)*. Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	Comparison	Mnemonic
000	Less Than	VPCOMLTUW
001	Less Than or Equal	VPCOMLEUW
010	Greater Than	VPCOMGTUW
011	Greater Than or Equal	VPCOMGEUW
100	Equal	VPCOMEQUW
101	Not Equal	VPCOMNEQUW
110	False	VPCOMFALSEUW
111	True	VPCOMTRUEUW

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMUW *xmm1*, *xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	6D /r ib

Related Instructions

VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPCOMW Compare Vector Signed Words

Compares corresponding packed signed words in the first and second sources and writes the result of each comparison in the corresponding word of the destination. The result of each comparison is a 16-bit value of all 1s (TRUE) or all 0s (FALSE).

There are four operands: VPCOMW *dest, src1, src2, imm8*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the results are written to the destination XMM register, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

The comparison type is specified by bits [2:0] of an immediate-byte operand (*imm8)*. Each type has an alias mnemonic to facilitate coding.

imm8[2:0]	**Comparison**	**Mnemonic**
000	Less Than	VPCOMLTW
001	Less Than or Equal	VPCOMLEW
010	Greater Than	VPCOMGTW
011	Greater Than or Equal	VPCOMGEW
100	Equal	VPCOMEQW
101	Not Equal	VPCOMNEQW
110	False	VPCOMFALSEW
111	True	VPCOMTRUEW

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPCOMW *xmm1*, *xmm2, xmm3*/*mem128, imm8*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	CD /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMD, VPCOMQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPERM2F128 Permute Floating-Point 128-bit

Copies 128-bit floating-point data elements from two 256-bit sources to two 128-bit elements of a 256-bit destination, as specified by an immediate byte operand.

The immediate operand is encoded as follows.

Destination	Immediate-Byte Bit Field	Value of Bit Field	Source 1 Bits Copied	Source 2 Bits Copied
[127:0]	[1:0]	00	[127:0]	—
		01	[255:128]	—
		10	—	[127:0]
		11	—	[255:128]
Setting *imm8* [3] clears bits [127:0] of the destination; i*mm8* [2] is ignored.				
[255:128]	[5:4]	00	[127:0]	—
		01	[255:128]	—
		10	—	[127:0]
		11	—	[255:128]
Setting *imm8* [7] clears bits [255:128] of the destination; i*mm8* [6] is ignored.				

This is a 256-bit extended-form instruction:

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPERM2F128 *ymm1*, *xmm2*, *xmm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	06 /r

Related Instructions

VEXTRACTF128, VINSERTF128, VPERMILPD, VPERMILPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VPERMIL2PD Permute Two-Source Double-Precision Floating-Point

Copies a selected quadword value from one of two source registers to a selected element of the destination or clears the selected element of the destination. Values in a third source operand and an immediate byte operand control operation.

There are XMM and YMM versions of this instruction. Both versions have four operands:

VPERMIL2PD *dest, src1, src2, src3*

The 128-bit version of the instruction concatenates the first source XMM register and the second source, which can be either an XMM register or a 128-bit memory location, to form a single operand partition consisting of four 64-bit double-precision floating point values.

The 256-bit version of the instruction concatenates the first source YMM register and the second source, which can be either a YMM register or a 256-bit memory location, to form two operand partitions, each containing four 64-bit double-precision floating point values.

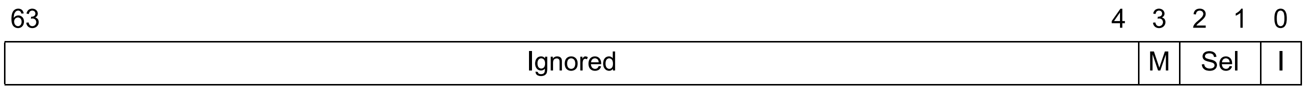
XOP.W and bits [7:4] of an immediate byte (*imm8*) configure *src2* and *src3*:

- When XOP.W = 0, *src2* is either a register or a memory location specified by MODRM.rm and *src3* is a register specified by *imm8[7:4]*.
- When XOP.W = 1, s*rc2* is a register specified by *imm8[7:4]* and *src3* is either a register or a memory location specified by MODRM.rm.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The third source operand can be either a register or a memory location. The operand of the 128-bit version of the instruction is divided into two quadword selector elements; the operand of the 256-bit version is divided into four quadword selector elements.

The bit field layout of each selector is as follows.



Bits	Mnemonic	Description
[63:4]	Ignored	—
[3]	M	Match
[2:1]	Sel	Select
[0]	Ignored	—

Bit [0] and bits [63:4] of the quadword selector element are ignored.

Each selector element has the following fields.

- Select — Selects the source to copy into the corresponding quadword element of the destination:

Value	Source Selected for Quadwords 0 and 1	Source Selected for Quadwords 2 and 3
00b	*src1*[63:0]	*src1*[191:128]
01b	*src1*[127:64]	*src1*[255:192]
10b	*src2*[63:0]	*src2*[191:128]
11b	*src2*[127:64]	*src2*[255:192]

- Match — This bit and the immediate byte Match field select destination quadwords to zero.

The immediate byte operand is defined as follows.

7 … 4	3 … 2	1 … 0
SRS	Ignore	M2Z

Bits	Mnemonic	Description
[7:4]	SRS	Source Register Select
[3:2]	Ignore	—
[1:0]	M2Z	Match to Zero

Bits [3:2] of the byte are ignored.

Byte fields are:

- Source Register Select — In 64-bit mode, when XOP.W = 0, the field identifies the third source register; when XOP.W = 1, it identifies the second source register. In non-64-bit mode, only bits [6:4] identify the register, and imm8[7] is ignored.
- Match to Zero — This field interacts with the Match bit of the selector element to determine the double-precision floating-point value that is written to the corresponding doubleword element of the destination operand.

The selector element Match bit and the immediate operand M2Z field operate as follows.

Immediate M2Z Field	Selector Match Bit	Value Loaded into Destination Quadword
0Xb	X	Operand value selected by selector element Sel field.
10b	0	Operand value selected by selector element Sel field.
10b	1	Zero
11b	0	Zero
11b	1	Operand value selected by selector element Sel field.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPERMIL2PD *xmm1*, *xmm2*, *xmm3/mem128*, *xmm4*	8F	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.0.01	49 /r ib
VPERMIL2PD *xmm1*, *xmm2*, *xmm3*, *xmm4/mem128*	8F	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.0.01	49 /r ib
VPERMIL2PD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	8F	$\overline{\text{RXB}}$.00011	0.$\overline{\textit{src}}$.1.01	49 /r ib
VPERMIL2PD *ymm1*, *ymm2*, *ymm3*, *ymm4/mem256*	8F	$\overline{\text{RXB}}$.00011	1.$\overline{\textit{src}}$.1.01	49 /r ib

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPD, VPERMILPS, VPPERM

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPERMIL2PS Permute Two-Source Single-Precision Floating-Point

Copies a selected doubleword value from one of two source registers to a selected element of the destination or clears the selected element of the destination. Values in a third source operand and an immediate byte operand control operation.

There are XMM and YMM versions of this instruction. Both versions have four operands:

VPERMIL2PS *dest, src1, src2, src3*

The 128-bit version of the instruction concatenates the first source XMM register and the second source, which can be either an XMM register or a 128-bit memory location, to form a single operand partition consisting of four 64-bit double-precision floating point values.

The 256-bit version of the instruction concatenates the first source YMM register and the second source, which can be either a YMM register or a 256-bit memory location, to form two operand partitions, each containing four 64-bit double-precision floating point values.

XOP.W and bits [7:4] of an immediate byte (*imm8*) configure *src2* and *src3*:

- When XOP.W = 0, *src2* is either a register or a memory location specified by MODRM.rm and *src3* is a register specified by *imm8[7:4]*.
- When XOP.W = 1, s*rc2* is a register specified by *imm8[7:4]* and *src3* is either a register or a memory location specified by MODRM.rm.

The destination is either an XMM register or a YMM register, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

The third source operand can be either a register or a memory location. The operand of the 128-bit version of the instruction is divided into two quadword selector elements; the operand of the 256-bit version is divided into four quadword selector elements.

The bit field layout of each selector is as follows.

31 4	3	2 1 0
Ignored	M	Sel

Bits	Mnemonic	Description
[31:4]	Ignored	—
[3]	M	Match
[2:0]	Sel	Select

Bits [31:4] of the doubleword selector element are ignored.

Selector elements have the following fields.

- Select — Selects the source to copy into the corresponding quadword element of the destination:

Selector	Source Selected for Doublewords 0, 1, 2 and 3	Source Selected for Doublewords 4, 5, 6 and 7
000b	*src1*[31:0]	*src1*[159:128]
001b	*src1*[63:32]	*src1*[191:160]
010b	*src1*[95:64]	*src1*[223:192]
011b	*src1*[127:96]	*src1*[255:224]
100b	*src2*[31:0]	*src2*[159:128]
101b	*src2*[63:32]	*src2*[191:160]
110b	*src2*[95:64]	*src2*[223:192]
111b	*src2*[127:96]	*src2*[255:224]

- Match — This bit and the immediate byte Match field select destination quadwords to zero.

Immediate Operand

The immediate byte layout is as follows.

7 4	3 2	1 0
SRS	Ignore	M2Z

Bits	Mnemonic	Description
[7:4]	SRS	Source Register Select
[3:2]	Ignore	—
[1:0]	M2Z	Match to Zero

Bits [3:2] of the byte are ignored.

Byte fields are:

- Source Register Select — In 64-bit mode, when XOP.W = 0, the field identifies the third source register; when XOP.W = 1, it identifies the second source register. In non-64-bit mode, only bits [6:4] identify the register, and imm8[7] is ignored.
- Match to Zero — This field interacts with the Match bit of the selector element to determine the double-precision floating-point value that is written to the corresponding doubleword element of the destination operand.

The selector element Match bit and the immediate operand M2Z field operate as follows.

Immediate M2Z Field	Selector Match Bit	Value Loaded into Destination Quadword
0Xb	X	Operand value selected by selector element Sel field.
10b	0	Operand value selected by selector element Sel field.
10b	1	Zero
11b	0	Zero
11b	1	Operand value selected by selector element Sel field.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPERMIL2PS *xmm1*, *xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.0.01	48 /r ib
VPERMIL2PS *xmm1*, *xmm2, xmm3, xmm4/mem128*	8F	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.0.01	48 /r ib
VPERMIL2PS *ymm1*, *ymm2, ymm3/mem256, ymm4*	8F	$\overline{\text{RXB}}$.00011	0.$\overline{src}$.1.01	48 /r ib
VPERMIL2PS *ymm1*, *ymm2, ymm3, ymm4/mem256*	8F	$\overline{\text{RXB}}$.00011	1.$\overline{src}$.1.01	48 /r ib

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMILPD, VPERMILPS, VPPERM

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPERMILPD Permute Double-Precision

Copies double-precision floating-point values from a source to a destination. Source and destination can be selected in two ways. There are different encodings for each selection method.

Selection by bits in a source register or memory location:

Each quadword of the operand is defined as follows.

63 … 2	1	0
	Sel	

A bit selects source and destination. Only bit [1] is used; bits [63:2} and bit [0] are ignored. Setting the bit selects the corresponding quadword element of the source and the destination.

Selection by bits in an immediate byte:

Each bit corresponds to a destination quadword. Only bits [3:2] and bits [1:0] are used; bits [7:4] are ignored. Selections are defined as follows.

Destination Quadword	Immediate-Byte Bit Field	Value of Bit Field	Source 1 Bits Copied
Used by 128-bit encoding and 256-bit encoding			
[63:0]	[0]	0	[63:0]
		1	[127:64]
[127:64]	[1]	0	[63:0]
		1	[127:64]
Used only by 256-bit encoding			
[191:128]	[2]	0	[191:128]
		1	[255:192]
[255:192]	[3]	0	[191:128]
		1	[255:192]

This extended-form instruction has both 128-bit and 256-bit encoding.

XMM Encoding

There are two encodings, one for each selection method:

- The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- The first source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. There is a third, immediate byte operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

There are two encodings, one for each selection method:

- The first source operand is a YMM register. The second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.
- The first source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register. There is a third, immediate byte operand.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
Selection by source register or memory:				
VPERMILPD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.0.01	0D /r
VPERMILPD *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src}$.1.01	0D /r
Selection by immediate byte operand:				
VPERMILPD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.1.01	05 /r ib
VPERMILPD *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.1.01	05 /r ib

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPS, VPPERM

Exceptions

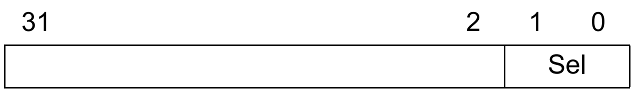
Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b (for versions with immediate byte operand only).
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VPERMILPS Permute Single-Precision

Copies single-precision floating-point values from a source to a destination. Source and destination can be selected in two ways. There are different encodings for each selection method.

Selection by bit fields in a source register or memory location:

Each doubleword of the operand is defined as follows.



Each bit field corresponds to a destination doubleword. Bit values select a source doubleword. Only bits [1:0] of each word are used; bits [31:2} are ignored. The 128-bit encoding uses four two-bit fields; the 256-bit version uses eight two-bit fields. Field encoding is as follows.

Destination Doubleword	Immediate Operand Bit Field	Value of Bit Field	Source Bits Copied
[31:0]	[1:0]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[63:32]	[33:32]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[95:64]	[65:64]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[127:96]	[97:96]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]

Destination Doubleword	Immediate Operand Bit Field	Value of Bit Field	Source Bits Copied
Upper 128 bits of 256-bit source and destination used by 256-bit encoding			
[159:128]	[129:128]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]
[191:160]	[161:160]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]
[223:192]	[193:192]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]
[255:224]	[225:224]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]

Selection by bit fields in an immediate byte:

Each bit field corresponds to a destination doubleword. For the 256-bit encoding, the fields specify sources and destinations in both the upper and lower 128 bits of the register. Selections are defined as follows.

Destination Doubleword	Bit Field	Value of Bit Field	Source Bits Copied
[31:0]	[1:0]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[63:32]	[3:2]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[95:64]	[5:4]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]
[127:96]	[7:6]	00	[31:0]
		01	[63:32]
		10	[95:64]
		11	[127:96]

Destination Doubleword	Bit Field	Value of Bit Field	Source Bits Copied
Upper 128 bits of 256-bit source and destination used by 256-bit encoding			
[159:128]	[1:0]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]
[191:160]	[3:2]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]
[223:192]	[5:4]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]
[255:224]	[7:6]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[255:224]

This extended-form instruction has both 128-bit and 256-bit encoding.

XMM Encoding

There are two encodings, one for each selection method:

- The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.
- The first source operand is either an XMM register or a 128-bit memory location. The destination is an XMM register. There is a third, immediate byte operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

There are two encodings, one for each selection method:

- The first source operand is a YMM register. The second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.
- The first source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register. There is a third, immediate byte operand.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
Selection by source register or memory:				
VPERMILPS *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{\textit{src}}$.0.01	0C /r
VPERMILPS *ymm1*, *ymm2*/*mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{\textit{src}}$.1.01	0C /r
Selection by immediate byte operand:				
VPERMILPS *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.1.01	04 /r ib
VPERMILPS *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.00011	0.1111.1.01	04 /r ib

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPD, VPPERM

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b (for versions with immediate byte operand only).
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

VPHADDBD Packed Horizontal Add Signed Byte to Signed Doubleword

Adds four sets of four 8-bit signed integer values of the source and packs the sign-extended sums into the corresponding doubleword of the destination.

There are two operands: VPHADDBD *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDBD *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	C2 /r

Related Instructions

VPHADDBW, VPHADDBQ, VPHADDWD, VPHADDWQ, VPHADDDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDBQ — Packed Horizontal Add Signed Byte to Signed Quadword

Adds two sets of eight 8-bit signed integer values of the source and packs the sign-extended sums into the corresponding quadword of the destination.

There are two operands: VPHADDBQ *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDBQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	C3 /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDWD, VPHADDWQ, VPHADDDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDBW Packed Horizontal Add Signed Byte to Signed Word

Adds each adjacent pair of 8-bit signed integer values of the source and packs the sign-extended 16-bit integer result of each addition into the corresponding word element of the destination.

There are two operands: VPHADDBW *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDBW *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	C1 /r

Related Instructions

VPHADDBD, VPHADDBQ, VPHADDWD, VPHADDWQ, VPHADDDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDDQ Packed Horizontal Add Signed Doubleword to Signed Quadword

Adds each adjacent pair of signed doubleword integer values of the source and packs the sign-extended sums into the corresponding quadword of the destination.

There are two operands: VPHADDDQ *dest*, *src*

The source is either an XMM register or a 128-bit memory location and the destination is an XMM register. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDDQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	CB /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDBQ, VPHADDWD, VPHADDWQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDUBD — Packed Horizontal Add Unsigned Byte to Doubleword

Adds four sets of four 8-bit unsigned integer values of the source and packs the sums into the corresponding doublewords of the destination.

There are two operands: VPHADDUBD *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDUBD *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	D2 /r

Related Instructions

VPHADDUBW, VPHADDUBQ, VPHADDUWD, VPHADDUWQ, VPHADDUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDUBQ Packed Horizontal Add Unsigned Byte to Quadword

Adds two sets of eight 8-bit unsigned integer values from the second source and packs the sums into the corresponding quadword of the destination.

There are two operands: VPHADDUBQ *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. When the destination XMM register is written, bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHADDUBQ *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	D3 /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUWD, VPHADDUWQ, VPHADDUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDUBW Packed Horizontal Add Unsigned Byte to Word

Adds each adjacent pair of 8-bit unsigned integer values of the source and packs the 16-bit integer sums to the corresponding word of the destination.

There are two operands: VPHADDUBW *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDUBWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	D1 /r

Related Instructions

VPHADDUBD, VPHADDUBQ, VPHADDUWD, VPHADDUWQ, VPHADDUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDUDQ Packed Horizontal Add Unsigned Doubleword to Quadword

Adds two adjacent pairs of 32-bit unsigned integer values of the source and packs the sums into the corresponding quadword of the destination.

There are two operands: VPHADDUDQ *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHADDUDQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	DB /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWD, VPHADDUWQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDUWD — Packed Horizontal Add Unsigned Word to Doubleword

Adds four adjacent pairs of 16-bit unsigned integer values of the source and packs the sums into the corresponding doubleword of the destination.

There are two operands: VPHADDUWD *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDUWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	D6 /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWQ, VPHADDUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDUWQ Packed Horizontal Add Unsigned Word to Quadword

Adds two pairs of 16-bit unsigned integer values of the source and packs the sums into the corresponding quadword element of the destination.

There are two operands: VPHADDUWQ *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHADDUWQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	D7 /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWD, VPHADDUDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDWD Packed Horizontal Add Signed Word to Signed Doubleword

Adds four adjacent pairs of 16-bit signed integer values of the source and packs the sign-extended sums to the corresponding doubleword of the destination.

There are two operands: VPHADDWD *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	C6 /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDBQ, VPHADDWQ, VPHADDDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHADDWQ Packed Horizontal Add Signed Word to Signed Quadword

Adds four successive pairs of 16-bit signed integer values of the source and packs the sign-extended sums to the corresponding quadword of the destination.

There are two operands: VPHADDWQ *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHADDWQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	C7 /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDBQ, VPHADDWD, VPHADDDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHSUBBW Packed Horizontal Subtract Signed Byte to Signed Word

Subtracts the most significant signed integer byte from the least significant signed integer byte of each word element in the source and packs the sign-extended 16-bit integer differences into the destination.

There are two operands: VPHSUBBW *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBBW *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	E1 /r

Related Instructions

VPHSUBWD, VPHSUBDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHSUBDQ Packed Horizontal Subtract Signed Doubleword to Signed Quadword

Subtracts the most significant signed integer doubleword from the least significant signed integer doubleword of each quadword in the source and packs the sign-extended 64-bit integer differences into the corresponding quadword element of the destination.

There are two operands: VPHSUBDQ *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHSUBDQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	DB /r

Related Instructions

VPHSUBBW, VPHSUBWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPHSUBWD Packed Horizontal Subtract Signed Word to Signed Doubleword

Subtracts the most significant signed integer word from the least significant signed integer word of each doubleword of the source and packs the sign-extended 32-bit integer differences into the destination.

There are two operands: VPHSUBWD *dest*, *src*

The destination is an XMM register and the source is either an XMM register or a 128-bit memory location. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPHSUBWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.01001	0.1111.0.00	E2 /r

Related Instructions

VPHSUBBW, VPHSUBDQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSDD Packed Multiply Accumulate Signed Doubleword to Signed Doubleword

Multiplies each packed 32-bit signed integer value of the first source by the corresponding value of the second source, adds the corresponding value of the third source to the 64-bit signed integer product, and writes four 32-bit sums to the destination.

No saturation is performed on the sum. When the result of the multiplication causes non-zero values to be set in the upper 32 bits of the 64-bit product, they are ignored. When the result of the add overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set). In both cases, only the signed low-order 32 bits of the result are written to the destination.

There are four operands: VPMACSDD *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When the third source designates the same XMM register as the destination*,* the XMM register behaves as an accumulator.

This is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMACSDD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	9E /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSDQH Packed Multiply Accumulate Signed High Doubleword to Signed Quadword

Multiplies the second 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the low-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Simultaneously, multiplies the fourth 32-bit signed integer value of the first source by the fourth 32-bit signed integer value of the second source, then adds the high-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Writes two 64-bit sums to the destination.

No saturation is performed on the sum. When the result of the add overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set).

There are four operands: VPMACSDQH *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When the third source designates the same XMM register as the destination, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSDQH *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	9F /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSDQL Packed Multiply Accumulate Signed Low Doubleword to Signed Quadword

Multiplies the low-order 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the low-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Simultaneously, multiplies the third 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the high-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Writes two 64-bit sums to the destination register.

No saturation is performed on the sum. When the result of the add overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set). Only the low-order 64 bits of each result are written to the destination.

There are four operands: VPMACSDQL *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination is a YMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMACSDQL *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	97 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSSDD Packed Multiply Accumulate with Saturation Signed Doubleword to Signed Doubleword

Multiplies each packed 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the corresponding packed 32-bit signed integer value of the third source to each 64-bit signed integer product. Writes four saturated 32-bit sums to the destination.

Out of range results of the addition are saturated to fit into a signed 32-bit integer. For each packed value of the destination, when the value is larger than the largest signed 32-bit integer, it is saturated to 7FFF_FFFFh, and when the value is smaller than the smallest signed 32-bit integer, it is saturated to 8000_0000h.

There are four operands: VPMACSSDD *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSSDD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	X.$\overline{src}$.0.00	8E /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSSDQH Packed Multiply Accumulate with Saturation Signed High Doubleword to Signed Quadword

Multiplies the second 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the low-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Simultaneously, multiplies the fourth 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the high-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Writes two saturated sums to the destination.

Out of range results of the addition are saturated to fit into a signed 64-bit integer. For each packed value of the destination, when the value is larger than the largest signed 64-bit integer, it is saturated to 7FFF_FFFF_FFFF_FFFFh, and when the value is smaller than the smallest signed 64-bit integer, it is saturated to 8000_0000_0000_0000h.

There are four operands: VPMACSSDQH *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the destination XMM register is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSSDQH *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	8F /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSSDQL Packed Multiply Accumulate with Saturation Signed Low Doubleword to Signed Quadword

Multiplies the low-order 32-bit signed integer value of the first source by the corresponding value of the second source, then adds the low-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Simultaneously, multiplies the third 32-bit signed integer value of the first source by the third 32-bit signed integer value of the second source, then adds the high-order 64-bit signed integer value of the third source to the 64-bit signed integer product. Writes two saturated sums to the destination.

Out of range results of the addition are saturated to fit into a signed 64-bit integer. For each packed value of the destination, when the value is larger than the largest signed 64-bit integer, it is saturated to 7FFF_FFFF_FFFF_FFFFh, and when the value is smaller than the smallest signed 64-bit integer, it is saturated to 8000_0000_0000_0000h.

There are four operands: VPMACSSDQL *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) register is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
PMACSSDQL *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	87 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSSWD Packed Multiply Accumulate with Saturation Signed Word to Signed Doubleword

Multiplies the odd-numbered packed 16-bit signed integer values of the first source by the corresponding values of the second source, then adds the corresponding packed 32-bit signed integer values of the third source to the 32-bit signed integer products. Writes four saturated sums to the destination.

Out of range results of the addition are saturated to fit into a signed 32-bit integer. For each packed value of the destination, when the value is larger than the largest signed 32-bit integer, it is saturated to 7FFF_FFFFh, and when the value is smaller than the smallest signed 32-bit integer, it is saturated to 8000_0000h.

There are four operands: VPMACSSWD d*est, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the destination XMM register is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMACSSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	86 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSSWW Packed Multiply Accumulate with Saturation Signed Word to Signed Word

Multiplies each packed 16-bit signed integer value of the first source by the corresponding packed 16-bit signed integer value of the second source, then adds the corresponding packed 16-bit signed integer value of the third source to the 32-bit signed integer products. Writes four saturated sums to the destination.

Out of range results of the addition are saturated to fit into a signed 16-bit integer. For each packed value of the destination, when the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and when the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

There are four operands: VPMACSSWW *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* and *dest* designate the same XMM register, this register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
PMACSSWW *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	X.$\overline{src}$.0.00	85 /r ib

Related Instructions

VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL,VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSWD Packed Multiply Accumulate Signed Word to Signed Doubleword

Multiplies each odd-numbered packed 16-bit signed integer value of the first source by the corresponding value of the second source, then adds the corresponding packed 32-bit signed integer value of the third source to the 32-bit signed integer products. Writes four 32-bit results to the destination.

When the result of the add overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set). Only the low-order 32 bits of the result are written to the destination.

There are four operands: VPMACSWD *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) register is an XMM register specified by MODRM.reg. When the destination XMM register is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	96 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSSDD, VPMACSDO, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMACSWW Packed Multiply Accumulate Signed Word to Signed Word

Multiplies each packed 16-bit signed integer value of the first source by the corresponding value of the second source, then adds the corresponding packed 16-bit signed integer value of the third source to each 32-bit signed integer product. Writes eight 16-bit results to the destination.

No saturation is performed on the sum. When the result of the multiplication causes non-zero values to be set in the upper 16 bits of the 32 bit result, they are ignored. When the result of the add overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set). In both cases, only the signed low-order 16 bits of the result are written to the destination.

There are four operands: VPMACSWW *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register specified by MODRM.reg. When the destination XMM register is written, bits [255:128] of the corresponding YMM register are cleared.

The first source (*src1*) is an XMM register specified by XOP.vvvv; the second source (*src2*) is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSWW *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	95 /r ib

Related Instructions

VPMACSSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMADCSSWD Packed Multiply Add Accumulate with Saturation Signed Word to Signed Doubleword

Multiplies each packed 16-bit signed integer value of the first source by the corresponding value of the second source, then adds the 32-bit signed integer products of the even-odd adjacent words. Each resulting sum is then added to the corresponding packed 32-bit signed integer value of the third source. Writes four 16-bit results to he destination.

Out of range results of the addition are saturated to fit into a signed 32-bit integer. For each packed value of the destination, when the value is larger than the largest signed 32-bit integer, it is saturated to 7FFF_FFFFh, and when the value is smaller than the smallest signed 32-bit integer, it is saturated to 8000_0000h.

There are four operands: VPMADCSSWD *dest, src1, src2, src3* *dest = src1* src2 + src3*

The destination is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source is an XMM register specified by XOP.vvvv; the second source is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPMADCSSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	A6 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPMADCSWD Packed Multiply Add Accumulate Signed Word to Signed Doubleword

Multiplies each packed 16-bit signed integer value of the first source by the corresponding value of the second source, then adds the 32-bit signed integer products of the even-odd adjacent words together and adds the sums to the corresponding packed 32-bit signed integer values of the third source. Writes four 32-bit sums to the destination.

No saturation is performed on the sum. When the result of the addition overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set). Only the signed 32-bits of the result are written to the destination.

There are four operands: VPMADCSWD *dest, src1, src2, src3*   *dest = src1* src2 + src3*

The destination is an XMM register specified by MODRM.reg. When the destination is written, bits [255:128] of the corresponding YMM register are cleared.

The first source is an XMM register specified by XOP.vvvv, the second source is either an XMM register or a 128-bit memory location specified by the MODRM.rm field; and the third source is an XMM register specified by bits [7:4] of an immediate byte operand.

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
PMADCSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{\textit{src}}$.0.00	B6 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPPERM Packed Permute Bytes

Selects 16 of 32 packed bytes from two concatenated sources, applies a logical transformation to each selected byte, then writes the byte to a specified position in the destination.

There are four operands: VPPERM *dest, src1, src2, src3*

The second (*src2*) and first (*src1*) sources are concatenated to form the 32-byte source.

The *src1* operand is an XMM register specified by XOP.vvvv.

The third source (*src3*) contains 16 control bytes. Each control byte specifies the source byte and the logical operation to perform on that byte. The order of the bytes in the destination is the same as that of the control bytes in the *src3*.

For each byte of the 16-byte result, the corresponding *src3* byte is used as follows:

- Bits [7:5] select a logical operation to perform on the selected byte.

Bit Value	Selected Operation
000	Source byte (no logical operation)
001	Invert source byte
010	Bit reverse of source byte
011	Bit reverse of inverted source byte
100	00h (zero-fill)
101	FFh (ones-fill)
110	Most significant bit of source byte replicated in all bit positions.
111	Invert most significant bit of source byte and replicate in all bit positions.

- Bits [4:0] select a source byte to move from *src2*:*src1*.

Bit Value	Source Byte	Bit Value	Source Byte	Bit Value	Source Byte	Bit Value	Source Byte
00000	*src1*[7:0]	01000	*src1*[71:64]	10000	*src2*[7:0]	11000	*src2*[71:64]
00001	*src1*[15:8]	01001	*src1*[79:72]	10001	*src2*[15:8]	11001	*src2*[79:72]
00010	*src1*[23:16]	01010	*src1*[87:80]	10010	*src2*[23:16]	11010	*src2*[87:80]
00011	*src1*[31:24]	01011	*src1*[95:88]	10011	*src2*[31:24]	11011	*src2*[95:88]
00100	*src1*[39:32]	01100	*src1*[103:96]	10100	*src2*[39:32]	11100	*src2*[103:96]
00101	*src1*[47:40]	01101	*src1*[111:104]	10101	*src2*[47:40]	11101	*src2*[111:104]
00110	*src1*[55:48]	01110	*src1*[119:112]	10110	*src2*[55:48]	11110	*src2*[119:112]
00111	*src1*[63:56]	01111	*src1*[127:120]	10111	*src2*[63:56]	11111	*src2*[127:120]

XOP.W and an immediate byte (*imm8*) determine register configuration.

- When XOP.W = 0, *src2* is either an XMM register or a 128-bit memory location specified by MODRM.rm and *src3* is an XMM register specified by *imm8[7:4]*.
- When XOP.W = 1, *src2* is an XMM register specified by *imm8[7:4]* and *src3* is either an XMM register or a 128-bit memory location specified by MODRM.rm.

The destination (*dest*) is an XMM register specified by MODRM.reg. When the result is written to the *dest* XMM register, bits [255:128] of the corresponding YMM register are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPPERM *xmm1*, *xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src}$.0.00	A3 /r ib
VPPERM *xmm1*, *xmm2, xmm3, xmm4/mem128*	8F	$\overline{\text{RXB}}$.01000	1.$\overline{src}$.0.00	A3 /r ib

Related Instructions

VPSHUFHW, VPSHUFD, VPSHUFLW, VPSHUFW, VPERMIL2PS, VPERMIL2PD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPROTB Packed Rotate Bytes

Rotates each byte of the source as specified by a count operand and writes the result to the corresponding byte of the destination.

There are two versions of the instruction, one for each source of the count byte:

- VPROTB *dest, src, fixed-count*
- VPROTB *dest, src, variable-count*

For both versions of the instruction, the destination (*dest*) operand is an XMM register specified by MODRM.reg.

The *fixed-count* version of the instruction rotates each byte of the source (*src*) the number of bits specified by the immediate *fixed-count* byte. All bytes are rotated the same amount. The source XMM register or memory location is selected by the MODRM.rm field.

The *variable-count* version of the instruction rotates each byte of the source the amount specified in the corresponding byte element of the *variable-count*. Both *src* and *variable-count* are configured by XOP.W.

- When XOP.W = 0, *variable-count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a 128-bit memory location specified by MODRM.rm.
- When XOP.W = 1, *variable-count* is either an XMM register or a 128-bit memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

When the count value is positive, bits are rotated to the left (toward the more significant bit positions). The bits rotated out left of the most significant bit are rotated back in at the right end (least-significant bit) of the byte.

When the count value is negative, bits are rotated to the right (toward the least significant bit positions). The bits rotated to the right out of the least significant bit are rotated back in at the left end (most-significant bit) of the byte.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTB *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	90 /r
VPROTB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	90 /r
VPROTB *xmm1*, *xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.01000	0.1111.0.00	C0 /r ib

Related Instructions

VPROTW, VPROTD, VPROTQ,VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.vvvv ! = 1111b (for immediate operand variant only)
			X	XOP.L field = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPROTD Packed Rotate Doublewords

Rotates each doubleword of the source as specified by a count operand and writes the result to the corresponding doubleword of the destination.

There are two versions of the instruction, one for each source of the count byte:

- VPROTD *dest, src, fixed-count*
- VPROTD *dest, src, variable-count*

For both versions of the instruction, the *dest* operand is an XMM register specified by MODRM.reg.

The fixed count version of the instruction rotates each doubleword of the source operand the number of bits specified by the immediate *fixed-count* byte operand. All doublewords are rotated the same amount. The *src* XMM register or memory location is selected by the MODRM.rm field.

The variable count version of the instruction rotates each doubleword of the source by the amount specified in the low order byte of the corresponding doubleword of the *variable-count* operand vector.

Both *src* and *variable-count* are configured by XOP.W.

- When XOP.W = 0, *src* is either an XMM register or a128-bit memory location specified by the MODRM.rm field and *variable-count* is an XMM register specified by XOP.vvvv.
- When XOP.W = 1, *src* is an XMM register specified by XOP.vvvv and *variable-count* is either an XMM register or a 128-bit memory location specified by the MODRM.rm field.

When the count value is positive, bits are rotated to the left (toward the more significant bit positions). The bits rotated out to the left of the most significant bit of each source doubleword operand are rotated back in at the right end (least-significant bit) of the doubleword.

When the count value is negative, bits are rotated to the right (toward the least significant bit positions). The bits rotated to the right out of the least significant bit of each source doubleword operand are rotated back in at the left end (most-significant bit) of the doubleword.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTD *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	92 /r
VPROTD *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	92 /r
VPROTD *xmm1, xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.01000	0.1111.0.00	C2 /r ib

Related Instructions

VPROTB, VPROTW, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.vvvv ! = 1111b (for immediate operand variant only)
			X	XOP.L field = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPROTQ Packed Rotate Quadwords

Rotates each quadword of the source operand as specified by a count operand and writes the result to the corresponding quadword of the destination.

There are two versions of the instruction, one for each source of the count byte:

- VPROTQ *dest, src, fixed-count*
- VPROTQ *dest, src, variable-count*

For both versions of the instruction, the *dest* operand is an XMM register specified by MODRM.reg.

The fixed count version of the instruction rotates each quadword in the source the number of bits specified by the immediate *fixed-count* byte operand. All quadword elements of the source are rotated the same amount. The *src* XMM register or memory location is selected by the MODRM.rm field.

The variable count version of the instruction rotates each quadword of the source the amount specified ny the low order byte of the corresponding quadword of the *variable-count* operand.

Both *src* and *variable-count* are configured by XOP.W.

- When XOP.W = 0, *src* is either an XMM register or a 128-bit memory location specified by MODRM.rm and *variable-count* is an XMM register specified by XOP.vvvv.
- When XOP.W = 1, *src* is an XMM register specified by XOP.vvvv and *variable-count* is either an XMM register or a128-bit memory location specified by MODRM.rm.

When the count value is positive, bits are rotated to the left (toward the more significant bit positions) of the operand element. The bits rotated out to the left of the most significant bit of the word element are rotated back in at the right end (least-significant bit).

When the count value is negative, operand element bits are rotated to the right (toward the least significant bit positions). The bits rotated to the right out of the least significant bit are rotated back in at the left end (most-significant bit) of the word element.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTQ *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	93 /r
VPROTQ *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	93 /r
VPROTQ *xmm1, xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.01000	0.1111.0.00	C3 /r ib

Related Instructions

VPROTB, VPROTW, VPROTD, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.vvvv ! = 1111b (for immediate operand variant only)
			X	XOP.L field = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPROTW Packed Rotate Words

Rotates each word of the source as specified by a count operand and writes the result to the corresponding word of the destination.

There are two versions of the instruction, one for each source of the count byte:

- VPROTW dest, *src*, *fixed-count*
- VPROTW dest, *src*, *variable-count*

For both versions of the instruction, the *dest* operand is an XMM register specified by MODRM.reg.

The fixed count version of the instruction rotates each word of the source the number of bits specified by the immediate *fixed-count* byte operand. All words of the source operand are rotated the same amount. The *src* XMM register or memory location is selected by the MODRM.rm field.

The variable count version of this instruction rotates each word of the source operand by the amount specified in the low order byte of the corresponding word of the *variable-count* operand.

Both *src* and *variable-count* are configured by XOP.W.

- When XOP.W = 0, *src* is either an XMM register or a 128-bit memory location specified by MODRM.rm and *variable-count* is an XMM register specified by XOP.vvvv.
- When XOP.W = 1, *src* is an XMM register specified by XOP.vvvv and *variable-count* is either an XMM register or a 128-bit memory location specified by MODRM.rm.

When the count value is positive, bits are rotated to the left (toward the more significant bit positions). The bits rotated out to the left of the most significant bit of an element are rotated back in at the right end (least-significant bit) of the word element.

When the count value is negative, bits are rotated to the right (toward the least significant bit positions) of the element. The bits rotated to the right out of the least significant bit of an element are rotated back in at the left end (most-significant bit) of the word element.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTW *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	91 /r
VPROTW *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	91 /r
VPROTW *xmm1, xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.01000	0.1111.0.00	C1 /r ib

Related Instructions

VPROTB, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.vvvv ! = 1111b (for immediate operand variant only)
			X	XOP.L field = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHAB Packed Shift Arithmetic Bytes

Shifts each signed byte of the source as specified by a count byte and writes the result to the corresponding byte of the destination.

The count bytes are 8-bit signed two's-complement values in the corresponding bytes of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the byte.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). The most significant bit (sign bit) is replicated and shifted in at the left end (most-significant bit) of the byte.

There are three operands: VPSHAB *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a128-bit memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a 128-bit memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSHAB *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	98 /r
VPSHAB *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	98 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHAD Packed Shift Arithmetic Doublewords

Shifts each signed doubleword of the source operand as specified by a count byte and writes the result to the corresponding doubleword of the destination.

The count bytes are 8-bit signed two's-complement values located in the low-order byte of the corresponding doubleword of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the doubleword.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). The most significant bit (sign bit) is replicated and shifted in at the left end (most-significant bit) of the doubleword.

There are three operands: VPSHAD *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSHAD *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{\textit{src}}$.0.00	9A /r
VPSHAD *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{\textit{src}}$.0.00	9A /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHAQ Packed Shift Arithmetic Quadwords

Shifts each signed quadword of the source as specified by a count byte and writes the result to the corresponding quadword of the destination.

The count bytes are 8-bit signed two's-complement values located in the low-order byte of the corresponding quadword element of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the quadword.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). The most significant bit is replicated and shifted in at the left end (most-significant bit) of the quadword.

The shift amount is stored in two's-complement form. The count is modulo 64.

There are three operands: VPSHAQ *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSHAQ *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	9B /r
VPSHAQ *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	9B /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHAW Packed Shift Arithmetic Words

Shifts each signed word of the source as specified by a count byte and writes the result to the corresponding word of the destination.

The count bytes are 8-bit signed two's-complement values located in the low-order byte of the corresponding word of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the word.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). The most significant bit (signed bit) is replicated and shifted in at the left end (most-significant bit) of the word.

The shift amount is stored in two's-complement form. The count is modulo 16.

There are three operands: VPSHAW *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSHAW *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	99 /r
VPSHAW *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	99 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHLB Packed Shift Logical Bytes

Shifts each packed byte of the source as specified by a count byte and writes the result to the corresponding byte of the destination.

The count bytes are 8-bit signed two's-complement values located in the corresponding byte element of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the byte.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). Zeros are shifted in at the left end (most-significant bit) of the byte.

There are three operands: VPSHLB *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLB *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	94 /r
VPSHLB *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	94 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHLD Packed Shift Logical Doublewords

Shifts each doubleword of the source operand as specified by a count byte and writes the result to the corresponding doubleword of the destination.

The count bytes are 8-bit signed two's-complement values located in the low-order byte of the corresponding doubleword element of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the doubleword.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). Zeros are shifted in at the left end (most-significant bit) of the doubleword.

The shift amount is stored in two's-complement form. The count is modulo 32.

There are three operands: VPSHLD *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLD *xmm1*, *xmm3/mem128, xmm2*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	96 /r
VPSHLD *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	96 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHLQ Packed Shift Logical Quadwords

Shifts each quadwords of the source by as specified by a count byte and writes the result in the corresponding quadword of the destination.

The count bytes are 8-bit signed two's-complement values located in the low-order byte of the corresponding quadword element of the *count* operand.

Bit 6 of the count byte is ignored.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the quadword.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). Zeros are shifted in at the left end (most-significant bit) of the quadword.

There are three operands: VPSHLQ *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	XOP	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VPSHLQ *xmm1*, *xmm3*/*mem128, xmm2*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{\textit{src}}$.0.00	97 /r
VPSHLQ *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{\textit{src}}$.0.00	97 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VPSHLW Packed Shift Logical Words

Shifts each word of the source operand as specified by a count byte and writes the result to the corresponding word of the destination.

The count bytes are 8-bit signed two's-complement values located in the low-order byte of the corresponding word element of the *count* operand.

When the count value is positive, bits are shifted to the left (toward the more significant bit positions). Zeros are shifted in at the right end (least-significant bit) of the word.

When the count value is negative, bits are shifted to the right (toward the least significant bit positions). Zeros are shifted in at the left end (most-significant bit) of the word.

There are three operands: VPSHLW *dest, src, count*

The destination (*dest*) is an XMM register specified by MODRM.reg.

Both *src* and *count* are configured by XOP.W.

- When XOP.W = 0, *count* is an XMM register specified by XOP.vvvv and *src* is either an XMM register or a memory location specified by MODRM.rm.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by MODRM.rm and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

This is an XOP instruction. Support for these instructions is indicated by CPUID Fn8000_00001_ECX[XOP] (see the CPUID Specification, order# 25481).

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLW *xmm1*, *xmm3/mem128, xmm2*	8F	$\overline{\text{RXB}}$.01001	0.$\overline{src}$.0.00	95 /r
VPSHLW *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.01001	1.$\overline{src}$.0.00	95 /r

Related Instructions

VPROTB, VPROLW, VPROTD, VPROTQ, VPSHLB, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

VTESTPD Packed Bit Test

First, performs a bitwise AND of the sign bits of each double-precision floating-point element of the first source operand with the sign bits of the corresponding elements of the second source operand. Sets rFLAGS.ZF when all bit operations = 0; else, clears ZF.

Second, performs a bitwise AND of the sign bits of each double-precision floating-point element of the first source with the complements (NOT) of the sign bits of the corresponding elements of the second source operand. Sets rFLAGS.CF when all bit operations = 0; else, clears CF.

Neither source operand is modified.

This extended-form instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location.

YMM Encoding

The first source operand is a YMM register. The second source operand is either a YMM register or a 256-bit memory location.

VTESTPD is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VTESTPD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	0.1111.0.01	0F /r
VTESTPD *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00010	0.1111.1.01	0F /r

Related Instructions

PTEST, VTESTPS

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				M	M	M	M	M
21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3 and 1 are reserved. A flag set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

VTESTPS Packed Bit Test

First, performs a bitwise AND of the sign bits of each single-precision floating-point element of the first source operand with the sign bits of the corresponding elements of the second source operand. Sets rFLAGS.ZF when all bit operations = 0; else, clears ZF.

Second, performs a bitwise AND of the sign bits of each single-precision floating-point element of the first source with the complements (NOT) of the sign bits of the corresponding elements of the second source operand. Sets rFLAGS.CF when all bit operations = 0; else, clears CF.

Neither source operand is modified.

This extended-form instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location.

YMM Encoding

The first source operand is a YMM register. The second source operand is either a YMM register or a 256-bit memory location.

VTESTPS is an AVX instruction. Support for these instructions is indicated by CPUID feature identifiers (see the CPUID Specification, order# 25481).

Mnemonic	**Encoding**			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VTESTPS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.00010	0.1111.0.01	0E /r
VTESTPS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.00010	0.1111.1.01	0E /r

Related Instructions

PTEST, VTESTPD

rFLAGS Affected

ID	VIP	VIF	AC	VM	RF	NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
								0				M	M	M	M	M
21	20	19	18	17	16	14	13:12	11	10	9	8	7	6	4	2	0

Note: *Bits 31:22, 15, 5, 3 and 1 are reserved. A flag set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

VZEROALL Clear All YMM Registers

Clears all XMM and YMM registers.

This extended-form instruction has 256-bit YMM encoding.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	**RXB.mmmmm**	**W.vvvv.L.pp**	**Opcode**
VZEROALL	C4	$\overline{\text{RXB}}$.00001	X.1111.1.00	77

Related Instructions

VZEROUPPER

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
A — AVX exception.				

VZEROUPPER Clear Bits [255:128] Of All YMM Registers

Clears bits [255:128] of all YMM registers. The corresponding XMM registers are not affected.

This extended-form instruction has 128-bit encoding.

This is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VZEROUPPER	C4	$\overline{\text{RXB}}$.00001	X.1111.0.00	77

Related Instructions

VZEROUPPER

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
A — AVX exception.				

XORPD VXORPD — XOR Packed Double-Precision Floating-Point

Performs bitwise XOR of two packed double-precision floating-point values in the first source operand with the corresponding values of the second source operand and writes the results into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

XORPD

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VXORPD

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

XORPD is an SSE2 instruction and VXORPD is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
XORPD *xmm1*, *xmm2/mem128*	66 0F 57 /r	Performs bitwise XOR of two packed double-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VXORPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	57 /r
VXORPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.01	57 /r

Related Instructions

(V)ANDNPS, (V)ANDPD, (V)ANDPS, (V)ORPD, (V)ORPS, (V)XORPS

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

XORPS / VXORPS — XOR Packed Single-Precision Floating-Point

Performs bitwise XOR of four packed single-precision floating-point values in the first source operand with the corresponding values of the second source operand and writes the results into the corresponding elements of the destination.

There are legacy and extended forms of the instruction:

XORPS

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The first source register is also the destination. Bits [255:128] of the YMM register that corresponds to the destination are not affected.

VXORPS

The extended form of the instruction has both 128-bit and 256-bit encoding.

XMM Encoding

The first source operand is an XMM register. The second source operand is either another XMM register or a 128-bit memory location. The destination is a third XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

YMM Encoding

The first source operand is a YMM register and the second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register.

XORPS is an SSE2 instruction and VXORPS is an AVX instruction. Support for these instructions is indicated by CPUID Fn0000_00001_EDX[SSE2] and Fn0000_00001_ECX[AVX] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
XORPS *xmm1*, *xmm2/mem128*	66 0F 57 /r	Performs bitwise XOR of four packed single-precision floating-point values in *xmm1* with corresponding values in *xmm2* or *mem128*. Writes the result to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VXORPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	57 /r
VXORPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.00	57 /r

Related Instructions

(V)ANDNPS, (V)ANDPD, (V)ANDPS, (V)ORPD, (V)ORPS, (V)XORPD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

XGETBV Get Extended Control Register Value

Copies the content of the extended control register (XCR) specified by the ECX register into the EDX:EAX register pair. The high-order 32 bits of the XCR are loaded into EDX and the low-order 32 bits are loaded into EAX. The corresponding high-order 32 bits of RAX and RDX are cleared.

This instruction and associated data structures extend the FXSAVE/FXRSTOR memory image used to manage processor states and provide additional functionality. See Section 1.3, "XSAVE/XRSTOR Instructions" for more information.

Values returned to EDX:EAX in unimplemented bit locations are undefined.

Specifying a reserved or unimplemented XCR in ECX causes a general protection exception.

Currently, only XCR0 (the XFEATURE_ENABLED_MASK register) is supported. All other values of ECX are reserved.

XGETBV is an XSAVE/XRSTOR instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[XSAVE] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
XGETBV	0F 01 D0	Copies content of the XCR specified by ECX into EDX:EAX.

Related Instructions

RDMSR

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	Lock prefix (F0h) preceding opcode.
General protection, #GP	X	X	X	ECX specifies a reserved or unimplemented XCR address.
X — exception generated				

XRSTOR Save Extended States

Restores processor state from memory.

This instruction and associated data structures extend the FXSAVE/FXRSTOR memory image used to manage processor states and provide additional functionality. See Section 1.3, “XSAVE/XRSTOR Instructions” for more information.

The XSAVE/XRSTOR save area consists of a header section and individual save areas for each processor state component. A component save area is updated when both the corresponding bits in the mask operand (EDX:EAX) and the XFEATURE_ENABLED_MASK (XCR0) register are set. A component save area is not updated when either of the corresponding bits in EDX:EAX or XCR0 is cleared. Updated state is either loaded from memory or set directly to hardware-specified initial values, depending on the corresponding xstate_bv bit in the save area header.

Software can set any bit in EDX:EAX, regardless of whether the bit position in XCR0 is valid for the processor. When the mask operand contains all 1's, all processor state components enabled in XCR0 are updated.

XRSTOR is an XSAVE/XRSTOR instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[XSAVE] (see the *CPUID Specification*, order# 25481).

Mnemonic	**Opcode**	**Description**
XRSTOR *mem*	0F AE /5	Restores user-specified processor state from memory.

Related Instructions

XGETBV, XSAVE, XSETBV

Exceptions

Exception	**Mode**			**Cause of Exception**
	Real	**Virt**	**Prot**	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	CR4.OSFXSR = 0.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	X	X	X	CR0.TS = 1.
Stack, #SS	X	X	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	X	X	X	Memory address exceeding data segment limit or non-canonical.
	X	X	X	Null data segment used to reference memory.
	X	X	X	Memory operand not aligned on 64-byte boundary.
	X	X	X	Any must be zero (MBZ) bits in the save area were set.
	X	X	X	Attempt to set reserved bits in MXCSR.
Page fault, #PF	X	X	X	Instruction execution caused a page fault.
X — exception generated				

XSAVE Save Extended States

Saves a user-defined subset of enabled processor state data to a specified memory address.

This instruction and associated data structures extend the FXSAVE/FXRSTOR memory image used to manage processor states and provide additional functionality. See Section 1.3, "XSAVE/XRSTOR Instructions" for more information.

The XSAVE/XRSTOR save area consists of a header section, and individual save areas for each processor state component. A component is saved when both the corresponding bits in the mask operand (EDX:EAX) and the XFEATURE_ENABLED_MASK (XCR0) register are set. A component is not saved when either of the corresponding bits in EDX:EAX or XCR0 is cleared.

Software can set any bit in EDX:EAX, regardless of whether the bit position in XCR0 is valid for the processor. When the mask operand contains all 1's, all processor state components enabled in XCR0 are saved.

For each component saved, XSAVE sets the corresponding bit in the XSTATE_BV field of the save area header. XSAVE does not clear XSTATE_BV bits or modify individual save areas for components that are not saved. If a saved component is in the hardware-specified initialized state, XSAVE may clear the corresponding XSTATE_BV bit instead of setting it. This optimization is implementation-dependent.

XSAVE is an XSAVE/XRSTOR instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[XSAVE] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
XSAVE *mem*	0F AE /4	Saves user-specified processor state to memory.

Related Instructions

XGETBV, XRSTOR, XSETBV

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	CR4.OSFXSR = 0.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	X	X	X	CR0.TS = 1.
Stack, #SS	X	X	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	X	X	X	Memory address exceeding data segment limit or non-canonical.
	X	X	X	Null data segment used to reference memory.
	X	X	X	Memory operand not aligned on 64-byte boundary.
	X	X	X	Attempt to write read-only memory.
Page fault, #PF	X	X	X	Instruction execution caused a page fault.
X — exception generated				

XSETBV Set Extended Control Register Value

Writes the content of the EDX:EAX register pair into the extended control register (XCR) specified by the ECX register. The high-order 32 bits of the XCR are loaded from EDX and the low-order 32 bits are loaded from EAX. The corresponding high-order 32 bits of RAX and RDX are ignored.

This instruction and associated data structures extend the FXSAVE/FXRSTOR memory image used to manage processor states and provide additional functionality. See Section 1.3, "XSAVE/XRSTOR Instructions" for more information.

Currently, only the XFEATURE_ENABLED_MASK register (XCR0) is supported. Specifying a reserved or unimplemented XCR in ECX causes a general protection exception (#GP).

Executing XSETBV at a privilege level other than 0 causes a general-protection exception. A general protection exception also occurs when software attempts to write to reserved bits of an XCR.

The XGETBV instruction is an XSAVE/XRSTOR instruction. Support for these instructions is indicated by CPUID Fn0000_00001_ECX[XSAVE] (see the *CPUID Specification*, order# 25481).

Mnemonic	Opcode	Description
XSETBV	0F 01 D1	Writes the content of the EDX:EAX register pair to the XCR specified by the ECX register.

Related Instructions

XGETBV, XRSTOR, XSAVE

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	CR4.OSFXSR = 0.
	X	X	X	Lock prefix (F0h) preceding opcode.
General protection, #GP	X	X	X	CPL != 0.
	X	X	X	ECX specifies a reserved or unimplemented XCR address.
	X	X	X	Any must be zero (MBZ) bits in the XCR were set.
				Setting XCR0[2:1] to 10b.
	X	X	X	Writing 0 to XCR[0].

X — exception generated

Note:
In virtual mode, only #UD for Instruction not supported and #GP for CPL != 0 are supported.

3 Exception Summary

This chapter provides a ready reference to instruction exceptions. Table 3-1 shows instructions grouped by exception class, with the extended and legacy instruction type (if applicable). Examples of the exception tables for each class, in numeric order, follow the table.

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 1 — AVX, SSE, 16/32-byte aligned, write to RO, VEX.vvvv != 1111b		
MOVAPD VMOVAPD	AVX	SSE2
MOVAPS VMOVAPS	AVX	SSE
MOVDQA VMOVDQA	AVX	SSE2
MOVNTDQ VMOVNTDQ	AVX	SSE2
MOVNTPD VMOVNTPD	AVX	SSE2
MOVNTPS VMOVNTPS	AVX	SSE
Class 1A — AVX, SSE, 16/32-byte aligned, write to RO, VEX.vvvv != 1111b, VEX.L = 1		
MOVNTDQA VMOVNTDQA	AVX	SSE4.1
Class 2 — AVX, SSE, 16/32-byte nonaligned, SIMD 111111		
DIVPD VDIVPD	AVX	SSE2
DIVPS VDIVPS	AVX	SSE
Class 2-1 — AVX, SSE, 16/32-byte nonaligned, SIMD 111011		
ADDPD VADDPD	AVX	SSE2
ADDPS VADDPS	AVX	SSE
ADDSUBPD VADDSUBPD	AVX	SSE2
ADDSUBPS VADDSUBPS	AVX	SSE
DPPS VDPPS	AVX	SSE4.1
HADDPD VHADDPD	AVX	SSE3
HADDPS VHADDPS	AVX	SSE3
HSUBPD VHSUBPD	AVX	SSE3
HSUBPS VHSUBPS	AVX	SSE3
SUBPD VSUBPD	AVX	SSE2
SUBPS VSUBPS	AVX	SSE
Class 2-2 — AVX, SSE, 16/32-byte nonaligned, SIMD 000011		
CMPPD VCMPPD	AVX	SSE2
CMPPS VCMPPS	AVX	SSE
MAXPD VMAXPD	AVX	SSE2
MAXPS VMAXPS	AVX	SSE
MINPD VMINPD	AVX	SSE2
MINPS VMINPS	AVX	SSE
MULPD VMULPD	AVX	SSE2
MULPS VMULPS	AVX	SSE
Class 2-3 — AVX, SSE, 16/32-byte nonaligned, SIMD 100001		
ROUNDPD, VROUNDPD	AVX	SSE4.1
ROUNDPS, VROUNDPS	AVX	SSE4.1

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 2A — AVX, SSE, 16/32-byte nonaligned, SIMD 111111, VEX.L = 1 [1]		
Class 2A-1 — AVX, SSE, 16/32-byte nonaligned, SIMD 111011, VEX.L = 1		
DPPD VDPPD	AVX	SSE4.1
Class 2B — AVX, SSE, 16/32-byte nonaligned, SIMD 111111, VEX.vvvv != 1111b [1]		
Class 2B-1 — AVX, SSE, 16/32-byte nonaligned, SIMD 100000, VEX.vvvv != 1111b		
CVTDQ2PS VCVTDQ2PS	AVX	SSE2
Class 2B-2 — AVX, SSE, 16/32-byte nonaligned, SIMD 100001, VEX.vvvv != 1111b		
CVTPD2DQ VCVTPD2DQ	AVX	SSE2
CVTPS2DQ VCVTPS2DQ	AVX	SSE2
CVTTPS2DQ VCVTTPS2DQ	AVX	SSE2
CVTTPD2DQ VCVTTPD2DQ	AVX	SSE2
Class 2B-3 — AVX, SSE, 16/32-byte nonaligned, SIMD 111011, VEX.vvvv != 1111b		
CVTPD2PS VCVTPD2PS	AVX	SSE2
Class 2B-4 — AVX, SSE, 16/32-byte nonaligned, SIMD 100011, VEX.vvvv != 1111b		
SQRTPD VSQRTPD	AVX	SSE2
SQRTPS VSQRTPS	AVX	SSE
Class 3 — AVX, SSE, <16-byte, SIMD		
DIVSD VDIVSD	AVX	SSE2
DIVSS VDIVSS	AVX	SSE
Class 3-1 — AVX, SSE, <16-byte, SIMD 111011		
ADDSD VADDSD	AVX	SSE2
ADDSS VADDSS	AVX	SSE
CVTSD2SS VCVTSD2SS	AVX	SSE2
SUBSD VSUBSD	AVX	SSE2
SUBSS VSUBSS	AVX	SSE
Class 3-2 — AVX, SSE, <16-byte, SIMD 000011		
CMPSD VCMPSD	AVX	SSE2
CMPSS VCMPSS	AVX	SSE
CVTSS2SD VCVTSS2SD	AVX	SSE2
MAXSD VMAXSD	AVX	SSE2
MAXSS VMAXSS	AVX	SSE
MINSD VMINSD	AVX	SSE2
MINSS VMINSS	AVX	SSE
MULSD VMULSD	AVX	SSE2
MULSS VMULSS	AVX	SSE
UCOMISD VUCOMISD	AVX	SSE2
UCOMISS VUCOMISS	AVX	SSE
Class 3-3 — AVX, SSE, <16-byte, SIMD 100000		
CVTSI2SD VCVTSI2SD	AVX	SSE2
CVTSI2SS VCVTSI2SS	AVX	SSE

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 3-4 — AVX, SSE, <16-byte, SIMD 100001		
ROUNDSD, VROUNDSD	AVX	SSE4.1
ROUNDSS, VROUNDSS	AVX	SSE4.1
Class 3-5 — AVX, SSE, <16-byte, SIMD 100011		
SQRTSD VSQRTSD	AVX	SSE2
SQRTSS VSQRTSS	AVX	SSE
Class 3A — AVX, SSE, <16-byte, SIMD 111111, VEX.vvvv != 1111b[1]		
Class 3A-1 — AVX, SSE, <16-byte, SIMD 000011, VEX.vvvv != 1111b		
COMISD VCOMISD	AVX	SSE2
COMISS VCOMISS	AVX	SSE
CVTPS2PD VCVTPS2PD	AVX	SSE2
Class 3A-2 — AVX, SSE, <16-byte, SIMD 100001, VEX.vvvv != 1111b		
CVTSD2SI VCVTSD2SI	AVX	SSE2
CVTSS2SI VCVTSS2SI	AVX	SSE
CVTTSD2SI VCVTTSD2SI	AVX	SSE2
CVTTSS2SI VCVTTSS2SI	AVX	SSE
Class 4 — AVX, SSE, 16/32-byte nonaligned		
AESDEC VAESDEC	AVX	AES
AESDECLAST VAESDECLAST	AES	AES
AESENC VAESENC	AES	AES
AESENCLAST VAESENCLAST	AES	AES
AESIMC VAESIMC	AES	AES
AESKEYGENASSIST VAESKEYGENASSIST	AES	AES
ANDNPD VANDNPD	AVX	SSE2
ANDNPS VANDNPS	AVX	SSE
ANDPD VANDPD	AVX	SSE2
ANDPS VANDPS	AVX	SSE
BLENDPD VBLENDPD	AVX	SSE4.1
BLENDPS VBLENDPS	AVX	SSE4.1
ORPD VORPD	AVX	SSE2
ORPS VORPS	AVX	SSE
PCLMULQDQ	—	CLMUL
SHUFPD VSHUFPD	AVX	SSE2
SHUFPS VSHUFPS	AVX	SSE2
UNPCKHPD VUNPCKHPD	AVX	SSE2
UNPCKHPS VUNPCKHPS	AVX	SSE
UNPCKLPD VUNPCKLPD	AVX	SSE2
UNPCKLPS VUNPCKLPS	AVX	SSE
XORPD VXORPD	AVX	SSE2
XORPS VXORPS	AVX	SSE

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 4A — AVX, SSE, 16/32-byte nonaligned, VEX.W = 1		
BLENDVPD VBLENDVPD	AVX	SSE4.1
BLENDVPS VBLENDVPS	AVX	SSE4.1
Class 4B — AVX, SSE, 16/32-byte nonaligned, VEX.L = 1		
MPSADBW VMPSADBW	AVX	SSE4.1
PACKSSDW VPACKSSDW	AVX	SSE2
PACKSSWB VPACKSSWB	AVX	SSE2
PACKUSDW VPACKUSDW	AVX	SSE4.1
PACKUSWB VPACKUSWB	AVX	SSE2
PADDB VPADDB	AVX	SSE2
PADDD VPADDD	AVX	SSE2
PADDQ VPADDQ	AVX	SSE2
PADDSB VPADDSB	AVX	SSE2
PADDSW VPADDSW	AVX	SSE2
PADDUSB VPADDUSB	AVX	SSE2
PADDUSW VPADDUSW	AVX	SSE2
PADDW VPADDW	AVX	SSE2
PALIGNR VPALIGNR	AVX	SSSE3
PANDN VPANDN	AVX	SSE2
PAND VPAND	AVX	SSE2
PAVGB VPAVGB	AVX	SSE
PAVGW VPAVGW	AVX	SSE
PBLENDW VPBLENDW	AVX	SSE4.1
PCMPEQB VPCMPEQB	AVX	SSE2
PCMPEQD VPCMPEQD	AVX	SSE2
PCMPEQQ VPCMPEQQ	AVX	SSE4.1
PCMPEQW VPCMPEQW	AVX	SSE2
PCMPGTB VPCMPGTB	AVX	SSE2
PCMPGTD VPCMPGTD	AVX	SSE2
PCMPGTQ VPCMPGTQ	AVX	SSE4.2
PCMPGTW VPCMPGTW	AVX	SSE2
PCMPISTRI VPCMPISTRI	AVX	SSE4.2
PCMPISTRM VPCMPISTRM	AVX	SSE4.2
PHADDD VPHADDD	AVX	SSSE3
PHADDSW VPHADDSW	AVX	SSSE3
PHADDW VPHADDW	AVX	SSSE3
PHSUBD VPHSUBD	AVX	SSSE3
PHSUBW VPHSUBW	AVX	SSSE3
PHSUBSW VPHSUBSW	AVX	SSSE3
PMADDUBSW VPMADDUBSW	AVX	SSSE3
PMADDWD VPMADDWD	AVX	SSE2
PMAXSB VPMAXSB	AVX	SSE4.1

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
PMAXSD VPMAXSD	AVX	SSE4.1
PMAXSW VPMAXSW	AVX	SSE
PMAXUB VPMAXUB	AVX	SSE
PMAXUD VPMAXUD	AVX	SSE4.1
PMAXUW VPMAXUW	AVX	SSE4.1
PMINSB VPMINSB	AVX	SSE4.1
PMINSD VPMINSD	AVX	SSE4.1
PMINSW VPMINSW	AVX	SSE
PMINUB VPMINUB	AVX	SSE
PMINUD VPMINUD	AVX	SSE4.1
PMINUW VPMINUW	AVX	SSE4.1
PMULDQ VPMULDQ	AVX	SSE4.1
PMULHRSW VPMULHRSW	AVX	SSSE3
PMULHUW VPMULHUW	AVX	SSE2
PMULHW VPMULHW	AVX	SSE2
PMULLD VPMULLD	AVX	SSE4.1
PMULLW VPMULLW	AVX	SSE2
PMULUDQ VPMULUDQ	AVX	SSE2
POR VPOR	AVX	SSE2
PSADBW VPSADBW	AVX	SSE
PSHUFB VPSHUFB	AVX	SSSE3
PSIGNB VPSIGNB	AVX	SSSE3
PSIGND VPSIGND	AVX	SSSE3
PSIGNW VPSIGNW	AVX	SSSE3
PSUBB VPSUBB	AVX	SSE2
PSUBD VPSUBD	AVX	SSE2
PSUBQ VPSUBQ	AVX	SSE2
PSUBSB VPSUBSB	AVX	SSE2
PSUBSW VPSUBSW	AVX	SSE2
PSUBUSB VPSUBUSB	AVX	SSE2
PSUBUSW VPSUBUSW	AVX	SSE2
PSUBW VPSUBW	AVX	SSE2
PUNPCKHBW VPUNPCKHBW	AVX	SSE2
PUNPCKHDQ VPUNPCKHDQ	AVX	SSE2
PUNPCKHQDQ VPUNPCKHQDQ	AVX	SSE2
PUNPCKHWD VPUNPCKHWD	AVX	SSE2
PUNPCKLBW VPUNPCKLBW	AVX	SSE2
PUNPCKLDQ VPUNPCKLDQ	AVX	SSE2
PUNPCKLQDQ VPUNPCKLQDQ	AVX	SSE2
PUNPCKLWD VPUNPCKLWD	AVX	SSE2
PXOR VPXOR	AVX	SSE2

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 4C — AVX, SSE, 16/32-byte nonaligned, VEX.vvvv != 1111b		
LDDQU VLDDQU	AVX	SSE3
MOVSHDUP VMOVSHDUP	AVX	SSE3
MOVSLDUP VMOVSLDUP	AVX	SSE3
PSHUFD VPSHUFD	AVX	SSE2
PSHUFHW VPSHUFHW	AVX	SSE2
PSHUFLW VPSHUFLW	AVX	SSE2
PTEST VPTEST	AVX	SSE4.1
RCPPS VRCPPS	AVX	SSE
RSQRTPS VRSQRTPS	AVX	SSE
Class 4C-1 — AVX, SSE, 16/32-byte nonaligned, write to RO, VEX.vvvv != 1111b		
MOVDQU VMOVDQU	AVX	SSE2
MOVUPD VMOVUPD	AVX	SSE2
MOVUPS VMOVUPS	AVX	SSE
Class 4D — AVX, SSE, 16/32-byte nonaligned, VEX.vvvv != 1111b, VEX.L = 1		
MASKMOVDQU VMASKMOVDQU	AVX	SSE2
PABSB VPABSB	AVX	SSSE3
PABSD VPABSD	AVX	SSSE3
PABSW VPABSW	AVX	SSSE3
PCMPESTRI VPCMPESTRI	AVX	SSE4.2
PCMPESTRM VPCMPESTRM	AVX	SSE4.2
PHMINPOSUW VPHMINPOSUW	AVX	SSE4.1
Class 4E — AVX, SSE, 16/32-byte nonaligned, VEX.W = 1, VEX.L = 1		
PBLENDVB VPBLENDVB	AVX	SSE4.1
Class 4F — AVX, SSE, 16/32-byte nonaligned, VEX.L = 1 (no memory argument for SSE)		
PSLLD VPSLLD	AVX	SSE2
PSLLQ VPSLLQ	AVX	SSE2
PSLLW VPSLLW	AVX	SSE2
PSRAD VPSRAD	AVX	SSE2
PSRAW VPSRAW	AVX	SSE2
PSRLD VPSRLD	AVX	SSE2
PSRLQ VPSRLQ	AVX	SSE2
PSRLW VPSRLW	AVX	SSE2
Class 4G — AVX, SSE, 16/32-byte nonaligned, VEX.W = 1, VEX.vvvv != 1111b		
VTESTPD	AVX	—
VTESTPS	AVX	—
Class 5 — AVX, SSE, <16-byte		
RCPSS VRCPSS	AVX	SSE
RSQRTSS VRSQRTSS	AVX	SSE

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 5A — AVX, SSE, <16-byte, VEX.L = 1		
INSERTPS VINSERTPS	AVX	SSE4.1
PMOVZXBD VPMOVZXBD	AVX	SSE4.1
PMOVZXBQ VPMOVZXBQ	AVX	SSE4.1
PMOVZXBW VPMOVZXBW	AVX	SSE4.1
PMOVZXDQ VPMOVZXDQ	AVX	SSE4.1
PMOVZXWD VPMOVZXWD	AVX	SSE4.1
PMOVZXWQ VPMOVZXWQ	AVX	SSE4.1
Class 5B — AVX, SSE, <16-byte, VEX.vvvv != 1111b		
CVTDQ2PD VCVTDQ2PD	AVX	SSE2
MOVDDUP VMOVDDUP	AVX	SSE3
Class 5C — AVX, SSE, <16-byte, VEX.vvvv != 1111b, VEX.L = 1		
PINSRB VPINSRB	AVX	SSE4.1
PINSRD VPINSRD	AVX	SSE4.1
PINSRQ VPINSRQ	AVX	SSE4.1
PINSRW VPINSRW	AVX	SSE
PMOVSXBD VPMOVSXBD	AVX	SSE4.1
PMOVSXBQ VPMOVSXBQ	AVX	SSE4.1
PMOVSXBW VPMOVSXBW	AVX	SSE4.1
PMOVSXDQ VPMOVSXDQ	AVX	SSE4.1
PMOVSXWD VPMOVSXWD	AVX	SSE4.1
PMOVSXWQ VPMOVSXWQ	AVX	SSE4.1
Class 5C-1 — AVX, SSE, <16-byte, write to RO, VEX.vvvv != 1111b, VEX.L = 1		
EXTRACTPS VEXTRACTPS	AVX	SSE4.1
MOVD VMOVD	AVX	SSE2
MOVQ VMOVQ	AVX	SSE2
PEXTRB VPEXTRB	AVX	SSE4.1
PEXTRD VPEXTRD	AVX	SSE4.1
PEXTRQ VPEXTRQ	AVX	SSE4.1
PEXTRW VPEXTRW	AVX	SSE4.1
Class 5D — AVX, SSE, <16-byte, write to RO, VEX.vvvv != 1111b (variant)		
MOVSD VMOVSD	AVX	SSE2
MOVSS VMOVSS	AVX	SSE
Class 5E — AVX, SSE, <16-byte, write to RO, VEX.vvvv != 1111b (variant), VEX.L = 1		
MOVHPD VMOVHPD	AVX	SSE2
MOVHPS VMOVHPS	AVX	SSE
MOVLPD VMOVLPD	AVX	SSE2
MOVLPS VMOVLPS	AVX	SSE
Class 6 — AVX, mixed memory argument[1]		
Class 6A — AVX, mixed memory argument, VEX.W = 1		
VBROADCASTSS	AVX	—

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 6A-1 — AVX, mixed memory argument, write to RO, VEX.W = 1		
VMASKMOVPD	AVX	—
VMASKMOVPS	AVX	—
Class 6B — AVX, mixed memory argument, VEX.W = 1, VEX.L=0		
VINSERTF128	AVX	—
VPERM2F128	AVX	—
Class 6B-1 — AVX, mixed memory argument, write to RO, VEX.W = 1, VEX.L=0		
VEXTRACTF128	AVX	—
Class 6C — AVX, mixed memory argument, VEX.W = 1, VEX.vvvv != 1111b, VEX.L=0		
VBROADCASTSD	AVX	—
VBROADCASTF128	AVX	—
Class 6D — AVX, mixed memory argument, VEX.W = 1, VEX.vvvv != 1111b		
VBROADCASTSS	AVX	—
Class 6E — AVX, mixed memory argument, VEX.W = 1, VEX.vvvv != 1111b (variant)		
VPERMILPD	AVX	—
VPERMILPS	AVX	—
Class 7 — AVX, SSE, no memory argument[1]		
Class 7A — AVX, SSE, no memory argument, VEX.L = 1		
MOVHLPS VMOVHLPS	AVX	SSE
MOVLHPS VMOVLHPS	AVX	SSE
PSLLDQ VPSLLDQ	AVX	SSE2
PSRLDQ VPSRLDQ	AVX	SSE2
Class 7B — AVX, SSE, no memory argument, VEX.vvvv != 1111b		
MOVMSKPD VMOVMSKPD	AVX	SSE2
MOVMSKPS VMOVMSKPS	AVX	SSE
Class 7C — AVX, SSE, no memory argument, VEX.vvvv != 1111b, VEX.L = 1		
PMOVMSKB VPMOVMSKB	AVX	SSE2
Class 8 — AVX, no memory argument, VEX.W = 1, VEX.vvvv != 1111b		
VZEROALL	AVX	—
VZEROUPPER	AVX	—
Class 9 — SSE, AVX, 4-byte argument, write to RO, VEX.vvvv != 1111b, VEX.L = 1		
STMXCSR VSTMXCSR	AVX	SSE
Class 9A — SSE, AVX, 4-byte argument, reserved MBZ=1 write, VEX.vvvv != 1111b, VEX.L = 1		
LDMXCSR VLDMXCSR	AVX	SSE

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 10 — XOP Base		
VPCMOV	XOP	
VPCOMB	XOP	—
VPCOMD	XOP	—
VPCOMQ	XOP	—
VPCOMUB	XOP	—
VPCOMUD	XOP	—
VPCOMUQ	XOP	—
VPCOMUW	XOP	—
VPCOMW	XOP	—
VPERMIL2PS	XOP	—
VPERMIL2PD	XOP	—
Class 10A — XOP Base, XOP.L = 1		
VPPERM	XOP	—
VPSHAB	XOP	—
VPSHAD	XOP	—
VPSHAQ	XOP	—
VPSHAW	XOP	—
VPSHLB	XOP	—
VPSHLD	XOP	—
VPSHLQ	XOP	—
VPSHLW	XOP	—
Class 10B — XOP Base, XOP.W = 1, XOP.L = 1		
VPMACSDD	XOP	—
VPMACSDQH	XOP	—
VPMACSDQL	XOP	—
VPMACSSDD	XOP	—
VPMACSSDQH	XOP	—
VPMACSSDQL	XOP	—
VPMACSSWD	XOP	—
VPMACSSWW	XOP	—
VPMACSWD	XOP	—
VPMACSWW	XOP	—
VPMADCSSWD	XOP	—
VPMADCSWD	XOP	—
Class 10C — XOP Base, XOP.W = 1, XOP.vvvv != 1111b, XOP.L = 1		
VPHADDBD	XOP	—
VPHADDBQ	XOP	—
VPHADDBW	XOP	—
VPHADDD	XOP	—
VPHADDDQ	XOP	—
VPHADDUBD	XOP	—

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
VPHADDUBQ	XOP	—
VPHADDUBW	XOP	—
VPHADDUDQ	XOP	—
VPHADDUWD	XOP	—
VPHADDUWQ	XOP	—
VPHADDWD	XOP	—
VPHADDWQ	XOP	—
VPHSUBBW	XOP	—
VPHSUBDQ	XOP	—
VPHSUBWD	XOP	—
Class 10D — XOP Base, XOP.W = 1, XOP.vvvv != 1111b, SIMD 110011		
VFRCZPD	XOP	—
VFRCZPS	XOP	—
VFRCZSD	XOP	—
VFRCZSS	XOP	—
Class 10E — XOP Base, XOP.vvvv != 1111b (variant), XOP.L = 1		
VPROTB	XOP	—
VPROTD	XOP	—
VPROTQ	XOP	—
VPROTW	XOP	—
Class 11		
Reserved for future use.		
Class 12 — FMA4, 16/32-byte nonaligned, SIMD 111011		
VFMADDPD	FMA4	—
VFMADDPS	FMA4	—
VFMADDSUBPD	FMA4	—
VFMADDSUBPS	FMA4	—
VFMSUBADDPD	FMA4	—
VFMSUBADDPS	FMA4	—
VFMSUBPD	FMA4	—
VFMSUBPS	FMA4	—
VFNMADDPD	FMA4	—
VFNMADDPS	FMA4	—
VFNMSUBPD	FMA4	—
VFNMSUBPS	FMA4	—

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 13 — FMA4, <16-byte, SIMD 111011		
VFMADDSD	FMA4	—
VFMADDSS	FMA4	—
VFMSUBSD	FMA4	—
VFMSUBSS	FMA4	—
VFNMADDSD	FMA4	—
VFNMADDSS	FMA4	—
VFNMSUBSD	FMA4	—
VFNMSUBSS	FMA4	—
Unique Cases		
XGETBV	—	—
XRSTOR	—	—
XSAVE	—	—
XSETBV	—	—

1.This base class does not apply to any instruction.It is shown for reference only.

Class 1 — AVX, SSE, 16/32-byte aligned, write to RO, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on 16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 1A — AVX, SSE, 16/32-byte aligned, write to RO, VEX.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L field = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
	S	S	X	Write to a read-only data segment.
			A	VEX256: Memory operand not 32-byte aligned. VEX128: Memory operand not 16-byte aligned.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2 — AVX, SSE, 16/32-byte nonaligned, SIMD 111111

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2-1 — AVX, SSE, 16/32-byte nonaligned, SIMD 111011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2-2 — AVX, SSE, 16/32-byte nonaligned, SIMD 000011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2-3 — AVX, SSE, 16/32-byte nonaligned, SIMD 100001

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2A — AVX, SSE, 16/32-byte nonaligned, SIMD 111111, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2A-1 — AVX, SSE, 16/32-byte nonaligned, SIMD 111011, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2B — AVX, SSE, 16/32-byte nonaligned, SIMD 111111, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2B-1 — AVX, SSE, 16/32-byte nonaligned, SIMD 100000, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2B-2 — AVX, SSE, 16/32-byte nonaligned, SIMD 100001, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

Class 2B-3 — AVX, SSE, 16/32-byte nonaligned, SIMD 111011, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 2B-4 — AVX, SSE, 16/32-byte nonaligned, SIMD 100011, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Memory operand not aligned on16-byte boundary while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3 — AVX, SSE, <16-byte, SIMD

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3-1 — AVX, SSE, <16-byte, SIMD 111011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3-2 — AVX, SSE, <16-byte, SIMD 000011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3-3 — AVX, SSE, <16-byte, SIMD 100000

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3-4 — AVX, SSE, <16-byte, SIMD 100001

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3-5 — AVX, SSE, <16-byte, SIMD 100011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.

X — AVX and SSE exception
A — AVX exception
S — SSE exception

Class 3A — AVX, SSE, <16-byte, SIMD 111111, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
Division by zero, ZE	S	S	X	Division of finite dividend by zero-value divisor.
Overflow, OE	S	S	X	Rounded result too large to fit into the format of the destination operand.
Underflow, UE	S	S	X	Rounded result too small to fit into the format of the destination operand.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3A-1 — AVX, SSE, <16-byte, SIMD 000011, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Denormalized operand, DE	S	S	X	A source operand was a denormal value.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 3A-2 — AVX, SSE, <16-byte, SIMD 100001, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE	S	S	X	A source operand was an SNaN value.
	S	S	X	Undefined operation.
Precision, PE	S	S	X	A result could not be represented exactly in the destination format.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4 — AVX, SSE, 16/32-byte nonaligned

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4A — AVX, SSE, 16/32-byte nonaligned, VEX.W = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4B — AVX, SSE, 16/32-byte nonaligned, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4C — AVX, SSE, 16/32-byte nonaligned, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4C-1 — AVX, SSE, 16/32-byte nonaligned, write to RO, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4D — AVX, SSE, 16/32-byte nonaligned, VEX.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4E — AVX, SSE, 16/32-byte nonaligned, VEX.W = 1, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4F — AVX, SSE, 16/32-byte nonaligned, VEX.L = 1 (no memory argument for SSE)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 4G — AVX, SSE, 16/32-byte nonaligned, VEX.W = 1, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5 — AVX, SSE, <16-byte

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5A — AVX, SSE, <16-byte, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5B — AVX, SSE, <16-byte, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5C — AVX, SSE, <16-byte, VEX.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5C-1 — AVX, SSE, <16-byte, write to RO, VEX.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5D — AVX, SSE, <16-byte, write to RO, VEX.vvvv != 1111b (variant)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination enoding only).
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 5E — AVX, SSE, <16-byte, write to RO, VEX.vvvv != 1111b (variant), VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b (for memory destination encoding only).
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 6 — AVX, mixed memory argument

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6A — AVX, mixed memory argument, VEX.W = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6A-1 — AVX, mixed memory argument, write to RO, VEX.W = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
	S	S	X	Write to a read-only data segment.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6B — AVX, mixed memory argument, VEX.W = 1, VEX.L = 0

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6B-1 — AVX, mixed memory argument, write to RO, VEX.W = 1, VEX.L = 0

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Write to a read-only data segment.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6C — AVX, mixed memory argument, VEX.W = 1, VEX.L = 0

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6D — AVX, mixed memory argument, VEX.W = 1, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 6E — AVX, mixed memory argument, VEX.W = 1, VEX.vvvv != 1111b (variant)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b (for versions with immediate byte operand only).
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
Stack, #SS			A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Page fault, #PF			A	Instruction execution caused a page fault.
Alignment check, #AC			A	4 or 8-byte unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
A — AVX exception.				

Class 7 — AVX, SSE, no memory argument

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 7A — AVX, SSE, no memory argument, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 7B — AVX, SSE, no memory argument, VEX.vvvv != 1111b

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 7C — AVX, SSE, no memory argument, VEX.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv field ! = 1111b.
			A	VEX.L field = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 8 — AVX, no memory argument, VEX.vvvv != 1111b, VEX.W = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
Device not available, #NM			A	CR0.TS = 1.
A — AVX exception.				

Class 9 — AVX, 4-byte argument, write to RO, vex.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	X	Write to a read-only data segment.
	S	S	S	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception *A — AVX exception* *S — SSE exception*				

Class 9A — AVX, 4-byte argument, reserved MBZ = 1, vex.vvvv != 1111b, VEX.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	XfeatureEnabledMask[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
	S	S	S	Null data segment used to reference memory.
	S	S	X	Attempt to load non-zero values into reserved MXCSR bits
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC		S	X	Unaligned memory reference with alignment checking enabled.
X — AVX and SSE exception A — AVX exception S — SSE exception				

Class 10 — XOP Base

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

Class 10A — XOP Base, XOP.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

Class 10B — XOP Base, XOP.W = 1, XOP.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

Class 10C — XOP Base, XOP.W = 1, XOP.vvvv != 1111b, XOP.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			A	XOP.vvvv ! = 1111b.
			X	XOP.L = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

Class 10D — XOP Base, SIMD 11011, XOP.vvvv != 1111b, XOP.W = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.W = 1.
			X	XOP.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF	S	S	X	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			X	A source operand was an SNaN value.
			X	Undefined operation.
Denormalized operand, DE			X	A source operand was a denormal value.
Underflow, UE			X	Rounded result too small to fit into the format of the destination operand.
Precision, PE			X	A result could not be represented exactly in the destination format.
X — XOP exception				

Class 10E — XOP Base, XOP.vvvv != 1111b (variant), XOP.L = 1

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XfeatureEnabledMask[2:1] ! = 11b.
			X	XOP.vvvv ! = 1111b (for immediate operand variant only)
			X	XOP.L field = 1.
			X	REX, F2, F3, or 66 prefix preceding XOP prefix.
			X	Lock prefix (F0h) preceding opcode.
Device not available, #NM			X	CR0.TS = 1.
Stack, #SS			X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory.
Page fault, #PF			X	Instruction execution caused a page fault.
Alignment check, #AC			X	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
X — XOP exception				

Class 11 — Reserved for future use

Class 12 — FMA4, 8/16-byte nonaligned, SIMD 111011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled and MXCSR.MM = 1.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

Class 13 — FMA4, <16-byte, SIMD 111011

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA4 instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XfeatureEnabledMask[2:1] ! = 11b.
			F	REX, F2, F3, or 66 prefix preceding VEX prefix.
			F	Lock prefix (F0h) preceding opcode.
			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0, see *SIMD Floating-Point Exceptions* below for details.
Device not available, #NM			F	CR0.TS = 1.
Stack, #SS			F	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP			F	Memory address exceeding data segment limit or non-canonical.
			F	Null data segment used to reference memory.
Page fault, #PF			F	Instruction execution caused a page fault.
Alignment check, #AC			F	Unaligned memory reference with alignment checking enabled.
SIMD floating-point, #XF			F	Unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 1, see *SIMD Floating-Point Exceptions* below for details.
SIMD Floating-Point Exceptions				
Invalid operation, IE			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized operand, DE			F	A source operand was a denormal value.
Overflow, OE			F	Rounded result too large to fit into the format of the destination operand.
Underflow, UE			F	Rounded result too small to fit into the format of the destination operand.
Precision, PE			F	A result could not be represented exactly in the destination format.
F — FMA4 exception				

XGETBV

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	Lock prefix (F0h) preceding opcode.
General protection, #GP	X	X	X	ECX specifies a reserved or unimplemented XCR address.
X — exception generated				

XRSTOR

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	CR4.OSFXSR = 0.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	X	X	X	CR0.TS = 1.
Stack, #SS	X	X	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	X	X	X	Memory address exceeding data segment limit or non-canonical.
	X	X	X	Null data segment used to reference memory.
	X	X	X	Memory operand not aligned on 64-byte boundary.
	X	X	X	Any must be zero (MBZ) bits in the save area were set.
	X	X	X	Attempt to set reserved bits in MXCSR.
Page fault, #PF	X	X	X	Instruction execution caused a page fault.
X — exception generated				

XSAVE

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	CR4.OSFXSR = 0.
	X	X	X	Lock prefix (F0h) preceding opcode.
Device not available, #NM	X	X	X	CR0.TS = 1.
Stack, #SS	X	X	X	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	X	X	X	Memory address exceeding data segment limit or non-canonical.
	X	X	X	Null data segment used to reference memory.
	X	X	X	Memory operand not aligned on 64-byte boundary.
	X	X	X	Attempt to write read-only memory.
Page fault, #PF	X	X	X	Instruction execution caused a page fault.
X — exception generated				

XSETBV

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X	X	CR4.OSFXSR = 0.
	X	X	X	Lock prefix (F0h) preceding opcode.
General protection, #GP	X	X	X	CPL != 0.
	X	X	X	ECX specifies a reserved or unimplemented XCR address.
	X	X	X	Any must be zero (MBZ) bits in the save area were set.
	X	X	X	Writing 0 to XCR0.

X — exception generated

Note:
In virtual mode, only #UD for Instruction not supported and #GP for CPL != 0 are supported.

Index