AMD

AMD64 Technology

AMD64 Architecture Programmer's Manual

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

Publication No.	Revision	Date
26568	3.22	May 2018

Advanced Micro Devices

© 2013 – 2018 Advanced Micro Devices Inc. All rights reserved.

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, and combinations thereof, and 3DNow! are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

MMX is a trademark and Pentium is a registered trademark of Intel Corporation.

Contents

Figures

Tables

Revision History

Date	Revision	Description
May 2018	3.22	Update Packed String Compare Algorithm Fixed a number of erroneous references to double precision that should be single precision Separate out MOVQ from MOVD
December 2017	3.21	Clarifications to XGETBV, XRSTOR, XRSTORS, XSAVE, XSAVEC, XSAVEOPT, XSAVES, and XSETBV instructions.
March 2017	3.20	Corrections to ROUNDPD, VROUNDPD, ROUNDPS, VROUNDPS, ROUNDSD, VROUNDSD, ROUNDSS, VROUNDSS, VPERMD, VPERMPD, VPERMPS, VPERMQ, VTESTPD, VTESTPS, XGETBV, XSETBV, XSAVE, and AVX instruction descriptions. Added SHA1RNDS4, SHA1NEXTE, SHA1MSG1, SHA1MSG2, SHA256RNDS2, SHA256MSG1, SHA256MSG2, XRSTOR, XRSTORS and XSAVEC instructions.
June 2015	3.19	Corrections to the MOVLPD, PHSUBW, PHSUBSW instruction descriptions.
October 2013	3.18	Added AVX2 Instructions. Added “Instruction Support” subsection to each instruction reference page that lists CPUID feature bit information in a table.
May 2013	3.17	Removed all references to the CPUID specification which has been superseded by Volume 3, Appendix E, "Obtaining Processor Information Via the CPUID Instruction." Corrected exceptions table for the explicitly-aligned load/store instructions. General protection exception does not depend on state of MXCSR.MM bit.
September 2012	3.16	Corrected REX.W bit encoding for the MOVD instruction. (See page 186.) Corrected L bit encoding for the VMOVQ (D6h opcode) instruction. (See page 222.) Corrected statement about zero extension for third encoding (11h opcode) of MOVSS instruction. (See page 230.)
March 2012	3.15	Corrected instruction encoding for VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMUW, and VPHSUBDQ instructions. Other minor corrections.

Date	Revision	Description
December 2011	3.14	Reworked Section 1.5, "String Compare Instructions" on page 10. Revised descriptions of the string compare instructions in instruction reference. Moved AES overview to Appendix A. Clarified trap and exception behavior for elements not selected for writing. See MASKMOVDQU VMASKMOVDQU on page 160. Additional minor corrections and clarifications.
September 2011	3.13	Moved discussion of extended instruction encoding; VEX and XOP prefixes to Volume 3. Added FMA instructions. Described on the corresponding FMA4 reference page. Moved BMI and TBM instructions to Volume 3. Added XSAVEOPT instruction. Corrected descriptions of VSQRTSD and VSQRTSS.
May 2011	3.12	Added F16C, BMI, and TBM instructions.
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, INSERTQ, MOVNTSD, and MOVNTSS. Added misaligned exception mask (MXCSR.MM) information. Added imm8 values with corresponding mnemonics to (V)CMPPD, (V)CMPPS, (V)CMPSD, and (V)CMPSS. 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.

Date	Revision	Description
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, instruction encoding, functions, affected flags, and possible exceptions.

The AMD64 instruction set is divided into five subsets:

- General-purpose instructions
- System instructions
- Streaming SIMD Extensions (includes 128-bit and 256-bit media instructions)
- 64-bit media instructions (MMX™)
- x87 floating-point instructions

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

This volume describes the Streaming SIMD Extensions (SSE) instruction set which includes 128-bit and 256-bit media instructions. SSE includes both legacy and extended forms. 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.

Conventions and Definitions

The section which follows, **Notational Conventions,** describes notational conventions used in this volume. The next section, **Definitions,** lists a number of terms used in this volume along with their technical definitions. Some of these definitions assume knowledge of the legacy x86 architecture. See "Related Documents" on page xl for further information about the legacy x86 architecture. Finally, the **Registers** section lists the registers which are a part of the system programming model.

Notational Conventions

Section 1.1, "Syntax and Notation" on page 2 describes notation relating specifically to instruction encoding.

#GP(0)

An instruction exception—in this example, a general-protection exception with error code of 0.

1011b

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

F0EA_0B40h

A hexadecimal value, in this example a 32-bit value. Underscore characters may be used to improve readability.

128

Numbers without an alpha suffix are decimal unless the context indicates otherwise.

7:4

A bit range, from bit 7 to 4, inclusive. The high-order bit is shown first. Commas may be inserted to indicate gaps.

#GP(0)

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

CPUID Fn*XXXX_XXXX_RRR*[*FieldName*]

Support for optional features or the value of an implementation-specific parameter of a processor can be discovered by executing the CPUID instruction on that processor. To obtain this value, software must execute the CPUID instruction with the function code *XXXX_XXXX*h in EAX and then examine the field *FieldName* returned in register *RRR*. If the "*_RRR*" notation is followed by "_x*YYY*", register ECX must be set to the value *YYY*h before executing CPUID. When *FieldName* is not given, the entire contents of register *RRR* contains the desired value. When determining optional feature support, if the bit identified by *FieldName* is set to a one, the feature is supported on that processor.

CR0–CR4

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

CR4[OSXSAVE], CR4.OSXSAVE

The OSXSAVE bit of the CR4 register.

CR0[PE] = 1, CR0.PE = 1

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

EFER[LME] = 0, EFER.LME = 0

The LME field of the EFER register is cleared (contains a value of 0).

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.

RFLAGS[13:12]

A field within a register identified by its bit range. In this example, corresponding to the IOPL field.

Definitions

128-bit media instruction

Instructions that operate on the various 128-bit vector data types. Supported within both the legacy SSE and extended SSE instruction sets.

256-bit media instruction

Instructions that operate on the various 256-bit vector data types. Supported within the extended SSE instruction set.

64-bit media instructions

Instructions that operate on the 64-bit vector data types. These are primarily a combination of MMX and 3DNow!™ instruction sets and their extensions, with some additional instructions from the SSE1 and SSE2 instruction sets.

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*.

AES

Advance Encryption Standard (AES) algorithm acceleration instructions; part of Streaming SIMD Extensions (SSE).

ASID

Address space identifier.

AVX

Extension of the SSE instruction set supporting 256-bit vector (packed) operands. See Streaming SIMD Extensions.

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*.

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 all exceptions except SSE floating-point exceptions and x87 floating-point exceptions, control is transferred to a handler (or service routine) for that exception as defined by the exception's 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 SSE instructions

Enhanced set of SIMD instructions supporting 256-bit vector data types and allowing the specification of up to four operands. A subset of the *Streaming SIMD Extensions (SSE)*. Includes the AVX, FMA, FMA4, and XOP instructions. Compare *legacy SSE*.

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."

FMA4

Fused Multiply Add, four operand. Part of the extended SSE instruction set.

FMA

Fused Multiply Add. Part of the extended SSE instruction set.

GDT

Global descriptor table.

GIF

Global interrupt flag.

IDT

Interrupt descriptor table.

IGN

Ignored. Value written is ignored by hardware. Value returned on a read is indeterminate. See *reserved*.

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 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*.

legacy SSE instructions

All Streaming SIMD Extensions instructions prior to AVX, XOP, and FMA4. Legacy SSE instructions primarily utilize operands held in XMM registers. The legacy SSE instructions include the original Streaming SIMD Extensions (SSE1) and the subsequent extensions SSE2, SSE3, SSSE3, SSE4, SSE4A, SSE4.1, and SSE4.2. See *Streaming SIMD instructions*.

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. See *reserved*.

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. Value returned on a read is always zero (0) regardless of what was previously written. See *reserved*.

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 marked as reserved may be used at some future time.

To preserve compatibility with future processors, reserved fields require special handling when read or written by software. Software must not depend on the state of a reserved field (unless qualified as RAZ), nor upon the ability of such fields to return a previously written state.

If a field is marked reserved without qualification, software must not change the state of that field; it must reload that field with the same value returned from a prior read.

Reserved fields may be qualified as IGN, MBZ, RAZ, or SBZ (see definitions).

REX

A legacy instruction modifier 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.

SBZ

Should be zero. An attempt by software to set an SBZ bit to 1 results in undefined behavior. See *reserved*.

scalar

An atomic value existing independently of any specification of location, direction, etc., as opposed to vectors.

set

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

SIMD

Single instruction, multiple data. See *vector*.

Streaming SIMD Extensions (SSE)

Instructions that operate on scalar or vector (packed) integer and floating point numbers. The SSE instruction set comprises the legacy SSE and extended SSE instruction sets.

SSE1

Original SSE instruction set. Includes instructions that operate on vector operands in both the MMX and the XMM registers.

SSE2

Extensions to the SSE instruction set.

SSE3

Further extensions to the SSE instruction set.

SSSE3

Further extensions to the SSE instruction set.

SSE4.1

Further extensions to the SSE instruction set.

SSE4.2

Further extensions to the SSE instruction set.

SSE4A

A minor extension to the SSE instruction set adding the instructions EXTRQ, INSERTQ, MOVNTSS, and MOVNTSD.

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*.

VEX prefix

Extended instruction encoding escape prefix. Introduces a two- or three-byte encoding escape sequence used in the encoding of AVX instructions. Opens a new extended instruction encoding space. Fields select the opcode map and allow the specification of operand vector length and an additional operand register. See *XOP prefix*.

virtual-8086 mode

A submode of *legacy mode*.

VMCB

Virtual machine control block.

VMM

Virtual machine monitor.

word

Two bytes, or 16 bits.

x86

See *legacy x86*.

XOP instructions

Part of the extended SSE instruction set using the XOP prefix. See Streaming SIMD Extensions.

XOP prefix

Extended instruction encoding escape prefix. Introduces a three-byte escape sequence used in the encoding of XOP instructions. Opens a new extended instruction encoding space distinct from the VEX opcode space. Fields select the opcode map and allow the specification of operand vector length and an additional operand register. See *VEX prefix*.

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.

CR*n*

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.

rAX–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.

YMM/XMM

Set of sixteen (eight accessible in legacy and compatibility modes) 256-bit wide registers that hold scalar and vector operands used by the SSE instructions.

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

Processors capable of performing the same mathematical operation simultaneously on multiple data streams are classified as *single-instruction, multiple-data (SIMD)*. Instructions that utilize this hardware capability are called SIMD instructions.

Software can utilize SIMD instructions to drastically increase the performance of media applications which typically employ algorithms that perform the same mathematical operation on a set of values in parallel. The original SIMD instruction set was called MMX and operated on 64-bit wide vectors of integer and floating-point elements. Subsequently a new SIMD instruction set called the Streaming SIMD Extensions (SSE) was added to the architecture.

The SSE instruction set defines a new programming model with its own array of vector data registers (YMM/XMM registers) and a control and status register (MXCSR). Most SSE instructions pull their operands from one or more YMM/XMM registers and store results in a YMM/XMM register, although some instructions use a GPR as either a source or destination. Most instructions allow one operand to be loaded from memory. The set includes instructions to load a YMM/XMM register from memory (aligned or unaligned) and store the contents of a YMM/XMM register.

An overview of the SSE instruction set is provided in Volume 1, Chapter 4.

This volume provides detailed descriptions of each instruction within the SSE instruction set. The SSE instruction set comprises the *legacy* SSE instructions and the *extended* SSE instructions.

Legacy SSE instructions comprise the following subsets:

- The original Streaming SIMD Extensions (herein referred to as SSE1)
- SSE2
- SSE3
- SSSE3
- SSE4.1
- SSE4.2
- SSE4A
- Advanced Encryption Standard (AES)

Extended SSE instructions comprise the following subsets:

- AVX
- AVX2
- FMA
- FMA4
- XOP

Legacy SSE architecture supports operations involving 128-bit vectors and defines the base programming model including the SSE registers, the Media eXtension Control and Status Register (MXCSR), and the instruction exception behavior.

The Streaming SIMD Extensions (SSE) instruction set is extended to include the AVX, FMA, FMA4, and XOP instruction sets. The AVX instruction set provides an extended form for most legacy SSE instructions and several new instructions. Extensions include providing for the specification of a unique destination register for operations with two or more source operands and support for 256-bit wide vectors. Some AVX instructions also provide enhanced functionality compared to their legacy counterparts.

A significant feature of the extended SSE instruction set architecture is the doubling of the width of the XMM registers. These registers are referred to as the YMM registers. The XMM registers overlay the lower octword (128 bits) of the YMM registers. Registers YMM/XMM0–7 are accessible in legacy and compatibility mode. Registers YMM/XMM8–15 are available in 64-bit mode (a subset of long mode). VEX/XOP instruction prefixes allow instruction encodings to address the additional registers.

The SSE instructions can be used in processor legacy mode or long (64-bit) mode. CPUID Fn8000_0001_EDX[LM] indicates the availability of long mode.

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

- Access to an additional eight YMM/XMM registers for a total of 16
- Access to an additional eight 64-bit general-purpose registers for a total of 16
- Access to the 64-bit virtual address space and the RIP-relative addressing mode

Hardware support for each of the subsets of SSE instructions listed above is indicated by CPUID feature flags. Refer to Volume 3, Appendix D, “Instruction Subsets and CPUID Feature Flags,” for a complete list of instruction-related feature flags. The CPUID feature flags that pertain to each instruction are also given in the instruction descriptions below. For information on using the CPUID instruction, see the instruction description in Volume 3.

Chapter 2, “Instruction Reference” contains detailed descriptions of each instruction, organized in alphabetic order by mnemonic. For those legacy SSE instructions that have an AVX form, the extended form of the instruction is described together with the legacy instruction in one entry. For these instructions, the instruction reference page is located based on the instruction mnemonic of the legacy SSE and not the extended (AVX) form. Those AVX instructions without a legacy form are listed in order by their AVX mnemonic. The mnemonic for all extended SSE instructions including the FMA and XOP instructions begin with the letter *V*.

1.1 Syntax and Notation

The descriptive synopsis of opcode syntax for legacy SSE instructions follows the conventions described in *Volume 3: General Purpose and System Instructions*. See Chapter 2 and the section entitled “Notation.”

For general information on the programming model and overview descriptions of the SSE instruction set, see:

- "Streaming SIMD Extensions Media and Scientific Programming" in Volume 1.
- "Instruction Encoding" in Volume 3
- "Summary of Registers and Data Types" 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 sequence fields. Figure 1-1 shows the descriptive synopsis of a typical XOP instruction. The synopsis of VEX-encoded instructions have the same format, differing only in regard to the instruction encoding escape prefix, that is, VEX instead of XOP.

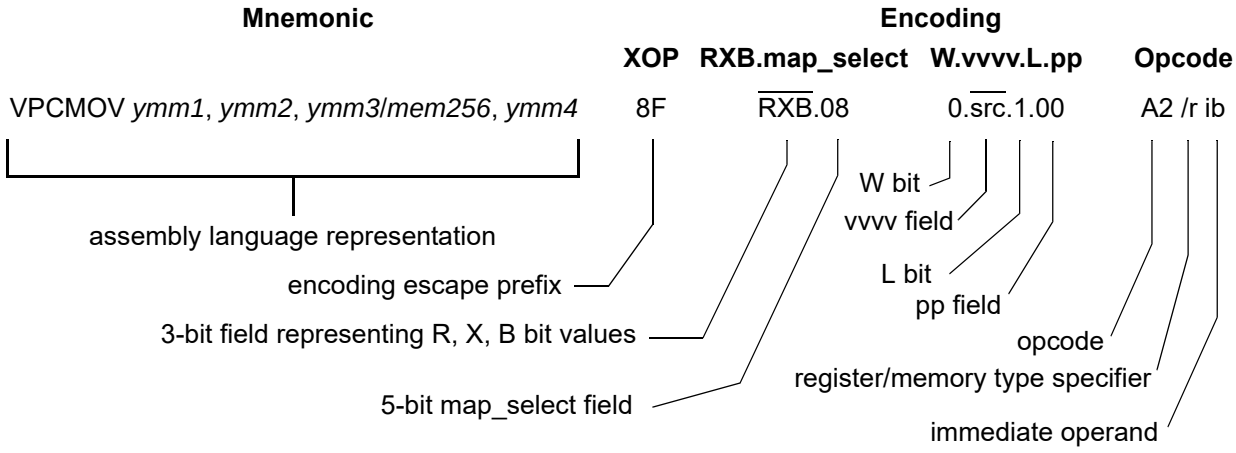


Figure 1-1. Typical Descriptive Synopsis - Extended SSE Instructions

1.2 Extended Instruction Encoding

The legacy SSE instructions are encoded using the legacy encoding syntax and the extended instructions are encoded using an enhanced encoding syntax which is compatible with the legacy syntax. Both are described in detail in Chapter 1 of Volume 3.

As described in Volume 3, the extended instruction encoding syntax utilizes multi-byte escape sequences to both select alternate opcode maps as well as augment the encoding of the instruction. Multi-byte escape sequences are introduced by one of the two VEX prefixes or the XOP prefix.

The AVX and AVX2 instructions utilize either the two-byte (introduced by the VEX C5h prefix) or the three-byte (introduced by the VEX C4h prefix) encoding escape sequence. XOP instructions are encoded using a three-byte encoding escape sequence introduced by the XOP prefix (except for the XOP instructions VPERMIL2PD and VPERMIL2PS which are encoded using the VEX prefix). The XOP prefix is 8Fh. The three-byte encoding escape sequences utilize the map_select field of the second byte to select the opcode map used to interpret the opcode byte.

The two-byte VEX prefix sequence implicitly selects the secondary (“two-byte”) opcode map.

1.2.1 Immediate Byte Usage Unique to the SSE instructions

An *immediate* is a value, typically an operand, explicitly provided within the instruction encoding. 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)*.

A one-byte immediate is generally shown in the instruction synopsis as “ib” suffix. For extended SSE instructions with four source operands, the suffix “is4” is used to indicate the presence of the immediate byte used to select the fourth source operand.

The VPERMIL2PD and VPERMIL2PS instructions utilize a fifth 2-bit operand which is encoded along with the fourth register select index in an immediate byte. For this special case the immediate byte will be shown in the instruction synopsis as “is5”.

1.2.2 Instruction Format Examples

The following sections provide examples of two-, three-, and four-operand extended instructions. These instructions generally perform *nondestructive-source operations*, meaning that the result of the operation is written to a separately specified destination register rather than overwriting one of the source operands. This preserves the contents of the source registers. Most legacy SSE instructions perform *destructive-source operations*, in which a single register is both source and destination, so source content is lost.

1.2.2.1 XMM Register Destinations

The following general properties apply to YMM/XMM 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.

1.2.2.2 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.map_select	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 a YMM/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.

1.2.2.3 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.map_select	W.vvvv.L.pp	Opcode
VPROTB *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{src}$.0.00	90 /r
VPROTB *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.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.r/m, 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.r/m.

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

Table 1-1. 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, $\overline{src}$ 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.

1.2.2.4 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.map_select	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 1-1 summarizes the effect of the VEX.W bit on operand selection.

Table 1-2. 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 VSIB Addressing

Specific AVX2 instructions utilize a vectorized form of indexed register-indirect addressing called vector SIB (VSIB) addressing. In contrast to the standard indexed register-indirect address mode, which generates a single effective address to access a single memory operand, VSIB addressing generates an array of effective addresses which is used to access data from multiple memory locations in a single operation.

VSIB addressing is encoded using three or six bytes following the opcode byte, augmented by the X and B bits from the VEX prefix. The first byte is the ModRM byte with the standard mod, reg, and r/m fields (although allowed values for the mod and r/m fields are restricted). The second is the VSIB byte which replaces the SIB byte in the encoding. The VSIB byte specifies a GPR which serves as a base address register and an XMM/YMM register that contains a packed array of index values. The two-bit scale field specifies a common scaling factor to be applied to all of the index values. A constant displacement value is encoded in the one or four bytes that follow the VSIB byte.

Figure 1-2 shows the format of the VSIB byte.

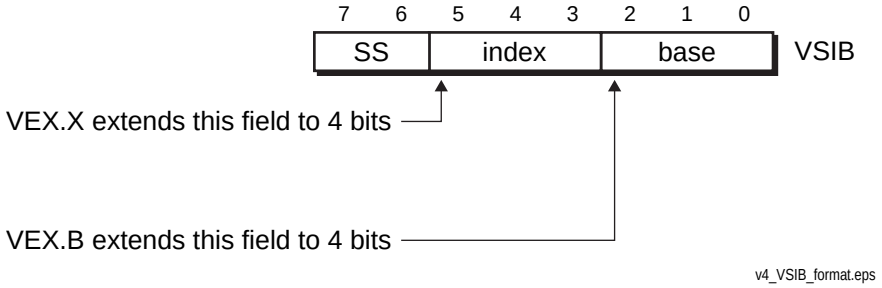


Figure 1-2. VSIB Byte Format

VSIB.SS (Bits [7:6]). The SS field is used to specify the scale factor to be used in the computation of each of the effective addresses. The scale factor *scale* is equal to 2^{SS} (two raised to power of the value of the SS field). Therefore, if SS = 00b, *scale* = 1; if SS = 01b, *scale* = 2; if SS = 10b, *scale* = 4; and if SS = 11b, *scale* = 8.

VSIB.index (Bits [5:3]). This field is concatenated with the complement of the VEX.X bit ({$\overline{\text{X}}$, index}) to specify the YMM/XMM register that contains the packed array of index values *index*[*i*] to be used in the computation of the array of effective addresses *effective address*[*i*].

VSIB.base (Bits [5:3]). This field is concatenated with the complement of the VEX.B bit ({$\overline{\text{B}}$, base}) to specify the general-purpose register (base GPR) that contains the base address *base* to be used in the computation of each of the effective addresses.

1.3.1 Effective Address Array Computation

Each element *i* of the effective address array is computed using the formula:

effective address[*i*] = *scale* * *index*[*i*] + *base* + *displacement*.

where *index*[*i*] is the *i*th element of the XMM/YMM register specified by {$\overline{\text{X}}$,VSIB.index}. An index element is either 32 or 64 bits wide and is treated as a signed integer.

Variants of this mode use either an eight-bit or a 32-bit displacement value. One variant sets the base to zero. The value of the ModRM.mod field specifies the specific variant of VSIB addressing mode, as shown in Table 1. In the table, the notation [XMM*n*/YMM*n*] indicates the XMM/YMM register that contains the packed index array and [base GPR] means the contents of the base GPR selected by {$\overline{\text{B}}$, base}.

Table 1: Vectorized Addressing Modes

Index[1]	ModRM.mod		
	00	01	10
0000	scale * [XMM0/YMM0] + Disp32	scale * [XMM0/YMM0] + Disp8 + [base GPR]	scale * [XMM0/YMM0] + Disp32 + [base GPR]
0001	scale * [XMM1/YMM1] + Disp32	scale * [XMM1/YMM1] + Disp8 + [base GPR]	scale * [XMM1/YMM1] + Disp32 + [base GPR]
0010	scale * [XMM2/YMM2] + Disp32	scale * [XMM2/YMM2] + Disp8 + [base GPR]	scale * [XMM2/YMM2] + Disp32 + [base GPR]
0011	scale * [XMM3/YMM3] + Disp32	scale * [XMM3/YMM3] + Disp8 + [base GPR]	scale * [XMM3/YMM3] + Disp32 + [base GPR]
0100	scale * [XMM4/YMM4] + Disp32	scale * [XMM4/YMM4] + Disp8 + [base GPR]	scale * [XMM4/YMM4] + Disp32 + [base GPR]
0101	scale * [XMM5/YMM5] + Disp32	scale * [XMM5/YMM5] + Disp8 + [base GPR]	scale * [XMM5/YMM5] + Disp32 + [base GPR]
0110	scale * [XMM6/YMM6] + Disp32	scale * [XMM6/YMM6] + Disp8 + [base GPR]	scale * [XMM6/YMM6] + Disp32 + [base GPR]
0111	scale * [XMM7/YMM7] + Disp32	scale * [XMM7/YMM7] + Disp8 + [base GPR]	scale * [XMM7/YMM7] + Disp32 + [base GPR]
1000	scale * [XMM8/YMM8] + Disp32	scale * [XMM8/YMM8] + Disp8 + [base GPR]	scale * [XMM8/YMM8] + Disp32 + [base GPR]
1001	scale * [XMM9/YMM9] + Disp32	scale * [XMM9/YMM9] + Disp8 + [base GPR]	scale * [XMM9/YMM9] + Disp32 + [base GPR]
1010	scale * [XMM10/YMM10] + Disp32	scale * [XMM10/YMM10] + Disp8 + [base GPR]	scale * [XMM10/YMM10] + Disp32 + [base GPR]
1011	scale * [XMM11/YMM11] + Disp32	scale * [XMM11/YMM11] + Disp8 + [base GPR]	scale * [XMM11/YMM11] + Disp32 + [base GPR]
1100	scale * [XMM12/YMM12] + Disp32	scale * [XMM12/YMM12] + Disp8 + [base GPR]	scale * [XMM12/YMM12] + Disp32 + [base GPR]
1101	scale * [XMM13/YMM13] + Disp32	scale * [XMM13/YMM13] + Disp8 + [base GPR]	scale * [XMM13/YMM13] + Disp32 + [base GPR]
1110	scale * [XMM14/YMM14] + Disp32	scale * [XMM14/YMM14] + Disp8 + [base GPR]	scale * [XMM14/YMM14] + Disp32 + [base GPR]
1111	scale * [XMM15/YMM15] + Disp32	scale * [XMM15/YMM15] + Disp8 + [base GPR]	scale * [XMM15/YMM15] + Disp32 + [base GPR]
Note 1. Index = {$\overline{\text{VEX.X}}$,VSIB.index}. In 32-bit mode, VEX.X = 1.			

1.3.2 Notational Conventions Related to VSIB Addressing Mode

In the instruction descriptions that follow, the notation vm32x indicates a packed array of four 32-bit index values contained in the specified XMM index register and vm32y indicates a packed array of eight 32-bit index values contained in the specified YMM index register. Depending on the instruction, these indices can be used to compute the effective address of up to four (vm32x) or eight (vm32y) memory-based operands.

The notation vm64x indicates a packed array of two 64-bit index values contained in the specified XMM index register and vm64y indicates a packed array of four 64-bit index values contained in the specified YMM index register. Depending on the instruction, these indices can be used to compute the effective address of up to two (vm64x) or four (vm64y) memory-based operands.

In body of the description of the instructions, the notation mem32[vm32x] is used to represent a sparse array of 32-bit memory operands where the packed array of four 32-bit indices used to calculate the effective addresses of the operands is held in an XMM register. The notation mem32[vm32y] refers to a similar array of 32-bit memory operands where the packed array of eight 32-bit indices is held in a YMM register. The notation mem32[vm64x] means a sparse array of 32-bit memory operands where the packed array of two 64-bit indices is held in an XMM register and mem32[vm64y] means a sparse array of 32-bit memory operands where the packed array of four 64-bit indices is held in a YMM register.

The notation mem64[*index_array*], where *index_array* is either vm32x, vm64x, or vm64y, specifies a sparse array of 64-bit memory operands addressed via a packed array of 32-bit or 64-bit indices held in an XMM/YMM register. If an instruction uses either an XMM or a YMM register, depending on operand size, to hold the index array, the notation vm32x/y or vm64x/y is used to represent the array.

In summary, given a maximum operand size of 256-bits, a sparse array of 32-bit memory-based operands can be addressed using a vm32x, vm32y, vm64x, or vm64y index array. A sparse array of 64-bit memory-based operands can be addressed using a vm32x, vm64x, or vm64y index array. Specific instructions may use fewer than the maximum number of memory operands that can be addressed using the specified index array.

VSIB addressing is only valid in 32-bit or 64-bit effective addressing mode and is only supported for instruction encodings using the VEX prefix. The ModRM.mod value of 11b is not valid in VSIB addressing mode and ModRM.r/m must be set to 100b.

1.3.3 Memory Ordering and Exception Behavior

VSIB addressing has some special considerations relative to memory ordering and the signaling of exceptions.

VSIB addressing specifies an array of addresses that allows an instruction to access multiple memory locations. The order in which data is read from or written to memory is not specified. Memory ordering with respect to other instructions follows the memory-ordering model described in Volume 2.

Data may be accessed by the instruction in any order, but access-triggered exceptions are delivered in right-to-left order. That is, if a exception is triggered by the load or store of an element of an XMM/YMM register and delivered, all elements to the right of that element (all the lower indexed elements) have been or will be completed without causing an exception. Elements to the left of the element causing the exception may or may not be completed. If the load or store of a given element triggers multiple exceptions, they are delivered in the conventional order.

Because data can be accessed in any order, elements to the left of the one that triggered the exception may be read or written before the exception is delivered. Although the ordering of accesses is not specified, it is repeatable in a specific processor implementation. Given the same input values and initial architectural state, the same set of elements to the left of the faulting one will be accessed.

VSIB addressing should not be used to access memory mapped I/O as the ordering of the individual loads is implementation-specific and some implementations may access data larger than the data element size or access elements more than once.

1.4 Enabling SSE Instruction Execution

Application software that utilizes the SSE instructions requires support from operating system software.

To enable and support SSE instruction execution, operating system software must:

- enable hardware for supported SSE subsets
- manage the SSE hardware architectural state, saving and restoring it as required during and after task switches
- provide exception handlers for all unmasked SSE exceptions.

See Volume 2, Chapter 11, for details on enabling SSE execution and managing its execution state.

1.5 String Compare Instructions

The legacy SSE instructions PCMPESTRI, PCMPESTRM, PCMPISTRI, and PCMPISTRM and the extended SSE instructions VPCMPESTRI, VPCMPESTRM, VPCMPISTRI, and VPCMPISTRM provide a versatile means of classifying characters of a string by performing one of several different types of comparison operations using a second string as a prototype.

This section describes the operation of the legacy string compare instructions. This discussion applies equally to the extended versions of the instructions. Any difference between the legacy and the extended version of a given instruction is described in the instruction reference entry for the instruction in the following chapter.

A character string is a vector of data elements that is normally used to represent an ordered arrangement of graphemes which may be stored, processed, displayed, or printed. Ordered strings of graphemes are most often used to convey information in a human-readable manner. The string compare instructions, however, do not restrict the use or interpretation of their operands.

The first source operand provides the prototype string and the second operand is the string to be scanned and characterized (referred to herein as the string under test, or SUT). Four string formats and four types of comparisons are supported. The intermediate result of this processing is a bit vector that summarizes the characterization of each character in the SUT. This bit vector is then post-processed based on options specified in the instruction encoding. Instruction variants determine the final result—either an index or a mask.

Instruction execution affects the arithmetic status flags (ZF, CF, SF, OF, AF, PF), but the significance of many of the flags is redefined to provide information tailored to the result of the comparison performed. See Section 1.5.6, “Affect on Flags” on page 19.

The instructions have a defined base function and additional functionality controlled by bit fields in an immediate byte operand (*imm8*). The base function determines whether the source strings have implicitly (PCMPISTRI and PCMPISTRM) or explicitly (PCMPESTRI and PCMPESTRM) defined lengths, and whether the result is an index (PCMPISTRI and PCMPESTRI) or a mask (PCMPISTRM and PCMPESTRM).

PCMPISTRI and PCMPESTRI return their final result (an integer value) via the ECX register, while PCMPISTRM and PCMPESTRM write a bit or character mask, depending on the option selected, to the XMM0 register.

There are a number of different schemes for encoding a set of graphemes, but the most common ones use either an 8-bit code (ASCII) or a 16-bit code (unicode). The string compare instructions support both character sizes.

Bit fields of the immediate operand control the following functions:

- Source data format — character size (byte or word), signed or unsigned values
- Comparison type
- Intermediate result postprocessing
- Output option selection

This overview description covers functions common to all of the string compare instructions and describes some of the differentiated features of specific instructions. Information on instruction encoding and exception behavior are covered in the individual instruction reference pages in the following chapter.

1.5.1 Source Data Format

The character strings that constitute the source operands for the string compare instructions are formatted as either 8-bit or 16-bit integer values packed into a 128-bit data type. The figure below illustrates how a string of byte-wide characters is laid out in memory and how these characters are arranged when loaded into an XMM register.

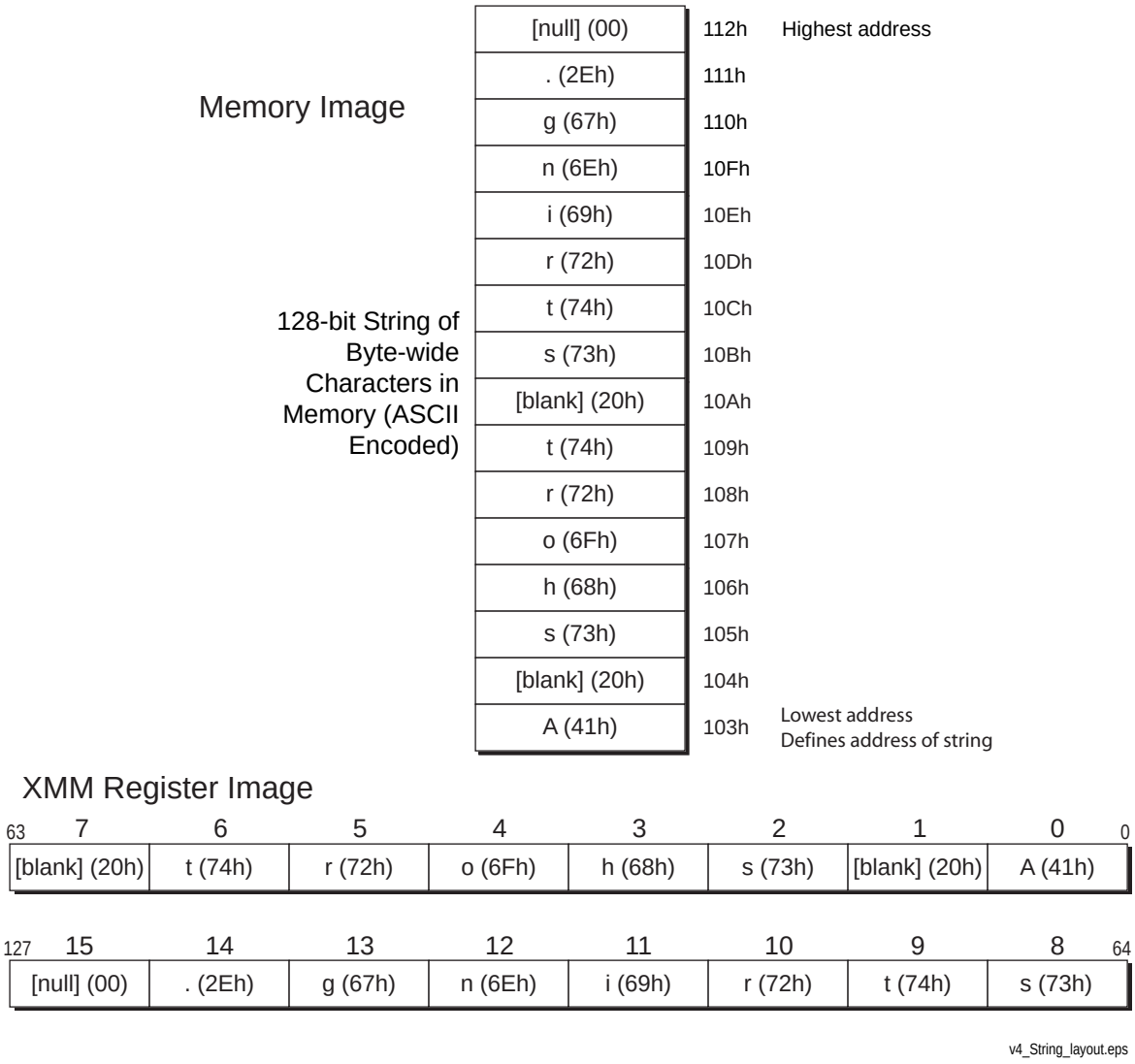


Figure 1-3. Byte-wide Character String – Memory and Register Image

Note from the figure that the longest string that can be packed in a 128-bit data object is either sixteen 8-bit characters (as illustrated) or eight 16-bit characters. When loaded from memory, the character read from the lowest address in memory is placed in the least-significant position of the register and the character read from the highest address is placed in the most-significant position. In other words, for character *i* of width *w*, bits [*w*−1:0] of the character are placed in bits [*iw* + (*w*−1):*iw*] of the register.

Bits [1:0] of the immediate byte operand specify the source string data format, as shown in Table 1-3.

Table 1-3. Source Data Format

Imm8[1:0]	Character Format	Maximum String Length
00b	unsigned bytes	16
01b	unsigned words	8
10b	signed bytes	16
11b	signed words	8

The string compare instructions are defined with the capability of operating on strings of lengths from 0 to the maximum that can be packed into the 128-bit data type as shown in the table above. Because strings being processed may be shorter than the maximum string length, a means is provided to designate the length of each string. As mentioned above, one pair of string compare instructions relies on an explicit method while the other utilizes an implicit method.

For the explicit method, the length of the first operand (the prototype string) is specified by the absolute value of the signed integer contained in rAX and the length of the second operand (the SUT) is specified by the absolute value of the signed integer contained in rDX. If a specified length is greater than the maximum allowed, the maximum value is used. Using the explicit method of length specification, null characters (characters whose numerical value is 0) can be included within a string.

Using the implicit method, a string shorter than the maximum length is terminated by a null character. If no null character is found in the string, its length is implied to be the maximum. For the example illustrated in Figure 1-3 above, the implicit length of the string is 15 because the final character is null. However, using the explicit method, a specified length of 16 would include the null character in the string.

In the following discussion, l_1 is the length of the first operand string (the prototype string), l_2 is the length of the second operand string (the SUT) and m is the maximum string length based on the selected character size.

1.5.2 Comparison Type

Although the string compare instructions can be implemented in many different ways, the instructions are most easily understood as the sequential processing of the SUT using the characters of the prototype string as a template. The template is applied at each character index of SUT, processing the string from the first character (index 0) to the last character (index l_2-1).

The result of each comparison is recorded in successive positions of a summary bit vector *CmprSumm*. When the sequence of comparisons is complete, this bit vector summarizes the results of comparison operations that were performed. The length of the *CmprSumm* bit vector is equal to the maximum input operand string length (*m)*. The rules for the setting of *CmprSumm* bits beyond the end of the SUT (*CmprSumm*[$m-1:l_2$]) are dependent on the comparison type (see Table 1-4 below.)

Bits [3:2] of the immediate byte operand determine the comparison type, as shown in Table 1-4.

Table 1-4. Comparison Type

Imm8[3:2]	Comparison Type	Description
00b	Subset	Tests each character of the SUT to determine if it is within the subset of characters specified by the prototype string. Each set bit of *CmprSumm* indicates that the corresponding character of the SUT is within the subset specified by the prototype. Bits [$m-1:l_2$] are cleared.
01b	Ranges	Tests each character of the SUT to determine if it lies within one or more ranges specified by pairs of values within the prototype string. The ranges are inclusive. Each set bit in *CmprSumm* indicates that the corresponding character of the SUT is within one or more of the inclusive ranges specified. Bits [$m-1:l_2$] are cleared. If the length of the prototype is odd, the last value in the prototype is effectively ignored.
10b	Match	Performs a character-by-character comparison between the SUT and the prototype string. Each set bit of *CmprSumm* indicates that the corresponding characters in the two strings match. If not, the bit is cleared. Bits [$m-1:\max(l_1, l_2)$] of *CmprSumm* are set.
11b	Sub-string	Searches for an exact match between the prototype string and an ordered sequence of characters (a sub-string) in the SUT beginning at the current index *i*. Bit *i* of *CmprSumm* is set for each value of *i* where the sub-string match is made, otherwise the bit is cleared. See discussion below.

In the Sub-string comparison type, any matching sub-string of the SUT must match the prototype string one-for-one, in order, and without gaps. Null characters in the SUT do not match non-null characters in the prototype. If the prototype and the SUT are equal in length and less than the max length, the two strings must be identical for the comparison to be TRUE. In this case, bit 0 of *CmprSumm* is set to one and the remainder are all 0s. If the length of the SUT is less than the prototype string, no match is possible and *CmprSumm* is all 0s.

If the prototype string is shorter than the SUT ($l_1 < l_2$), a sequential search of the SUT is performed. For each *i* from 0 to $l_2 - l_1$, the prototype is compared to characters [$i + l_1 - 1:i$] of the SUT. If the prototype and the sub-string SUT[$i + l_1 - 1:i$] match exactly, then *CmprSumm*[*i*] is set, otherwise the bit is cleared. When the comparison at $i = l_2 - l_1$ is complete, no further testing is required because there are not enough characters remaining in the SUT for a match to be possible. The remaining bits $l_2 - l_1 + 1$ through *m-1* are all set to 0.

For the Match comparison type, the character-by-character comparison is performed on all *m* characters in the 128-bit operand data, which may extend beyond the end of one or both strings. A null character at index *i* within one string is not considered a match when compared with a character beyond the end of the other string. In this case, *CmprSumm*[*i*] is cleared. For index positions beyond the end of both strings, *CmprSumm*[*i*] is set.

The following section provides more detail on the generation of the comparison summary bit vector based on the specified comparison type.

1.5.3 Comparison Summary Bit Vector

The following pseudo code provides more detail on the generation of the comparison summary bit vector *CmprSumm*. The function `CompareStrgs` defined below returns a bit vector of length *m*, the maximum length of the operand data strings.

```
bit vector CompareStrgs(ProtoType, length1, SUT, length2, CmpType, signed, m)
   doubleword vector StrUndTst   // temp vector; holds string under test
   doubleword vector StrProto    // temp vector; holds prototype string
   bit vector[m] Result          // length of vector is m

   StrProto = m{0}         //initialize m elements of StrProto to 0
   StrUndTst = m{0}        //initialize m elements of StrUndTst to 0
   Result = m{0}           //initialize result bit vector

   FOR i = 0 to length1
      StrProto[i] = signed ? SignExtend(ProtoType[i]) : ZeroExtend(ProtoType[i])
   FOR i = 0 to length2
      StrUndTst[i] = signed ? SignExtend(SUT[i]) : ZeroExtend(SUT[i])

   IF CmpType == Subset
      FOR j = 0 to length2 - 1            // j indexes SUT
         FOR i = 0 to length1 - 1         // i indexes prototype
            Result[j] |= (StrProto[i] == StrUndTst[j])

   IF CmpType == Ranges
      FOR j = 0 to length2 - 1                    // j indexes SUT
         FOR i = 0 to length1 - 2, BY 2           // i indexes prototype
            Result[j] |= (StrProto[i] <= StrUndTst[j])
                       && (StrProto[i+1] >= StrUndTst[j])

   IF CmpType == Match
      FOR i = 0 to (min(length1, length2)-1)
         Result[i] = (StrProto[i] == StrUndTst[i])
      FOR i = min(length1, length2) to (max(length1, length2)-1)
         Result[i] = 0
      FOR i = max(length1, length2) to (m-1)
         Result[i] = 1

   IF CmpType == Sub-string
      IF (length2==16)&& (length1==16)
         maxlength=15
      else
          maxlength = length2-length1
   IF length2 >= lenght1
    FOR j = 0 to maxlength             // j indexes result bit vector
         Result[j] = 1
         k = j                         // k scans the SUT
         FOR i = 0 to length1 - 1      // i scans the Prototype
            Result[j] &= (StrProto[i] == StrUndTst[k])// Result[j] is cleared if
any of the comparisons do not match
            k++

Return Result
```

Given the above definition of `CompareStrgs()`, the following pseudo code computes the value of *CmprSumm*:

```
ProtoType = contents of first source operand (xmm1)
SUT = contents of xmm2 or 128-bit value read from the specified memory location
length1 = length of first operand string     //specified implicitly or explicitly
length2 = length of second operand string    //specified implicitly or explicitly
m = Maximum String Length from Table 1-3 above
CmpType = Comparison Type from Table 1-4 above
signed = (imm8[1] == 1) ? TRUE : FALSE
bit vector [m] CmprSumm    // CmprSumm is m bits long

CmprSumm = CompareStrgs(ProtoType, length1, SUT, length2, CmpType, signed, m)
```

The following examples demonstrate the comparison summary bit vector *CmprSumm* for each comparison type. For the sake of illustration, the operand strings are represented as ASCII-encoded strings. Each character value is represented by its ASCII grapheme. Strings are displayed with the lowest indexed character on the left as they would appear when printed or displayed. *CmprSumm* is shown in reverse order with the least significant bit on the left to agree with the string presentation.

Comparison Type = Subset

```
Prototype:  ZCx
SUT:        aCx%xbZreCx
CmprSumm:   0110101001100000
```

Comparison Type = Ranges

```
Prototype:  ACax
SUT:        aCx%xbZreCx
CmprSumm:   1110110111100000
```

Comparison Type = Match

```
Prototype:  ZCx
SUT:        aCx%xbZreCx
CmprSumm:   0110000000011111
```

Comparison Type = Sub-string

```
Prototype:  ZCx
SUT:        aZCx%xCZreZCxCZ
CmprSumm:   0100000000100000
```

1.5.4 Intermediate Result Post-processing

Post-processing of the *CmprSumm* bit vector is controlled by *imm8*[5:4]. The result of this step is designated *pCmprSumm*.

Bit [4] of the immediate operand determines whether a ones' complement (bit-wise inversion) is performed on *CmprSumm*; bit [5] of the immediate operand determines whether the inversion applies to the entire comparison summary bit vector (*CmprSumm*) or just to those bits that correspond to characters within the SUT. See Table 1-5 below for the encoding of the *imm8*[5:4] field.

Table 1-5. Post-processing Options

Imm8[5:4]	Post-processing Applied
x0b	pCmprSumm = CmprSumm
01b	pCmprSumm = NOT CmprSumm
11b	pCmprSumm[*i*] = **!**CmprSumm[*i*] for $i < I_2$, pCmprSumm[*i*] = CmprSumm[*i*], for $I_2 \leq i < m$

1.5.5 Output Option Selection

For PCMPESTRI and PCMPISTRI, *imm8*[6] determines whether the index of the lowest set bit or the highest set bit of *pCmprSumm* is written to ECX, as shown in Table 1-6.

Table 1-6. Indexed Output Option Selection

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

For PCMPESTRM and PCMPISTRM, *imm8*[6] specifies whether the output from the instruction is a bit mask or an expanded mask. The bit mask is a copy of *pCmprSumm* zero-extended to 128 bits. The expanded mask is a packed vector of byte or word elements, as determined by the string operand format (as indicated by *imm8*[0]). The expanded mask is generated by copying each bit of *pCmprSumm* to all bits of the element of the same index. Table 1-7 below shows the encoding of *imm8*[6].

Table 1-7. Masked Output Option Selection

Imm8[6]	Description
0b	Return pCmprSumm as the output with zero extension to 128 bits.
1b	Return expanded pCmprSumm byte or word mask.

The PCMPESTRM and PCMPISTRM instructions return their output in register XMM0. For the extended forms of the instructions, bits [127:64] of YMM0 are cleared.

1.5.6 Effect on Flags

The execution of a string compare instruction updates the state of the CF, PF, AF, ZF, SF, and OF flags within the rFLAGs register. All other flags are unaffected. The PF and AF flags are always cleared. The ZF and SF flags are set or cleared based on attributes of the source strings and the CF and OF flags are set or cleared based on attributes of the summary bit vector after post processing.

The CF flag is cleared if the summary bit vector, after post processing, is zero; the flag is set if one or more of the bits in the post-processed bit vector are 1. The OF flag is updated to match the value of the least significant bit of the post-processed summary bit vector.

The ZF flag is set if the length of the second string operand (SUT) is shorter than *m*, the maximum number of 8-bit or 16-bit characters that can be packed into 128 bits. Similarly, the SF flag is set if the length of the first string operand (prototype) is shorter than *m*.

This information is summarized in Table 1-8 below.

Table 1-8. State of Affected Flags After Execution

Unconditional		Source String Length		Post-processed Bit Vector	
PF	**AF**	**SF**	**ZF**	**CF**	**OF**
0	0	($l_1 < m$)	($l_2 < m$)	pCmprSumm ≠ 0	pCmprSumm [0]

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 **Instruction**
VINST **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	$\overline{RXB}$.11	0.$\overline{src}$.0.00	FF */r*
VINST *ymm1*, *ymm2*/*mem256, ymm3*	C4	$\overline{RXB}$.11	0.$\overline{src}$.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

Instruction Exceptions

Under various conditions instructions described below can cause exceptions. The conditions that cause these exceptions can differ based on processor mode and instruction subset. This information is summarized at the end of each instruction reference page in an Exception Table. Rows list the applicable exceptions and the different conditions that trigger each exception for the instruction. For each processor mode (real, virtual, and protected) a symbol in the table indicates whether this exception condition applies.

Each AVX instruction has a legacy form that comes from one of the legacy (SSE1, SSE2, ...) subsets. An “X” at the intersection of a processor mode column and an exception cause row indicates that the causing condition and potential exception applies to both the AVX instruction and the legacy SSE instruction. “A” indicates that the causing condition applies only to the AVX instruction and “S” indicates that the condition applies to the SSE legacy instruction.

Note that XOP and FMA4 instructions do not have corresponding instructions from the SSE legacy subsets. In the exception tables for these instructions, “X” represents the XOP instruction and “F” represents the FMA4 instruction.

ADDPD Add
VADDPD 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 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.

VADDPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Adds two pairs of values.

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.

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.

Instruction Support

Form	Subset	Feature Flag
ADDPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VADDPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Add
VADDPS 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 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.

VADDPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Adds four pairs of values.

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.

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.

Instruction Support

Form	Subset	Feature Flag
ADDPS	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VADDPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Add
VADDSD 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
ADDSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VADDSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Add
VADDSS 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
ADDSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VADDSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 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.

VADDSUBPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ADDSUBPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VADDSUBPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 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.

VADDSUBPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ADDSUBPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VADDSUBPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

(V)ADDSUBPD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Appendix A on page 973 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 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.

VAESDEC

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

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.

Instruction Support

Form	Subset	Feature Flag
AESDEC	AES	CPUID Fn0000_0001_ECX[AES] (bit 25)
VAESDEC	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

AESDECLAST AES
VAESDECLAST 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 Appendix A on page 973 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.

There are legacy and extended forms of the instruction:

AESDECLAST

The first source operand is an XMM register. 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.

VAESDECLAST

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

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.

Instruction Support

Form	Subset	Feature Flag
AESDECLAST	AES	CPUID Fn0000_0001_ECX[AES] (bit 25)
VAESDECLAST	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Appendix A on page 973 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 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.

VAESENC

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

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.

Instruction Support

Form	Subset	Feature Flag
AESENC	AES	CPUID Fn0000_0001_ECX[AES] (bit 25)
VAESENC	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

AESENCLAST AES
VAESENCLAST 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 Appendix A on page 973 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:

AESENCLAST

The first source operand is an XMM register. 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.

VAESENCLAST

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

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.

Instruction Support

Form	Subset	Feature Flag
AESENCLAST	AES	CPUID Fn0000_0001_ECX[AES] (bit 25)
VAESENCLAST	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

AESIMC AES
VAESIMC 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 Appendix A on page 973 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 legacy and extended forms of the instruction:

AESIMC

The first source operand is an XMM register. 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.

VAESIMC

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

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.

Instruction Support

Form	Subset	Feature Flag
AESIMC	AES	CPUID Fn0000_0001_ECX[AES] (bit 25)
VAESIMC	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

AESKEYGENASSIST AES
VAESKEYGENASSIST 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 Appendix A on page 973 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 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.

VAESKEYGENASSIST

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

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.

Instruction Support

Form	Subset	Feature Flag
AESKEYGENASSIST	AES	CPUID Fn0000_0001_ECX[AES] (bit 25)
VAESKEYGENASSIST	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 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.

VANDNPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ANDNPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VANDNPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

ANDNPS AND NOT
VANDNPS 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 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.

VANDNPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ANDNPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VANDNPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

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

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

ANDPD AND
VANDPD 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 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.

VANDPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ANDPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VANDPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

ANDPS AND
VANDPS 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 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.

VANDPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ANDPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VANDPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
ANDPS *xmm1*, *xmm2/mem128*	0F 54 /r	Performs bitwise AND 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.map_select	W.vvvv.L.pp	Opcode
VANDPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	54 /r
VANDPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.00	54 /r

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

BLENDPD Blend
VBLENDPD 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 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. Only mask bits [1:0] are used.

VBLENDPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
BLENDPD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VBLENDPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

BLENDPS Blend
VBLENDPS 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 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. Only mask bits [3:0] are used.

VBLENDPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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 a YMM register or a 256-bit memory location. The destination is a third YMM register. All 8 bits of the mask are used.

Instruction Support

Form	Subset	Feature Flag
BLENDPS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VBLENDPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VBLENDPS *xmm1*, *xmm2*, *xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	0C /r ib
VBLENDPS *ymm1, ymm2, ymm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.1.01	0C /r ib

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

BLENDVPD Variable Blend
VBLENDVPD 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 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. 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 encodings:

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. 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 a 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.

Instruction Support

Form	Subset	Feature Flag
BLENDVPD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VBLENDVPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

BLENDVPS Variable Blend
VBLENDVPS 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 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. 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 encodings:

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. 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 a 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.

Instruction Support

Form	Subset	Feature Flag
BLENDVPS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VBLENDVPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

CMPPD Compare
VCMPPD 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 an 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 encodings:

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. 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

Instruction Support

Form	Subset	Feature Flag
CMPPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCMPPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Compare
VCMPPS 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 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. 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 encodings:

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. 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

Instruction Support

Form	Subset	Feature Flag
CMPPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VCMPPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Compare
VCMPSD 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 an 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 a 128-bit encoding only.

The first source operand is an XMM register. 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 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

Instruction Support

Form	Subset	Feature Flag
CMPSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCMPSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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

Related Instructions

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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Compare
VCMPSS 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 an 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 a 128-bit encoding only.

The first source operand is an XMM register. 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 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

Instruction Support

Form	Subset	Feature Flag
CMPSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VCMPSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Compare Ordered
VCOMISD Scalar Double-Precision Floating-Point

Compares a double-precision floating-point value in the low-order 64 bits of the first operand with a double-precision floating-point value in the low-order 64 bits of the second operand 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 an XMM register or a 64-bit memory location.

VCOMISD

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

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

Instruction Support

Form	Subset	Feature Flag
COMISD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCOMISD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 the first operand with a single-precision floating-point value in the low-order 32 bits of the second operand 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 an XMM register or a 32-bit memory location.

VCOMISS

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

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

Instruction Support

Form	Subset	Feature Flag
COMISS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VCOMISS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTDQ2PD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTDQ2PD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VCVTDQ2PD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.0.10	E6 /r
VCVTDQ2PD *ymm1*, *ymm2/mem256*	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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.

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 encodings:

XMM Encoding

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 cleared.

YMM Encoding

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

Instruction Support

Form	Subset	Feature Flag
CVTDQ2PS	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTDQ2PS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTPD2DQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTPD2DQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Convert Packed Double-Precision Floating-Point
VCVTPD2PS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTPD2PS	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTPD2PS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTPS2DQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTPS2DQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTPS2PD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTPS2PD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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

The legacy form has two encodings:

- 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 two 128-bit encodings:

- 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.

Instruction Support

Form	Subset	Feature Flag
CVTSD2SI	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTSD2SI	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 the second 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 a 128-bit encoding only.

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 the 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.

Instruction Support

Form	Subset	Feature Flag
CVTSD2SS	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTSD2SS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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

The legacy form as two encodings:

- 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 two 128-bit encodings:

- 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.

Instruction Support

Form	Subset	Feature Flag
CVTSI2SD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTSI2SD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VCVTSI2SD *xmm1*, *xmm2*, *reg32*/*mem32*	C4	$\overline{\text{RXB}}$.00001	0.$\overline{src}$.X.11	2A /r
VCVTSI2SD *xmm1*, *xmm2*, *reg64*/*mem64*	C4	$\overline{\text{RXB}}$.00001	1.$\overline{src}$.X.11	2A /r

Related Instructions

(V)CVTDQ2PD, (V)CVTPD2DQ, (V)CVTPI2PD, (V)CVTSD2SI, (V)CVTTPD2DQ, (V)CVTTSD2SI

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Convert Signed Doubleword or Quadword Integer
VCVTSI2SS 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

The legacy form has two encodings:

- 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 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.

VCVTSI2SS

The extended form of the instruction has two 128-bit encodings:

- 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 single-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 single-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.

Instruction Support

Form	Subset	Feature Flag
CVTSI2SS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VCVTSI2SS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 VCVTSS2SD Convert Scalar Single-Precision Floating-Point 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 a 128-bit encoding only.

Converts a scalar single-precision floating-point value in the low-order 32 bits of the second source XMM register or 32-bit memory location to a scalar 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 destination are copied from the first source XMM register. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
CVTSS2SD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTSS2SD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VCVTSS2SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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 when 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

The legacy form has two encodings:

- 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 two 128-bit encodings:

- 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.

Instruction Support

Form	Subset	Feature Flag
CVTSS2SI	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VCVTSS2SI	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Convert Packed Double-Precision Floating-Point VCVTTPD2DQ 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTTPD2DQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTTPD2DQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
CVTTPS2DQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTTPS2DQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Convert Scalar Double-Precision Floating-Point VCVTTSD2SI 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

The legacy form of the instruction has two encodings:

- 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 two 128-bit encodings.

- 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.

Instruction Support

Form	Subset	Feature Flag
CVTTSD2SI	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VCVTTSD2SI	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Convert Scalar Single-Precision Floating-Point VCVTTSS2SI 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

The legacy form of the instruction has two encodings:

- 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 two 128-bit encodings:

- 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.

Instruction Support

Form	Subset	Feature Flag
CVTTSS2SI	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VCVTTSS2SI	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Divide
VDIVPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
DIVPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VDIVPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VDIVPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	5E /r
VDIVPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Divide
VDIVPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
DIVPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VDIVPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Divide
VDIVSD 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
DIVSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VDIVSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VDIVSD *xmm1*, *xmm2*, *xmm3/mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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 when 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
DIVSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VDIVSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 a second 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 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.

VDPPD

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

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.

Instruction Support

Form	Subset	Feature Flag
DPPD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VDPPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VDPPD *xmm1*, *xmm2*, *xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Dot Product
VDPPS 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 a second 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 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.

VDPPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
DPPS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VDPPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Extract
VEXTRACTPS 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 a single 128-bit encoding.

The source operand is an XMM register. The destination can be a general purpose register or a 32-bit memory location.

Instruction Support

Form	Subset	Feature Flag
EXTRACTPS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VEXTRACTPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

EXTRQ Extract Field From Register

Extracts specified bits from the lower 64 bits of the first operand (the destination XMM register). The extracted bits are saved in the least-significant bit positions of the lower quadword of the destination; the remaining bits in the lower quadword of the destination register are cleared to 0. The upper quadword of the destination register is undefined.

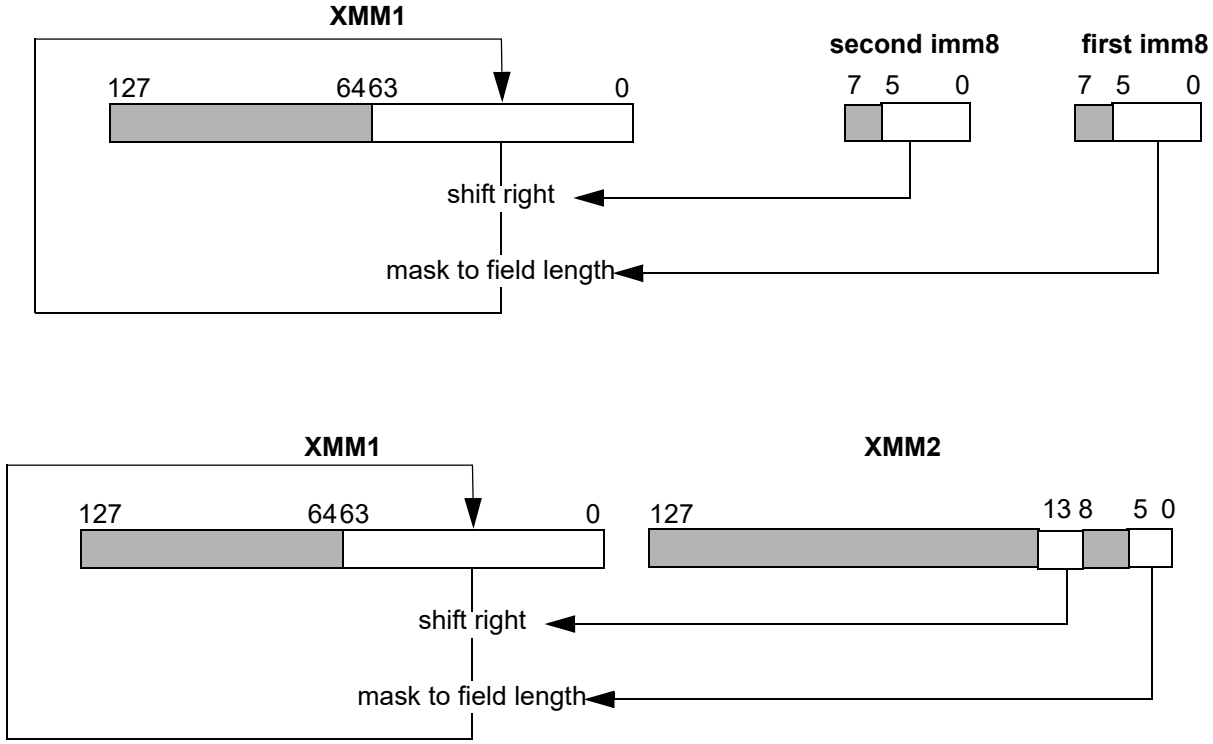
The portion of the source data being extracted is defined by the bit index and the field length. The bit index defines the least-significant bit of the source operand being extracted. Bits [*bit index* + *length field* – 1]:[*bit index*] are extracted. If the sum of the *bit index* + *length field* is greater than 64, the results are undefined.

For example, if the bit index is 32 (20h) and the field length is 16 (10h), then the result in the destination register will be source [47:32] in bits 15:0, with zeros in bits 63:16.

A value of zero in the field length is defined as a length of 64. If the *length field* is 0 and the *bit index* is 0, bits 63:0 of the source are extracted. For any other value of the *bit index*, the results are undefined.

The *bit index* and *field length* can be specified as immediate values (second and first immediate operands, respectively, in the case of the three argument version of the instruction), or they can both be specified by fields in an XMM source operand. In the latter case, bits [5:0] of the XMM register specify the number of bits to extract (the *field length*) and bits [13:8] of the XMM register specify the index of the first bit in the field to extract. The *bit index* and *field length* are each six bits in length; other bits of the field are ignored.

The diagram below illustrates the operation of this instruction.



Instruction Support

Form	Subset	Feature Flag
EXTRQ	SSE4A	CPUID Fn8000_0001_ECX[SSE4A] (bit 6)

Software *must* check the CPUID bit once per program or library initialization before using the instruction, or inconsistent behavior may result. For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
EXTRQ *xmm1*, *imm8*, *imm8*	66 0F 78 /0 ib ib	Extract field from *xmm1*, with the least significant bit of the extracted data starting at the bit index specified by [5:0] of the second immediate byte, with the length specified by [5:0] of the first immediate byte.
EXTRQ *xmm1*, *xmm2*	66 0F 79 /r	Extract field from *xmm1*, with the least significant bit of the extracted data starting at the bit index specified by xmm2[13:8], with the length specified by xmm2[5:0].

Related Instructions

INSERTQ, PINSRW, PEXTRW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	SSE4A instructions are not supported, as indicated by CPUID Fn8000_0001_ECX[SSE4A] = 0.
	X	X	X	The emulate bit (EM) of CR0 was set to 1.
	X	X	X	The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.

HADDPD Horizontal Add
VHADDPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
HADDPD	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VHADDPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Horizontal Add
VHADDPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
HADDPS	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VHADDPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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	RXB.map_select	W.vvvv.L.pp	Opcode
VHADDPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	RXB.00001	X.*src*.0.11	7C /r
VHADDPS *ymm1, ymm2, ymm3/mem256*	C4	RXB.00001	X.*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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Horizontal Subtract
VHSUBPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
HSUBPD	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VHSUBPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
HSUBPS	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VHSUBPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Insert
VINSERTPS 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 a 128-bit encoding only.

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 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.

Instruction Support

Form	Subset	Feature Flag
INSERTPS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VINSERTPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VINSERTPS *xmm1*, *xmm2*, *xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

INSERTQ Insert Field

Inserts bits from the lower 64 bits of the source operand into the lower 64 bits of the destination operand. No other bits in the lower 64 bits of the destination are modified. The upper 64 bits of the destination are undefined.

The least-significant *l* bits of the source operand are inserted into the destination, with the least-significant bit of the source operand inserted at bit position *n*, where *l* and *n* are defined as the *field length* and *bit index*, respectively.

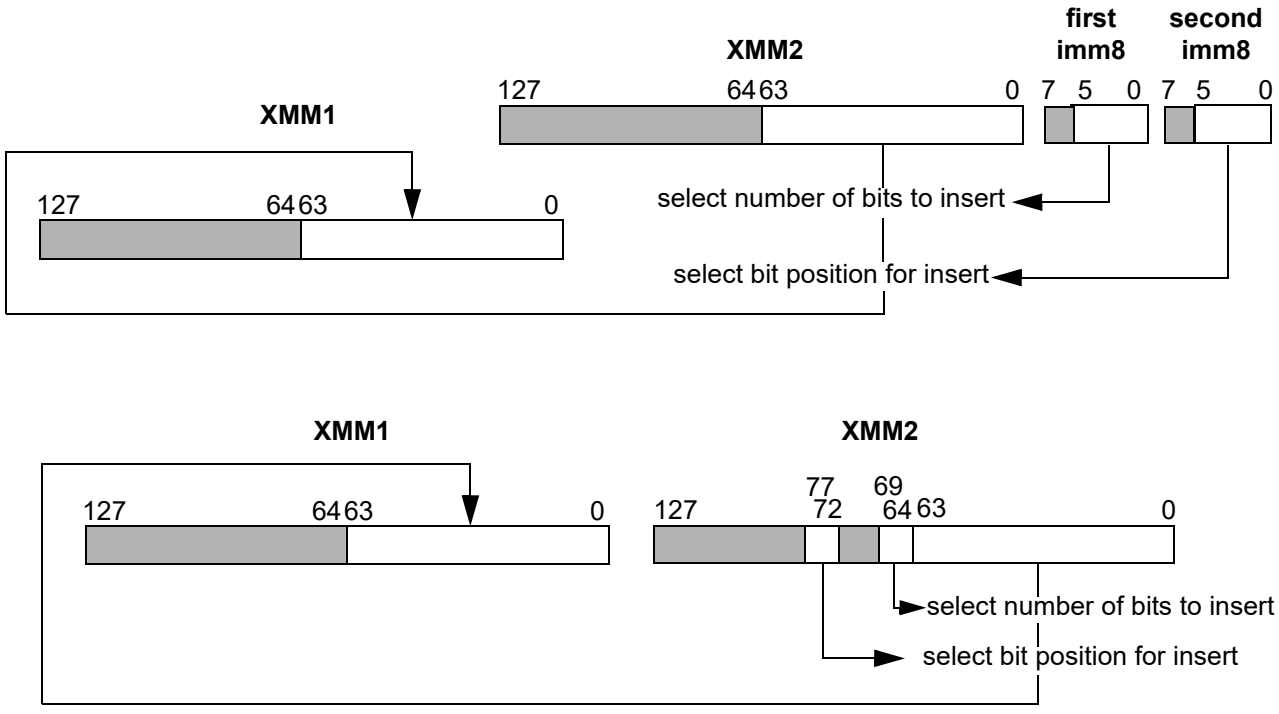
Bits (*field length* – 1):0 of the source operand are inserted into bits (*bit index* + *field length* – 1):(*bit index*) of the destination. If the sum of the *bit index* + *length field* is greater than 64, the results are undefined.

For example, if the *bit index* is 32 (20h) and the field length is 16 (10h), then the result in the destination register will be *source operand*[15:0] in bits 47:32. Bits 63:48 and bits 31:0 are not modified.

A value of zero in the *field length* is defined as a length of 64. If the *length field* is 0 and the *bit index* is 0, bits 63:0 of the source operand are inserted. For any other value of the *bit index*, the results are undefined.

The bits to insert are located in the XMM2 source operand. The *bit index* and *field length* can be specified as immediate values or can be specified in the XMM source operand. In the immediate form, the *bit index* and the *field length* are specified by the fourth (second immediate byte) and third operands (first immediate byte), respectively. In the register form, the *bit index* and *field length* are specified in bits [77:72] and bits [69:64] of the source XMM register, respectively. The *bit index* and *field length* are each six bits in length; other bits in the field are ignored.

The diagram below illustrates the operation of this instruction.



Instruction Support

Form	Subset	Feature Flag
INSERTQ	SSE4A	CPUID Fn8000_0001_ECX[SSE4A] (bit 6)

Software *must* check the CPUID bit once per program or library initialization before using the instruction, or inconsistent behavior may result. For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
INSERTQ *xmm1*, *xmm2*, *imm8*, *imm8*	F2 0F 78 /r ib ib	Insert field starting at bit 0 of xmm2 with the length specified by [5:0] of the first immediate byte. This field is inserted into xmm1 starting at the bit position specified by [5:0] of the second immediate byte.
INSERTQ *xmm1*, *xmm2*	F2 0F 79 /r	Insert field starting at bit 0 of xmm2 with the length specified by xmm2[69:64]. This field is inserted into xmm1 starting at the bit position specified by xmm2[77:72].

Related Instructions

EXTRQ, PINSRW, PEXTRW

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	SSE4A instructions are not supported, as indicated by CPUID Fn8000_0001_ECX[SSE4A] = 0.
	X	X	X	The emulate bit (EM) of CR0 was set to 1.
	X	X	X	The operating-system FXSAVE/FXRSTOR support bit (OSFXSR) of CR4 is cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (TS) of CR0 was set to 1.

LDDQU Load
VLDDQU 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
LDDQU	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VLDDQU	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	X	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

LDMXCSR Load
VLDMXCSR 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.

Instruction Support

Form	Subset	Feature Flag
LDMXCSR	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VLDMXCSR	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
LDMXCSR *mem32*	0F AE */2*	Loads MXCSR register with 32-bit value from memory.

Mnemonic	Encoding			
	VEX	RXB.map_select	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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

MASKMOVDQU Masked Move
VMASKMOVDQU 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.

Exception and trap behavior for elements not selected for storage to memory is implementation dependent. For instance, a given implementation may signal a data breakpoint or a page fault for bytes that are zero-masked and not actually written.

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 an XMM register. The destination is a 128-bit memory location.

VMASKMOVDQU

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

The first source operand is an XMM register and the second source operand is an XMM register. The destination is a 128-bit memory location.

Instruction Support

Form	Subset	Feature Flag
MASKMOVDQU	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMASKMOVDQU	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

MAXPD Maximum
VMAXPD 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 an 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 encodings:

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 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

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.

Instruction Support

Form	Subset	Feature Flag
MAXPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMAXPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMAXPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	5F /r
VMAXPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Maximum
VMAXPS 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 an 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 encodings:

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 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

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.

Instruction Support

Form	Subset	Feature Flag
MAXPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMAXPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMAXPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	5F /r
VMAXPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Maximum
VMAXSD 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 an 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 a 128-bit encoding only.

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 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.

Instruction Support

Form	Subset	Feature Flag
MAXSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMAXSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Maximum
VMAXSS 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 an 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 a 128-bit encoding only.

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 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.

Instruction Support

Form	Subset	Feature Flag
MAXSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMAXSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Minimum
VMINPD 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 an 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 encodings:

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 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

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.

Instruction Support

Form	Subset	Feature Flag
MINPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMINPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMINPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.01	5D /r
VMINPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Minimum
VMINPS 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 an 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 encodings:

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 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

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.

Instruction Support

Form	Subset	Feature Flag
MINPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMINPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMINPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	5D /r
VMINPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Minimum
VMINSD 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 an 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 a 128-bit encoding only.

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 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.

Instruction Support

Form	Subset	Feature Flag
MINSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMINSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Minimum
VMINSS 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 an 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 a 128-bit encoding only.

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 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.

Instruction Support

Form	Subset	Feature Flag
MINSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMINSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Move Aligned
VMOVAPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVAPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVAPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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 Move Aligned
VMOVAPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVAPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVAPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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 Move
VMOVD 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 = 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 not affected.

VMOVD

The extended form of the instruction has four 128-bit encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
MOVD *xmm*, *reg32*/*mem32*	66 (W0) 0F 6E /r	Move a 32-bit value from *reg32*/*mem32* to *xmm*.
MOVD *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*
MOVD *reg64*/*mem64*, *xmm*	66 (W1) 0F 7E /r	Move a 64-bit value from *xmm* to *reg64*/*mem64*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VMOVD[1] *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[1] r*eg32*/*mem32*, *xmm*	C4	$\overline{\text{RXB}}$.00001	0.1111.0.01	7E /r
VMOVQ r*eg64*/*mem64*, *xmm*	C4	$\overline{\text{RXB}}$.00001	1.1111.0.01	7E /r

Note: *1. Also known as MOVQ in some developer tools.*

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 an 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 encodings:

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 an 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 a 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.

Instruction Support

Form	Subset	Feature Flag
MOVDDUP	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VMOVDDUP	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.

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

MOVDQA Move Aligned
VMOVDQA Double Quadword

Moves aligned packed integer values. Values can be moved from a register or a memory location to a register, or from a register to a 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 an 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.

VMOVDQA

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Moves two aligned quadwords (128-bit move). There are two encodings.

- The source operand is an XMM register. The destination is either an 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 a 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.

Instruction Support

Form	Subset	Feature Flag
MOVDQA	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVDQA	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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 Move
VMOVDQU Unaligned Double Quadword

Moves unaligned packed integer values. Values can be moved from a register or a memory location to a register, or from a register to a 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 an 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.

VMOVDQU

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Moves two unaligned quadwords (128-bit move). There are two encodings:

- The source operand is an XMM register. The destination is either an 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 a 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.

Instruction Support

Form	Subset	Feature Flag
MOVDQU	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVDQU	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	X	Memory operand not 16-byte aligned when alignment checking enabled.
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 an 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 an 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
MOVHLPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVHLPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 Move High
VMOVHPD 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.

Instruction Support

Form	Subset	Feature Flag
MOVHPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVHPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMOVHPD *xmm1*, *xmm2*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

MOVHPS Move High
VMOVHPS 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.

Instruction Support

Form	Subset	Feature Flag
MOVHPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVHPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

MOVLHPS Move Low to High
VMOVLHPS 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 a second 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 an 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
MOVLHPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVLHPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMOVLHPS *xmm1*, *xmm2*, *xmm3*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	16 /r

Related Instructions

(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	XFEATURE_ENABLED_MASK[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

MOVLPD Move Low
VMOVLPD 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.

Instruction Support

Form	Subset	Feature Flag
MOVLPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVLPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
MOVLPD *xmm1*, *mem64*	66 0F 12 /r	Moves a packed double-precision floating-point value from *mem64* to *xmm1[63:0].*
MOVLPD *mem64*, *xmm1*	66 0F 13 /r	Moves a packed double-precision floating-point value from *xmm1[63:0]* to *mem64*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VMOVLPD *xmm1*, *xmm2*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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.

Instruction Support

Form	Subset	Feature Flag
MOVLPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVLPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

MOVMSKPD Extract Sign Mask
VMOVMSKPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVMSKPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVMSKPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMOVMSKPD *reg*, x*mm*	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	XFEATURE_ENABLED_MASK[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 Extract Sign Mask
VMOVMSKPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVMSKPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVMSKPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.

An attempted store to a non-aligned memory location results in a #GP exception.

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.

VMOVNTDQ

The extended form of the instruction has both 128-bit and 256-bit encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVNTDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVNTDQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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

Loads an XMM/YMM register from a naturally-aligned 128-bit or 256-bit memory location.

Indicates to the processor that the data is non-temporal, and is unlikely to be used again soon. The processor treats the load as a write-combining (WC) memory read, 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 MOVNTDQA with respect to other reads.

An attempted load from a non-aligned memory location results in a #GP exception.

There are legacy and extended forms of the instruction:

MOVNTDQA

Loads a 128-bit value into the specified XMM register from a 16-byte aligned memory location.

VMOVNTDQA

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Loads a 128-bit value into the specified XMM register from a 16-byte aligned memory location.

YMM Encoding

Loads a 256-bit value into the specified YMM register from a 32-byte aligned memory location.

Instruction Support

Form	Subset	Feature Flag
MOVNTDQA	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VMOVNTDQA 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VMOVNTDQA 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
MOVNTDQA *xmm, mem128*	66 0F 38 2A /r	Loads xmm from an aligned memory location, minimizing cache pollution.

	Encoding			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VMOVNTDQA *xmm, mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	2A /r
VMOVNTDQA *ymm, mem256*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 a 16-byte boundary.
	S	S	X	Write to a read-only data segment.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit 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, AVX2, and SSE exception
A — AVX, AVX2 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.

An attempted store to a non-aligned memory location results in a #GP exception.

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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVNTPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVNTPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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 Move Non-Temporal
VMOVNTPS Packed Single-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.

An attempted store to a non-aligned memory location results in a #GP exception.

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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVNTPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVNTPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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

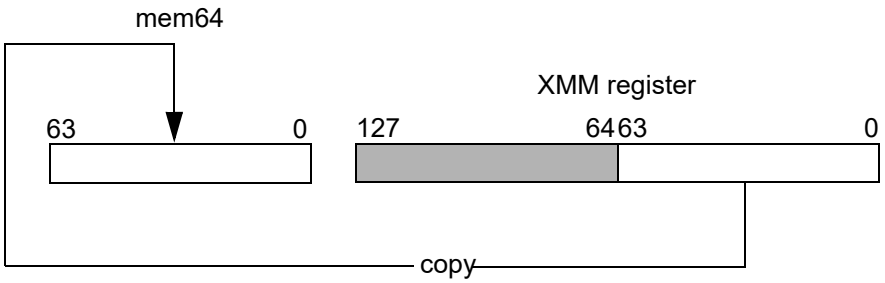
Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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

MOVNTSD Move Non-Temporal Scalar Double-Precision Floating-Point

Stores one double-precision floating-point value from an XMM register to a 64-bit memory location. This instruction 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 memory write, which minimizes cache pollution.

The diagram below illustrates the operation of this instruction:



Instruction Support

Form	Subset	Feature Flag
MOVNTSD	SSE4A	CPUID Fn8000_0001_ECX[SSE4A] (bit 6)

Software *must* check the CPUID bit once per program or library initialization before using the instruction, or inconsistent behavior may result. For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic		Opcode	Description
MOVNTSD	*mem64*, *xmm*	F2 0F 2B /r	Stores one double-precision floating-point XMM register value into a 64 bit memory location. Treat as a non-temporal store.

Related Instructions

MOVNTDQ, MOVNTI, MOVNTPD, MOVNTPS, MOVNTQ, MOVNTSS

rFLAGS Affected

None

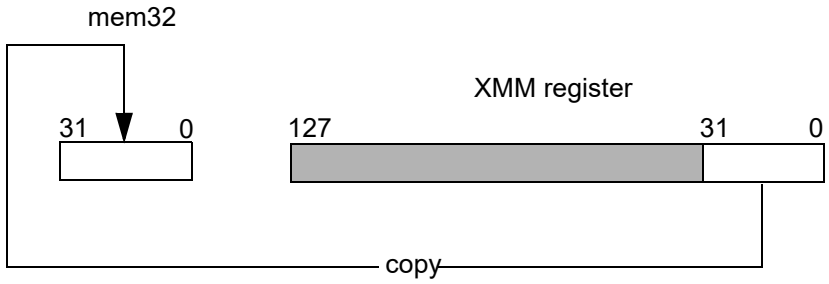
Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The SSE4A instructions are not supported, as indicated by CPUID Fn8000_0001_ECX[SSE4A] = 0.
	X	X	X	The emulate bit (CR0.EM) was set to 1.
	X	X	X	The operating-system FXSAVE/FXRSTOR support bit (CR4.OSFXSR) was cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (CR0.TS) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
			X	The destination operand was in a non-writable segment.
Page fault, #PF		X	X	A page fault resulted from executing the instruction.
Alignment check, #AC		X	X	An unaligned memory reference was performed while alignment checking was enabled.

MOVNTSS Move Non-Temporal Scalar Single-Precision Floating-Point

Stores one single-precision floating-point value from an XMM register to a 32-bit memory location. This instruction 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 memory write, which minimizes cache pollution.

The diagram below illustrates the operation of this instruction:



Instruction Support

Form	Subset	Feature Flag
MOVNTSS	SSE4A	CPUID Fn8000_0001_ECX[SSE4A] (bit 6)

Software *must* check the CPUID bit once per program or library initialization before using the instruction, or inconsistent behavior may result. For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic		Opcode	Description
MOVNTSS	*mem32*, *xmm*	F3 0F 2B /r	Stores one single-precision floating-point XMM register value into a 32-bit memory location. Treat as a non-temporal store.

Related Instructions

MOVNTDQ, MOVNTI, MOVNTOPD, MOVNTPS, MOVNTQ, MOVNTSD

rFLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	The SSE4A instructions are not supported, as indicated by CPUID Fn8000_0001_ECX[SSE4A] = 0.
	X	X	X	The emulate bit (CR0.EM) was set to 1.
	X	X	X	The operating-system FXSAVE/FXRSTOR support bit (CR4.OSFXSR) was cleared to 0.
Device not available, #NM	X	X	X	The task-switch bit (CR0.TS) was set to 1.
Stack, #SS	X	X	X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP	X	X	X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
			X	The destination operand was in a non-writable segment.
Page fault, #PF		X	X	A page fault resulted from executing the instruction.
Alignment check, #AC		X	X	An unaligned memory reference was performed while alignment checking was enabled.

MOVQ Move
VMOVQ 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.

Instruction Support

Form	Subset	Feature Flag
MOVQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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.1111.0.01	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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

MOVSD Move
VMOVSD 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 two encodings.

- The source operand is either an XMM register or a 64-bit memory location. The destination is an XMM register. If the source operand is a register, bits [127:64] of the destination are not affected. If the source operand is a 64-bit memory location, the upper 64 bits of the destination are cleared.
- 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, bits [127:64] of the destination are not affected.

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.

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.

Instruction Support

Form	Subset	Feature Flag
MOVSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
MOVSD *xmm1, xmm2/mem64*	F2 0F 10 /r	Moves a 64-bit value from *xmm2 or mem64* to *xmm1*. Zero extends to 128 bits when source operand is memory.
MOVSD *xmm1*/*mem64*, *xmm2*	F2 0F 11 /r	Moves a 64-bit value from *xmm2* to *xmm1* or *mem64*.

	Encoding [1]			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VMOVSD *xmm1*, *mem64*	C4	$\overline{\text{RXB}}$.00001	X.1111.X.11	10 /r
VMOVSD *mem64*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.X.11	11 /r
VMOVSD *xmm1, xmm2*, *xmm3* [2]	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	10 /r
VMOVSD *xmm1, xmm2*, *xmm3* [2]	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.11	11 /r

Note 1: The addressing mode differentiates between the two operand form (where one operand is a memory location) and the three operand form (where all operands are held in registers).

Note 2: These two encodings are functionally equivalent.

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVSHDUP	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VMOVSHDUP	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVSLDUP	SSE3	CPUID Fn0000_0001_ECX[SSE3] (bit 0)
VMOVSLDUP	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

MOVSS Move
VMOVSS 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, bits [127:32] of the destination are not affected.

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.

Instruction Support

Form	Subset	Feature Flag
MOVSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.*

Mnemonic	Encoding[1]			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VMOVSS *xmm1*, *mem32*	C4	$\overline{\text{RXB}}$.00001	X.1111.X.10	10 /r
VMOVSS *mem32*, *xmm1*	C4	$\overline{\text{RXB}}$.00001	X.1111.X.10	11 /r
VMOVSS *xmm1, xmm2*, *xmm3* [2]	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	10 /r
VMOVSS *xmm1, xmm2*, *xmm3* [2]	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.X.10	11 /r

Note 1: The addressing mode differentiates between the two operand form (where one operand is a memory location) and the three operand form (where all operands are held in registers).

Note 2: These two encodings are functionally equivalent.

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVUPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMOVUPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	X	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

MOVUPS Move Unaligned
VMOVUPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MOVUPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMOVUPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	X	Memory operand not 16-byte aligned when alignment checking enabled.
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 8 or 16 sums of absolute differences of sequentially selected groups of four contiguous unsigned byte integers in the first source operand and a selected group of four contiguous unsigned byte integers in a second source operand and writes the eight or sixteen 16-bit unsigned integer sums to sequential words of the destination register. The 256-bit form of the instruction additionally performs a similar but independent calculation using the upper 128 bits of the source operands.

Figure 2-2 on page 238 provides a graphical representation of the operation of the instruction. The following description accompanies it.

The computation uses as inputs 11 bytes from the first source operand and 4 bytes in the second source operand. Bit fields in the imm8 operand specify the index of the right-most byte of each group.

Bits [1:0] of the immediate operand determine the index of the right-most byte of four contiguous bytes within the second source operand used in the operation that produces the result (or, in the case of the 256-bit form of the instruction, the lower 128 bits of the result). Bit 2 of the immediate operand determines the right-most index of the 11contiguous bytes in the first source operand used in the same calculation. In the 128-bit form of the instruction, bits [7:3] of the immediate operand are ignored.

Bits [4:3] of the immediate operand determine the index of the right-most byte of four contiguous bytes within the second source operand used in the operation that produces the upper 128 bits of the result in the 256-bit form of the instruction. Bit 5 of the immediate operand determines the right-most index of the 11 contiguous bytes within in the upper half of the first 256-bit source operand used in the same calculation. In the 256-bit form of the instruction, bits [7:6] of the immediate operand are ignored.

Each word of the destination register receives the result of a separate computation of the sum of absolute differences function applied to a specific pair of four-element vectors derived from the source operands. The sum of absolute differences function SumAbsDiff (A, B) takes as input two 4-element unsigned 8-bit integer vectors and produces a single unsigned 16-bit integer result. The function is defined as:

SumAbsDiff(A, B) = | A[0]-B[0] | + | A[1]-B[1] | + | A[2]-B[2] | + | A[3]-B[3] |

The sum of absolute differences function produces a quantitative measure of the difference between two 4-element vectors. Each of the calculations that generates a result uses this metric to assess the difference between the selected 4-byte vector from operand 2 (B in the above equation) with each of eight overlapping 4-byte vectors (A in the equation) selected sequentially from the first source operand.

The right-most word (Word 0) of the destination receives the result of the comparison of the right-most 4 bytes of the selected group of 11 from operand 1 (src1[i1+3 : i1], as shown in the figure) to the selected 4 bytes from operand 2 (src2[j1+3:j1], in the figure). Word 1 of the destination receives the result of the comparison of the four bytes starting at an offset of 1 from the right-most byte of the group of 11 (src1[i1+4 : i1+1] in the figure) to the 4 bytes from operand 2. Word 2 of the destination receives the result of the comparison of the four bytes starting at an offset of 2 from the right-most byte of the group of 11 (src1[i1+5 : i1+2], in the figure) to the selected 4 bytes from operand 2. This continues in like manner until the left-most four bytes of the 11 are compared to the 4 bytes from operand 2 with the result being written to Word 7. This completes the generation of the lower 128 bits of the result.

The generation of the upper 128 bits of the result for the 256-bit form of the instruction is performed in like manner using separately selected groups of bytes from the upper half of the 256-bit operands, as described above.

The following is a more formal description of the operation of the (V)MPSADBW instruction:

For both the 128-bit and 256-bit form of the instruction, the following set of operations is performed:

```
src1 and src2 are byte vectors that overlay the first and second source operand respectively.
dest is a word vector that overlays the destination register.
tmp1[ ] is an array of 4-element vectors derived from the first source operand.
tmp2 and tmp3 are 4-element vectors derived from the second source operand.

i1 = imm8[2] * 4
j1= imm8[1:0] * 4

tmp1[0] = {src1[i1+3], src1[i1+2], src1[i1+1], src1[i1]}
tmp1[1] = {src1[i1+4], src1[i1+3], src1[i1+2], src1[i1+1]}
tmp1[2] = {src1[i1+5], src1[i1+4], src1[i1+3], src1[i1+2]}
tmp1[3] = {src1[i1+6], src1[i1+5], src1[i1+4], src1[i1+3]}
tmp1[4] = {src1[i1+7], src1[i1+6], src1[i1+5], src1[i1+4]}
tmp1[5] = {src1[i1+8], src1[i1+7], src1[i1+6], src1[i1+5]}
tmp1[6] = {src1[i1+9], src1[i1+8], src1[i1+7], src1[i1+6]}
tmp1[7] = {src1[i1+10], src1[i1+9], src1[i1+8], src1[i1+7]}
tmp2 = {src2[j1+3], src2[j1+2], src2[j1+1], src2[j1]}

dest[0] = SumAbsDiff(tmp1[0], tmp2)
dest[1] = SumAbsDiff(tmp1[1], tmp2)
dest[2] = SumAbsDiff(tmp1[2], tmp2)
dest[3] = SumAbsDiff(tmp1[3], tmp2)
dest[4] = SumAbsDiff(tmp1[4], tmp2)
dest[5] = SumAbsDiff(tmp1[5], tmp2)
dest[6] = SumAbsDiff(tmp1[6], tmp2)
dest[7] = SumAbsDiff(tmp1[7], tmp2)
```

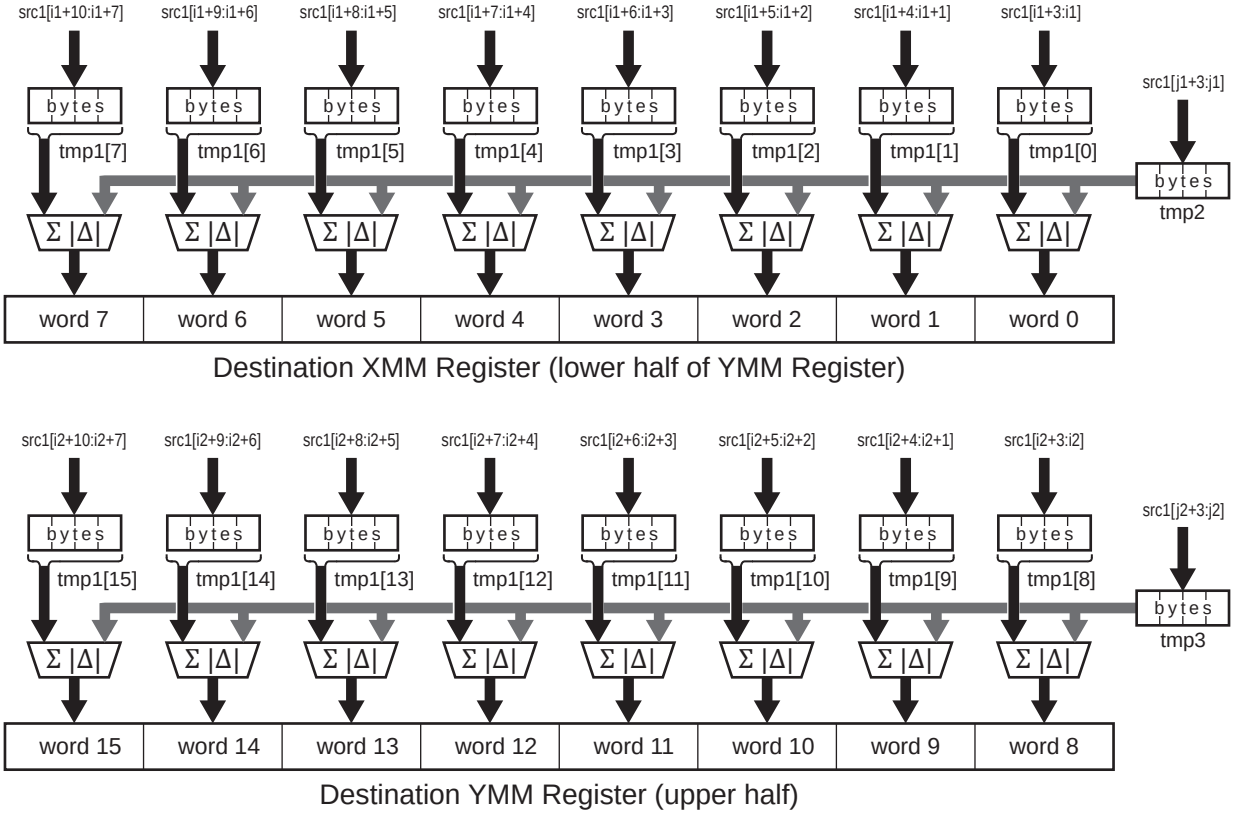
Additionally, for the 256-bit form of the instruction, the following set of operations is performed:

```
i2 = imm8[5] * 4 + 16
j2= imm8[4:3] * 4 +16

tmp1[8] = {src1[i2+3], src1[i2+2], src1[i2+1], src1[i2]}
tmp1[9] = {src1[i2+4], src1[i2+3], src1[i2+2], src1[i2+1]}
tmp1[10] = {src1[i2+5], src1[i2+4], src1[i2+3], src1[i2+2]}
tmp1[11] = {src1[i2+6], src1[i2+5], src1[i2+4], src1[i2+3]}
tmp1[12] = {src1[i2+7], src1[i2+6], src1[i2+5], src1[i2+4]}
tmp1[13] = {src1[i2+8], src1[i2+7], src1[i2+6], src1[i2+5]}
tmp1[14] = {src1[i2+9], src1[i2+8], src1[i2+7], src1[i2+6]}
tmp1[15] = {src1[i2+10], src1[i2+9], src1[i2+8], src1[i2+7]}
tmp3 = {src2[j2+3], src2[j2+2], src2[j2+1], src2[j2]}

dest[8] = SumAbsDiff(tmp1[8], tmp3)
dest[9] = SumAbsDiff(tmp1[9], tmp3)
dest[10] = SumAbsDiff(tmp1[10], tmp3)
dest[11] = SumAbsDiff(tmp1[11], tmp3)
```

dest[12] = SumAbsDiff(tmp1[12], tmp3)
dest[13] = SumAbsDiff(tmp1[13], tmp3)
dest[14] = SumAbsDiff(tmp1[14], tmp3)
dest[15] = SumAbsDiff(tmp1[15], tmp3)



Notes:
- i1 is a byte offset into source operand 1 (i1 = imm8[2] * 4).
- j1 is a byte offset into source operand 2 (j1 = imm8[1:0] * 4)
- i2 is a second byte offset into source operand 1 (i2 = imm8[5] * 4 + 16)
- j2 is a second byte offset into source operand 2 (j2 = imm8[4:3] * 4 + 16)
- Σ |Δ| represents the sum of absolute differences function which operates on two 4-element unsigned packed byte values and produces an unsigned 16-bit integer.

MPSADBW_instruct2.eps

Figure 2-2. (V)MPSADBW Instruction

There are legacy and extended forms of the instruction:

MPSADBW

The first source operand is an XMM register. 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.

VMPSADBW

The extended form of the instruction has 128-bit and 256-bit encodings:

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. 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. The second source operand is either a YMM register or a 256-bit memory location. The destination is a third YMM register. Bits [127:0] of the destination receive the results of the first 8 sums of absolute differences calculation using the selected bytes of the lower halves of the two source operands. Bits [255:128] of the destination receive the results of the second 8 sums of absolute differences calculation using selected bytes of the upper halves of the two source operands.

Instruction Support

Form	Subset	Feature Flag
MPSADBW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VMPSADBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VMPSADBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VMPSADBW *xmm1*, *xmm2*, *xmm3*/*mem128, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{\textit{src1}}$.0.01	42 /r ib
VMPSADBW *ymm1*, *ymm2*, *ymm3*/*mem256, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{\textit{src1}}$.1.01	42 /r ib

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MULPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMULPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMULPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src}$.0.01	59 /r
VMULPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Multiply
VMULPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
MULPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMULPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMULPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.00	59 /r
VMULPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
MULSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VMULSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMULSD *xmm1*, *xmm2*, *xmm3*/*mem64*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.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	XFEATURE_ENABLED_MASK[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 when 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
MULSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VMULSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMULSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.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	XFEATURE_ENABLED_MASK[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 when 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 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.

VORPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ORPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VORPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VORPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	56 /r
VORPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	56 /r

Related Instructions

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

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 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.

VORPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
ORPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VORPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VORPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.00	56 /r
VORPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.00	56 /r

Related Instructions

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

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 or 32 packed 8-bit signed integers in the source operand. Each byte of the destination receives an unsigned 8-bit integer that is the absolute value of the signed 8-bit integer in the corresponding byte of the source operand.

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 an 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 and 256-bit encodings.

XMM Encoding

The source operand is 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

The source operand is a YMM register or a 256-bit memory location. The destination is a YMM register. All 32 bytes of the destination are written.

Instruction Support

Form	Subset	Feature Flag
PABSB	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPABSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPABSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPABSB *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	1C /r
VPABSB *ymm1, ymm2/mem256*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	1C /r

Related Instructions

(V)PABSW, (V)PABSD

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PABSD
VPABSD
Packed Absolute Value Signed Doubleword

Computes the absolute value of four or eight packed 32-bit signed integers in the source operand. Each doubleword of the destination receives an unsigned 32-bit integer that is the absolute value of the signed 32-bit integer in the corresponding doubleword of the source operand.

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 an 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 and 256-bit encodings.

XMM Encoding

The source operand is 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

The source operand is a YMM register or a 256-bit memory location. The destination is a YMM register. All four doublewords of the destination are written.

Instruction Support

Form	Subset	Feature Flag
PABSD	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPABSD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPABSD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPABSD *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	1E /r
VPABSD *ymm1, ymm2/mem256*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	1E /r

Related Instructions

(V)PABSB, (V)PABSW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PABSW VPABSW Packed Absolute Value Signed Word

Computes the absolute value of eight or sixteen packed 16-bit signed integers in the source operand. Each word of the destination receives an unsigned 16-bit integer that is the absolute value of the signed 16-bit integer in the corresponding word of the source operand.

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 an 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 and 256-bit encodings.

XMM Encoding

The source operand is 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

The source operand is a YMM register or a 256-bit memory location. The destination is a YMM register. All 16 words of the destination are written.

Instruction Support

Form	Subset	Feature Flag
PABSW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPABSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPABSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPABSW *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	1D /r
VPABSW *ymm1, ymm2/mem256*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	1D /r

Related Instructions

(V)PABSB, (V)PABSD

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PACKSSDW VPACKSSDW — Pack with Signed Saturation Doubleword to Word

Converts four or eight 32-bit signed integers from the first source operand and the second source operand into 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 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.

VPACKSSDW

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

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

Instruction Support

Form	Subset	Feature Flag
PACKSSDW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPACKSSDW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPACKSSDW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPACKSSDW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	6B /r
VPACKSSDW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	6B /r

Related Instructions

(V)PACKSSWB, (V)PACKUSDW, (V)PACKUSWB

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PACKSSWB VPACKSSWB — Pack with Signed Saturation Word to Byte

Converts eight or sixteen 16-bit signed integers from the first source operand and the second source operand into sixteen or thirty two 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 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.

VPACKSSWB

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

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

Instruction Support

Form	Subset	Feature Flag
PACKSSWB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPACKSSWB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPACKSSWB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPACKSSWB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	63 /r
VPACKSSWB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	63 /r

Related Instructions

(V)PACKSSDW, (V)PACKUSDW, (V)PACKUSWB

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PACKUSDW VPACKUSDW — Pack with Unsigned Saturation Doubleword to Word

Converts four or eight 32-bit signed integers from the first source operand and the second source operand into eight or sixteen 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 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.

VPACKUSDW

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

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

Instruction Support

Form	Subset	Feature Flag
PACKUSDW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPACKUSDW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPACKUSDW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPACKUSDW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	2B /r
VPACKUSDW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	2B /r

Related Instructions

(V)PACKSSDW, (V)PACKSSWB, (V)PACKUSWB

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PACKUSWB VPACKUSWB — Pack with Unsigned Saturation Word to Byte

Converts eight or sixteen 16-bit signed integers from the first source operand and the second source operand into sixteen or thirty two 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 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.

VPACKUSWB

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

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

Instruction Support

Form	Subset	Feature Flag
PACKUSWB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPACKUSWB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPACKUSWB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPACKUSWB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	67 /r
VPACKUSWB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	67 /r

Related Instructions

(V)PACKSSDW, (V)PACKSSWB, (V)PACKUSDW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDB VPADDB — Packed Add Bytes

Adds 16 or 32 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 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.

VPADDB

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

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

Instruction Support

Form	Subset	Feature Flag
PADDB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	FC /r
VPADDB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	FC /r

Related Instructions

(V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDD VPADDD Packed Add Doublewords

Adds 4 or 8 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 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.

VPADDD

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

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

Instruction Support

Form	Subset	Feature Flag
PADDD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	FE /r
VPADDD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	FE /r

Related Instructions

(V)PADDB, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDQ VPADDQ Packed Add Quadwords

Adds 2 or 4 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 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.

VPADDQ

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

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

Instruction Support

Form	Subset	Feature Flag
PADDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDQ *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src1}$.0.01	D4 /r
VPADDQ *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src1}$.1.01	D4 /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDSB VPADDSB Packed Add with Signed Saturation Bytes

Adds 16 or 32 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 7Fh; 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 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.

VPADDSB

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

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

Instruction Support

Form	Subset	Feature Flag
PADDSB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDSB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	EC /r
VPADDSB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	EC /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSW, (V)PADDUSB, (V)PADDUSW, (V)PADDW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDSW VPADDSW Packed Add with Signed Saturation Words

Adds 8 or 16 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 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.

VPADDSW

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

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

Instruction Support

Form	Subset	Feature Flag
PADDSW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDSW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src1}}$.0.01	ED /r
VPADDSW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{\textit{src1}}$.1.01	ED /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDUSB, (V)PADDUSW, (V)PADDW

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDUSB VPADDUSB Packed Add with Unsigned Saturation Bytes

Adds 16 or 32 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 FFh are saturated to FFh.

There are legacy and extended forms of the instruction:

PADDUSB

The first source operand is an XMM register. 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.

VPADDUSB

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

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

Instruction Support

Form	Subset	Feature Flag
PADDUSB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDUSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDUSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDUSB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	DC /r
VPADDUSB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	DC /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSW, (V)PADDW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDUSW VPADDUSW Packed Add with Unsigned Saturation Words

Adds 8 or 16 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.

There are legacy and extended forms of the instruction:

PADDUSW

The first source operand is an XMM register. 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.

VPADDUSW

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

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

Instruction Support

Form	Subset	Feature Flag
PADDUSW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDUSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDUSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDUSW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	DD /r
VPADDUSW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	DD /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PADDW VPADDW — Packed Add Words

Adds or 16 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 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.

VPADDW

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

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

Instruction Support

Form	Subset	Feature Flag
PADDW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPADDW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPADDW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPADDW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	FD /r
VPADDW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	FD /r

Related Instructions

(V)PADDB, (V)PADDD, (V)PADDQ, (V)PADDSB, (V)PADDSW, (V)PADDUSB, (V)PADDUSW

RFlags Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PALIGNR Packed Align Right
VPALIGNR

Concatenates one or two pairs of 16-byte values from the first and second source operands and right-shifts the concatenated values the number of bytes specified by the unsigned immediate operand. Writes the least-significant 16 bytes of the shifted result to the destination or writes the least-significant 16 bytes of the two shifted results to the upper and lower halves of the destination.

For the 128-bit form of the instruction, the first and second 128-bit source operands are concatenated to form a temporary 256-bit value with the first source operand occupying the most-significant half of the temporary value. After the right-shift operation, the lower 128 bits of the result are written to the destination.

For the 256-bit form of the instruction, the lower 16 bytes of the first and second source operands are concatenated to form a first temporary 256-bit value with the bytes from the first source operand occupying the most-significant half of the temporary value. The upper 16 bytes of the first and second source operands are concatenated to form a second temporary 256-bit value with the bytes from the first source operand occupying the most-significant half of the second temporary value. Both temporary values are right-shifted the number of bytes specified by the immediate operand. After the right-shift operation, the lower 16 bytes of the first temporary value are written to the lower 128 bits of the destination and the lower 16 bytes of the second temporary value are written to the upper 128 bits of the destination.

The binary value of the immediate operand determines the byte shift value. On each shift the most-significant byte is set to zero. When the byte shift value is greater than 31, 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 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.

VPALIGNR

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

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

Instruction Support

Form	Subset	Feature Flag
PALIGNR	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPALIGNR 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPALIGNR 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPALIGNR *xmm1, xmm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{src1}$.0.01	0F /r ib
VPALIGNR *ymm1, ymm2, ymm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{src1}$.1.01	0F /r ib

Related Instructions

None

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PAND Packed AND
VPAND

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 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.

VPAND

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

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

Instruction Support

Form	Subset	Feature Flag
PAND	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPAND 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPAND 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPAND *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	DB /r
VPAND *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	DB /r

Related Instructions

(V)PANDN, (V)POR, (V)PXOR

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PANDN Packed AND NOT
VPANDN

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 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.

VPANDN

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

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

Instruction Support

Form	Subset	Feature Flag
PANDN	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPANDN 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPANDN 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPANDN *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src}$.0.01	DF /r
VPANDN *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src}$.1.01	DF /r

Related Instructions

(V)PAND, (V)POR, (V)PXOR

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PAVGB
VPAVGB
Packed Average Unsigned Bytes

Computes the rounded averages of 16 or 32 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 8-bit integer values in corresponding positions in the two 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 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.

VPAVGB

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

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

Instruction Support

Form	Subset	Feature Flag
PAVGB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPAVGB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPAVGB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPAVGB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E0 /r
VPAVGB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E0 /r

Related Instructions

PAVGW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 16-bit integer values in corresponding positions in the two 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 an 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 has128-bit and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PAVGW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPAVGW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPAVGW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPAVGW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E3 /r
VPAVGW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E3 /r

Related Instructions

(V)PAVGB

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PBLENDVB Variable Blend
VPBLENDVB 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 most significant bit 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 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. The mask operand is the implicit register XMM0.

VPBLENDVB

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

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. 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 by bits [7:4] of an immediate byte.

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. The destination is a third YMM register. The mask operand is a fourth YMM register selected by bits [7:4] of an immediate byte.

Instruction Support

Form	Subset	Feature Flag
PBLENDVB	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPBLENDVB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPBLENDVB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPBLENDVB *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	4C /r is4
VPBLENDVB *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	4C /r is4

Related Instructions

(V)BLENDVPD, (V)BLENDVPS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PBLENDW Blend
VPBLENDW Packed Words

Copies packed words from either of two sources to a destination, as specified by an immediate 8-bit mask operand. For the 256-bit form, the same 8-bit mask is applied twice; once to select words to be written to the lower 128 bits of the destination and again to select words to be written to the upper 128 bits of the destination.

Each bit of the mask selects a word from one of the source operands based on the position of the word within the operand. Bit 0 of the mask selects the least-significant word (word 0) to be copied, bit 1 selects the next-most significant word (word 1), and so forth. Bit 7 selects word 7 (the most-significant word for 128-bit operands).

For the 256-bit operands, the mask is reused to select words in the upper 128-bits of the source operands to be copied. Bit 0 of the mask selects word 8, bit 1 selects word 9, and so forth. Finally, bit 7 of the mask selects the word from position 15.

- 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 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.

VPBLENDW

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

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

Instruction Support

Form	Subset	Feature Flag
PBLENDW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPBLENDW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPBLENDW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPBLENDW *xmm1, xmm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{src1}$.0.01	0E /r /ib
VPBLENDW *ymm1, ymm2, ymm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{src1}$.1.01	0E /r /ib

Related Instructions

(V)BLENDPD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 Appendix A on page 973 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 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.

VPCLMULQDQ

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

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.

Instruction Support

Form	Subset	Feature Flag
PCLMULQDQ	PCLMULQDQ	CPUID Fn0000_0001_ECX[PCLMULQDQ] (bit 1)
VPCLMULQDQ	AVX or PCLMULQDQ	CPUID Fn0000_0001_ECX[PCLMULQDQ] (bit 1) or CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCLMULQDQ *xmm1*, *xmm2*, *xmm3/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.00011	X.$\overline{src}$.0.01	44 /r ib

Related Instructions

(V)PMULDQ, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 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 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.

VPCMPEQB

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

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

Instruction Support

Form	Subset	Feature Flag
PCMPEQB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPCMPEQB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPEQB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPEQB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	74 /r
VPCMPEQB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	74 /r

Related Instructions

(V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPEQD VPCMPEQD — Packed Compare Equal Doublewords

Compares 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 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.

VPCMPEQD

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

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

Instruction Support

Form	Subset	Feature Flag
PCMPEQD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPCMPEQD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPEQD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPEQD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	76 /r
VPCMPEQD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	76 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPEQQ VPCMPEQQ — Packed Compare Equal Quadwords

Compares 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 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.

VPCMPEQQ

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

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

Instruction Support

Form	Subset	Feature Flag
PCMPEQQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPCMPEQQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPEQQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPEQQ *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	29 /r
VPCMPEQQ *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	29 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPEQW VPCMPEQW — Packed Compare Equal Words

Compares 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 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.

VPCMPEQW

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

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

Instruction Support

Form	Subset	Feature Flag
PCMPEQW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPCMPEQW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPEQW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPEQW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	75 /r
VPCMPEQW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	75 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPGTB, (V)PCMPGTD, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPESTRI Packed Compare
VPCMPESTRI Explicit Length Strings Return Index

Compares character string data in the first and second source operands. Comparison operations are carried out as specified by values encoded in the immediate operand. Writes an index to the ECX register.

Source operands are formatted as a packed characters in one of two supported widths: 8 or 16 bits. Characters may be treated as either signed or unsigned values. Each operand has associated with it a separate integer value specifying the length of the string.

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.

If the absolute value of the data in either register is greater than the maximum string length that fits in 128 bits, the length is set to the maximum: 8, for 16-bit characters, or 16, for 8-bit characters.

The comparison operations between the two operand strings are summarized in an intermediate result—a comparison summary bit vector that is post-processed to produce the final output. Data fields within the immediate byte specify the source data format, comparison type, comparison summary bit vector post-processing, and output option selection.

The index of either the most significant or least significant set bit of the post-processed comparison summary bit vector is returned in ECX. If no bits are set in the post-processed comparison summary bit vector, ECX is set to 16 for source operand strings composed of 8-bit characters or 8 for 16-bit character strings.

See Section 1.5, “String Compare Instructions” for information about source string data format, comparison operations, comparison summary bit vector generation, post-processing, and output selection options.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if the comparison summary bit vector is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	Set if the specified length of the second string is less than the maximum; otherwise cleared.
SF	Set if the specified length of the first string is less than the maximum; otherwise cleared.
OF	Equal to the value of the lsb of the post-processed comparison summary bit vector.

There are legacy and extended forms of the instruction:

PCMPESTRI

The first source operand is an XMM register. The second source operand is either an 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 a 128-bit encoding only.

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. A result index is written to the ECX register.

Instruction Support

Form	Subset	Feature Flag
PCMPESTRI	SSE4.2	CPUID Fn0000_0001_ECX[SSE42] (bit 20)
VPCMPESTRI	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	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 that is set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

PCMPESTRM Packed Compare
VPCMPESTRM Explicit Length Strings Return Mask

Compares character string data in the first and second source operands. Comparison operations are carried out as specified by values encoded in the immediate operand. Writes a mask value to the YMM0/XMM0 register.

Source operands are formatted as a packed characters in one of two supported widths: 8 or 16 bits. Characters may be treated as either signed or unsigned values. Each operand has associated with it a separate integer value specifying the length of the string.

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.

If the absolute value of the data in either register is greater than the maximum string length that fits in 128 bits, the length is set to the maximum: 8, for 16-bit characters, or 16, for 8-bit characters.

The comparison operations between the two operand strings are summarized in an intermediate result—a comparison summary bit vector that is post-processed to produce the final output. Data fields within the immediate byte specify the source data format, comparison type, comparison summary bit vector post-processing, and output option selection.

Depending on the output option selected, the post-processed comparison summary bit vector is either zero-extended to 128 bits or expanded into a byte/word-mask and then written to XMM0.

See Section 1.5, “String Compare Instructions” for information about source string data format, comparison operations, comparison summary bit vector generation, post-processing, and output selection options.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if the comparison summary bit vector is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	Set if the specified length of the second string is less than the maximum; otherwise cleared.
SF	Set if the specified length of the first string is less than the maximum; otherwise cleared.
OF	Equal to the value of the lsb of the post-processed summary bit vector.

There are legacy and extended forms of the instruction:

PCMPESTRM

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. The mask result is written to the XMM0 register.

VPCMPESTRM

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

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. The mask result is written to the XMM0 register. Bits [255:128] of the YMM0 register are cleared.

Instruction Support

Form	Subset	Feature Flag
PCMPESTRM	SSE4.2	CPUID Fn0000_0001_ECX[SSE42] (bit 20)
VPCMPESTRM	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	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 to 0 is M (modified). Unaffected flags are blank. Undefined flags are U.*

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 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 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.

VPCMPGTB

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

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

Instruction Support

Form	Subset	Feature Flag
PCMPGTB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPCMPGTB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPGTB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PCMPGTB *xmm1*, *xmm2/mem128*	66 0F 64 /r	Compares packed bytes in *xmm1* to packed bytes in *xmm2* or *mem128*. Writes results to *xmm1.*

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPCMPGTB *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	64 /r
VPCMPGTB *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	64 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTD, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPGTD VPCMPGTD Packed Compare Greater Than Signed Doublewords

Compares 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 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.

VPCMPGTD

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

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

Instruction Support

Form	Subset	Feature Flag
PCMPGTD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPCMPGTD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPGTD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPGTD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	66 /r
VPCMPGTD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	66 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPGTQ VPCMPGTQ Packed Compare Greater Than Signed Quadwords

Compares 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 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.

VPCMPGTQ

The extended form of the instruction has 128-bit and 256-bit encodings:

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

Instruction Support

Form	Subset	Feature Flag
PCMPGTQ	SSE4.2	CPUID Fn0000_0001_ECX[SSE42] (bit 20)
VPCMPGTQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPGTD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPGTQ *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	37 /r
VPCMPGTQ *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	37 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPGTW Packed Compare Greater Than Signed Words
VPCMPGTW

Compares packed signed 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 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 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.

VPCMPGTW

The extended form of the instruction has 128-bit and 256-bit encodings:

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

Instruction Support

Form	Subset	Feature Flag
PCMPGTW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPCMPGTW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPCMPGTW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPGTW *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	65 /r
VPCMPGTW *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	65 /r

Related Instructions

(V)PCMPEQB, (V)PCMPEQD, (V)PCMPEQW, (V)PCMPGTB, (V)PCMPGTD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PCMPISTRI Packed Compare
VPCMPISTRI Implicit Length Strings Return Index

Compares character string data in the first and second source operands. Comparison operations are carried out as specified by values encoded in the immediate operand. Writes an index to the ECX register.

Source operands are formatted as a packed characters in one of two supported widths: 8 or 16 bits. Characters may be treated as either signed or unsigned values.

Source operand strings shorter than the maximum that can be packed into a 128-bit value are terminated by a null character (value of 0). The characters prior to the null character constitute the string. If the first (lowest indexed) character is null, the string length is 0.

The comparison operations between the two operand strings are summarized in an intermediate result—a comparison summary bit vector that is post-processed to produce the final output. Data fields within the immediate byte specify the source data format, comparison type, comparison summary bit vector post-processing, and output option selection.

The index of either the most significant or least significant set bit of the post-processed comparison summary bit vector is returned in ECX. If no bits are set in the post-processed comparison summary bit vector, ECX is set to 16 for source operand strings composed of 8-bit characters or 8 for 16-bit character strings.

See Section 1.5, “String Compare Instructions” for information about source string data format, comparison operations, comparison summary bit vector generation, post-processing, and output selection options.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if the comparison summary bit vector is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	Set if any byte (word) in the second operand is null; otherwise cleared.
SF	Set if any byte (word) in the first operand is null; otherwise cleared
OF	Equal to the value of the lsb of the post-processed summary bit vector.

There are legacy and extended forms of the instruction:

PCMPISTRI

The first source operand is an XMM register. The second source operand is either an 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 a 128-bit encoding only.

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. A result index is written to the ECX register.

Instruction Support

Form	Subset	Feature Flag
PCMPISTRI	SSE4.2	CPUID Fn0000_0001_ECX[SSE42] (bit 20)
VPCMPISTRI	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PCMPISTRI *xmm1*, *xmm2/mem128, imm8*	66 0F 3A 63 /r ib	Compares packed string data in *xmm1* and *xmm2* or *mem128*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPCMPISTRI *xmm1, xmm2/mem128,* imm8	C4	$\overline{\text{RXB}}$.03	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	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 that is set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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. Comparison operations are carried out as specified by values encoded in the immediate operand. Writes a mask value to the YMM0/XMM0 register

Source operands are formatted as a packed characters in one of two supported widths: 8 or 16 bits. Characters may be treated as either signed or unsigned values.

Source operand strings shorter than the maximum that can be packed into a 128-bit value are terminated by a null character (value of 0). The characters prior to the null character constitute the string. If the first (lowest indexed) character is null, the string length is 0.

The comparison operations between the two operand strings are summarized in an intermediate result—a comparison summary bit vector that is post-processed to produce the final output. Data fields within the immediate byte specify the source data format, comparison type, comparison summary bit vector post-processing, and output option selection.

Depending on the output option selected, the post-processed comparison summary bit vector is either zero-extended to 128 bits or expanded into a byte/word-mask and then written to XMM0.

See Section 1.5, “String Compare Instructions” for information about source string data format, comparison operations, comparison summary bit vector generation, post-processing, and output selection options.

The rFLAGS are set to indicate the following conditions:

Flag	Condition
CF	Cleared if the comparison summary bit vector is zero; otherwise set.
PF	cleared.
AF	cleared.
ZF	Set if any byte (word) in the second operand is null; otherwise cleared.
SF	Set if any byte (word) in the first operand is null; otherwise cleared.
OF	Equal to the value of the lsb of the post-processed summary bit vector.

There are legacy and extended forms of the instruction:

PCMPISTRM

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. The mask result is written to the XMM0 register.

VPCMPISTRM

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

The first source operand is an XMM register. The second source operand is either an XMM register or a 128-bit memory location. The mask result is written to the XMM0 register. Bits [255:128] of the YMM0 register are cleared.

Instruction Support

Form	Subset	Feature Flag
PCMPISTRM	SSE4.2	CPUID Fn0000_0001_ECX[SSE42] (bit 20)
VPCMPISTRM	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPCMPISTRM *xmm1, xmm2/mem128,* imm8	C4	$\overline{\text{RXB}}$.03	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	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 that is set or cleared is M (modified). Unaffected flags are blank. Undefined flags are U.*

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

PEXTRB Extract
VPEXTRB 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PEXTRB	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPEXTRB	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPEXTRB *reg/mem8*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.0.01	14 /r ib

Related Instructions

(V)PEXTRD, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

PEXTRD Extract
VPEXTRD 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PEXTRD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPEXTRD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPEXTRD *reg32/mem32*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.0.01	16 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRW, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

PEXTRQ Extract
VPEXTRQ 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PEXTRD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPEXTRD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPEXTRQ *reg64/mem64*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	1.1111.0.01	16 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRW, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

PEXTRW Extract Packed Word
VPEXTRW

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.

Instruction Support

Form	Subset	Feature Flag
PEXTRW reg	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
PEXTRW reg/mem16	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPEXTRW	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPEXTRW *reg*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.0.01	C5 /r ib
VPEXTRW *reg/mem16*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.0.01	15 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PINSRB, (V)PINSRD, (V)PINSRW, (V)PINSRQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

PHADDD Packed Horizontal Add
VPHADDD Doubleword

Adds adjacent 32-bit signed integers in each of two source operands and packs the sums into the 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.

Adds the 32-bit signed integer values in bits [63:32] and bits [31:0] of the first source operand 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 operand and packs the sum into bits [63:32] of the destination. Adds the corresponding values in the second source operand and packs the sums into bits [95:64] and [127:96] of the destination.

Additionally, for the 256-bit form, adds the 32-bit signed integer values in bits [191:160] and bits [159:128] of the first source operand and packs the sum into bits [159:128] of the destination; adds the 32-bit signed integer values in bits [255:224] and bits [223:192] of the first source operand and packs the sum into bits [191:160] of the destination. Adds the corresponding values in the second source operand and packs the sums into bits [223:192] and [255:224] of the destination.

There are legacy and extended forms of the instruction:

PHADDD

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

VPHADDD

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

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

Instruction Support

Form	Subset	Feature Flag
PHADDD	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPHADDD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPHADDD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPHADDD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	02 /r
VPHADDD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	02 /r

Related Instructions

(V)PHADDW, (V)PHADDSW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PHADDSW VPHADDSW Packed Horizontal Add with Saturation Word

Adds adjacent 16-bit signed integers in each of two source operands, with saturation, and packs the 16-bit signed sums into the destination.

Positive sums greater than 7FFFh are saturated to 7FFFh; negative sums less than 8000h are saturated to 8000h.

For the 128-bit form of the instruction, the following operations are performed:

```
dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.
Ssum() is a function that returns the saturated 16-bit signed sum of its arguments.

dest[15:0] = Ssum(src1[31:16], src1[15:0])
dest[31:16] = Ssum(src1[63:48], src1[47:32])
dest[47:32] = Ssum(src1[95:80], src1[79:64])
dest[63:48] = Ssum(src1[127:112], src1[111:96])
dest[79:64] = Ssum(src2[31:16], src2[15:0])
dest[95:80] = Ssum(src2[63:48], src2[47:32])
dest[111:96] = Ssum(src2[95:80], src2[79:64])
dest[127:112] = Ssum(src2[127:112], src2[111:96])
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = Ssum(src1[159:144], src1[143:128])
dest[159:144] = Ssum(src1[191:176], src1[175:160])
dest[175:160] = Ssum(src1[223:208], src1[207:192])
dest[191:176] = Ssum(src1[255:240], src1[239:224])
dest[207:192] = Ssum(src2[159:144], src2[143:128])
dest[223:208] = Ssum(src2[191:176], src2[175:160])
dest[239:224] = Ssum(src2[223:208], src2[207:192])
dest[255:240] = Ssum(src2[255:240], src2[239:224])
```

There are legacy and extended forms of the instruction:

PHADDSW

The first source operand is an XMM register. 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.

VPHADDSW

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

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

Instruction Support

Form	Subset	Feature Flag
PHADDSW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPHADDSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPHADDSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPHADDSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	03 /r
VPHADDSW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	03 /r

Related Instructions

(V)PHADDD, (V)PHADDW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PHADDW VPHADDW — Packed Horizontal Add Word

Adds adjacent 16-bit signed integers in each of two source operands and packs the 16-bit sums into the destination. If a sum overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set).

For the 128-bit form of the instruction, the following operations are performed:

dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.

```
dest[15:0] = src1[31:16] + src1[15:0]
dest[31:16] = src1[63:48] + src1[47:32]
dest[47:32] = src1[95:80] + src1[79:64]
dest[63:48] = src1[127:112] + src1[111:96]
dest[79:64] = src2[31:16] + src2[15:0]
dest[95:80] = src2[63:48] + src2[47:32]
dest[111:96] = src2[95:80] + src2[79:64]
dest[127:112] = src2[127:112] + src2[111:96]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = src1[159:144] + src1[143:128]
dest[159:144] = src1[191:176] + src1[175:160]
dest[175:160] = src1[223:208] + src1[207:192]
dest[191:176] = src1[255:240] + src1[239:224]
dest[207:192] = src2[159:144] + src2[143:128]
dest[223:208] = src2[191:176] + src2[175:160]
dest[239:224] = src2[223:208] + src2[207:192]
dest[255:240] = src2[255:240] + src2[239:224]
```

There are legacy and extended forms of the instruction:

PHADDW

The first source operand is an XMM register. 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.

VPHADDW

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

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

Instruction Support

Form	Subset	Feature Flag
PHADDW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPHADDW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPHADDW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPHADDW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	01 /r
VPHADDW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	01 /r

Related Instructions

(V)PHADDD, (V)PHADDSW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PHMINPOSUW Horizontal Minimum and Position
VPHMINPOSUW

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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PHMINPOSUW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPHMINPOSUW	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPHMINPOSUW *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	41 /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUD, (V)PMINUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 32-bit signed integers in each of two source operands and packs the differences into the destination. The higher-order doubleword of each pair is subtracted from the lower-order doubleword.

Subtracts the 32-bit signed integer value in bits [63:32] of the first source operand from the 32-bit signed integer value in bits [31:0] of the first source operand and packs the difference into bits [31:0] of the destination; subtracts the 32-bit signed integer value in bits [127:96] of the first source operand from the 32-bit signed integer value in bits [95:64] of the first source operand and packs the difference into bits [63:32] of the destination. Performs the corresponding operations on pairs of 32-bit signed integer values in the second source operand and packs the differences into bits [95:64] and [127:96] of the destination.

Additionally, for the 256-bit form, subtracts the 32-bit signed integer value in bits [191:160] of the first source operand from the 32-bit signed integer value in bits [159:128] of the first source operand and packs the difference into bits [159:128] of the destination; subtracts the 32-bit signed integer value in bits [255:224] of the first source operand from the 32-bit integer value in bits [223:192] of the first source operand and packs the difference into bits [191:160] of the destination. Performs the corresponding operations on pairs of 32-bit signed integer values in the second source operand and packs the differences into bits [223:192] and [255:224] of the destination.

There are legacy and extended forms of the instruction:

PHSUBD

The first source operand is an XMM register. 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.

VPHSUBD

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

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

Instruction Support

Form	Subset	Feature Flag
PHSUBD	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPHSUBD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPHSUBD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PHSUBD *xmm1*, *xmm2/mem128*	66 0F 38 06 /r	Subtracts adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*. Writes packed differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPHSUBD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	06 /r
VPHSUBD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	06 /r

Related Instructions

(V)PHSUBW, (V)PHSUBSW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PHSUBSW Packed Horizontal Subtract with Saturation
VPHSUBSW Word

Subtracts adjacent 16-bit signed integers in each of two source operands, with saturation, and packs the differences into the 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.

For the 128-bit form of the instruction, the following operations are performed:

```
dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.
Sdiff(A,B) is a function that returns the saturated 16-bit signed difference A – B.

dest[15:0] = Sdiff(src1[15:0], src1[31:16])
dest[31:16] = Sdiff(src1[47:32], src1[63:48])
dest[47:32] = Sdiff(src1[79:64], src1[95:80])
dest[63:48] = Sdiff(src1[111:96], src1[127:112])
dest[79:64] = Sdiff(src2[15:0], src2[31:16])
dest[95:80] = Sdiff(src2[47:32], src2[63:48])
dest[111:96] = Sdiff(src2[79:64], src2[95:80])
dest[127:112] = Sdiff(src2[111:96], src2[127:112])
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = Sdiff(src1[143:128], src1[159:144])
dest[159:144] = Sdiff(src1[175:160], src1[191:176])
dest[175:160] = Sdiff(src1[207:192], src1[223:208])
dest[191:176] = Sdiff(src1[239:224], src1[255:240])
dest[207:192] = Sdiff(src2[143:128], src2[159:144])
dest[223:208] = Sdiff(src2[175:160], src2[191:176])
dest[239:224] = Sdiff(src2[207:192], src2[223:208])
dest[255:240] = Sdiff(src2[239:224], src2[255:240])
```

There are legacy and extended forms of the instruction:

PHSUBSW

The first source operand is an XMM register. 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.

VPHSUBSW

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

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

Instruction Support

Form	Subset	Feature Flag
PHSUBSW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPHSUBSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPHSUBSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPHSUBSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	07 /r
VPHSUBSW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	07 /r

Related Instructions

(V)PHSUBD, (V)PHSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PHSUBW VPHSUBW — Packed Horizontal Subtract Word

Subtracts adjacent 16-bit signed integers in each of two source operands and packs the differences into a destination. The higher-order word of each pair is subtracted from the lower-order word.

For the 128-bit form of the instruction, the following operations are performed:

dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.

```
dest[15:0] = src1[15:0] − src1[31:16
dest[31:16] = src1[47:32] − src1[63:48]
dest[47:32] = src1[79:64] − src1[95:80]
dest[63:48] = src1[111:96] − src1[127:112]
dest[79:64] = src2[15:0] − src2[31:16]
dest[95:80] = src2[47:32] − src2[63:48]
dest[111:96] = src2[79:64] − src2[95:80]
dest[127:112] = src2[111:96] − src2[127:112]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = src1[143:128] − src1[159:144]
dest[159:144] = src1[175:160] − src1[191:176]
dest[175:160] = src1[207:192] − src1[223:208]
dest[191:176] = src1[239:224] − src1[255:240]
dest[207:192] = src2[143:128] − src2[159:144]
dest[223:208] = src2[175:160] − src2[191:176]
dest[239:224] = src2[207:192] − src2[223:208]
dest[255:240] = src2[239:224] − src2[255:240]
```

There are legacy and extended forms of the instruction:

PHSUBW

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

VPHSUBW

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

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

Instruction Support

Form	Subset	Feature Flag
PHSUBW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPHSUBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPHSUBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PHSUBW *xmm1*, *xmm2/mem128*	66 0F 38 05 /r	Subtracts adjacent pairs of signed integers in *xmm1* and *xmm2* or *mem128*. Writes packed differences to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPHSUBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	05 /r
VPHSUBW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	05 /r

Related Instructions

(V)PHSUBD, (V)PHSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PINSRB	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPINSRB	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPINSRB *xmm*, *reg/mem8*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.0.01	20 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRD, (V)PINSRQ, (V)PINSRW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PINSRD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPINSRD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPINSRD *xmm*, *reg32/mem32*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.0.01	22 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRB, (V)PINSRQ, (V)PINSRW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
PINSRQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPINSRQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPINSRQ *xmm*, *reg64/mem64*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.03	1.1111.0.01	22 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRB, (V)PINSRD, (V)PINSRW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 a 128-bit encoding only.

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 an 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.

Instruction Support

Form	Subset	Feature Flag
PINSRW	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VPINSRW	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPINSRW *xmm*, *reg32/mem16*, *xmm, imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.0.01	C4 /r ib

Related Instructions

(V)PEXTRB, (V)PEXTRD, (V)PEXTRQ, (V)PEXTRW, (V)PINSRB, (V)PINSRD, (V)PINSRQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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 sets of two packed 8-bit unsigned values from the first source operand and two packed 8-bit signed values from the second source operand, with signed saturation; writes eight 16-bit sums to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.
Ssum() is a function that returns the saturated 16-bit signed sum of its arguments.

dest[15:0] = Ssum(src1[7:0] * src2[7:0], src1[15:8] * src2[15:8])
dest[31:16] = Ssum(src1[23:16] * src2[23:16], src1[31:24] * src2[31:24])
dest[47:32] = Ssum(src1[39:32] * src2[39:32], src1[47:40] * src2[47:40])
dest[63:48] = Ssum(src1[55:48] * src2[55:48], src1[63:56] * src2[63:56])
dest[79:64] = Ssum(src1[71:64] * src2[71:64], src1[79:72] * src2[79:72])
dest[95:80] = Ssum(src1[87:80] * src2[87:80], src1[95:88] * src2[95:88])
dest[111:96] = Ssum(src1[103:96] * src2[103:96]], src1[111:104] * src2[111:104])
dest[127:112] = Ssum(src1[119:112] * src2[119:112], src1[127:120] * src2[127:120])
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = Ssum(src1[135:128] * src2[135:128], src1[143:136] * src2[143:136])
dest[159:144] = Ssum(src1[151:144] * src2[151:144], src1[159:152] * src2[159:152])
dest[175:160] = Ssum(src1[167:160] * src2[167:160], src1[175:168] * src2[175:168])
dest[191:176] = Ssum(src1[183:176] * src2[183:176], src1[191:184] * src2[191:184])
dest[207:192] = Ssum(src1[199:192] * src2[199:192], src1[207:200] * src2[207:200])
dest[223:208] = Ssum(src1[215:208] * src2[215:208], src1[223:216] * src2[223:216])
dest[239:224] = Ssum(src1[231:224] * src2[231:224], src1[239:232] * src2[239:232])
dest[255:240] = Ssum(src1[247:240] * src2[247:240], src1[255:248] * src2[255:248])
```

There are legacy and extended forms of the instruction:

PMADDUBSW

The first source operand is an XMM register. 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.

VPMADDUBSW

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

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

Instruction Support

Form	Subset	Feature Flag
PMADDUBSW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPMADDUBSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMADDUBSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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* / *mem128*, adds the products, and writes saturated sums to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPMADDUBSW *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	04 /r
VPMADDUBSW *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	04 /r

Related Instructions

(V)PMADDWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMADDWD VPMADDWD Packed Multiply and Add Word to Doubleword

Multiplies and adds sets of four packed 16-bit signed values from two source registers; writes four 32-bit sums to the destination.

For the 128-bit form of the instruction, the following operations are performed:

dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.

```
dest[31:0] = (src1[15:0] * src2[15:0]) + (src1[31:16] * src2[31:16])
dest[63:32] = (src1[47:32] * src2[47:32]) + (src1[63:48] * src2[63:48])
dest[95:64] = (src1[79:64] * src2[79:64]) + (src1[95:80] * src2[95:80])
dest[127:96] = (src1[111:96] * src2[111:96]) + (src1[127:112] * src2[127:112])
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[159:128] = (src1[143:128] * src2[143:128]) + (src1[159:144] * src2[159:144])
dest[191:160] = (src1[175:160] * src2[175:160]) + (src1[191:176] * src2[191:176])
dest[223:192] = (src1[207:192] * src2[207:192]) + (src1[223:208] * src2[223:208])
dest[255:224] = (src1[239:224] * src2[239:224]) + (src1[255:240] * src2[255:240])
```

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 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.

VPMADDWD

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

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

Instruction Support

Form	Subset	Feature Flag
PMADDWD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMADDWD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMADDWD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMADDWD *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	F5 /r
VPMADDWD *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	F5 /r

Related Instructions

(V)PMADDUBSW, (V)PMULHUW, (V)PMULHW, (V)PMULLW, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 byte of the destination.

The 128-bit form of the instruction compares 16 pairs of 8-bit signed integer values; the 256-bit form compares 32 pairs.

There are legacy and extended forms of the instruction:

PMAXSB

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMAXSB	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMAXSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMAXSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMAXSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	3C /r
VPMAXSB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	3C /r

Related Instructions

(V)PMAXSD, (V)PMAXSW, (V)PMAXUB, (V)PMAXUD, (V)PMAXUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 doubleword of the destination.

The 128-bit form of the instruction compares four pairs of 32-bit signed integer values; the 256-bit form compares eight.

There are legacy and extended forms of the instruction:

PMAXSD

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMAXSD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMAXSD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMAXSD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMAXSD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	3D /r
VPMAXSD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	3D /r

Related Instructions

(V)PMAXSB, (V)PMAXSW, (V)PMAXUB, (V)PMAXUD, (V)PMAXUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 word of the destination.

The 128-bit form of the instruction compares eight pairs of 16-bit signed integer values; the 256-bit form compares 16 pairs.

There are legacy and extended forms of the instruction:

PMAXSW

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMAXSW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMAXSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMAXSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMAXSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	EE /r
VPMAXSW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	EE /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXUB, (V)PMAXUD, (V)PMAXUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 byte of the destination.

The 128-bit form of the instruction compares 16 pairs of 8-bit unsigned integer values; the 256-bit form compares 32 pairs.

There are legacy and extended forms of the instruction:

PMAXUB

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMAXUB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMAXUB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMAXUB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMAXUB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	DE /r
VPMAXUB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	DE /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXSW, (V)PMAXUD, (V)PMAXUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

None

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 doubleword of the destination.

The 128-bit form of the instruction compares four pairs of 32-bit unsigned integer values; the 256-bit form compares eight.

There are legacy and extended forms of the instruction:

PMAXUD

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMAXUD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMAXUD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMAXUD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMAXUD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	3F /r
VPMAXUD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	3F /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXSW, (V)PMAXUB, (V)PMAXUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 word of the destination.

The 128-bit form of the instruction compares eight pairs of 16-bit unsigned integer values; the 256-bit form compares 16 pairs.

There are legacy and extended forms of the instruction:

PMAXUW

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMAXUW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMAXUW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMAXUW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMAXUW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	3E /r
VPMAXUW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	3E /r

Related Instructions

(V)PMAXSB, (V)PMAXSD, (V)PMAXSW, (V)PMAXUB, (V)PMAXUD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 byte of the destination.

The 128-bit form of the instruction compares 16 pairs of 8-bit signed integer values; the 256-bit form compares 32 pairs.

There are legacy and extended forms of the instruction:

PMINSB

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMINSB	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMINSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMINSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMINSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	38 /r
VPMINSB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	38 /r

Related Instructions

(V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUD, (V)PMINUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 doubleword of the destination.

The 128-bit form of the instruction compares four pairs of 32-bit signed integer values; the 256-bit form compares eight.

There are legacy and extended forms of the instruction:

PMINSD

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMINSD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMINSD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMINSD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMINSD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	39 /r
VPMINSD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	39 /r

Related Instructions

(V)PMINSB, (V)PMINSW, (V)PMINUB, (V)PMINUD, (V)PMINUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 word of the destination.

The 128-bit form of the instruction compares eight pairs of 16-bit signed integer values; the 256-bit form compares 16 pairs.

There are legacy and extended forms of the instruction:

PMINSW

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMINSW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMINSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMINSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMINSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	EA /r
VPMINSW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	EA /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINUB, (V)PMINUD, (V)PMINUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 byte of the destination.

The 128-bit form of the instruction compares 16 pairs of 8-bit unsigned integer values; the 256-bit form compares 32 pairs.

There are legacy and extended forms of the instruction:

PMINUB

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMINUB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMINUB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMINUB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMINUB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	DA /r
VPMINUB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	DA /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUD, (V)PMINUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 doubleword of the destination.

The 128-bit form of the instruction compares four pairs of 32-bit unsigned integer values; the 256-bit form compares eight.

There are legacy and extended forms of the instruction:

PMINUD

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMINUD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMINUD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMINUD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMINUD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	3B /r
VPMINUD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	3B /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 word of the destination.

The 128-bit form of the instruction compares eight pairs of 16-bit unsigned integer values; the 256-bit form compares 16 pairs.

There are legacy and extended forms of the instruction:

PMINUW

The first source operand is an XMM register. The second source operand is either an 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PMINUW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMINUW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMINUW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMINUW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	3A /r
VPMINUW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	3A /r

Related Instructions

(V)PMINSB, (V)PMINSD, (V)PMINSW, (V)PMINUB, (V)PMINUD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVMSKB VPMOVMSKB — Packed Move Mask Byte

Copies the value of the most-significant bit of each byte element of the source operand to create a 16 or 32 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 extended form of the instruction has 128-bit and 256-bit encodings.

XMM Encoding

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].

YMM Encoding

The source operand is a YMM register. The destination is a 64-bit general purpose register. The mask is zero-extended to fill the destination register, the mask occupies bits [31:0].

Instruction Support

Form	Subset	Feature Flag
PMOVMSKB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMOVMSKB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVMSKB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VMOVMSKB *reg64*, *xmm1*	C4	$\overline{\text{RXB}}$.01	X.1111.0.01	D7 /r
VMOVMSKB *reg64*, *ymm1*	C4	$\overline{\text{RXB}}$.01	X.1111.1.01	D7 /r

Related Instructions

(V)MOVMSKPD, (V)MOVMSKPS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv field ! = 1111b.
			A	VEX.L field = 1 when AVX2 not supported.
			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 — SSE, AVX and AVX2 exception
A — AVX, AVX2exception
S — SSE exception

PMOVSXBD VPMOVSXBD — Packed Move with Sign-Extension Byte to Doubleword

Sign-extends four or eight packed 8-bit signed integers in the source operand to 32 bits and writes the packed doubleword signed integers to the destination.

If the source operand is a register, the 8-bit signed integers are taken from the least-significant bytes of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 64-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVSXBD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVSXBD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVSXBD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVSXBD *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	21 /r
VPMOVSXBD *ymm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	21 /r

Related Instructions

(V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWD, (V)PMOVSXW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVSXBQ VPMOVSXBQ — Packed Move with Sign Extension Byte to Quadword

Sign-extends two or four packed 8-bit signed integers in the source operand to 64 bits and writes the packed quadword signed integers to the destination.

If the source operand is a register, the 8-bit signed integers are taken from the least-significant bytes of the register.

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 and 256-bit encodings.

XMM Encoding

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 cleared.

YMM Encoding

The source operand is either an XMM register or a 32-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVSXBQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVSXBQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVSXBQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVSXBQ *xmm1*, *xmm2/mem16*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	22 /r
VPMOVSXBQ *ymm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	22 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWD, (V)PMOVSXW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVSXBW VPMOVSXBW Packed Move with Sign Extension Byte to Word

Sign-extends eight or sixteen packed 8-bit signed integers in the source operand to 16 bits and writes the packed word signed integers to the destination.

If the source operand is a register, the eight 8-bit signed integers are taken from the lower half of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVSXBW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVSXBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVSXBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PMOVSXBW *xmm1*, *xmm2/mem64*	66 0F 38 20 /r	Sign-extends eight packed signed 8-bit integers in the eight low bytes of *xmm2* or *mem64* and writes eight packed signed 16-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPMOVSXBW *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	20 /r
VPMOVSXBW *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	20 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXDQ, (V)PMOVSXWD, (V)PMOVSXW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVSXDQ VPMOVSXDQ Packed Move with Sign-Extension Doubleword to Quadword

Sign-extends two or four packed 32-bit signed integers in the source operand to 64 bits and writes the packed quadword signed integers to the destination.

If the source operand is a register, the two 32-bit signed integers are taken from the lower half of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVSXDQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVSXDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVSXDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVSXDQ *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	25 /r
VPMOVSXDQ *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	25 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXWD, (V)PMOVSXWQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVSXWD VPMOVSXWD Packed Move with Sign-Extension Word to Doubleword

Sign-extends four or eight packed 16-bit signed integers in the source operand to 32 bits and writes the packed doubleword signed integers to the destination.

If the source operand is a register, the four 16-bit signed integers are taken from the lower half of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVSXWD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVSXWD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVSXWD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVSXWD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	23 /r
VPMOVSXWD *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	23 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVSXWQ VPMOVSXWQ Packed Move with Sign-Extension Word to Quadword

Sign-extends two or four packed 16-bit signed integers to 64 bits and writes the packed quadword signed integers to the destination.

If the source operand is a register, the 16-bit signed integers are taken from least-significant words of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 64-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVSXWQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVSXWQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVSXWQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVSXWQ *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	24 /r
VPMOVSXWQ *ymm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	24 /r

Related Instructions

(V)PMOVSXBD, (V)PMOVSXBQ, (V)PMOVSXBW, (V)PMOVSXDQ, (V)PMOVSXWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVZXBD VPMOVZXBD — Packed Move with Zero-Extension Byte to Doubleword

Zero-extends four or eight packed 8-bit unsigned integers in the source operand to 32 bits and writes the packed doubleword positive-signed integers to the destination.

If the source operand is a register, the 8-bit signed integers are taken from the least-significant bytes of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 64-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVZXBD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVZXBD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVZXBD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVZXBD *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	31 /r
VPMOVZXBD *ymm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	31 /r

Related Instructions

(V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWD, (V)PMOVZXW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVZXBQ VPMOVZXBQ — Packed Move Byte to Quadword with Zero-Extension

Zero-extends two or four packed 8-bit unsigned integers in the source operand to 64 bits and writes the packed quadword positive-signed integers to the destination.

If the source operand is a register, the 8-bit signed integers are taken from the least-significant bytes of the register.

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 and 256-bit encodings.

XMM Encoding

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 cleared.

YMM Encoding

The source operand is either an XMM register or a 32-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVZXBQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVZXBQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVZXBQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVZXBQ *xmm1, xmm2/mem16*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	32 /r
VPMOVZXBQ *ymm1, xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	32 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWD, (V)PMOVZXW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVZXBW Packed Move Byte to Word with Zero-Extension VPMOVZXBW

Zero-extends eight or sixteen packed 8-bit unsigned integers in the source operand to 16 bits and writes the packed word positive-signed integers to the destination.

If the source operand is a register, the eight 8-bit signed integers are taken from the lower half of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVZXBW	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVZXBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVZXBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PMOVZXBW *xmm1*, *xmm2/mem64*	66 0F 38 30 /r	Zero-extends eight packed unsigned 8-bit integers in the eight low bytes of *xmm2* or *mem64* and writes eight packed positive-signed 16-bit integers to *xmm1*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPMOVZXBW *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	30 /r
VPMOVZXBW *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	30 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXDQ, (V)PMOVZXWD, (V)PMOVZXW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVZXDQ VPMOVZXDQ Packed Move with Zero-Extension Doubleword to Quadword

Zero-extends two or four packed 32-bit unsigned integers in the source operand to 64 bits and writes the packed quadword positive-signed integers to the destination.

If the source operand is a register, the two 32-bit signed integers are taken from the lower half of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVZXDQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVZXDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVZXDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVZXDQ *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	35 /r
VPMOVZXDQ *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	35 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXWD, (V)PMOVZXWQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVZXWD VPMOVZXWD Packed Move Word to Doubleword with Zero-Extension

Zero-extends four or eight packed 16-bit unsigned integers in the source operand to 32 bits and writes the packed doubleword positive-signed integers to the destination.

If the source operand is a register, the four 16-bit signed integers are taken from the lower half of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVZXWD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVZXWD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVZXWD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVZXWD *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	33 /r
VPMOVZXWD *ymm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	33 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMOVZXWQ VPMOVZXWQ — Packed Move with Zero-Extension Word to Quadword

Zero-extends two or four packed 16-bit unsigned integers to 64 bits and writes the packed quadword positive signed integers to the destination.

If the source operand is a register, the 16-bit signed integers are taken from least-significant words of the register.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either an XMM register or a 64-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PMOVZXWQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMOVZXWQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMOVZXWQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMOVZXWQ *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	X.1111.0.01	34 /r
VPMOVZXWQ *ymm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	X.1111.1.01	34 /r

Related Instructions

(V)PMOVZXBD, (V)PMOVZXBQ, (V)PMOVZXBW, (V)PMOVZXDQ, (V)PMOVZXWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMULDQ Packed Multiply
VPMULDQ Signed Doubleword to Quadword

Multiplies two or four pairs of 32-bit signed integers in the first and second source operands and writes two or four packed quadword signed integer products to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.

dest[63:0] = (src1[31:0] * src2[31:0])
dest[127:64] = (src1[95:64] * src2[95:64])
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[191:128] = (src1[159:128] * src2[159:128])
dest[255:192] = (src1[223:192] * src2[223:192])
```

There are legacy and extended forms of the instruction:

PMULDQ

The first source operand is an XMM register. 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.

VPMULDQ

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

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

Instruction Support

Form	Subset	Feature Flag
PMULDQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMULDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	28 /r
VPMULDQ *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	28 /r

Related Instructions

(V)PMULLD, (V)PMULHW, (V)PMULHUW,(V)PMULUDQ, (V)PMULLW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 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.

VPMULHRSW

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

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

Instruction Support

Form	Subset	Feature Flag
PMULHRSW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPMULHRSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULHRSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULHRSW *xmm1*, *xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.2	X.$\overline{src1}$.0.01	0B /r
VPMULHRSW *ymm1*, *ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.2	X.$\overline{src1}$.1.01	0B /r

Related Instructions

None

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 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.

VPMULHUW

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

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

Instruction Support

Form	Subset	Feature Flag
PMULHUW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMULHUW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULHUW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULHUW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E4 /r
VPMULHUW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E4 /r

Related Instructions

(V)PMULDQ, (V)PMULHW, (V)PMULLD, (V)PMULLW, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 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.

VPMULHW

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

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

Instruction Support

Form	Subset	Feature Flag
PMULHW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMULHW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULHW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULHW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E5 /r
VPMULHW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E5 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULLD, (V)PMULLW, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 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.

VPMULLD

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

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

Instruction Support

Form	Subset	Feature Flag
PMULLD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPMULLD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULLD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULLD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.0.01	40 /r
VPMULLD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{src1}$.1.01	40 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULHW, (V)PMULLW, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 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.

VPMULLW

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

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

Instruction Support

Form	Subset	Feature Flag
PMULLW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMULLW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULLW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULLW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	D5 /r
VPMULLW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	D5 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULHW, (V)PMULLD, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PMULUDQ Packed Multiply
VPMULUDQ Unsigned Doubleword to Quadword

Multiplies two or four pairs of 32-bit unsigned integers in the first and second source operands and writes two or four packed quadword unsigned integer products to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest is the destination register – either an XMM register or the corresponding YMM register.
src1 is the first source operand. src2 is the second source operand.

dest[63:0] = (src1[31:0] * src2[31:0])
dest[127:64] = (src1[95:64] * src2[95:64])
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[191:128] = (src1[159:128] * src2[159:128])
dest[255:192] = (src1[223:192] * src2[223:192])
```

There are legacy and extended forms of the instruction:

PMULUDQ

The first source operand is an XMM register. 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.

VPMULUDQ

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

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

Instruction Support

Form	Subset	Feature Flag
PMULUDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPMULUDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPMULUDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPMULUDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	F4 /r
VPMULUDQ *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	F4 /r

Related Instructions

(V)PMULDQ, (V)PMULHUW, (V)PMULHW, (V)PMULLD, (V)PMULLW, (V)PMULUDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

POR Packed OR
VPOR

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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
POR	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPOR 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPOR 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPOR *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	EB /r
VPOR *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	EB /r

Related Instructions

(V)PAND, (V)PANDN, (V)PXOR

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSADBW Packed Sum of Absolute Differences
VPSADBW Bytes to Words

Subtracts the 16 or 32 packed 8-bit unsigned integers in the second source operand from the corresponding values in the first source operand and computes the absolute value of the differences. Computes two or four unsigned 16-bit integer sums of groups of eight absolute differences and writes the sums to specific words of the destination.

For the 128-bit form of the instruction:

- The unsigned 16-bit integer sum of absolute differences of the eight bytes [7:0] 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 absolute differences of the eight bytes [15:8] of the source operands is written to bits [79:64] of the destination; bits [127:80] are cleared.

Additionally, for the 256-bit form of the instruction:

- The unsigned 16-bit integer sum of absolute differences of the eight bytes [23:16] of the source operands is written to bits [143:128] of the destination; bits [191:144] are cleared.
- The unsigned 16-bit integer sum of absolute differences of the eight bytes [24:31] of the source operands is written to bits [207:192] of the destination; bits [255:208] 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PSADBW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSADBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSADBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSADBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	F6 /r
VPSADBW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	F6 /r

Related Instructions

(V)MPSADBW

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

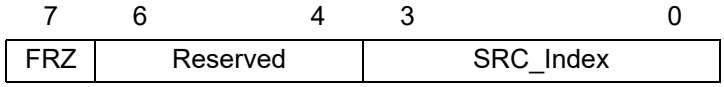
Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSHUFB Packed Shuffle
VPSHUFB 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.



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.

For the 256-bit form of the instruction, the SRC_Index fields in the upper 16 bytes of the second source operand select bytes in the upper 16 bytes of the first source operand to be copied.

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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PSHUFB	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPSHUFB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSHUFB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSHUFB *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	00 /r
VPSHUFB *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	00 /r

Related Instructions

(V)PSHUFD, (V)PSHUFW, (V)PSHUHW, (V)PSHUFLW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 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]

For the 256-bit form of the instruction, the same immediate byte selects doublewords in the upper 128-bits of the source operand to be copied to the destination.

Destination Doubleword	Immediate-Byte Bit Field	Value of Bit Field	Source Doubleword
[159:128]	[1:0]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[225:224]
[191:160]	[3:2]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[225:224]

Destination Doubleword	Immediate-Byte Bit Field	Value of Bit Field	Source Doubleword
[223:192]	[5:4]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[225:224]
[255:224]	[7:6]	00	[159:128]
		01	[191:160]
		10	[223:192]
		11	[225:224]

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PSHUFD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSHUFD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSHUFD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
PSHUFD *xmm1*, *xmm2*/*mem128*, *imm8*	66 0F 70 /r ib	Copies packed 32-bit values from *xmm2* or *mem128* to *xmm1*, as specified by *imm8*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHUFD *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.0.01	70 /r ib
VPSHUFD *ymm1*, *ymm2/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.1.01	70 /r ib

Related Instructions

(V)PSHUFHW, (V)PSHUFLW, (V)PSHUFW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSHUFHW VPSHUFHW — Packed Shuffle High Words

Copies packed word values from the high quadword of the source operand or the upper quadwords of two halves of the source operand to a word in the high quadword of the destination or the upper quadwords of two halves of the destination, as specified by bit fields of an immediate byte operand. A source word can be copied more than once.

Source words are selected by two-bit fields in the immediate-byte operand. Each 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]

The least-significant quadword of the source is copied to the corresponding quadword of the destination.

For the 256-bit form of the instruction, the same immediate byte selects words in the most-significant quadword of the source operand to be copied to the destination:

Destination Word	Immediate-Byte Bit Field	Value of Bit Field	Source Word
[207:192]	[1:0]	00	[207:192]
		01	[223:208]
		10	[239:224]
		11	[255:240]

Destination Word	Immediate-Byte Bit Field	Value of Bit Field	Source Word
[223:208]	[3:2]	00	[207:192]
		01	[223:208]
		10	[239:224]
		11	[255:240]
[239:224]	[5:4]	00	[207:192]
		01	[223:208]
		10	[239:224]
		11	[255:240]
[255:240]	[7:6]	00	[207:192]
		01	[223:208]
		10	[239:224]
		11	[255:240]

The least-significant quadword of the upper 128 bits of the source is copied to the corresponding quadword of the destination.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PSHUFHW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSHUFHW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSHUFHW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSHUFHW *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.0.10	70 /r ib
VPSHUFHW *ymm1*, *ymm2/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.1.10	70 /r ib

Related Instructions

(V)PSHUFD, (V)PSHUFLW, (V)PSHUFW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSHUFLW VPSHUFLW — Packed Shuffle Low Words

Copies packed word values from the low quadword of the source operand or the lower quadwords of two halves of the source operand to a word in the low quadword of the destination or the lower quadwords of two halves of the destination, as specified by bit fields of an immediate byte operand. A source word can be copied more than once.

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]

The most-significant quadword of the source is copied to the corresponding quadword of the destination.

For the 256-bit form of the instruction, the same immediate byte selects words in the lower quadword of the upper 128 bits of the source operand to be copied to the destination:

Destination Word	Immediate-Byte Bit Field	Value of Bit Field	Source Word
[143:128]	[1:0]	00	[143:128]
		01	[159:144]
		10	[175:160]
		11	[191:176]

Destination Word	Immediate-Byte Bit Field	Value of Bit Field	Source Word
[159:144]	[3:2]	00	[143:128]
		01	[159:144]
		10	[175:160]
		11	[191:176]
[175:160]	[5:4]	00	[143:128]
		01	[159:144]
		10	[175:160]
		11	[191:176]
[191:176]	[7:6]	00	[143:128]
		01	[159:144]
		10	[175:160]
		11	[191:176]

The most-significant quadword of the upper 128 bits of the source is copied to the corresponding quadword of the destination.

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 and 256-bit encodings.

XMM 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.

YMM Encoding

The source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
PSHUFLW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSHUFLW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSHUFLW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSHUFLW *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.0.11	70 /r ib
VPSHUFLW *ymm1*, *ymm2/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.1111.1.11	70 /r ib

Related Instructions

(V)PSHUFD, (V)PSHUFHW, (V)PSHUFW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PSIGNB	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPSIGNB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSIGNB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSIGNB *xmm1*, xmm2*, xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	08 /r
VPSIGNB *ymm1*, ymm2*, ymm2/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	08 /r

Related Instructions

(V)PSIGNW, (V)PSIGND

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PSIGND	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPSIGND 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSIGND 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSIGND *xmm1*, xmm2*, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	0A /r
VPSIGND *ymm1*, ymm2*, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	0A /r

Related Instructions

(V)PSIGNB, (V)PSIGNW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 and 256-bit encodings.

XMM 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.

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. The destination is a third YMM register.

Instruction Support

Form	Subset	Feature Flag
PSIGNW	SSSE3	CPUID Fn0000_0001_ECX[SSSE3] (bit 9)
VPSIGNW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSIGNW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSIGNW *xmm1*, xmm2*, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.0.01	09 /r
VPSIGNW *ymm1*, ymm2*, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	X.$\overline{\textit{src1}}$.1.01	09 /r

Related Instructions

(V)PSIGNB, (V)PSIGND

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

Low-order bits emptied by shifting are cleared. When the shift count 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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSLLD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSLLD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSLLD 256-bit	AVX2	CPUID Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSLLD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	F2 /r
VPSLLD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	72 /6 ib
VPSLLD *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	F2 /r
VPSLLD *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	72 /6 ib

Related Instructions

(V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 exception
S — SSE exception

PSLLDQ Packed Shift Left Logical
VPSLLDQ Double Quadword

Left-shifts the one or each of the two double quadword values in the source operand the number of bytes specified by an immediate byte operand and writes the shifted values to the destination.

The immediate byte operand supplies an unsigned shift count. Low-order bytes emptied by shifting are cleared. When the shift value is greater than 15, the destination is cleared. For the 256-bit form of the instruction, the shift count is applied to both the upper and the lower double quadword. Bytes shifted out of the lower 128 bits are not shifted into the upper.

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 extended form of the instruction has 128-bit and 256-bit encodings.

XMM Encoding

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.

YMM Encoding

The source operand is a YMM register. The destination is a YMM register specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSLLDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSLLDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSLLDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSLLDQ *xmm1*, xmm2, *imm8*	C4	$\overline{\text{RXB}}$.01	0.$\overline{\text{dest}}$.0.01	73 /7 ib
VPSLLDQ *ymm1*, ymm2, *imm8*	C4	$\overline{\text{RXB}}$.01	0.$\overline{\text{dest}}$.1.01	73 /7 ib

Related Instructions

(V)PSLLD, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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 — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

Low-order bits emptied by shifting are cleared. When the shift value is greater than 63, 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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSLLQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSLLQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSLLQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSLLQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	F3 /r
VPSLLQ *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	73 /6 ib
VPSLLQ *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	F3 /r
VPSLLQ *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	73 /6 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQLLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

Low-order bits emptied by shifting are cleared. When the shift count is greater than 15, 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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSLLW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSLLW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSLLW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSLLW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	F1 /r
VPSLLW *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	71 /6 ib
VPSLLW *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	F1 /r
VPSLLW *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	71 /6 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSRAD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSRAD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSRAD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSRAD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E2 /r
VPSRAD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	72 /4 ib
VPSRAD *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E2 /r
VPSRAD *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	72 /4 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

High-order bits emptied by shifting are filled with the sign bit of the initial value. When the shift value is greater than 16, 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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSRAW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSRAW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSRAW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSRAW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E1 /r
VPSRAW *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	71 /4 ib
VPSRAW *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E1 /r
VPSRAW *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	71 /4 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

High-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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSRLD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSRLD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSRLD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSRLD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	D2 /r
VPSRLD *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	72 /2 ib
VPSRLD *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	D2 /r
VPSRLD *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	72 /2 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 exception
S — SSE exception

PSRLDQ VPSRLDQ — Packed Shift Right Logical Double Quadword

Right-shifts one or each of two double quadword values in the source operand the number of bytes specified by an immediate byte operand and writes the shifted values to the destination.

The immediate byte operand supplies an unsigned shift count. High-order bytes emptied by shifting are cleared. When the shift value is greater than 15, the destination is cleared. For the 256-bit form of the instruction, the shift count is applied to both the upper and the lower double quadword. Bytes shifted out of the upper 128 bits are not shifted into the lower.

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 extended form of the instruction has 128-bit and 256-bit encodings.

XMM Encoding

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.

YMM Encoding

The source operand is a YMM register. The destination is a YMM register specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSRLDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSRLDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSRLDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSRLDQ *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	73 /3 ib
VPSRLDQ *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	73 /3 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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 — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

High-order bits emptied by shifting are cleared. When the shift value is greater than 63, 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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSRLQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSRLQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSRLQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSRLQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	D3 /r
VPSRLQ *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	73 /2 ib
VPSRLQ *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	D3 /r
VPSRLQ *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	73 /2 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 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 second register, or a memory location. The shift count is treated as an unsigned integer. When the shift count is provided by a register or memory location, only bits [63:0] of the value are considered.

High-order bits 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:

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 and 256-bit encodings.

XMM Encoding

There are two 128-bit encodings. These differ 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.

YMM Encoding

There are two 256-bit encodings. These differ based on the type of count operand.

The first source operand is a YMM 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 a YMM register. For the immediate operand encoding, the destination is specified by VEX.vvvv.

Instruction Support

Form	Subset	Feature Flag
PSRLW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSRLW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSRLW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSRLW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	D1 /r
VPSRLW *xmm1*, *xmm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.0.01	71 /2 ib
VPSRLW *ymm1*, *ymm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	D1 /r
VPSRLW *ymm1*, *ymm2*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\text{dest}}$.1.01	71 /2 ib

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 exception
S — SSE exception

PSUBB VPSUBB — Packed Subtract Bytes

Subtracts 16 or 32 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 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.

VPSUBB

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	F8 /r
VPSUBB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	F8 /r

Related Instructions

(V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBD VPSUBD — Packed Subtract Doublewords

Subtracts four or eight 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 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.

VSUBD

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	FA /r
VPSUBD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	FA /r

Related Instructions

(V)PSUBB, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBQ VPSUBQ Packed Subtract Quadword

Subtracts two or four 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 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.

VSUBQ

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	FB /r
VPSUBQ *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	FB /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBSB VPSUBSB Packed Subtract Signed With Saturation Bytes

Subtracts 16 or 32 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 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.

VPSUBSB

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBSB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E8 /r
VPSUBSB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E8 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBSW VPSUBSW Packed Subtract Signed With Saturation Words

Subtracts eight or sixteen 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 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.

VPSUBSW

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBSW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	E9 /r
VPSUBSW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	E9 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBUSB, (V)PSUBUSW, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBUSB VPSUBUSB Packed Subtract Unsigned With Saturation Bytes

Subtracts 16 or 32 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 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 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.

VPSUBUSB

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBUSB	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBUSB 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBUSB 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBUSB *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	D8 /r
VPSUBUSB *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	D8 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSW, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBUSW VPSUBUSW Packed Subtract Unsigned With Saturation Words

Subtracts eight or sixteen 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 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 an 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PSUBUSW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBUSW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBUSW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBUSW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	D9 /r
VPSUBUSW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	D9 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PSUBW VPSUBW — Packed Subtract Words

Subtracts eight or sixteen 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 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.

VPSUBW

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

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

Instruction Support

Form	Subset	Feature Flag
PSUBW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPSUBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPSUBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPSUBW *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	F9 /r
VPSUBW *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	F9 /r

Related Instructions

(V)PSUBB, (V)PSUBD, (V)PSUBQ, (V)PSUBSB, (V)PSUBSW, (V)PSUBUSB, (V)PSUBUSW

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PTEST Packed Bit Test
VPTEST

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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
PTEST	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPTEST	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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.*

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 8 high-order bytes of each octword the first and second source operands and interleaves the bytes as they are copied to the destination. The low-order bytes of each octword of the source operands are ignored.

Bytes are interleaved in ascending order from the least-significant byte of the upper 8 bytes of each octword of the source operands with bytes from the first source operand occupying the lower byte of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[7:0] = src1[71:64]
dest[15:8] = src2[71:64]
dest[23:16] = src1[79:72]
dest[31:24] = src2[79:72]
dest[39:32] = src1[87:80]
dest[47:40] = src2[87:80]
dest[55:48] = src1[95:88]
dest[63:56] = src2[95:88]
dest[71:64] = src1[103:96]
dest[79:72] = src2[103:96]
dest[87:80] = src1[111:104]
dest[95:88] = src2[111:104]
dest[103:96] = src1[119:112]
dest[111:104] = src2[119:112]
dest[119:112] = src1[127:120]
dest[127:120] = src2[127:120]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[135:128] = src1[199:192]
dest[143:136] = src2[199:192]
dest[151:144] = src1[207:200]
dest[159:152] = src2[207:200]
dest[167:160] = src1[215:208]
dest[175:168] = src2[215:208]
dest[183:176] = src1[223:216]
dest[191:184] = src2[223:216]
dest[199:192] = src1[231:224]
dest[207:200] = src2[231:224]
dest[215:208] = src1[239:232]
dest[223:216] = src2[239:232]
dest[231:224] = src1[247:240]
dest[239:232] = src2[247:240]
dest[247:240] = src1[255:248]
dest[255:248] = src2[255:248]
```

When the second source operand is all 0s, the destination effectively contains the 8 high-order bytes from the first source operand or the 8 high-order bytes from both octwords of 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKHBW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKHBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKHBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKHBW *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	68 /r
VPUNPCKHBW *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	68 /r

Related Instructions

(V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKHDQ VPUNPCKHDQ — Unpack and Interleave High Doublewords

Unpacks the two high-order doublewords of each octword of the first and second source operands and interleaves the doublewords as they are copied to the destination. The low-order doublewords of each octword of the source operands are ignored.

Doublewords are interleaved in ascending order from the least-significant doubleword of the high quadword of each octword with doublewords from the first source operand occupying the lower doubleword of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[31:0] = src1[95:64]
dest[63:32] = src2[95:64]
dest[95:64] = src1[127:96]
dest[127:96] = src2[127:96]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[159:128] = src1[223:192]
dest[191:160] = src2[223:192]
dest[223:192] = src1[255:224]
dest[255:224] = src2[255:224]
```

When the second source operand is all 0s, the destination effectively receives the 2 high-order doublewords from the first source operand or the 2 high-order doublewords from both octwords of 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 an 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKHDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKHDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKHDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKHDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	6A /r
VPUNPCKHDQ *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	6A /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKHQDQ VPUNPCKHQDQ — Unpack and Interleave High Quadwords

Unpacks the high-order quadword of each octword of the first and second source operands and interleaves the quadwords as they are copied to the destination. The low-order quadword of each octword of the source operands is ignored.

Quadwords are interleaved in ascending order with the high-order quadword from the first source operand or each octword of the first source operand occupying the lower quadword of corresponding octword of the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[63:0] = src1[127:64]
dest[127:64] = src2[127:64]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[191:128] = src1[255:192]
dest[255:192] = src2[255:192]
```

When the second source operand is all 0s, the destination effectively receives the quadword from upper half of the first source operand or the high-order quadwords from each octword of 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 an 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKHQDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKHQDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKHQDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKHQDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	6D /r
VPUNPCKHQDQ *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	6D /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKHWD VPUNPCKHWD — Unpack and Interleave High Words

Unpacks the 4 high-order words of each octword of the first and second source operands and interleaves the words as they are copied to the destination. The low-order words of each octword of the source operands are ignored.

Words are interleaved in ascending order from the least-significant word of the high quadword of each octword with words from the first source operand occupying the lower word of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[15:0] = src1[79:64]
dest[31:16] = src2[79:64]
dest[47:32] = src1[95:80]
dest[63:48] = src2[95:80]
dest[79:64] = src1[111:96]
dest[95:80] = src2[111:96]
dest[111:96] = src1[127:112]
dest[127:112] = src2[127:112]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = src1[207:192]
dest[159:144] = src2[207:192]
dest[175:160] = src1[223:208]
dest[191:176] = src2[223:208]
dest[207:192] = src1[239:224]
dest[223:208] = src2[239:224]
dest[239:224] = src1[255:240]
dest[255:240] = src2[255:240]
```

When the second source operand is all 0s, the destination effectively receives the 4 high-order words from the first source operand or the 4 high-order words from both octwords of 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKHWD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKHWD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKHWD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKHWD *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	69 /r
VPUNPCKHWD *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	69 /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	S	S	X	Lock prefix (F0h) preceding opcode.

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Device not available, #NM	S	S	X	CR0.TS = 1.
Stack, #SS	S	S	X	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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKLBW VPUNPCKLBW — Unpack and Interleave Low Bytes

Unpacks the 8 low-order bytes of each octword of the first and second source operands and interleaves the bytes as they are copied to the destination. The high-order bytes of each octword are ignored.

Bytes are interleaved in ascending order from the least-significant byte of source operands with bytes from the first source operand occupying the lower byte of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[7:0] = src1[7:0]
dest[15:8] = src2[7:0]
dest[23:16] = src1[15:8]
dest[31:24] = src2[15:8]
dest[39:32] = src1[23:16]
dest[47:40] = src2[23:16]
dest[55:48] = src1[31:24]
dest[63:56] = src2[31:24]
dest[71:64] = src1[39:32]
dest[79:72] = src2[39:32]
dest[87:80] = src1[47:40]
dest[95:88] = src2[47:40]
dest[103:96] = src1[55:48]
dest[111:104] = src2[55:48]
dest[119:112] = src1[63:56]
dest[127:120] = src2[63:56]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[135:128] = src1[135:128]
dest[143:136] = src2[135:128]
dest[151:144] = src1[143:136]
dest[159:152] = src2[143:136]
dest[167:160] = src1[151:144]
dest[175:168] = src2[151:144]
dest[183:176] = src1[159:152]
dest[191:184] = src2[159:152]
dest[199:192] = src1[167:160]
dest[207:200] = src2[167:160]
dest[215:208] = src1[175:168]
dest[223:216] = src2[175:168]
dest[231:224] = src1[183:176]
dest[239:232] = src2[183:176]
dest[247:240] = src1[191:184]
dest[255:248] = src2[191:184]
```

When the second source operand is all 0s, the destination effectively receives the eight low-order bytes from the first source operand or the eight low-order bytes from both octwords of 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKLBW	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKLBW 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKLBW 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKLBW *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	60 /r
VPUNPCKLBW *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	60 /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLDQ, (V)PUNPCKLQDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKLDQ VPUNPCKLDQ — Unpack and Interleave Low Doublewords

Unpacks the two low-order doublewords of each octword of the first and second source operands and interleaves the doublewords as they are copied to the destination. The high-order doublewords of each octword of the source operands are ignored.

Doublewords are interleaved in ascending order from the least-significant doubleword of the sources with doublewords from the first source operand occupying the lower doubleword of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[31:0] = src1[31:0]
dest[63:32] = src2[31:0]
dest[95:64] = src1[63:32]
dest[127:96] = src2[63:32]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[159:128] = src1[159:128]
dest[191:160] = src2[159:128]
dest[223:192] = src1[191:160]
dest[255:224] = src2[191:160]
```

When the second source operand is all 0s, the destination effectively receives the two low-order doublewords from the first source operand or the two low-order doublewords from both octwords of the 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 an 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKLDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKLDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKLDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKLDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	62 /r
VPUNPCKLDQ *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	62 /r

Related Instructions

(V)PUNPCKHW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLQDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKLQDQ VPUNPCKLQDQ — Unpack and Interleave Low Quadwords

Unpacks the low-order quadword of each octword of the first and second source operands and interleaves the quadwords as they are copied to the destination. The high-order quadword of each octword of the source operands is ignored.

Quadwords are interleaved in ascending order from the least-significant quadword of the sources with quadwords from the first source operand occupying the lower quadword of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[63:0] = src1[63:0]
dest[127:64] = src2[63:0]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[191:128] = src1[191:128]
dest[255:192] = src2[191:128]
```

When the second source operand is all 0s, the destination effectively receives the low-order quadword from the first source operand or the low-order quadword of both octwords of 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 an 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKLQDQ	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKLQDQ 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKLQDQ 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKLQDQ *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	6C /r
VPUNPCKLQDQ *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	6C /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLWD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PUNPCKLWD VPUNPCKLWD — Unpack and Interleave Low Words

Unpacks the four low-order words of each octword of the first and second source operands and interleaves the words as they are copied to the destination. The high-order words of each octword of the source operands are ignored.

Words are interleaved in ascending order from the least-significant word of the source operands with words from the first source operand occupying the lower word of each pair copied to the destination.

For the 128-bit form of the instruction, the following operations are performed:

```
dest[15:0] = src1[15:0]
dest[31:16] = src2[15:0]
dest[47:32] = src1[31:16]
dest[63:48] = src2[31:16]
dest[79:64] = src1[47:32]
dest[95:80] = src2[47:32]
dest[111:96] = src1[63:48]
dest[127:112] = src2[63:48]
```

Additionally, for the 256-bit form of the instruction, the following operations are performed:

```
dest[143:128] = src1[143:128]
dest[159:144] = src2[143:128]
dest[175:160] = src1[159:144]
dest[191:176] = src2[159:144]
dest[207:192] = src1[175:160]
dest[223:208] = src2[175:160]
dest[239:224] = src1[191:176]
dest[255:240] = src2[191:176]
```

When the second source operand is all 0s, the destination effectively receives the 4 low-order words from the first source operand or the 4 low-order words of each octword of 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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PUNPCKLWD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPUNPCKLWD 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPUNPCKLWD 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPUNPCKLWD *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	61 /r
VPUNPCKLWD *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	61 /r

Related Instructions

(V)PUNPCKHBW, (V)PUNPCKHDQ, (V)PUNPCKHQDQ, (V)PUNPCKHWD, (V)PUNPCKLBW, (V)PUNPCKLDQ, (V)PUNPCKLQDQ

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

PXOR Packed Exclusive OR
VPXOR

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 and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
PXOR	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VPXOR 128-bit	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VPXOR 256-bit	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VPXOR *xmm1*, *xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.0.01	EF /r
VPXOR *ymm1*, *ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.1.01	EF /r

Related Instructions

(V)PAND, (V)PANDN, (V)POR

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

RCPPS Reciprocal
VRCPPS 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 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.

VRCPPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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 source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
RCPPS	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VRCPPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VRCPPS *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.01	X.1111.0.00	53 /r
VRCPPS *ymm1*, *ymm2*/*mem256*	C4	$\overline{\text{RXB}}$.01	X.1111.1.00	53 /r

Related Instructions

(V)RCPSS, (V)RSQRTPS, (V)RSQRTSS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

RCPSS Reciprocal
VRCPSS 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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
RCPSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VRCPSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VRCPSS *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.X.10	53 /r

Related Instructions

(V)RCPPS, (V)RSQRTPS, (V)RSQRTSS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

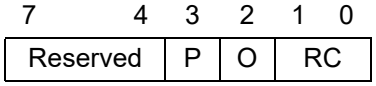
X — AVX and SSE exception
A — AVX exception
S — SSE exception

ROUNDPD Round
VROUNDPD 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.



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	Use RC from immediate operand
1	Use RC from MXCSR

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 an 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 encodings:

XMM Encoding

Rounds two source values. The first source operand is an XMM register. The second source operand is either an 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.

Instruction Support

Form	Subset	Feature Flag
PCMPEQQ	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VPCMPEQQ	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VROUNDPD *xmm1*, *xmm2*/*mem128, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.0.01	09 /r ib
VROUNDPD *ymm1*, *xmm2*/*mem256, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.1.01	09 /r ib

Related Instructions

(V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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.

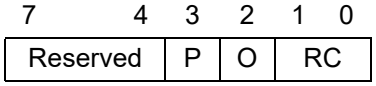
X — AVX and SSE exception
A — AVX exception
S — SSE exception

ROUNDPS Round
VROUNDPS 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.



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	Use RC from immediate operand
1	Use RC from MXCSR

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 an 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 encodings:

XMM Encoding

Rounds four source values. The first source operand is an XMM register. The second source operand is either an 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.

Instruction Support

Form	Subset	Feature Flag
ROUNDPS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VROUNDPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VROUNDPS *xmm1*, *xmm2*/*mem128, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.0.01	08 /r ib
VROUNDPS *ymm1*, *xmm2*/*mem256, imm8*	C4	$\overline{\text{RXB}}$.03	X.1111.1.01	08 /r ib

Related Instructions

(V)ROUNDPD, (V)ROUNDSD, (V)ROUNDSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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.

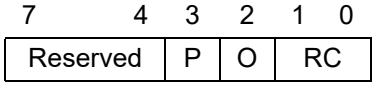
X — AVX and SSE exception
A — AVX exception
S — SSE exception

ROUNDSD Round
VROUNDSD 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.



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	Use RC from immediate operand
1	Use RC from MXCSR

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 quadword. 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 a 128-bit encoding only.

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

Instruction Support

Form	Subset	Feature Flag
ROUNDSD	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VROUNDSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VROUNDSD *xmm1*, *xmm2*, *xmm3*/*mem64, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{\textit{src1}}$.X.01	0B /r ib

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSS

rFLAGS Affected

None

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

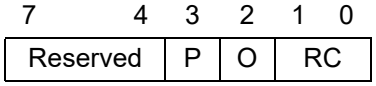
Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[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 when 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 Round
VROUNDSS 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.



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	Use RC from immediate operand
1	Use RC from MXCSR

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 doubleword. 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 a 128-bit encoding only.

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

Instruction Support

Form	Subset	Feature Flag
ROUNDSS	SSE4.1	CPUID Fn0000_0001_ECX[SSE41] (bit 19)
VROUNDSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VROUNDSS *xmm1*, *xmm2*, *xmm3*/*mem64, imm8*	C4	$\overline{\text{RXB}}$.03	X.$\overline{\textit{src1}}$.X.01	0A /r ib

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Reciprocal Square Root
VRSQRTPS 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 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.

VRSQRTPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Computes four values. The destination is an XMM register. The source 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

Computes eight values. The destination is a YMM register. The source operand is either a YMM register or a 256-bit memory location.

Instruction Support

Form	Subset	Feature Flag
RSQRTPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VRSQRTPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VRSQRTPS *xmm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.01	X.1111.0.00	52 /r
VRSQRTPS *ymm1*, *ymm2*/*mem256*	C4	$\overline{\text{RXB}}$.01	X.1111.1.00	52 /r

Related Instructions

(V)RSQRTSS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSD, (V)SQRTSS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

RSQRTSS Reciprocal Square Root
VRSQRTSS 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.

There are legacy and extended forms of the instruction:

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 a 128-bit encoding only.

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 square root of the single-precision floating-point value held in bits [31:0] of the second source operand; 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.

Instruction Support

Form	Subset	Feature Flag
RSQRTSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VRSQRTSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VRSQRTSS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.X.10	52 /r

Related Instructions

(V)RSQRTPS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSD, (V)SQRTSS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

SHA1RNDS4 Four Rounds of SHA1

Execute 4 rounds of a SHA1 operation using the 4 double words (A, B, C, D) from the first source operand, and value E from the second operand. The lower two bits of the immediate are used to specify the function and constant appropriate for the current round of processing. The resulting (A, B, C, D) is placed in the destination register which is the same as the first source register.

The following function is performed:

```
A ← SRC1[127:96];
B ← SRC1[95:64];
C ← SRC1[63:32];
D ← SRC1[31:0];

W0E ← SRC2[127:96];
W1  ← SRC2[95:64];
W2  ← SRC2[63:32];
W3  ← SRC2[31:0];

i=imm[1:0]  which determines f_i and K_i

First Round operation:
A_1 ← f_ 0(B, C, D) + (A Rotate Left 5) +W0E +K_0;
B_1 ← A;
C_1 ← B Rotate Left 30;
D_1 ← C;
E_1 ← D;

FOR j = 1 to 3
{       A_(j +1) ← f_j(B_j, C_j, D_j) + (A_j Rotate Left 5) +Wj+ E_j +K_i;

        B_(j+1) <- A_j;

        C_(j +1) ← B_j Rotate Left 30;

        D_(j +1) ← C_j;
        E_(j +1) ← D_j;

}

DEST[127:96] ← A_4;
DEST[95:64]  ← B_4;
DEST[63:32]  ← C_4;
DEST[31:0]   ← D_4;
```

Mnemonic	Opcode	Description
SHA1RNDS4 xmm1, xmm2/m128, imm8	0F 3A CC /r ib	Executes 4 Rounds of SHA1

Related Instructions

SHA1NEXTE, SHA1MSG1, SHA1MSG2

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction

X - SSE, AVX, and AVX2 exception
A - AVX, AVX2 exception
S - SSE exception

SHA1NEXTE Calculate Next E SHA1

Calculate what the next E register values should be after 4 rounds of a SHA1 operation using the 4 double words from the second source operand, and value A from the first operand. The resulting E is placed in the destination register which is the same as the first source register.

```
DEST[127:96] ← SRC2[127:96] + (SRC1[127:96] rotated left 30)
DEST[95:0]   ← SRC2[95:0];
```

Mnemonic	Opcode	Description
SHA1NEXTE xmm1,xmm2/m128	0F 38 C8 /r	Calculate Next E of SHA1

Related Instructions

SHA1RNDS4, SHA1MSG1, SHA1MSG2

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction
X - SSE, AVX, and AVX2 exception A - AVX, AVX2 exception S - SSE exception				

SHA1MSG1 Message Intermediate 1

Performs the 1st of two intermediate calculations necessary before doing the next four rounds of the SHA1 message.

```
DEST[127:96] ← SRC1[63:32]  XOR SRC1[127:96]
DEST[95:64]  ← SRC1[31:0]   XOR SRC1[95:64]
DEST[63:32]  ← SRC2[127:96] XOR SRC1[63:32]
DEST[31:0]   ← SRC2[95:64]  XOR SRC1[31:0]
```

Mnemonic	Opcode	Description
SHA1MSG1 xmm1, xmm2/m128	0F 38 C9 /r	Calculate Message Intermediate 1

Related Instructions

SHA1RNDS4, SHA1NEXTE, SHA1MSG2

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction
X - SSE, AVX, and AVX2 exception A - AVX, AVX2 exception S - SSE exception				

SHA1MSG2 Message Calculation 2

Performs the 2nd of two intermediate calculations necessary before doing the next four rounds of the SHA1 message.

```
Temp[31:0]  ← (SRC1[127:96]   XOR  SRC2[95:64]) Rotate Left 1

DEST[127:96] ← Temp[31:0]
DEST[95:64]  ← (SRC1[95:64]   XOR  SRC2[63:32]) Rotate Left 1
DEST[63:32]  ← (SRC1{63:32]   XOR  SRC2[31:0])  Rotate Left 1
DEST[31:0]   ← (SRC1[31:0]    XOR  Temp[31:0])  Rotate Left 1
```

Mnemonic	Opcode	Description
SHA1MSG2 xmm1, xmm2/m128	0F 38 CA /r	CCalculate Message Intermediate 2

Related Instructions

SHA1RNDS4, SHA1NEXTE, SHA1MSG1

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS	S	S	X	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
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction
X - SSE, AVX, and AVX2 exception A - AVX, AVX2 exception S - SSE exception				

SHA256RNDS2 Two Rounds of SHA256

Performs 2 rounds of SHA256 operation with the first operand holding the initial SHA256 state (C, D, G, H), the second operand holding the initial SHA256 state (A, B, E, F), and the implicit operand xmm0 holding a pre-computed sum of the next two double word round 2 message as well as the corresponding round constants. The resulting SHA256 state (A, B, E, F) is placed in the destination register.

```
A_0 ← SRC2[127:96];
B_0 ← SRC2[95:64];
C_0 ← SRC1[127:96];
D_0 ← SRC1[95:64];
E_0 ← SRC2[63:32];
F_0 ← SRC2[31:0];
G_0 ← SRC1[63:32];
H_0 ← SRC1[31:0];
K0 ← XMM0[31: 0];
K1 ← XMM0[63: 32];

FOR i = 0 to 1
{       A_(i +1) ← Ch (E_i, F_i, G_i) + Perm1(E_i) +K_i + H_i + Ma(A_i , B_i, C_i) + Perm0(A_i);
        B_(i +1) ← A_i;
        C_(i +1) ← B_i ;
        D_(i +1) ← C_i;
        E_(i +1) ← Ch (E_i, F_i, G_i) + Perm1(E_i) + K_i + H_i + D_i;
        F_(i +1) ← E_i ;
        G_(i +1) ← F_i;
        H_(i +1) ← G_i;
}

DEST[127:96] ← A_2;
DEST[95:64] ← B_2;
DEST[63:32] ← E_2;
DEST[31:0] ← F_2;
```

Mnemonic	Opcode	Description
SHA256RNDS2xmm1, xmm2/m128, xmm0	0F 38 CB /r	Execute 2 rounds of SHA256

Related Instructions

SHA256MSG1, SHA256MSG2

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction

X - SSE, AVX, and AVX2 exception
A - AVX, AVX2 exception
S - SSE exception

SHA256MSG1 Message Intermediate 1

Performs the 1st of two intermediate calculations necessary for the next four SHA256 message dwords.

```
DEST[127:96] ← SRC1[127:96] + Perm2( SRC2[31:0])
DEST[95:64]  ← SRC1[95:64]  + Perm2( SRC1[127:96])
DEST[63:32]  ← SRC1[63:32]  + Perm2( SRC1[95:64]
DEST[31:0]   ← SRC1[31:0]   + Perm2( SRC1[63:62])
```

Mnemonic	Opcode	Description
SHA256MSG1xmm1, xmm2/m128	0F 38 CC /r	Calculate Message Intermediate 1

Related Instructions

SHA256RNDS2, SHA256MSG2

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction
X - SSE, AVX, and AVX2 exception A - AVX, AVX2 exception S - SSE exception				

SHA256MSG2 Message Intermediate 2

Performs the 2nd of two intermediate calculations necessary for the next four SHA256 message dwords.

```
Temp0       ← SRC1[31:0]    + Perm3( SRC2[95:64])
Temp1       ← SRC1[63:32]   + Perm3( SRC2[127:96])

DEST[127:96] ← SRC1[127:96] + Perm3( Temp1)
DEST[95:64]  ← SRC1[95:64]  + Perm3( Temp0)
DEST[63:32]  ← SRC1[63:32]  + Perm3( SRC2[127:96])
DEST[31:0]   ← SRC1[31:0]   + Perm3( SRC2[95:624])
```

Mnemonic	Opcode	Description
SHA256MSG1 xmm1, xmm2/m128	0F 38 CD /r	Calculate Message Intermediate 2

Related Instructions

SHA256RNDS2, SHA256MSG1

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X	X	Instruction not supported by CPUID
	A	A		AVX instructions are only recognized in protected mode
	S	S	S	CR0.EM=1 OR CR4.OSFXSR=0
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE]
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.

Exceptions	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP	S	S	X	Memory address exceeding data segment limit or non-canonical.
			X	Null data segment used to reference memory
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page Fault, #PF		S	X	A page fault resulted from the execution of the instruction
X - SSE, AVX, and AVX2 exception A - AVX, AVX2 exception S - SSE exception				

SHUFPD Shuffle
VSHUFPD 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 an 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 encodings:

XMM Encoding

Shuffles four source values. 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. 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.

Instruction Support

Form	Subset	Feature Flag
SHUFPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VSHUFPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSHUFPD *xmm1*, *xmm2, xmm3*/*mem128*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	C6 /r
VSHUFPD *ymm1*, *ymm2, ymm3*/*mem256*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	C6 /r

Related Instructions

(V)SHUFPS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

SHUFPS Shuffle
VSHUFPS 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 an 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 encodings:

XMM Encoding

Shuffles eight source values. 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. 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.

Instruction Support

Form	Subset	Feature Flag
SHUFPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VSHUFPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSHUFPS *xmm1*, *xmm2, xmm3*/*mem128*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.00	C6 /r
VSHUFPS *ymm1*, *ymm2, ymm3*/*mem256*, *imm8*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.00	C6 /r

Related Instructions

(V)SHUFPD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

SQRTPD Square Root
VSQRTPD 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 destination is an XMM register. The source 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.

VSQRTPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Computes two 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 four values. The source operand is either a YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
SQRTPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VSQRTPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSQRTPD *xmm1, xmm2/mem128*	C4	$\overline{\text{RXB}}$.01	X.1111.0.01	51 /r
VSQRTPD *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.01	X.1111.1.01	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPS, (V)SQRTSD, (V)SQRTSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 destination is an XMM register. The source 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.

VSQRTPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Computes four values. The destination is an XMM register. The source 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

Computes eight values. The destination is a YMM register. The source operand is either a YMM register or a 256-bit memory location.

Instruction Support

Form	Subset	Feature Flag
SQRTPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VSQRTPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSQRTPS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.01	X.1111.0.00	51 /r
VSQRTPS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.01	X.1111.1.00	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPD, (V)SQRTSD, (V)SQRTSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Square Root
VSQRTSD Scalar Double-Precision Floating-Point

Computes the square root of a double-precision floating-point value and writes the result to the low quadword of the destination. The three-operand form of the instruction also writes a copy of the upper quadword of a second source operand to the upper quadword 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 quadword. 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 a single 128-bit encoding that requires three operands:

VSQRTSD *xmm1, xmm2, xmm3/mem64*

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 quadword. The destination is a third XMM register. The square root of the second source operand is written to bits [63:0] of the destination register. Bits [127:64] of the destination are copied from the corresponding bits of the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
SQRTSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VSQRTSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSQRTSD *xmm1*, *xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.X.11	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Square Root
VSQRTSS Scalar Single-Precision Floating-Point

Computes the square root of a single-precision floating-point value and writes the result to the low doubleword of the destination. The three-operand form of the instruction also writes a copy of the three most significant doublewords of a second source operand to the upper 96 bits 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 doubleword. 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 has a single 128-bit encoding that requires three operands:

VSQRTSS *xmm1, xmm2, xmm3/mem64*

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 doubleword. The destination is a third XMM register. The square root of the second source operand is written to bits [31:0] of the destination register. Bits [127:32] of the destination are copied from the corresponding bits of the first source operand. Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
SQRTSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VSQRTSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSQRTSS *xmm1*, *xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.X.10	51 /r

Related Instructions

(V)RSQRTPS, (V)RSQRTSS, (V)SQRTPD, (V)SQRTPS, (V)SQRTSD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Store MXCSR
VSTMXCSR

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.

There is one encoding for each instruction form.

Instruction Support

Form	Subset	Feature Flag
STMXCSR	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VSTMXCSR	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
STMXCSR *mem32*	0F AE /3	Stores content of MXCSR in *mem32*.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VSTMXCSR *mem32*	C4	$\overline{\text{RXB}}$.01	X.1111.0.00	AE /3

Related Instructions

(V)LDMXCSR

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

SUBPD Subtract
VSUBPD 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 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.

VSUBPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Subtracts two pairs of values. 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.

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.

Instruction Support

Form	Subset	Feature Flag
SUBPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VSUBPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSUBPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	5C /r
VSUBPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	5C /r

Related Instructions

(V)SUBPS, (V)SUBSD, (V)SUBSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Subtract
VSUBPS 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 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.

VSUBPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Subtracts four pairs of values. 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.

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.

Instruction Support

Form	Subset	Feature Flag
SUBPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VSUBPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSUBPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.0.00	5C /r
VSUBPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.00001	X.$\overline{src}$.1.00	5C /r

Related Instructions

(V)SUBPD, (V)SUBSD, (V)SUBSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Subtract
VSUBSD 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

There are legacy and extended forms of the instruction:

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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
SUBSD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VSUBSD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSUBSD *xmm1*, *xmm2*, *xmm3*/*mem64*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.X.11	5C /r

Related Instructions

(V)SUBPD, (V)SUBPS, (V)SUBSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Subtract
VSUBSS 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

There are legacy and extended forms of the instruction:

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 a 128-bit encoding only.

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.

Instruction Support

Form	Subset	Feature Flag
SUBSS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VSUBSS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VSUBSS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.01	X.$\overline{\textit{src1}}$.X.10	5C /r

Related Instructions

(V)SUBPD, (V)SUBPS, (V)SUBSD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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 when 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 Unordered Compare
VUCOMISD 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 an 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.

Instruction Support

Form	Subset	Feature Flag
UCOMISD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VUCOMISD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	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	XFEATURE_ENABLED_MASK[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 when 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 Unordered Compare
VUCOMISS 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 single-precision floating-point value in the low-order 32 bits of an 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.

Instruction Support

Form	Subset	Feature Flag
UCOMISS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VUCOMISS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
UCOMISS *xmm1*, *xmm2/mem32*	0F 2E /r	Compares scalar single-precision floating-point values in *xmm1* and *xmm2* or *mem64*. Sets rFLAGS.

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VUCOMISS *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.01	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	XFEATURE_ENABLED_MASK[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 when 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 Unpack High
VUNPCKHPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
UNPCKHPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VUNPCKHPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VUNPCKHPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	15 /r
VUNPCKHPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	15 /r

Related Instructions

(V)UNPCKHPS, (V)UNPCKLPD, (V)UNPCKLPS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

UNPCKHPS Unpack High
VUNPCKHPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
UNPCKHPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VUNPCKHPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VUNPCKHPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.00	15 /r
VUNPCKHPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.00	15 /r

Related Instructions

(V)UNPCKHPD, (V)UNPCKLPD, (V)UNPCKLPS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

UNPCKLPD Unpack Low
VUNPCKLPD 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
UNPCKLPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VUNPCKLPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VUNPCKLPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	14 /r
VUNPCKLPD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	14 /r

Related Instructions

(V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPS

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

UNPCKLPS Unpack Low
VUNPCKLPS 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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
UNPCKLPS	SSE1	CPUID Fn0000_0001_EDX[SSE] (bit 25)
VUNPCKLPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VUNPCKLPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.00	14 /r
VUNPCKLPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.00	14 /r

Related Instructions

(V)UNPCKHPD, (V)UNPCKHPS, (V)UNPCKLPD

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 a single 256-bit encoding.

The source operand is a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VBROADCASTF128	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VBROADCASTF128 *ymm1*, *mem128*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	1A /r

Related Instructions

VBROADCASTSD, VBROADCASTSS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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.

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VBROADCASTI128 Load With Broadcast Integer From 128-bit Memory Location

Loads data from a 128-bit memory location and writes it to the two 128-bit elements of a YMM register

There is a single form of this instruction:

VBROADCASTI128 *dest, mem128*

There is a single VEX.L = 1 encoding of this instruction.

The source operand is a 128-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VBROADCASTI128	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

Instruction Encoding

		Encoding		
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VBROADCASTI128 *ymm1*, *mem128*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	5A /r

Related Instructions

VBROADCASTF128, VEXTRACTF128, VEXTRACTI128, VINSERTF128, VINSERTI128

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	Register-based source operand specified (MODRM.mod = 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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VBROADCASTSD Load With Broadcast Scalar Double

Loads a double-precision floating-point value from a register or memory and writes it to the four 64-bit elements of a YMM register

This extended-form instruction has a single 256-bit encoding.

The source operand is the lower half of an XMM register or a 64-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VBROADCASTSD *ymm1*, *mem64*	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VBROADCASTSD *ymm1*, *xmm*	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VBROADCASTSD *ymm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	19 /r

Related Instructions

VBROADCASTF128, VBROADCASTSS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	Register-based source operand specified when AVX2 not supported.
			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	Unaligned memory reference when alignment checking enabled.
A — AVX, AVX2 exception.				

VBROADCASTSS Load With Broadcast Scalar Single

Loads a single-precision floating-point value from a register or memory and writes it to all 4 or 8 doublewords of an XMM or YMM register.

This extended-form instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Copies the source operand to all four 32-bit elements of the destination.

The source operand is the least-significant 32 bits of an XMM register or a 32-bit memory location. The destination is an XMM register.

YMM Encoding

Copies the source operand to all eight 32-bit elements of the destination.

The source operand is the least-significant 32 bits of an XMM register or a 32-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VBROADCASTSS *mem32*	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)
VBROADCASTSS *xmm*	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VBROADCASTSS *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	18 /r
VBROADCASTSS *ymm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	18 /r

Related Instructions

VBROADCASTF128, VBROADCASTSD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	MODRM.mod = 11b when AVX2 not supported.
			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	Unaligned memory reference when alignment checking enabled.
A — AVX, AVX2 exception.				

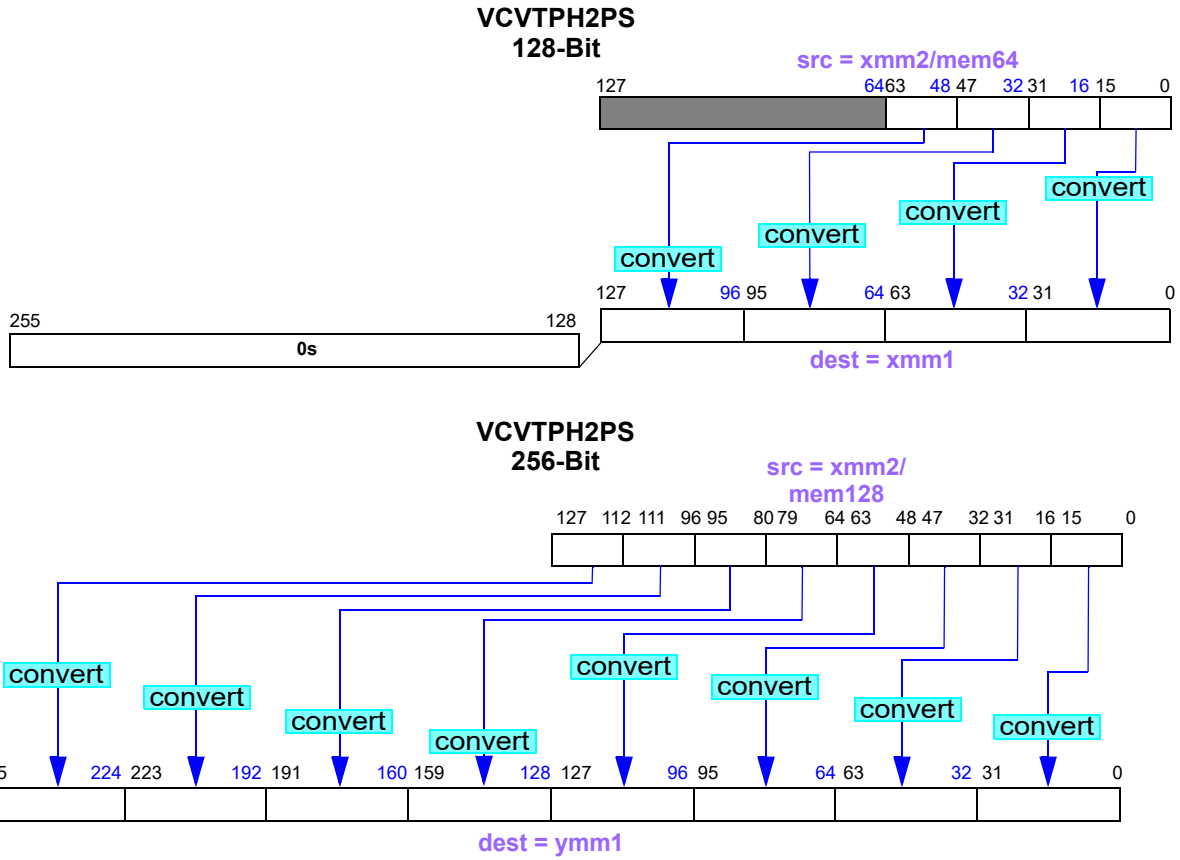
VCVTPH2PS Convert Packed 16-Bit Floating-Point to Single-Precision Floating-Point

Converts packed 16-bit floating point values to single-precision floating point values.

A denormal source operand is converted to a normal result in the destination register. MXCSR.DAZ is ignored and no MXCSR denormal exception is reported.

Because the full range of 16-bit floating-point encodings, including denormal encodings, can be represented exactly in single-precision format, rounding, inexact results, and denormalized results are not applicable.

The operation of this instruction is illustrated in the following diagram.



This extended-form instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Converts four packed 16-bit floating-point values in the low-order 64 bits of an XMM register or in a 64-bit memory location to four packed single-precision floating-point values and writes the converted values to an XMM destination register. When the result operand is written to the destination register, the upper 128 bits of the corresponding YMM register are zeroed.

YMM Encoding

Converts eight packed 16-bit floating-point values in the low-order 128 bits of a YMM register or in a 128-bit memory location to eight packed single-precision floating-point values and writes the converted values to a YMM destination register.

Instruction Support

Form	Subset	Feature Flag
VCVTPH2PS	F16C	CPUID Fn0000_0001_ECX[F16C] (bit 29)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VCVTPH2PS *xmm1*, *xmm2*/*mem64*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	13 /r
VCVTPH2PS *ymm1*, *xmm2*/*mem128*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	13 /r

Related Instructions

VCVTPS2PH

rFLAGS Affected

None

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: A flag that may be set to one or cleared to zero is M (modified). Unaffected flags are blank.

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		AVX instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			F	VEX.W field = 1.
			A	VEX.vvvv ! = 1111b.
			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.
Alignment check, #AC			F	Unaligned memory reference when alignment checking enabled.
Page fault, #PF			F	Instruction execution caused a page fault.
SIMD Floating-Point Exception, #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 exception (IE)			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized-operand exception (DE)			F	A source operand was a denormal value.
Overflow exception (OE)			F	Rounded result too large to fit into the format of the destination operand.
Underflow exception (UE)			F	Rounded result too small to fit into the format of the destination operand.
Precision exception (PE)			F	A result could not be represented exactly in the destination format.

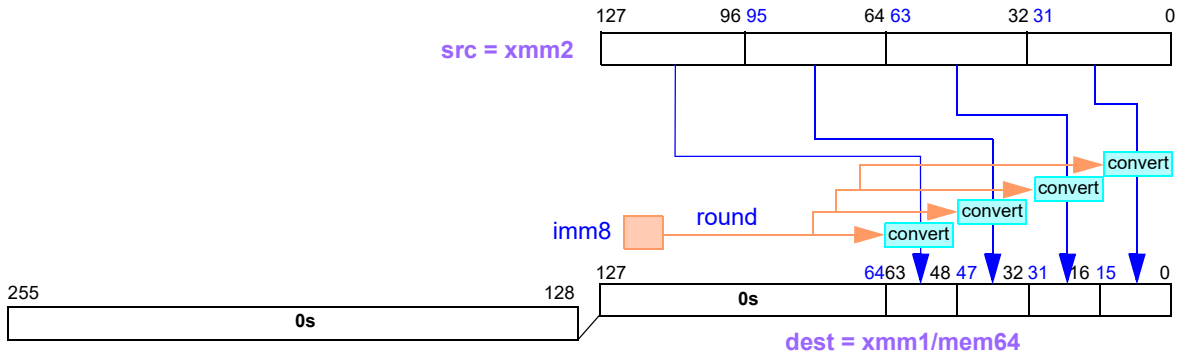
F — F16C exception.

VCVTPS2PH Convert Packed Single-Precision Floating-Point to 16-Bit Floating-Point

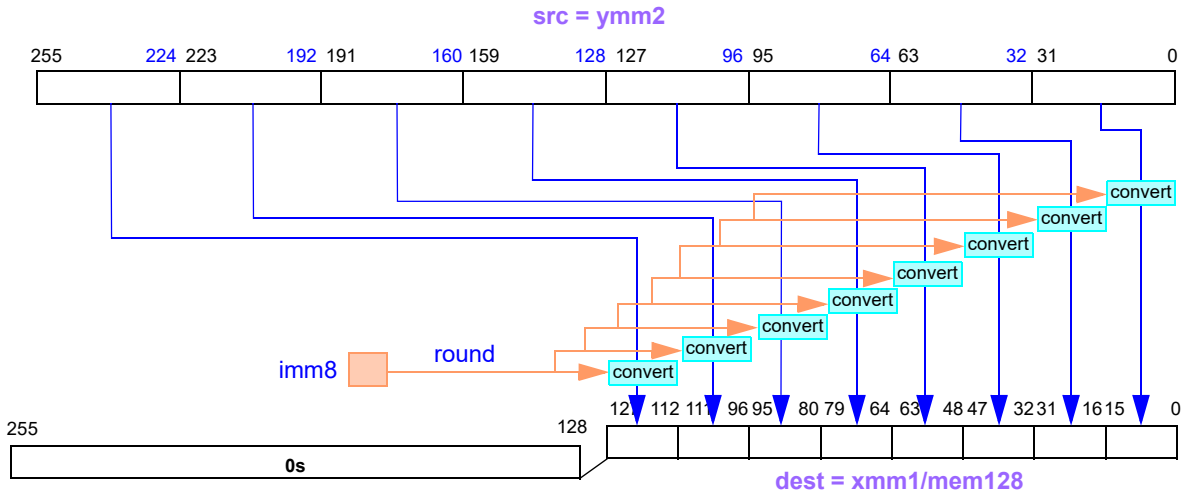
Converts packed single-precision floating-point values to packed 16-bit floating-point values and writes the converted values to the destination register or to memory. An 8-bit immediate operand provides dynamic control of rounding.

The operation of this instruction is illustrated in the following diagram.

VCVTPS2PH
128-Bit



VCVTPS2PH
256-Bit



The handling of rounding is controlled by fields in the immediate byte, as shown in the following table.

Rounding Control with Immediate Byte Operand

Mnemonic	Rounding Source (RS)	Rounding Control (RC)		Description	Notes
Bit	2	1	0		
Value	0	0	0	Nearest	Ignore MXCSR.RC.
		0	1	Down	
		1	0	Up	
		1	1	Truncate	
	1	X	X	Use MXCSR.RC for rounding.	

MXCSR[FTZ] has no effect on this instruction. Values within the half-precision denormal range are unconditionally converted to denormals.

This extended-form instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Converts four packed single-precision floating-point values in an XMM register to four packed 16-bit floating-point values and writes the converted values to the low-order 64 bits of the destination XMM register or to a 64-bit memory location. When the result is written to the destination XMM register, the high-order 64 bits in the destination XMM register and the upper 128 bits of the corresponding YMM register are cleared to 0s.

YMM Encoding

Converts eight packed single-precision floating-point values in a YMM register to eight packed 16-bit floating-point values and writes the converted values to the low-order 128 bits of a YMM register or to a 128-bit memory location. When the result is written to the destination YMM register, the high-order 128 bits in the register are cleared to 0s.

Instruction Support

Form	Subset	Feature Flag
VCVTPH2PH	F16C	CPUID Fn0000_0001_ECX[F16C] (bit 29)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VCVTPS2PH *xmm1*/*mem64*, *xmm2, imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.0.01	1D /r /imm8
VCVTPS2PH *xmm1*/*mem128*, *ymm2, imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.1.01	1D /r /imm8

Related Instructions

VCVTPH2PS

rFLAGS Affected

None

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 to one or cleared to zero is M (modified). Unaffected flags are blank.

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		AVX instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			F	VEX.W field = 1.
			A	VEX.vvvv ! = 1111b.
			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.
Alignment check, #AC			F	Unaligned memory reference when alignment checking enabled.
Page fault, #PF			F	Instruction execution caused a page fault.
SIMD Floating-Point Exception, #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 exception (IE)			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized-operand exception (DE)			F	A source operand was a denormal value.
Overflow exception (OE)			F	Rounded result too large to fit into the format of the destination operand.
Underflow exception (UE)			F	Rounded result too small to fit into the format of the destination operand.
Precision exception (PE)			F	A result could not be represented exactly in the destination format.

F — F16C exception.

VEXTRACTF128 Extract Packed Floating-Point Values

Extracts 128 bits of packed 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 a single 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.

Instruction Support

Form	Subset	Feature Flag
VEXTRACTF128	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VEXTRACTF128 *xmm/mem128, ymm, imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.1.01	19 /r ib

Related Instructions

VBROADCASTF128, VINSERTF128

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
A — AVX exception.				

VEXTRACTI128 Extract 128-bit Integer

Writes a selected 128-bit half of a YMM register to an XMM register or a 128-bit memory location based on the value of bit 0 of an immediate byte.

There is a single form of this instruction:

VEXTRACTI128 *dest, src, imm8*

If imm8[0] = 0, the lower half of the source YMM register is selected; if imm8[0] = 1, the upper half of the source register is selected.

There is a single VEX.L = 1 encoding of this instruction.

The source operand is a YMM register. The destination is either an XMM register or a 128-bit memory location. When the destination is a register, bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VEXTRACTI128	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

Instruction Encoding

	Encoding			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VEXTRACTI128 *xmm1/mem128, ymm2, imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.1.01	39 /r ib

Related Instructions

VBROADCASTF128, VBROADCASTI128, VEXTRACTF128, VINSERTF128, VINSERTI128

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VFMADDPD Multiply and Add
VFMADD132PD Packed Double-Precision Floating-Point
VFMADD213PD
VFMADD231PD

Multiplies together two double-precision floating-point vectors and adds the unrounded product to a third double-precision floating-point vector producing a precise result which is then rounded to double-precision based on the mode specified by the MXCSR[RC] field. The rounded sum is written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFMADDPD dest, src1, src2/mem, src3     // dest = (src1* src2/mem) + src3
VFMADDPD dest, src1, src2, src3/mem     // dest = (src1* src2) + src3/mem
```

and three three-operand forms:

```
VFMADD132PD scr1, src2, src3/mem        // src1 = (src1* src3/mem) + src2
VFMADD213PD scr1, src2, src3/mem        // src1 = (src2* src1) + src3/mem
VFMADD231PD scr1, src2, src3/mem        // src1 = (src2* src3/mem) + src1
```

When VEX.L = 0, the vector size is 128 bits (two double-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (four double-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMADDPD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMADD*nnn*PD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	69 /r /is4
VFMADDPD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	69 /r /is4
VFMADDPD *xmm1*, *xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	69 /r /is4
VFMADDPD *ymm1*, *ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	69 /r /is4
VFMADD132PD *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	98 /r
VFMADD132PD *ymm0, ymm1, ymm2/m256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	98 /r
VFMADD213PD *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	A8 /r
VFMADD213PD *ymm0, ymm1, ymm2/m256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	A8 /r
VFMADD231PD *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	B8 /r
VFMADD231PD *ymm0, ymm1, ymm2/m256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	B8 /r

Related Instructions

VFMADDPS, VFMADD132PS, VFMADD213PS, VFMADD231PS, VFMADDSD, VFMADD132SD, VFMADD213SD, VFMADD231SD, VFMADDSS, VFMADD132SS, VFMADD213SS, VFMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMADDPS VFMADD132PS VFMADD213PS VFMADD231PS — Multiply and Add Packed Single-Precision Floating-Point

Multiplies together two single-precision floating-point vectors and adds the unrounded product to a third single-precision floating-point vector producing a precise result which is then rounded to single-precision based on the mode specified by the MXCSR[RC] field. The rounded sum is written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFMADDPS dest, src1, src2/mem, src3     // dest = (src1* src2/mem) + src3
VFMADDPS dest, src1, src2, src3/mem     // dest = (src1* src2) + src3/mem
```

and three three-operand forms:

```
VFMADD132PS scr1, src2, src3/mem        // src1 = (src1* src3/mem) + src2
VFMADD213PS scr1, src2, src3/mem        // src1 = (src2* src1) + src3/mem
VFMADD231PS scr1, src2, src3/mem        // src1 = (src2* src3/mem) + src1
```

When VEX.L = 0, the vector size is 128 bits (four single-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (eight single-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMADDPS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMADD*nnn*PS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	68 /r /is4
VFMADDPS *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	68 /r /is4
VFMADDPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	68 /r /is4
VFMADDPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	68 /r /is4
VFMADD132PS *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	98 /r
VFMADD132PS *ymm0, ymm1, ymm2/m256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	98 /r
VFMADD213PS *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	A8 /r
VFMADD213PS *ymm0, ymm1, ymm2/m256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	A8 /r
VFMADD231PS *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	B8 /r
VFMADD231PS *ymm0, ymm1, ymm2/m256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	B8 /r

Related Instructions

VFMADDPD, VFMADD132PD, VFMADD213PD, VFMADD231PD, VFMADDSD, VFMADD132SD, VFMADD213SD, VFMADD231SD, VFMADDSS, VFMADD132SS, VFMADD213SS, VFMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMADDSD Multiply and Add
VFMADD132SD Scalar Double-Precision Floating-Point
VFMADD213SD
VFMADD231SD

Multiplies together two double-precision floating-point values and adds the unrounded product to a third double-precision floating-point value producing a precise result which is then rounded to double-precision based on the mode specified by the MXCSR[RC] field. The rounded sum is written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

```
VFMADDSD dest, src1, src2/mem64, src3     // dest = (src1* src2/mem64) + src3
VFMADDSD dest, src1, src2, src3/mem64     // dest = (src1* src2) + src3/mem64
```

and three three-operand forms:

```
VFMADD132SD scr1, src2, src3/mem64        // src1 = (src1* src3/mem64) + src2
VFMADD213SD scr1, src2, src3/mem64        // src1 = (src2* src1) + src3/mem64
VFMADD231SD scr1, src2, src3/mem64        // src1 = (src2* src3/mem64) + src1
```

All 64-bit double-precision floating-point register-based operands are held in the lower quadword of XMM registers. The result is written to the lower quadword of the destination register. For those instructions that use a memory-based operand, one of the source operands is a 64-bit value read from memory.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third operand is either 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.

Instruction Support

Form	Subset	Feature Flag
VFMADDSD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMADD*nnn*SD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMADDSD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	6B /r /is4
VFMADDSD *xmm1*, *xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	6B /r /is4
VFMADD132SD *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	99 /r
VFMADD213SD *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	A9 /r
VFMADD231SD *xmm0, xmm1, xmm2/m128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	B9 /r

Related Instructions

VFMADDPD, VFMADD132PD, VFMADD213PD, VFMADD231PD, VFMADDPS, VFMADD132PS, VFMADD213PS, VFMADD231PS, VFMADDSS, VFMADD132SS, VFMADD213SS, VFMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFMADDSS VFMADD132SS VFMADD213SS VFMADD231SS — Multiply and Add Scalar Single-Precision Floating-Point

Multiplies together two single-precision floating-point values and adds the unrounded product to a third single-precision floating-point value producing a precise result which is then rounded to single-precision based on the mode specified by the MXCSR[RC] field. The rounded sum is written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

```
VFMADDSS dest, src1, src2/mem32, src3     // dest = (src1* src2/mem32) + src3
VFMADDSS dest, src1, src2, src3/mem32     // dest = (src1* src2) + src3/mem32
```

and three three-operand forms:

```
VFMADD132SS scr1, src2, src3/mem32        // src1 = (src1* src3/mem32) + src2
VFMADD213SS scr1, src2, src3/mem32        // src1 = (src2* src1) + src3/mem32
VFMADD231SS scr1, src2, src3/mem32        // src1 = (src2* src3/mem32) + src1
```

All 32-bit single-precision floating-point register-based operands are held in the lower doubleword of XMM registers. The result is written to the low doubleword of the destination register. For those instructions that use a memory-based operand, one of the source operands is a 32-bit value read from memory.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third operand 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.

Instruction Support

Form	Subset	Feature Flag
VFMADDSS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMADD*nnn*SS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMADDSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	6A /r /is4
VFMADDSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	6A /r /is4
VFMADD132SS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	99 /r
VFMADD213SS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	A9 /r
VFMADD231SS *xmm1*, *xmm2*, *xmm3*/*mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	B9 /r

Related Instructions

VFMADDPD, VFMADD132PD, VFMADD213PD, VFMADD231PD, VFMADDPS, VFMADD132PS, VFMADD213PS, VFMADD231PS, VFMADDSD, VFMADD132SD, VFMADD213SD, VFMADD231SD

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFMADDSUBPD VFMADDSUB132PD VFMADDSUB213PD VFMADDSUB231PD — Multiply with Alternating Add/Subtract Packed Double-Precision Floating-Point

Multiplies together two double-precision floating-point vectors, adds odd elements of the unrounded product to odd elements of a third double-precision floating-point vector, and subtracts even elements of the third floating point vector from even elements of unrounded product. The precise result of each addition or subtraction is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the corresponding element of the destination.

The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

VFMADDSUBPD *dest, src1, src2/mem, src3*	// $dest_{odd} = (src1_{odd} * src2_{odd}/mem_{odd}) + src3_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}/mem_{even}) - src3_{even}$
VFMADDSUBPD *dest, src1, src2, src3/mem*	// $dest_{odd} = (src1_{odd} * src2_{odd}) + src3_{odd}/mem_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}) - src3_{even}/mem_{even}$

and three three-operand forms:

VFMADDSUB132PD *scr1, src2, src3/mem*	// $src1_{odd} = (src1_{odd} * src3_{odd}/mem_{odd}) + src2_{odd}$ // $src1_{even} = (src1_{even} * src3_{even}/mem_{even}) - src2_{even}$
VFMADDSUB213PD *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src1_{odd}) + src3_{odd}/mem_{odd}$ // $src1_{even} = (src2_{even} * src1_{even}) - src3_{even}/mem_{even}$
VFMADDSUB231PD *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src3_{odd}/mem_{odd}) + src1_{odd}$ // $src1_{even} = (src2_{even} * src3_{even}/mem_{even}) - src1_{even}$

When VEX.L = 0, the vector size is 128 bits (two double-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (four double-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMADDSUBPD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMADDSUB*nnn*PD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMADDSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	5D /r /is4
VFMADDSUBPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	5D /r /is4
VFMADDSUBPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	5D /r /is4
VFMADDSUBPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	5D /r /is4
VFMADDSUB132PD *xmm1*, *xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	96 /r
VFMADDSUB132PD *ymm1*, *ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	96 /r
VFMADDSUB213PD *xmm1*, *xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	A6 /r
VFMADDSUB213PD *ymm1*, *ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	A6 /r
VFMADDSUB231PD *xmm1*, *xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	B6 /r
VFMADDSUB231PD *ymm1*, *ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	B6 /r

Related Instructions

VFMSUBADDPD, VFMSUBADD132PD, VFMSUBADD213PD, VFMSUBADD231PD, VFMADDSUBPS, VFMADDSUB132PS, VFMADDSUB213PS, VFMADDSUB231PS, VFMSUBADDPS, VFMSUBADD132PS, VFMSUBADD213PS, VFMSUBADD231PS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMADDSUBPS VFMADDSUB132PS VFMADDSUB213PS VFMADDSUB231PS — Multiply with Alternating Add/Subtract Packed Single-Precision Floating-Point

Multiplies together two single-precision floating-point vectors, adds odd elements of the unrounded product to odd elements of a third single-precision floating-point vector, and subtracts even elements of the third floating point vector from even elements of unrounded product. The precise result of each addition or subtraction is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the corresponding element of the destination.

The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

VFMADDSUBPS *dest, src1, src2/mem, src3*	// $dest_{odd} = (src1_{odd} * src2_{odd}/mem_{odd}) + src3_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}/mem_{even}) - src3_{even}$
VFMADDSUBPS *dest, src1, src2, src3/mem*	// $dest_{odd} = (src1_{odd} * src2_{odd}) + src3_{odd}/mem_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}) - src3_{even}/mem_{even}$

and three three-operand forms:

VFMADDSUB132PS *scr1, src2, src3/mem*	// $src1_{odd} = (src1_{odd} * src3_{odd}/mem_{odd}) + src2_{odd}$ // $src1_{even} = (src1_{even} * src3_{even}/mem_{even}) - src2_{even}$
VFMADDSUB213PS *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src1_{odd}) + src3_{odd}/mem_{odd}$ // $src1_{even} = (src2_{even} * src1_{even}) - src3_{even}/mem_{even}$
VFMADDSUB231PS *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src3_{odd}/mem_{odd}) + src1_{odd}$ // $src1_{even} = (src2_{even} * src3_{even}/mem_{even}) - src1_{even}$

When VEX.L = 0, the vector size is 128 bits (four single-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (eight single-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMADDSUBPS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMADDSUB*nnn*PS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMADDSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	5C /r /is4
VFMADDSUBPS *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	5C /r /is4
VFMADDSUBPS *xmm1*, *xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	5C /r /is4
VFMADDSUBPS *ymm1*, *ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	5C /r /is4
VFMADDSUB132PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	96 /r
VFMADDSUB132PS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	96 /r
VFMADDSUB213PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	A6 /r
VFMADDSUB213PS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	A6 /r
VFMADDSUB231PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	B6 /r
VFMADDSUB231PS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	B6 /r

Related Instructions

VFMADDSUBPD, VFMADDSUB132PD, VFMADDSUB213PD, VFMADDSUB231PD, VFMSUBADDPD, VFMSUBADD132PD, VFMSUBADD213PD, VFMSUBADD231PD, VFMSUBADDPS, VFMSUBADD132PS, VFMSUBADD213PS, VFMSUBADD231PS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMSUBADDPD VFMSUBADD132PD VFMSUBADD213PD VFMSUBADD231PD — Multiply with Alternating Subtract/Add Packed Double-Precision Floating-Point

Multiplies together two double-precision floating-point vectors, adds even elements of the unrounded product to even elements of a third double-precision floating-point vector, and subtracts odd elements of the third floating point vector from odd elements of unrounded product. The precise result of each addition or subtraction is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the corresponding element of the destination.

The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

VFMSUBADDPD *dest, src1, src2/mem, src3*	// $dest_{odd} = (src1_{odd} * src2_{odd}/mem_{odd}) - src3_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}/mem_{even}) + src3_{even}$
VFMSUBADDPD *dest, src1, src2, src3/mem*	// $dest_{odd} = (src1_{odd} * src2_{odd}) - src3_{odd}/mem_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}) + src3_{even}/mem_{even}$

and three three-operand forms:

VFMSUBADD132PD *scr1, src2, src3/mem*	// $src1_{odd} = (src1_{odd} * src3_{odd}/mem_{odd}) - src2_{odd}$ // $src1_{even} = (src1_{even} * src3_{even}/mem_{even}) + src2_{even}$
VFMSUBADD213PD *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src1_{odd}) - src3_{odd}/mem_{odd}$ // $src1_{even} = (src2_{even} * src1_{even}) + src3_{even}/mem_{even}$
VFMSUBADD231PD *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src3_{odd}/mem_{odd}) - src1_{odd}$ // $src1_{even} = (src2_{even} * src3_{even}/mem_{even}) + src1_{even}$

For VEX.L = 0, vector size is 128 bits and register-based operands are held in XMM registers. For VEX.L = 1, vector size is 256 bits and register-based operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMSUBADDPD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMSUBADD*nnn*PD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMSUBADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	5F /r /is4
VFMSUBADDPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	5F /r /is4
VFMSUBADDPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	5F /r /is4
VFMSUBADDPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	5F /r /is4
VFMSUBADD132PD *xmm1, xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	97 /r
VFMSUBADD132PD *ymm1, ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	97 /r
VFMSUBADD213PD *xmm1, xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	A7 /r
VFMSUBADD213PD *ymm1, ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	A7 /r
VFMSUBADD231PD *xmm1, xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	B7 /r
VFMSUBADD231PD *ymm1, ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	B7 /r

Related Instructions

VFMADDSUBPD, VFMADDSUB132PD, VFMADDSUB213PD, VFMADDSUB231PD, VFMADDSUBPS, VFMADDSUB132PS, VFMADDSUB213PS, VFMADDSUB231PS, VFMSUBADDPS, VFMSUBADD132PS, VFMSUBADD213PS, VFMSUBADD231PS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMSUBADDPS VFMSUBADD132PS VFMSUBADD213PS VFMSUBADD231PS — Multiply with Alternating Subtract/Add Packed Single-Precision Floating-Point

Multiplies together two single-precision floating-point vectors, adds even elements of the unrounded product to even elements of a third single-precision floating-point vector, and subtracts odd elements of the third floating point vector from odd elements of unrounded product. The precise result of each addition or subtraction is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the corresponding element of the destination.

The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

VFMSUBADDPS *dest, src1, src2/mem, src3*	// $dest_{odd} = (src1_{odd} * src2_{odd}/mem_{odd}) - src3_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}/mem_{even}) + src3_{even}$
VFMSUBADDPS *dest, src1, src2, src3/mem*	// $dest_{odd} = (src1_{odd} * src2_{odd}) - src3_{odd}/mem_{odd}$ // $dest_{even} = (src1_{even} * src2_{even}) + src3_{even}/mem_{even}$

and three three-operand forms:

VFMSUBADD132PS *scr1, src2, src3/mem*	// $src1_{odd} = (src1_{odd} * src3_{odd}/mem_{odd}) - src2_{odd}$ // $src1_{even} = (src1_{even} * src3_{even}/mem_{even}) + src2_{even}$
VFMSUBADD213PS *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src1_{odd}) - src3_{odd}/mem_{odd}$ // $src1_{even} = (src2_{even} * src1_{even}) + src3_{even}/mem_{even}$
VFMSUBADD231PS *scr1, src2, src3/mem*	// $src1_{odd} = (src2_{odd} * src3_{odd}/mem_{odd}) - src1_{odd}$ // $src1_{even} = (src2_{even} * src3_{even}/mem_{even}) + src1_{even}$

When VEX.L = 0, the vector size is 128 bits (four single-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (eight single-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMSUBADDPS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMSUBADD*nnn*PS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMSUBADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	5E /r /is4
VFMSUBADDPS *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	5E /r /is4
VFMSUBADDPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	5E /r /is4
VFMSUBADDPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	5E /r /is4
VFMSUBADD132PS *xmm1, xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src2}$.0.01	97 /r
VFMSUBADD132PS *ymm1, ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src2}$.1.01	97 /r
VFMSUBADD213PS *xmm1, xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src2}$.0.01	A7 /r
VFMSUBADD213PS *ymm1, ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src2}$.1.01	A7 /r
VFMSUBADD231PS *xmm1, xmm2, xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src2}$.0.01	B7 /r
VFMSUBADD231PS *ymm1, ymm2, ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.00010	0.$\overline{src2}$.1.01	B7 /r

Related Instructions

VFMADDSUBPD, VFMADDSUB132PD, VFMADDSUB213PD, VFMADDSUB231PD, VFMADDSUBPS, VFMADDSUB132PS, VFMADDSUB213PS, VFMADDSUB231PS, VFMSUBADDPD, VFMSUBADD132PD, VFMSUBADD213PD, VFMSUBADD231PD

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMSUBPD VFMSUB132PD VFMSUB213PD VFMSUB231PD — Multiply and Subtract Packed Double-Precision Floating-Point

Multiplies together two double-precision floating-point vectors and subtracts a third double-precision floating-point vector from the unrounded product to produce a precise intermediate result. The intermediate result is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFMSUBPD dest, src1, src2/mem, src3     // dest = (src1* src2/mem) − src3
VFMSUBPD dest, src1, src2, src3/mem     // dest = (src1* src2) − src3/mem
```

and three three-operand forms:

```
VFMSUB132PD scr1, src2, src3/mem        // src1 = (src1* src3/mem) − src2
VFMSUB213PD scr1, src2, src3/mem        // src1 = (src2* src1) − src3/mem
VFMSUB231PD scr1, src2, src3/mem        // src1 = (src2* src3/mem) − src1
```

For VEX.L = 0, vector size is 128 bits and register-based operands are held in XMM registers. For VEX.L = 1, vector size is 256 bits and register-based operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMSUBPD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMSUB*nnn*PD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	6D /r /is4
VFMSUBPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	6D /r /is4
VFMSUBPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	6D /r /is4
VFMSUBPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	6D /r /is4
VFMSUB132PD *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	9A /r
VFMSUB132PD *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	9A /r
VFMSUB213PD *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	AA /r
VFMSUB213PD *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	AA /r
VFMSUB231PD *xmm1*, *xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	BA /r
VFMSUB231PD *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	BA /r

Related Instructions

VFMSUBPS, VFMSUB132PS, VFMSUB213PS, VFMSUB231PPS, VFMSUBSD, VFMSUB132SD, VFMSUB213SD, VFMSUB2P31SD, VFMSUBSS, VFMSUB132SS, VFMSUB213SS, VFMSUBP231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMSUBPS VFMSUB132PS VFMSUB213PS VFMSUB231PS — Multiply and Subtract Packed Single-Precision Floating-Point

Multiplies together two single-precision floating-point vectors and subtracts a third single-precision floating-point vector from the unrounded product to produce a precise intermediate result. The intermediate result is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFMSUBPS dest, src1, src2/mem, src3     // dest = (src1* src2/mem) − src3
VFMSUBPS dest, src1, src2, src3/mem     // dest = (src1* src2) − src3/mem
```

and three three-operand forms:

```
VFMSUB132PS scr1, src2, src3/mem        // src1 = (src1* src3/mem) − src2
VFMSUB213PS scr1, src2, src3/mem        // src1 = (src2* src1) − src3/mem
VFMSUB231PS scr1, src2, src3/mem        // src1 = (src2* src3/mem) − src1
```

When VEX.L = 0, the vector size is 128 bits (four single-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (eight single-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFMSUBPS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMSUB*nnn*PS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	6C /r /is4
VFMSUBPS *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	6C /r /is4
VFMSUBPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	6C /r /is4
VFMSUBPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	6C /r /is4
VFMSUB132PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	9A /r
VFMSUB132PS *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	9A /r
VFMSUB213PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	AA /r
VFMSUB213PS *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	AA /r
VFMSUB231PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	BA /r
VFMSUB231PS *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	BA /r

Related Instructions

VFMSUBPD, VFMSUB132PD, VFMSUB213PD, VFMSUB231PD, VFMSUBSD, VFMSUB132SD, VFMSUB213SD, VFMSUB231SD, VFMSUBSS, VFMSUB132SS, VFMSUB213SS, VFMSUB231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFMSUBSD Multiply and Subtract
VFMSUB132SD Scalar Double-Precision Floating-Point
VFMSUB213SD
VFMSUB231SD

Multiplies together two double-precision floating-point values and subtracts a third double-precision floating-point value from the unrounded product to produce a precise intermediate result. The intermediate result is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFMSUBSD dest, src1, src2/mem, src3     // dest = (src1* src2/mem) − src3
VFMSUBSD dest, src1, src2, src3/mem     // dest = (src1* src2) − src3/mem
```

and three three-operand forms:

```
VFMSUB132SD scr1, src2, src3/mem        // src1 = (src1* src3/mem) − src2
VFMSUB213SD scr1, src2, src3/mem        // src1 = (src2* src1) − src3/mem
VFMSUB231SD scr1, src2, src3/mem        // src1 = (src2* src3/mem) − src1
```

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third operand is either a register or a 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.

Instruction Support

Form	Subset	Feature Flag
VFMSUBSD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMSUB*nnn*SD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMSUBSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	6F /r /is4
VFMSUBSD *xmm1*, *xmm2, xmm3, xmm4*/*mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	6F /r /is4
VFMSUB132SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	9B /r
VFMSUB213SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	AB /r
VFMSUB231SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	BB /r

Related Instructions

VFMSUBPD, VFMSUB132PD, VFMSUB213PD, VFMSUB231PD, VFMSUBPS, VFMSUB132PS, VFMSUB213PS, VFMSUB231PS, VFMSUBSS, VFMSUB132SS, VFMSUB213SS, VFMSUB231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFMSUBSS VFMSUB132SS VFMSUB213SS VFMSUB231SS — Multiply and Subtract Scalar Single-Precision Floating-Point

Multiplies together two single-precision floating-point values and subtracts a third single-precision floating-point value from the unrounded product to produce a precise intermediate result. The intermediate result is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFMSUBSS dest, src1, src2/mem, src3     // dest = (src1* src2/mem) − src3
VFMSUBSS dest, src1, src2, src3/mem     // dest = (src1* src2) − src3/mem
```

and three three-operand forms:

```
VFMSUB132SS scr1, src2, src3/mem        // src1 = (src1* src3/mem) − src2
VFMSUB213SS scr1, src2, src3/mem        // src1 = (src2* src1) − src3/mem
VFMSUB231SS scr1, src2, src3/mem        // src1 = (src2* src3/mem) − src1
```

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third operand is either a register or a 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.

Instruction Support

Form	Subset	Feature Flag
VFMSUBSS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFMSUB*nnn*SS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFMSUBSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	6E /r /is4
VFMSUBSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	6E /r /is4
VFMSUB132SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	9B /r
VFMSUB213SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	AB /r
VFMSUB231SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	BB /r

Related Instructions

VFMSUBPD, VFMSUB132PD, VFMSUB213PD, VFMSUB231PD, VFMSUBPS, VFMSUB132PS, VFMSUB213PS, VFMSUB231PS, VFMSUBSD, VFMSUB132SD, VFMSUB213SD, VFMSUB231SD

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFNMADDPD VFNMADD132PD VFNMADD213PD VFNMADD231PD — Negative Multiply and Add Packed Double-Precision Floating-Point

Multiplies together two double-precision floating-point vectors, negates the unrounded product, and adds it to a third double-precision floating-point vector. The precise result is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFNMADDPD dest, src1, src2/mem, src3     // dest = −(src1* src2/mem) + src3
VFNMADDPD dest, src1, src2, src3/mem     // dest = −(src1* src2) + src3/mem
```

and three three-operand forms:

```
VFNMADD132PD scr1, src2, src3/mem        // src1 = −(src1* src3/mem) + src2
VFNMADD213PD scr1, src2, src3/mem        // src1 = −(src2* src1) + src3/mem
VFNMADD231PD scr1, src2, src3/mem        // src1 = −(src2* src3/mem) + src1
```

When VEX.L = 0, the vector size is 128 bits (two double-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (four double-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFNMADDPD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMADD*nnn*PD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\textit{src1}}$.0.01	79 /r /is4
VFNMADDPD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\textit{src1}}$.1.01	79 /r /is4
VFNMADDPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\textit{src1}}$.0.01	79 /r /is4
VFNMADDPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\textit{src1}}$.1.01	79 /r /is4
VFNMADD132PD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src2}}$.0.01	9C /r
VFNMADD132PD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src2}}$.1.01	9C /r
VFNMADD213PD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src2}}$.0.01	AC /r
VFNMADD213PD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src2}}$.1.01	AC /r
VFNMADD231PD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src2}}$.0.01	BC /r
VFNMADD231PD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src2}}$.1.01	BC /r

Related Instructions

VFNMADDPS, VFNMADD132PS, VFNMADD213PS, VFNMADD231PS, VFNMADDSD, VFNMADD132SD, VFNMADD213SD, VFNMADD231SD, VFNMADDSS, VFNMADD132SS, VFNMADD213SS, VFNMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFNMADDPS VFNMADD132PS VFNMADD213PS VFNMADD231PS — Negative Multiply and Add Packed Single-Precision Floating-Point

Multiplies together two single-precision floating-point vectors, negates the unrounded product, and adds it to a third single-precision floating-point vector. The precise result is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFNMADDPS dest, src1, src2/mem, src3    // dest = −(src1* src2/mem) + src3
VFNMADDPS dest, src1, src2, src3/mem    // dest = −(src1* src2) + src3/mem
```

and three three-operand forms:

```
VFNMADD132PS scr1, src2, src3/mem    // src1 = −(src1* src3/mem) + src2
VFNMADD213PS scr1, src2, src3/mem    // src1 = −(src2* src1) + src3/mem
VFNMADD231PS scr1, src2, src3/mem    // src1 = −(src2* src3/mem) + src1
```

When VEX.L = 0, the vector size is 128 bits (four single-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (eight single-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFNMADDPS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMADD*nnn*PS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	78 /r /is4
VFNMADDPS *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	78 /r /is4
VFNMADDPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	78 /r /is4
VFNMADDPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	78 /r /is4
VFNMADD132PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	9C / r
VFNMADD132PS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	9C / r
VFNMADD213PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	AC / r
VFNMADD213PS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	AC / r
VFNMADD231PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	BC / r
VFNMADD231PS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	BC / r

Related Instructions

VFNMADDPD, VFNMADD132PD, VFNMADD213PD, VFNMADD231PD, VFNMADDSD, VFNMADD132SD, VFNMADD213SD, VFNMADD231SD, VFNMADDSS, VFNMADD132SS, VFNMADD213SS, VFNMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFNMADDSD VFNMADD132SD VFNMADD213SD VFNMADD231SD — Negative Multiply and Add Scalar Double-Precision Floating-Point

Multiplies together two double-precision floating-point values, negates the unrounded product, and adds it to a third double-precision floating-point value. The precise result is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

```
VFNMADDSD dest, src1, src2/mem, src3     // dest = −(src1* src2/mem) + src3
VFNMADDSD dest, src1, src2, src3/mem     // dest = −(src1* src2) + src3/mem
```

and three three-operand forms:

```
VFNMADD132SD scr1, src2, src3/mem        // src1 = −(src1* src3/mem) + src2
VFNMADD213SD scr1, src2, src3/mem        // src1 = −(src2* src1) + src3/mem
VFNMADD231SD scr1, src2, src3/mem        // src1 = −(src2* src3/mem) + src1
```

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third operand is either a register or a 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.

Instruction Support

Form	Subset	Feature Flag
VFNMADDSD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMADD*nnn*SD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMADDSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	7B /r /is4
VFNMADDSD *xmm1*, *xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	7B /r /is4
VFNMADD132SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	9D /r
VFNMADD213SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	AD /r
VFNMADD231SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	BD /r

Related Instructions

VFNMADDPD, VFNMADD132PD, VFNMADD213PD, VFNMADD231PD, VFNMADDPS, VFNMADD132PS, VFNMADD213PS, VFNMADD231PS, VFNMADDSS, VFNMADD132SS, VFNMADD213SS, VFNMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFNMADDSS VFNMADD132SS VFNMADD213SS VFNMADD231SS — Negative Multiply and Add Scalar Single-Precision Floating-Point

Multiplies together two single-precision floating-point values, negates the unrounded product, and adds it to a third single-precision floating-point value. The precise result is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

```
VFNMADDSS dest, src1, src2/mem, src3    // dest = −(src1* src2/mem) + src3
VFNMADDSS dest, src1, src2, src3/mem    // dest = −(src1* src2) + src3/mem
```

and three three-operand forms:

```
VFNMADD132SS scr1, src2, src3/mem    // src1 = −(src1* src3/mem) + src2
VFNMADD213SS scr1, src2, src3/mem    // src1 = −(src2* src1) + src3/mem
VFNMADD231SS scr1, src2, src3/mem    // src1 = −(src2* src3/mem) + src1
```

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third operand is either a register or a 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.

Instruction Support

Form	Subset	Feature Flag
VFNMADDSS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMADD*nnn*SS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMADDSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	7A /r /is4
VFNMADDSS *xmm1*, *xmm2*, *xmm3, xmm4/mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	7A /r /is4
VFNMADD132SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	9D /r
VFNMADD213SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	AD /r
VFNMADD231SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	BD /r

Related Instructions

VFNMADDPD, VFNMADD132PD, VFNMADD213PD, VFNMADD231PD, VFNMADDPS, VFNMADD132PS, VFNMADD213PS, VFNMADD231PS, VFNMADDSS, VFNMADD132SS, VFNMADD213SS, VFNMADD231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFNMSUBPD VFNMSUB132PD VFNMSUB213PD VFNMSUB231PD — Negative Multiply and Subtract Packed Double-Precision Floating-Point

Multiplies together two double-precision floating-point vectors, negates the unrounded product, and subtracts a third double-precision floating-point vector from it. The precise result is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFNMSUBPD dest, src1, src2/mem, src3     // dest = −(src1* src2/mem) − src3
VFNMSUBPD dest, src1, src2, src3/mem     // dest = −(src1* src2) − src3/mem
```

and three three-operand forms:

```
VFNMSUB132PD scr1, src2, src3/mem        // src1 = −(src1* src3/mem) − src2
VFNMSUB213PD scr1, src2, src3/mem        // src1 = −(src2* src1) − src3/mem
VFNMSUB231PD scr1, src2, src3/mem        // src1 = −(src2* src3/mem) − src1
```

When VEX.L = 0, the vector size is 128 bits (two double-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (four double-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFNMSUBPD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMSUB*nnn*PD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	7D /r /is4
VFNMSUBPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	7D /r /is4
VFNMSUBPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	7D /r /is4
VFNMSUBPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	7D /r /is4
VFNMSUB132PD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	9E /r
VFNMSUB132PD *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	9E /r
VFNMSUB213PD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	AE /r
VFNMSUB213PD *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	AE /r
VFNMSUB231PD *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	BE /r
VFNMSUB231PD *ymm1*, *ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	BE /r

Related Instructions

VFNMSUBPS, VFNMSUB132PS, VFNMSUB213PS, VFNMSUB231PS, VFNMSUBSD, VFNMSUB132SD, VFNMSUB213SD, VFNMSUB231SD, VFNMSUBSS, VFNMSUB132SS, VFNMSUB213SS, VFNMSUB231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFNMSUBPS VFNMSUB132PS VFNMSUB213PS VFNMSUB231PS — Negative Multiply and Subtract Packed Single-Precision Floating-Point

Multiplies together two single-precision floating-point vectors, negates the unrounded product, and subtracts a third single-precision floating-point vector from it. The precise result is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the vector equation in the comment on the right.

There are two four-operand forms:

```
VFNMADDPS dest, src1, src2/mem, src3    // dest = −(src1* src2/mem) − src3
VFNMADDPS dest, src1, src2, src3/mem    // dest = −(src1* src2) − src3/mem
```

and three three-operand forms:

```
VFNMADD132PS scr1, src2, src3/mem    // src1 = −(src1* src3/mem) − src2
VFNMADD213PS scr1, src2, src3/mem    // src1 = −(src2* src1) − src3/mem
VFNMADD231PS scr1, src2, src3/mem    // src1 = −(src2* src3/mem) − src1
```

When VEX.L = 0, the vector size is 128 bits (four single-precision elements per vector) and register-based source operands are held in XMM registers.

When VEX.L = 1, the vector size is 256 bits (eight single-precision elements per vector) and register-based source operands are held in YMM registers.

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third 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 the destination is an XMM register (L = 0), bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFNMSUBPS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMSUB*nnn*PS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	7C /r /is4
VFNMSUBPS *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	7C /r /is4
VFNMSUBPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	7C /r /is4
VFNMSUBPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	7C /r /is4
VFNMSUB132PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	9E /r
VFNMSUB132PS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	9E /r
VFNMSUB213PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	AE /r
VFNMSUB213PS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	AE /r
VFNMSUB231PS *xmm1, xmm2, xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	BE /r
VFNMSUB231PS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	BE /r

Related Instructions

VFNMSUBPD, VFNMSUB132PD, VFNMSUB213PD, VFNMSUB231PD, VFNMSUBSD, VFNMSUB132SD, VFNMSUB213SD, VFNMSUB231SD, VFNMSUBSS, VFNMSUB132SS, VFNMSUB213SS, VFNMSUB231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

VFNMSUBSD VFNMSUB132SD VFNMSUB213SD VFNMSUB231SD — Negative Multiply and Subtract Scalar Double-Precision Floating-Point

Multiplies together two double-precision floating-point values, negates the unrounded product, and subtracts a third double-precision floating-point value from it. The precise result is then rounded to double-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

```
VFNMSUBSD dest, src1, src2/mem, src3    // dest = −(src1* src2/mem) − src3
VFNMSUBSD dest, src1, src2, src3/mem    // dest = −(src1* src2) − src3/mem
```

and three three-operand forms:

```
VFNMSUB132SD scr1, src2, src3/mem    // src1 = −(src1* src3/mem) − src2
VFNMSUB213SD scr1, src2, src3/mem    // src1 = −(src2* src1) − src3/mem
VFNMSUB231SD scr1, src2, src3/mem    // src1 = −(src2* src3/mem) − src1
```

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 1. The first and second operands are registers and the third operand is either a register or a 64-bit memory location.

The destination is an XMM register. Bits [127:64] of the destination XMM register and bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFNMSUBSD	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMSUB*nnn*SD	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMSUBSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	7F /r /is4
VFNMSUBSD *xmm1*, *xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	7F /r /is4
VFNMSUB132SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	9F /r
VFNMSUB213SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	AF /r
VFNMSUB231SD *xmm1, xmm2, xmm3/mem64*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.X.01	BF /r

Related Instructions

VFNMSUBPD, VFNMSUB132PD, VFNMSUB213PD, VFNMSUB231PD, VFNMSUBPS, VFNMSUB132PS, VFNMSUB213PS, VFNMSUB231PS, VFNMSUBSS, VFNMSUB132SS, VFNMSUB213SS, VFNMSUB231SS

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, FMA4 exception				

VFNMSUBSS VFNMSUB132SS VFNMSUB213SS VFNMSUB231SS — Negative Multiply and Subtract Scalar Single-Precision Floating-Point

Multiplies together two single-precision floating-point values, negates the unrounded product, and subtracts a third single-precision floating-point value from it. The precise result is then rounded to single-precision based on the mode specified by the MXCSR[RC] field and written to the destination register. The role of each of the source operands specified by the assembly language prototypes given below is reflected in the equation in the comment on the right.

There are two four-operand forms:

```
VFNMSUBSS dest, src1, src2/mem, src3    // dest = −(src1* src2/mem) − src3
VFNMSUBSS dest, src1, src2, src3/mem    // dest = −(src1* src2) − src3/mem
```

and three three-operand forms:

```
VFNMSUB132SS scr1, src2, src3/mem    // src1 = −(src1* src3/mem) − src2
VFNMSUB213SS scr1, src2, src3/mem    // src1 = −(src2* src1) − src3/mem
VFNMSUB231SS scr1, src2, src3/mem    // src1 = −(src2* src3/mem) − src1
```

For the four-operand forms, 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.

For the three-operand forms, VEX.W is 0. The first and second operands are registers and the third operand is either a register or a 32-bit memory location.

The destination is an XMM register. Bits[127:32] of the destination XMM register and bits [255:128] of the corresponding YMM register are cleared.

Instruction Support

Form	Subset	Feature Flag
VFNMSUBSS	FMA4	CPUID Fn8000_0001_ECX[FMA4] (bit 16)
VFNMSUB*nnn*SS	FMA	CPUID Fn0000_0001_ECX[FMA] (bit 12)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VFNMSUBSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.X.01	7E /r /is4
VFNMSUBSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.X.01	7E /r /is4
VFNMSUB132SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	9F /r
VFNMSUB213SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	AF /r
VFNMSUB231SS *xmm1, xmm2, xmm3/mem32*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.X.01	BF /r

Related Instructions

VFNMSUBPD, VFNMSUB132PD, VFNMSUB213PD, VFNMSUB231PD, VFNMSUBPS, VFNMSUB132PS, VFNMSUB213PS, VFNMSUB231PS, VFNMSUBSD, VFNMSUB132SD, VFNMSUB213SD, VFNMSUB231SD

rFLAGS Affected

None

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		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, 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 – ∞) 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.

Instruction Support

Form	Subset	Feature Flag
VFRCZPD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VFRCZPD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	81 /r
VFRCZPD *ymm1*, *ymm2*/*mem256*	8F	$\overline{\text{RXB}}$.09	0.1111.1.00	81 /r

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS, VFRCZPS, VFRCZSS, VFRCZSD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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			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 – ∞) 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.

Instruction Support

Form	Subset	Feature Flag
VFRCZPS	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VFRCZPS *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	80 /r
VFRCZPS *ymm1*, *ymm2*/*mem256*	8F	$\overline{\text{RXB}}$.09	0.1111.1.00	80 /r

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS, VFRCZPD, VFRCZSS, VFRCZSD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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			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 – ∞) 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.

Instruction Support

Form	Subset	Feature Flag
VFRCZSD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VFRCZSD *xmm1*, *xmm2*/*mem64*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	83 /r

Related Instructions

(V)ROUNDPD, (V)ROUNDPS, (V)ROUNDSD, (V)ROUNDSS, VFRCZPS, VFRCZPD, VFRCZSS

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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			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 double-word of an XMM register or 32-bit memory location and writes the result in the low-order double-word 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 – ∞) 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.

Instruction Support

Form	Subset	Feature Flag
VFRCZSS	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VFRCZSS *xmm1*, *xmm2*/*mem32*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	82 /r

Related Instructions

ROUNDPD, ROUNDPS, ROUNDSD, ROUNDSS, VFRCZPS, VFRCZPD, VFRCZSD

rFLAGS Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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			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				

VGATHERDPD Conditionally Gather Double-Precision Floating-Point Values, Doubleword Indices

Conditionally loads double-precision (64-bit) values from memory using VSIB addressing with doubleword indices.

The instruction is of the form:

VGATHERDPD *dest, mem64[vm32x], mask*

Loading of each element of the destination register is conditional based on the value of the corresponding element of the mask operand. If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 32-bit values. Quadword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to two 64-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the two low-order doublewords of an XMM register; the two high-order doublewords of the index register are not used. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is a YMM register. The first source operand is up to four 64-bit values located in memory. The second source operand (the mask) is a YMM register. The index vector is the four doublewords of an XMM register.

Instruction Support

Form	Subset	Feature Flag
VGATHERDPD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VGATHERDPD *xmm1, vm32x, xmm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	92 /r
VGATHERDPD *ymm1, vm32x, ymm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	92 /r

Related Instructions

VGATHERDPS, VGATHERQPD, VGATHERQPS, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VGATHERDPS Conditionally Gather Single-Precision Floating-Point Values, Doubleword Indices

Conditionally loads single-precision (32-bit) values from memory using VSIB addressing with doubleword indices.

The instruction is of the form:

VGATHERDPS *dest, mem32[vm32x/y], mask*

Loading of each element of the destination register is conditional based on the value of the corresponding element of the mask operand. If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 32-bit values. Doubleword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to four 32-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the four doublewords of an XMM register. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is a YMM register. The first source operand is up to eight 32-bit values located in memory. The second source operand (the mask) is a YMM register. The index vector is the eight doublewords of a YMM register.

Instruction Support

Form	Subset	Feature Flag
VGATHERDPS	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VGATHERDPS *xmm1, vm32x, xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	92 /r
VGATHERDPS *ymm1, vm32y, ymm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	92 /r

Related Instructions

VGATHERDPD, VGATHERQPD, VGATHERQPS, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VGATHERQPD Conditionally Gather Double-Precision Floating-Point Values, Quadword Indices

Conditionally loads double-precision (64-bit) values from memory using VSIB addressing with quadword indices.

The instruction is of the form:

VGATHERQPD *dest, mem64[vm64x/y], mask*

Loading of each element of the destination register is conditional based on the value of the corresponding element of the mask operand. If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 64-bit values. Quadword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to two 64-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the two quadwords of an XMM register. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is a YMM register. The first source operand is up to four 64-bit values located in memory. The second source operand (the mask) is a YMM register. The index vector is the four quadwords of a YMM register.

Instruction Support

Form	Subset	Feature Flag
VGATHERQPD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VGATHERQPD *xmm1, vm64x, xmm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	93 /r
VGATHERQPD *ymm1, vm64y, ymm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	93 /r

Related Instructions

VGATHERDPD, VGATHERDPS, VGATHERQPS, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VGATHERQPS Conditionally Gather Single-Precision Floating-Point Values, Quadword Indices

Conditionally loads single-precision (32-bit) values from memory using VSIB addressing with quadword indices.

The instruction is of the form:

VGATHERQPS *dest, mem32[vm64x/y], mask*

Loading of each element of the destination register is conditional based on the value of the corresponding element of the mask operand. If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 64-bit values. Doubleword elements of the destination for which the corresponding mask element is zero are not affected by the operation. The upper half of the destination is zeroed. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to two 32-bit values located in memory. The second source operand (the mask) is an XMM register. Only the lower half of the mask is used. The index vector is the two quadwords of an XMM register. Bits [255:64] of the YMM register that corresponds to the destination and bits [255:64] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is an XMM register. The first source operand is up to four 32-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the four quadwords of a YMM register. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

Instruction Support

Form	Subset	Feature Flag
VGATHERQPS	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VGATHERQPS *xmm1, vm64x, xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	93 /r
VGATHERQPS *xmm1, vm64y, xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	93 /r

Related Instructions

VGATHERDPD, VGATHERDPS, VGATHERQPD, VPGATHERDD, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VINSERTF128 Insert Packed Floating-Point 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 as 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 a single 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.

Instruction Support

Form	Subset	Feature Flag
VINSERTF128	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VINSERTF128 *ymm1, ymm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src}$.1.01	18 /r ib

Related Instructions

VBROADCASTF128, VBROADCASTI128, VEXTRACTF128, VEXTRACTI128, VINSERTI128

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
A — AVX exception.				

VINSERTI128 Insert Packed Integer 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.

Bit [0] of the immediate operand controls how the 128-bit values from the source operands are merged into the destination. The operation is as 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 instruction has a single 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. The immediate byte is encoded in the instruction.

Instruction Support

Form	Subset	Feature Flag
VINSERTI128	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VINSERTI128 *ymm1, ymm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	38 /r ib

Related Instructions

VBROADCASTF128, VBROADCASTI128, VEXTRACTF128, VEXTRACTI128, VINSERTF128

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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 most-significant bit 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.

Exception and trap behavior for elements not selected for loading or storing from/to memory is implementation dependent. For instance, a given implementation may signal a data breakpoint or a page fault for quadwords that are zero-masked and not actually written.

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 an 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 a YMM register, and the destination is a 128-bit memory location.

Instruction Support

Form	Subset	Feature Flag
VMASKMOVPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
Loads:				
VMASKMOVPD *xmm1*, *xmm2*, *mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	2D /r
VMASKMOVPD *ymm1*, *ymm2*, *mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	2D /r
Stores:				
VMASKMOVPD *mem128, xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	2F /r
VMASKMOVPD *mem256, ymm1*, *ymm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	2F /r

Related Instructions

VMASKMOVPS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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.
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 most-significant bits 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.

Exception and trap behavior for elements not selected for loading or storing from/to memory is implementation dependent. For instance, a given implementation may signal a data breakpoint or a page fault for doublewords that are zero-masked and not actually written.

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 an 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 a YMM register, and the destination is a 128-bit memory location.

Instruction Support

Form	Subset	Feature Flag
VMASKMOVPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
Loads:				
VMASKMOVPS *xmm1*, *xmm2*, *mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	2C /r
VMASKMOVPS *ymm1*, *ymm2*, *mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	2C /r
Stores:				
VMASKMOVPS *mem128, xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	2E /r
VMASKMOVPS *mem256, ymm1*, *ymm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	2E /r

Related Instructions

VMASKMOVPS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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.
A — AVX exception.				

VPBLENDD Blend Packed Doublewords

Copies packed doublewords from either of two sources to a destination, as specified by an immediate 8-bit mask operand.

Each bit of the mask selects a doubleword from one of the source operands to be copied to the destination. The least-significant bit controls the selection of the doubleword to be copied to the lowest doubleword of the destination. For each doubleword *i* of the destination:

- When mask bit [*i*] = 0, doubleword *i* of the first source operand is copied to the corresponding doubleword of the destination.
- When mask bit [*i*] = 1, doubleword *i* of the second source operand is copied to the corresponding doubleword of the destination.

VPBLENDD

The instruction has 128-bit and 256-bit encodings.

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

Instruction Support

Form	Subset	Feature Flag
VPBLENDD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPBLENDD *xmm1, xmm2, xmm3/mem128, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	02 /r /ib
VPBLENDD *ymm1, ymm2, ymm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	02 /r /ib

Related Instructions

VBLENDW

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VPBROADCASTB Broadcast Packed Byte

Loads a byte from a register or memory and writes it to all 16 or 32 bytes of an XMM or YMM register.

This instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Copies the source operand to all 16 bytes of the destination.

The source operand is the least-significant 8 bits of an XMM register or an 8-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

Copies the source operand to all 32 bytes of the destination.

The source operand is the least-significant 8 bits of an XMM register or an 8-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPBROADCASTB	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPBROADCASTB *xmm1*, *xmm2/mem8*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	78 /r
VPBROADCASTB *ymm1*, *xmm2/mem8*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	78 /r

Related Instructions

VPBROADCASTD, VPBROADCASTQ, VPBROADCASTW

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VPBROADCASTD Broadcast Packed Doubleword

Loads a doubleword from a register or memory and writes it to all 4 or 8 doublewords of an XMM or YMM register.

This instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Copies the source operand to all 4 doublewords of the destination.

The source operand is the least-significant 32 bits of 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.

YMM Encoding

Copies the source operand to all 8 doublewords of the destination.

The source operand is the least-significant 32 bits of an XMM register or a 32-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPBROADCASTD	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPBROADCASTD *xmm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	58 /r
VPBROADCASTD *ymm1*, *xmm2/mem32*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	58 /r

Related Instructions

VPBROADCASTB, VPBROADCASTQ, VPBROADCASTW

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VPBROADCASTQ Broadcast Packed Quadword

Loads a quadword from a register or memory and writes it to all 2 or 4 quadwords of an XMM or YMM register.

This instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Copies the source operand to both quadwords of the destination.

The source operand is the least-significant 64 bits of 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.

YMM Encoding

Copies the source operand to all 4 quadwords of the destination.

The source operand is the least-significant 64 bits of an XMM register or a 64-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPBROADCASTQ	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPBROADCASTQ *xmm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	59 /r
VPBROADCASTQ *ymm1*, *xmm2/mem64*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	59 /r

Related Instructions

VPBROADCASTB, VPBROADCASTD, VPBROADCASTW

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VPBROADCASTW Broadcast Packed Word

Loads a word from a register or memory and writes it to all 8 or 16 words of an XMM or YMM register.

This instruction has both 128-bit and 256-bit encodings:

XMM Encoding

Copies the source operand to all 8 words of the destination.

The source operand is the least-significant 16 bits of 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 cleared.

YMM Encoding

Copies the source operand to all 16 words of the destination.

The source operand is the least-significant 16 bits of an XMM register or a 16-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPBROADCASTW	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPBROADCASTW *xmm1*, *xmm2/mem16*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	79 /r
VPBROADCASTW *ymm1*, *xmm2/mem16*	C4	$\overline{\text{RXB}}$.02	0.1111.1.01	79 /r

Related Instructions

VPBROADCASTB, VPBROADCASTD, VPBROADCASTQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
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.r/m 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.r/m.

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.

Instruction Support

Form	Subset	Feature Flag
VPCMOV	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCMOV *xmm1*, *xmm2, xmm3*/*mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	A2 /r ib
VPCMOV *ymm1*, *ymm2, ymm3*/*mem256, ymm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.1.00	A2 /r ib
VPCMOV *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	8F	$\overline{\text{RXB}}$.08	1.$\overline{src1}$.0.00	A2 /r ib
VPCMOV *ymm1*, *ymm2, ymm3*, *ymm4*/*mem256*	8F	$\overline{\text{RXB}}$.08	1.$\overline{src1}$.1.00	A2 /r ib

Related Instructions

VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMUW, VCMPPD, VCMPPS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMB	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMB *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	CC /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMD xmm1, xmm2, xmm3/mem128, imm8	8F	$\overline{\text{RXB}}$.08	0.$\overline{\textit{src1}}$.0.00	CE /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMQ *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	CF /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMUB	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMUB *xmm1*, *xmm2, xmm3*/*mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	EC /r ib

Related Instructions

VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMUD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMUD *xmm1*, *xmm2, xmm3*/*mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	EE /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMUQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMUQ *xmm1*, *xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	EF /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMUW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMUW *xmm1*, *xmm2, xmm3*/*mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	ED /r ib

Related Instructions

VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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

Instruction Support

Form	Subset	Feature Flag
VPCOMW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPCOMW *xmm1*, *xmm2, xmm3*/*mem128, imm8*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\textit{src1}}$.0.00	CD /r ib

Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
X — XOP exception				

VPERM2F128 Permute Floating-Point 128-bit

Copies 128 bits of floating-point data from a selected octword of two 256-bit source operands or zero to each octword 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.

Instruction Support

Form	Subset	Feature Flag
VPERM2F128	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPERM2F128 *ymm1*, *ymm2*, *ymm3*/*mem256, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	06 /r ib

Related Instructions

VEXTRACTF128, VINSERTF128, VPERMILPD, VPERMILPS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
A — AVX exception.				

VPERM2I128 Permute Integer 128-bit

Copies 128 bits of integer data from a selected octword of two 256-bit source operands or zero to each octword 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; *imm8* [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; *imm8* [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. Bits 2 and 6 of the immediate byte are ignored.

Instruction Support

Form	Subset	Feature Flag
VPERM2I128	AVX2	CPUID Fn0000_0007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPERM2I128 *ymm1*, *ymm2*, *ymm3/mem256, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\textit{src1}}$.1.01	46 /r ib

Related Instructions

VEXTRACTI128, VEXTRACTF128, VINSERTI128, VINSERTF128, VPERMILPD, VPERMILPS

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
A — AVX exception.				

VPERMD Packed Permute Doubleword

Copies selected doublewords from a 256-bit value located either in memory or a YMM register to specific doublewords of the destination YMM register. For each doubleword of the destination, selection of which doubleword to copy from the source is specified by a selector field in the corresponding doubleword of a YMM register.

There is a single form of this instruction:

VPERMD *dest, src1, src2*

The first source operand provides eight 3-bit selectors, each selector occupying the least-significant bits of a doubleword. Each selector specifies the index of the doubleword of the second source operand to be copied to the destination. The doubleword in the destination that each selector controls is based on its position within the first source operand.

The index value may be the same in multiple selectors. This results in multiple copies of the same source doubleword being copied to the destination.

There is no 128-bit form of this instruction.

YMM Encoding

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

Instruction Support

Form	Subset	Feature Flag
VPERMD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

Instruction Encoding

	Encoding			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VPERMD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	36 /r

Related Instructions

VPERMQ, VPERMPD, VPERMPS

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	A	A	A	CR0.EM = 1.
	A	A	A	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L= 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	A	A	A	Lock prefix (F0h) preceding opcode.
Device not available, #NM	A	A	A	CR0.TS = 1.
Stack, #SS	A	A	A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	A	A	A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Alignment check, #AC			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPERMIL2PD Permute Two-Source Double-Precision Floating-Point

Copies a selected quadword from one of two source operands to a selected quadword of the destination or clears the selected quadword of the destination. Values in a third source operand and an immediate two-bit operand control the operation.

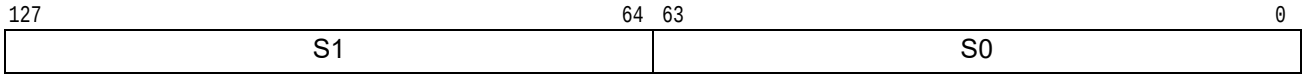
There are 128-bit and 256-bit versions of this instruction. Both versions have five operands:

VPERMIL2PD *dest, src1, src2, src3, m2z.*

The first four operands are either 128 bits or 256 bits wide, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

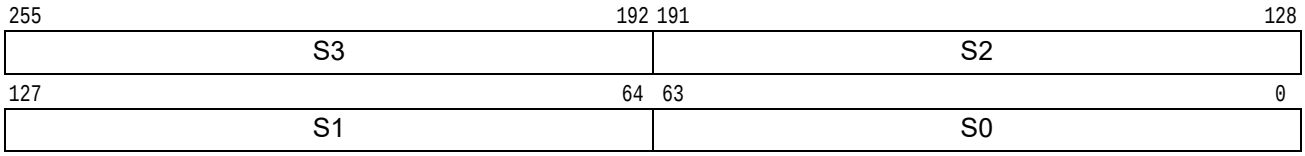
The third source operand is a selector that specifies how quadwords are copied or cleared in the destination. The selector contains one selector element for each quadword of the destination register.

Selector for 128-bit Instruction Form



The selector for the 128-bit instruction form is an octword composed of two quadword selector elements S0 and S1. S0 (the lower quadword) controls the value written to destination quadword 0 (bits [63:0]) and S1 (the upper quadword) controls the destination quadword 1 (bits [127:64]).

Selector for 256-bit Instruction Form



The selector for the 256-bit instruction form is a double octword and adds two more selector elements S2 and S3. S0 controls the value written to the destination quadword 0 (bits [63:0]), S1 controls the destination quadword 1 (bits [127:64]), S2 controls the destination quadword 2 (bits [191:128]), and S3 controls the destination quadword 3 (bits [255:192]).

The layout of each selector element is as follows:

Bits	Mnemonic	Description
[63:4]	—	Reserved, IGN
[3]	M	Match
[2:1]	Sel	Select
[0]	—	Reserved, IGN

The fields are defined as follows:

- Sel — Select. Selects the source quadword to copy into the corresponding quadword of the destination:

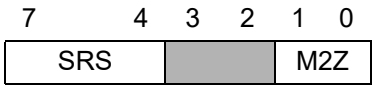
Sel Value	Source Selected for Destination Quadwords 0 and 1 (both forms)	Source Selected for Destination Quadwords 2 and 3 (256-bit form)
00b	*src1*[63:0]	*src1*[191:128]
01b	*src1*[127:64]	*src1*[255:192]
10b	*src2*[63:0]	*src2*[191:128]
11b	*src2*[127:64]	*src2*[255:192]

- M — Match bit. The combination of the Match bit in each selector element and the value of the M2Z field determines if the Select field is overridden. This is described below.

m2z immediate operand

The fifth operand is *m2z*. The assembler uses this 2-bit value to encode the M2Z field in the instruction. M2Z occupies bits [1:0] of an immediate byte. Bits [7:4] of the same byte are used to select one of 16 YMM/XMM registers. This dual use of the immediate byte is indicated in the instruction synopsis by the symbol “is5”.

The immediate byte is defined as follows.



Bits	Mnemonic	Description
[7:4]	SRS	Source Register Select
[3:2]	—	Reserved, IGN
[1:0]	M2Z	Match to Zero

Fields are defined as follows:

- SRS — Source Register Select. As with many other extended instructions, bits in the immediate byte are used to select a source operand register. This field is set by the assembler based on the operands listed in the instruction. See discussion in “*src2* and *src3* Operand Addressing” below.
- M2Z — Match to Zero. This field, combined with the M bit of the selector element, controls the function of the Sel field as follows:

M2Z Field	Selector M Bit	Value Loaded into Destination Quadword
0Xb	X	Source quadword selected by selector element Sel field.
10b	0	Source quadword selected by selector element Sel field.
10b	1	Zero
11b	0	Zero
11b	1	Source quadword selected by selector element Sel field.

src2 and *src3* Operand Addressing

In 64-bit mode, VEX.W and bits [7:4] of the immediate byte specify *src2* and *src3*:

- 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, s*rc2* 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.

In non-64-bit mode, bit 7 is ignored.

Instruction Support

Form	Subset	Feature Flag
VPERMIL2PD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

		Encoding		
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VPERMIL2PD *xmm1*, *xmm2*, *xmm3/mem128*, *xmm4, m2z*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	49 /r is5
VPERMIL2PD *xmm1*, *xmm2*, *xmm3*, *xmm4/mem128, m2z*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	49 /r is5
VPERMIL2PD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4, m2z*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	49 /r is5
VPERMIL2PD *ymm1*, *ymm2*, *ymm3*, *ymm4/mem256, m2z*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	49 /r is5

NOTE: VPERMIL2PD is encoded using the VEX prefix even though it is an XOP instruction.

Related Instructions

VPERM2F128, VPERMIL2PS, VPERMILPD, VPERMILPS, VPPERM

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
X — XOP exception				

VPERMIL2PS Permute Two-Source Single-Precision Floating-Point

Copies a selected doubleword from one of two source operands to a selected doubleword of the destination or clears the selected doubleword of the destination. Values in a third source operand and an immediate two-bit operand control operation.

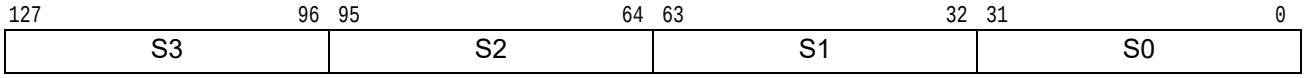
There are 128-bit and 256-bit versions of this instruction. Both versions have five operands:

VPERMIL2PS *dest, src1, src2, src3, m2z*

The first four operands are either 128 bits or 256 bits wide, as determined by VEX.L. When the destination is an XMM register, bits [255:128] of the corresponding YMM register are cleared.

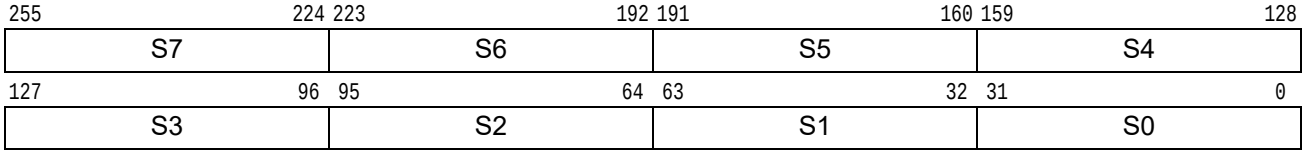
The third source operand is a selector that specifies how doublewords are copied or cleared in the destination. The selector contains one selector element for each doubleword of the destination register.

Selector for 128-bit Instruction Form



The selector for the 128-bit instruction form is an octword containing four selector elements S0–S3. S0 controls the value written to the destination doubleword 0 (bits [31:0]), S1 controls the destination doubleword 1 (bits [63:32]), S2 controls the destination doubleword 2 (bits [95:64]), and S3 controls the destination doubleword 3 (bits [127:96]).

Selector for 256-bit Instruction Form



The selector for the 256-bit instruction form is a double octword and adds four more selector elements S4–S7. S4 controls the value written to the destination doubleword 4 (bits [159:128]), S5 controls the destination doubleword 5 (bits [191:160]), S6 controls the destination doubleword 6 (bits [223:192]), and S7 controls the destination doubleword 7 (bits [255:224]).

The layout of each selector element is as follows.

Bits	Mnemonic	Description
[31:4]	—	Reserved, IGN
[3]	M	Match
[2:0]	Sel	Select

The fields are defined as follows:

- Sel — Select. Selects the source doubleword to copy into the corresponding doubleword of the destination:

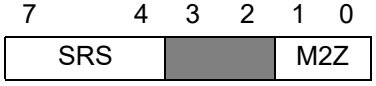
Sel Value	Source Selected for Destination Doublewords 0, 1, 2 and 3 (both forms)	Source Selected for Destination Doublewords 4, 5, 6 and 7 (256-bit form)
000b	*src1*[31:0]	*src1*[159:128]
001b	*src1*[63:32]	*src1*[191:160]
010b	*src1*[95:64]	*src1*[223:192]
011b	*src1*[127:96]	*src1*[255:224]
100b	*src2*[31:0]	*src2*[159:128]
101b	*src2*[63:32]	*src2*[191:160]
110b	*src2*[95:64]	*src2*[223:192]
111b	*src2*[127:96]	*src2*[255:224]

- M — Match. The combination of the M bit in each selector element and the value of the M2Z field determines if the Sel field is overridden. This is described below.

m2z immediate operand

The fifth operand is *m2z*. The assembler uses this 2-bit value to encode the M2Z field in the instruction. M2Z occupies bits [1:0] of an immediate byte. Bits [7:4] of the same byte are used to select one of 16 YMM/XMM registers. This dual use of the immediate byte is indicated in the instruction synopsis by the symbol “is5”.

The immediate byte is defined as follows.



Bits	Mnemonic	Description
[7:4]	SRS	Source Register Select
[3:2]	—	Reserved, IGN
[1:0]	M2Z	Match to Zero

Fields are defined as follows:

- SRS — Source Register Select. As with many other extended instructions, bits in the immediate byte are used to select a source operand register. This field is set by the assembler based on the operands listed in the instruction. See discussion in “*src2* and *src3* Operand Addressing” below.
- M2Z — Match to Zero. This field, combined with the M bit of the selector element, controls the function of the Sel field as follows:

M2Z Field	Selector M Bit	Value Loaded into Destination Doubleword
0Xb	X	Source doubleword selected by Sel field.
10b	0	Source doubleword selected by Sel field.

M2Z Field	Selector M Bit	Value Loaded into Destination Doubleword
10b	1	Zero
11b	0	Zero
11b	1	Source doubleword selected by Sel field.

src2 and *src3* Operand Addressing

In 64-bit mode, VEX.W and bits [7:4] of the immediate byte specify *src2* and *src3*:

- 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, s*rc2* 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.

In non-64-bit mode, bit 7 is ignored.

Instruction Support

Form	Subset	Feature Flag
VPERMIL2PS	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

		Encoding		
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VPERMIL2PS *xmm1*, *xmm2, xmm3*/*mem128, xmm4, m2z*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.0.01	48 /r is5
VPERMIL2PS *xmm1*, *xmm2, xmm3, xmm4*/*mem128, m2z*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.0.01	48 /r is5
VPERMIL2PS *ymm1*, *ymm2, ymm3*/*mem256, ymm4, m2z*	C4	$\overline{\text{RXB}}$.03	0.$\overline{src1}$.1.01	48 /r is5
VPERMIL2PS *ymm1*, *ymm2, ymm3, ymm4*/*mem256, m2z*	C4	$\overline{\text{RXB}}$.03	1.$\overline{src1}$.1.01	48 /r is5

NOTE: VPERMIL2PS is encoded using the VEX prefix even though it is an XOP instruction.

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMILPD, VPERMILPS, VPPERM

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

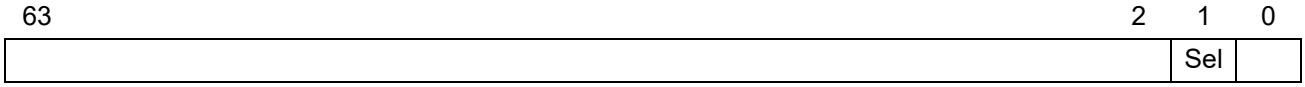
Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			X	Instruction not supported, as indicated by CPUID feature identifier.
	X	X		XOP instructions are only recognized in protected mode.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XFEATURE_ENABLED_MASK[2:1] ! = 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	Memory operand not 16-byte aligned when alignment checking enabled.
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.



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.

Instruction Support

Form	Subset	Feature Flag
VPERMILPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
Selection by source register or memory:				
VPERMILPD *xmm1, xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{\textit{src1}}$.0.01	0D /r
VPERMILPD *ymm1, ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{\textit{src1}}$.1.01	0D /r
Selection by immediate byte operand:				
VPERMILPD *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.0.01	05 /r ib
VPERMILPD *ymm1*, *ymm2*/mem256, *imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.1.01	05 /r ib

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPS, VPPERM

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

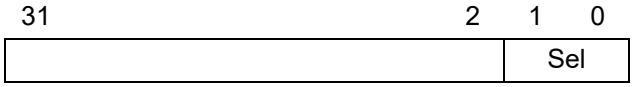
Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 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	Unaligned memory reference when alignment checking enabled.
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 encodings:

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.

Instruction Support

Form	Subset	Feature Flag
VPERMILPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
Selection by source register or memory:				
VPERMILPS *xmm1, xmm2*, *xmm3*/*mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	0C /r
VPERMILPS *ymm1, ymm2*, *ymm3*/*mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	0C /r
Selection by immediate byte operand:				
VPERMILPS *xmm1*, *xmm2/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.0.01	04 /r ib
VPERMILPS *ymm1*, *ymm2*/mem256, *imm8*	C4	$\overline{\text{RXB}}$.03	0.1111.1.01	04 /r ib

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPD, VPPERM

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

VPERMPD Packed Permute Double-Precision Floating-Point

Copies selected quadwords from a 256-bit value located either in memory or a YMM register to specific quadwords of the destination. For each quadword of the destination, selection of which quadword to copy from the source is specified by a 2 bit selector field in an immediate byte.

There is a single form of this instruction:

VPERMPD *dest, src, imm8*

The selection of which quadword of the source operand to copy to each quadword of the destination is specified by four 2-bit selector fields in the immediate byte. Bits [1:0] specify the index of the quadword to be copied to the destination quadword 0. Bits [3:2] select the quadword to be copied to quadword 1, bits [5:4] select the quadword to be copied to quadword 2, and bits [7:6] select the quadword to be copied to quadword 3.

The index value may be the same in multiple selectors. This results in multiple copies of the same source quadword being copied to the destination.

There is no 128-bit form of this instruction.

YMM Encoding

The destination is a YMM register. The source operand is a YMM register or a 256-bit memory location.

Instruction Support

Form	Subset	Feature Flag
VPERMPD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

	Encoding			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VPERMPD *ymm1*, *ymm2/mem256, imm8*	C4	$\overline{\text{RXB}}$.03	1.1111.1.01	01 /r ib

Related Instructions

VPERMD, VPERMQ, VPERMPS

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	A	A	A	CR0.EM = 1.
	A	A	A	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L= 0.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	A	A	A	Lock prefix (F0h) preceding opcode.
Device not available, #NM	A	A	A	CR0.TS = 1.
Stack, #SS	A	A	A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	A	A	A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Alignment check, #AC			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPERMPS Packed Permute Single-Precision Floating-Point

Copies selected doublewords from a 256-bit value located either in memory or a YMM register to specific doublewords of the destination YMM register. For each doubleword of the destination, selection of which doubleword to copy from the source is specified by a selector field in the corresponding doubleword of a YMM register.

There is a single form of this instruction:

VPERMPS *dest, src1, src2*

The first source operand provides eight 3-bit selectors, each selector occupying the least-significant bits of a doubleword. Each selector specifies the index of the doubleword of the second source operand to be copied to the destination. The doubleword in the destination that each selector controls is based on its position within the first source operand.

The index value may be the same in multiple selectors. This results in multiple copies of the same source doubleword being copied to the destination.

There is no 128-bit form of this instruction.

YMM Encoding

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

Instruction Support

Form	Subset	Feature Flag
VPERMPS	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

	Encoding			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VPERMPS *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	16 /r

Related Instructions

VPERMD, VPERMQ, VPERMPD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	A	A	A	CR0.EM = 1.
	A	A	A	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L= 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	A	A	A	Lock prefix (F0h) preceding opcode.
Device not available, #NM	A	A	A	CR0.TS = 1.
Stack, #SS	A	A	A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	A	A	A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Alignment check, #AC			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPERMQ Packed Permute Quadword

Copies selected quadwords from a 256-bit value located either in memory or a YMM register to specific quadwords of the destination. For each quadword of the destination, selection of which quadword to copy from the source is specified by a 2 bit selector field in an immediate byte.

There is a single form of this instruction:

VPERMQ *dest, src, imm8*

The selection of which quadword of the source operand to copy to each quadword of the destination is specified by four 2-bit selector fields in the immediate byte. Bits [1:0] specify the index of the quadword to be copied to the destination quadword 0. Bits [3:2] select the quadword to be copied to quadword 1, bits [5:4] select the quadword to be copied to quadword 2, and bits [7:6] select the quadword to be copied to quadword 3.

The index value may be the same in multiple selectors. This results in multiple copies of the same source quadword being copied to the destination.

There is no 128-bit form of this instruction.

YMM Encoding

The destination is a YMM register. The source operand is a YMM register or a 256-bit memory location.

Instruction Support

Form	Subset	Feature Flag
VPERMQ	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

	Encoding			
Mnemonic	**VEX**	**RXB.map_select**	**W.vvvv.L.pp**	**Opcode**
VPERMQ *ymm1*, *ymm2/mem256, imm8*	C4	$\overline{\text{RXB}}$.03	1.1111.1.01	00 /r ib

Related Instructions

VPERMD, VPERMPD, VPERMPS

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	A	A	A	CR0.EM = 1.
	A	A	A	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L= 0.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	A	A	A	Lock prefix (F0h) preceding opcode.
Device not available, #NM	A	A	A	CR0.TS = 1.
Stack, #SS	A	A	A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	A	A	A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Alignment check, #AC			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPGATHERDD Conditionally Gather Doublewords, Doubleword Indices

Conditionally loads doubleword values from memory using VSIB addressing with doubleword indices.

The instruction is of the form:

VPGATHERDD *dest, mem32[vm32x/y], mask*

The loading of each element of the destination register is conditional based on the value of the corresponding element of the mask (second source operand). If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 32-bit values. Doubleword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to four 32-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the four doublewords of an XMM register. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is a YMM register. The first source operand is up to eight 32-bit values located in memory. The second source operand (the mask) is a YMM register. The index vector is the eight doublewords of a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPGATHERDD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPGATHERDD *xmm1, vm32x, xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	90 /r
VPGATHERDD *ymm1, vm32y, ymm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	90 /r

Related Instructions

VGATHERDPD, VGATHERDPS, VGATHERQPD, VGATHERQPS, VPGATHERDQ, VPGATHERQD, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPGATHERDQ Conditionally Gather Quadwords, Doubleword Indices

Conditionally loads quadword values from memory using VSIB addressing with doubleword indices.

The instruction is of the form:

VPGATHERDQ *dest, mem64[vm32x], mask*

The loading of each element of the destination register is conditional based on the value of the corresponding element of the mask (second source operand). If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 32-bit values. Quadword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to two 64-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the two low-order doublewords of an XMM register; the two high-order doublewords of the index register are not used. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is a YMM register. The first source operand is up to four 64-bit values located in memory. The second source operand (the mask) is a YMM register. The index vector is the four doublewords of an XMM register.

Instruction Support

Form	Subset	Feature Flag
VPGATHERDQ	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPGATHERDQ *xmm1, vm32x, xmm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	90 /r
VPGATHERDQ *ymm1, vm32x, ymm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	90 /r

Related Instructions

VGATHERDPD, VGATHERDPS, VGATHERQPD, VGATHERQPS, VPGATHERDD, VPGATHERQD, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPGATHERQD Conditionally Gather Doublewords, Quadword Indices

Conditionally loads doubleword values from memory using VSIB addressing with quadword indices.

The instruction is of the form:

VPGATHERQD *dest, mem32[vm64x/y], mask*

The loading of each element of the destination register is conditional based on the value of the corresponding element of the mask (second source operand). If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 64-bit values. Doubleword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to two 32-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the two quadwords of an XMM register. The upper half of the destination register and the mask register are cleared. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the mask register are cleared.

YMM Encoding

The destination is an XMM register. The first source operand is up to four 32-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the four quadwords of a YMM register. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the mask register are cleared.

Instruction Support

Form	Subset	Feature Flag
VPGATHERQD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPGATHERQD *xmm1, vm64x, xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.0.01	91 /r
VPGATHERQD *xmm1, vm64y, xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src2}$.1.01	91 /r

Related Instructions

VGATHERDPD, VGATHERDPS, VGATHERQPD, VGATHERQPS, VPGATHERDD, VPGATHERDQ, VPGATHERQQ

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

VPGATHERQQ Conditionally Gather Quadwords, Quadword Indices

Conditionally loads quadword values from memory using VSIB addressing with quadword indices.

The instruction is of the form:

VPGATHERQQ *dest, mem64[vm64x/y], mask*

The loading of each element of the destination register is conditional based on the value of the corresponding element of the mask (second source operand). If the most-significant bit of the *i*th element of the mask is set, the *i*th element of the destination is loaded from memory using the *i*th address of the array of effective addresses calculated using VSIB addressing.

The index register is treated as an array of signed 64-bit values. Quadword elements of the destination for which the corresponding mask element is zero are not affected by the operation. If no exceptions occur, the mask register is set to zero.

Execution of the instruction can be suspended by an exception if the exception is triggered by an element other than the rightmost element loaded. When this happens, the destination register and the mask operand may be observed as partially updated. Elements that have been loaded will have their mask elements set to zero. If any traps or faults are pending from elements that have been loaded, they will be delivered in lieu of the exception; in this case, the RF flag is set so that an instruction breakpoint is not re-triggered when the instruction execution is resumed.

See Section 1.3, “VSIB Addressing,” on page 6 for a discussion of the VSIB addressing mode.

There are 128-bit and 256-bit forms of this instruction.

XMM Encoding

The destination is an XMM register. The first source operand is up to two 64-bit values located in memory. The second source operand (the mask) is an XMM register. The index vector is the two quadwords of an XMM register. Bits [255:128] of the YMM register that corresponds to the destination and bits [255:128] of the YMM register that corresponds to the second source (mask) operand are cleared.

YMM Encoding

The destination is a YMM register. The first source operand is up to four 64-bit values located in memory. The second source operand (the mask) is a YMM register. The index vector is the four quadwords of a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPGATHERQQ	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPGATHERQQ *xmm1, vm64x, xmm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.0.01	91 /r
VPGATHERQQ *ymm1, vm64y, ymm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src2}$.1.01	91 /r

Related Instructions

VGATHERDPD, VGATHERDPS, VGATHERQPD, VGATHERQPS, VPGATHERDD, VPGATHERDQ, VPGATHERQD

rFLAGS Affected

RF

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 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.

Instruction Support

Form	Subset	Feature Flag
VPHADDBD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDBD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C2 /r

Related Instructions

VPHADDBW, VPHADDBQ, VPHADDWD, VPHADDWQ, VPHADDDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDBQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDBQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C3 /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDWD, VPHADDWQ, VPHADDDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDBW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDBW *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C1 /r

Related Instructions

VPHADDBD, VPHADDBQ, VPHADDWD, VPHADDWQ, VPHADDDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDDQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDDQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	CB /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDBQ, VPHADDWD, VPHADDWQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDUBD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDUBD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	D2 /r

Related Instructions

VPHADDUBW, VPHADDUBQ, VPHADDUWD, VPHADDUWQ, VPHADDUDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDUBQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDUBQ *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	D3 /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUWD, VPHADDUWQ, VPHADDUDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDUBW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDUBW *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	D1 /r

Related Instructions

VPHADDUBD, VPHADDUBQ, VPHADDUWD, VPHADDUWQ, VPHADDUDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDUDQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDUDQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	DB /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWD, VPHADDUWQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDUWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDUWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	D6 /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWQ, VPHADDUDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDUWQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDUWQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	D7 /r

Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWD, VPHADDUDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C6 /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDBQ, VPHADDWQ, VPHADDDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHADDWQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHADDWQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C7 /r

Related Instructions

VPHADDBW, VPHADDBD, VPHADDBQ, VPHADDWD, VPHADDDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHSUBBW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHSUBBW *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	E1 /r

Related Instructions

VPHSUBWD, VPHSUBDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHSUBDQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHSUBDQ *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	E3 /r

Related Instructions

VPHSUBBW, VPHSUBWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.

Instruction Support

Form	Subset	Feature Flag
VPHSUBWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPHSUBWD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	E2 /r

Related Instructions

VPHSUBBW, VPHSUBDQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSDD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSDD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	9E /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSDQH	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSDQH *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.01000	0.$\overline{src1}$.0.00	9F /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSDQL	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSDQL *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	97 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSSDD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSSDD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	X.$\overline{src1}$.0.00	8E /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSSDQH	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSSDQH *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	8F /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSSDQL	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSSDQL *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	87 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSSWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	86 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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 eight 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.r/m field; and the third source (*src3*) is an XMM register specified by bits [7:4] of an immediate byte.

When *src3* and *dest* designate the same XMM register, this register behaves as an accumulator.

Instruction Support

Form	Subset	Feature Flag
VPMACSSWW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSSWW *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	X.$\overline{\textit{src1}}$.0.00	85 /r ib

Related Instructions

VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL,VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	96 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSSDD, VPMACSDO, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMACSWW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMACSWW *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	95 /r ib

Related Instructions

VPMACSSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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 32-bit signed-integer results 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: 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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPMADCSSWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPMADCSSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\textit{src1}}$.0.00	A6 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
PMADCSWD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
PMADCSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	B6 /r ib

Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
X — XOP exception				

VPMASKMOVD Masked Move Packed Doubleword

Moves packed doublewords from a second source operand to a destination, as specified by mask bits in a first source operand. There are load and store versions of the instruction.

The mask bits are the most-significant bit of each doubleword in the first source operand (mask).

- For loads, when a mask bit = 1, the corresponding doubleword 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 doubleword 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.

Exception and trap behavior for elements not selected for loading or storing from/to memory is implementation dependent. For instance, a given implementation may signal a data breakpoint or a page fault for doublewords that are zero-masked and not actually written.

This instruction provides no non-temporal access hint.

This instruction has both 128-bit and 256-bit forms:

XMM Encoding

There are load and store encodings.

- For loads, the four doublewords that make up the source operand are located 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, the four doublewords that make up the source operand are located in an XMM register, the mask operand is an XMM register, and the destination is a 128-bit memory location.

YMM Encoding

There are load and store encodings.

- For loads, the eight doublewords that make up the source operand are located in a 256-bit memory location, the mask operand is a YMM register, and the destination is a YMM register.
- For stores, the eight doublewords that make up the source operand are located in a YMM register, the mask operand is a YMM register, and the destination is a 256-bit memory location.

Instruction Support

Form	Subset	Feature Flag
VPMASKMOVD	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
Loads:				
VPMASKMOVD *xmm1*, *xmm2*, *mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{\textit{src1}}$.0.01	8C /r
VPMASKMOVD *ymm1*, *ymm2*, *mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{\textit{src1}}$.1.01	8C /r
Stores:				
VPMASKMOVD *mem128, xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{\textit{src1}}$.0.01	8E /r
VPMASKMOVD *mem256, ymm1*, *ymm2*	C4	$\overline{\text{RXB}}$.02	0.$\overline{\textit{src1}}$.1.01	8E /r

Related Instructions

VPMASKMOVQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VPMASKMOVQ Masked Move Packed Quadword

Moves packed quadwords from a second source operand to a destination, as specified by mask bits in a first source operand. There are load and store versions of the instruction.

The mask bits are the most-significant bit of each quadword in the mask first source operand (mask).

- For loads, when a mask bit = 1, the corresponding quadword 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 quadword 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.

Exception and trap behavior for elements not selected for loading or storing from/to memory is implementation dependent. For instance, a given implementation may signal a data breakpoint or a page fault for quadwords that are zero-masked and not actually written.

This instruction provides no non-temporal access hint.

This instruction has both 128-bit and 256-bit forms:

XMM Encoding

There are load and store encodings.

- For loads, the two quadwords that make up the source operand are located 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, the two quadwords that make up the source operand are located in an XMM register, the mask operand is an XMM register, and the destination is a 128-bit memory location.

YMM Encoding

There are load and store encodings.

- For loads, the four quadwords that make up the source operand are located in a 256-bit memory location, the mask operand is a YMM register, and the destination is a YMM register.
- For stores, the four quadwords that make up the source operand are located in a YMM register, the mask operand is a YMM register, and the destination is a 256-bit memory location.

Instruction Support

Form	Subset	Feature Flag
VPMASKMOVQ	AVX2	Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
Loads:				
VPMASKMOVQ *xmm1*, *xmm2*, *mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src1}}$.0.01	8C /r
VPMASKMOVQ *ymm1*, *ymm2*, *mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src1}}$.1.01	8C /r
Stores:				
VPMASKMOVQ *mem128, xmm1*, *xmm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src1}}$.0.01	8E /r
VPMASKMOVQ *mem256, ymm1*, *ymm2*	C4	$\overline{\text{RXB}}$.02	1.$\overline{\textit{src1}}$.1.01	8E /r

Related Instructions

VPMASKMOVD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 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.r/m 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.r/m.

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.

Instruction Support

Form	Subset	Feature Flag
VPPERM	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPPERM *xmm1*, *xmm2, xmm3*/*mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{src1}$.0.00	A3 /r ib
VPPERM *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	8F	$\overline{\text{RXB}}$.08	1.$\overline{src1}$.0.00	A3 /r ib

Related Instructions

VPSHUFHW, VPSHUFD, VPSHUFLW, VPSHUFW, VPERMIL2PS, VPERMIL2PD

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.r/m.
- When XOP.W = 1, *variable-count* is either an XMM register or a 128-bit memory location specified by ModRM.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPROTB	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPROTB *xmm1*, *xmm2/mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	90 /r
VPROTB *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	90 /r
VPROTB *xmm1*, *xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C0 /r ib

Related Instructions

VPROTW, VPROTD, VPROTQ,VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.r/m 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.r/m 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.

Instruction Support

Form	Subset	Feature Flag
VPROTD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPROTD *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	92 /r
VPROTD *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	92 /r
VPROTD *xmm1, xmm2*/*mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C2 /r ib

Related Instructions

VPROTB, VPROTW, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.r/m 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.r/m.

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.

Instruction Support

Form	Subset	Feature Flag
VPROTQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPROTQ *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	93 /r
VPROTQ *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	93 /r
VPROTQ *xmm1, xmm2*/*mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C3 /r ib

Related Instructions

VPROTB, VPROTW, VPROTD, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m 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.r/m 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.r/m.

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.

Instruction Support

Form	Subset	Feature Flag
VPROTW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPROTW *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	91 /r
VPROTW *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	91 /r
VPROTW *xmm1, xmm2*/*mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C1 /r ib

Related Instructions

VPROTB, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a 128-bit memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHAB	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHAB *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	98 /r
VPSHAB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	98 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHAD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHAD *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	9A /r
VPSHAD *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	9A /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHAQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHAQ *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	9B /r
VPSHAQ *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHAW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHAW *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	99 /r
VPSHAW *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	99 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHLB	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHLB *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	94 /r
VPSHLB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	94 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHLD	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHLD *xmm1*, *xmm3*/*mem128, xmm2*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	96 /r
VPSHLD *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	96 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHLQ	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHLQ *xmm1*, *xmm3/mem128, xmm2*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	97 /r
VPSHLQ *xmm1*, *xmm2, xmm3/mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	97 /r

Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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.r/m.
- When XOP.W = 1, *count* is either an XMM register or a memory location specified by ModRM.r/m and *src* is an XMM register specified by XOP.vvvv.

Bits [255:128] of the YMM register that corresponds to the destination are cleared.

Instruction Support

Form	Subset	Feature Flag
VPSHLW	XOP	CPUID Fn8000_0001_ECX[XOP] (bit 11)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	XOP	RXB.map_select	W.vvvv.L.pp	Opcode
VPSHLW *xmm1*, *xmm3*/*mem128, xmm2*	8F	$\overline{\text{RXB}}$.09	0.$\overline{count}$.0.00	95 /r
VPSHLW *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{src}$.0.00	95 /r

Related Instructions

VPROTB, VPROLW, VPROTD, VPROTQ, VPSHLB, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
X — XOP exception				

VPSLLVD Variable Shift Left Logical Doublewords

Left-shifts the bits of each doubleword in the first source operand by a count specified in the corresponding doubleword of a second source operand and writes the shifted values to the destination.

The second source operand is treated as an array of unsigned 32-bit integers. Each integer specifies the shift count of the corresponding doubleword of the first source operand. Each doubleword is shifted independently.

Low-order bits emptied by shifting are cleared. High-order bits shifted out of each doubleword are discarded. When the shift count for any doubleword is greater than 31, that doubleword is cleared in the destination.

This instruction has 128-bit and 256-bit encodings:

XMM Encoding

The first source operand is an XMM register. The shift count array is specified by either a second 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

The first source operand is a YMM register. The shift count array is specified by either a second YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPSLLVD	AVX2	CPUID Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPSLLVD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	47 /r
VPSLLVD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	47 /r

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVQ, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VPSLLVQ Variable Shift Left Logical Quadwords

Left-shifts the bits of each quadword in the first source operand by a count specified in the corresponding quadword of a second source operand and writes the shifted values to the destination.

The second source operand is treated as an array of unsigned 64-bit integers. Each integer specifies the shift count of the corresponding quadword of the first source operand. Each quadword is shifted independently.

Low-order bits emptied by shifting are cleared. High-order bits shifted out of each quadword are discarded. When the shift count for any quadword is greater than 63, that quadword is cleared in the destination.

This instruction has 128-bit and 256-bit encodings:

XMM Encoding

The first source operand is an XMM register. The shift count array is specified by either a second 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

The first source operand is a YMM register. The shift count array is specified by either a second YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPSLLVQ	AVX2	CPUID Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPSLLVQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src1}$.0.01	47 /r
VPSLLVQ *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src1}$.1.01	47 /r

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSRAVD, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VPSRAVD Variable Shift Right Arithmetic Doublewords

Performs a right arithmetic shift of each signed 32-bit integer in the first source operand by a count specified in the corresponding doubleword of a second source operand and writes the shifted values to the destination.

The second source operand is treated as an array of unsigned 32-bit integers. Each integer specifies the shift count of the corresponding doubleword of the first source operand. Each doubleword is shifted independently.

A copy of the sign bit is shifted into the most-significant bit of the element on each right-shift. Low-order bits shifted out of each element are discarded. If a doubleword contains a positive integer and the shift count is greater than 31, that doubleword is cleared in the destination. If a doubleword contains a negative integer and the shift count is greater than 31, that doubleword is set to -1 in the destination.

This instruction has 128-bit and 256-bit encodings:

XMM Encoding

The first source operand is an XMM register. The shift count array is specified by either a second 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

The first source operand is a YMM register. The shift count array is specified by either a second YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPSRAVD	AVX2	CPUID Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPSRAVD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	46 /r
VPSRAVD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	46 /r

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRLVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VPSRLVD Variable Shift Right Logical Doublewords

Right-shifts each doubleword in the first source operand by a count specified in the corresponding doubleword of a second source operand and writes the shifted values to the destination.

The second source operand is treated as an array of unsigned 32-bit integers. Each integer specifies the shift count of the corresponding doubleword of the first source operand. Each doubleword is shifted independently.

Zero is shifted into the most-significant bit of the element on each right-shift. Low-order bits shifted out of each element are discarded. If the shift count for any doubleword is greater than 31, that doubleword is cleared in the destination.

This instruction has 128-bit and 256-bit encodings:

XMM Encoding

The first source operand is an XMM register. The shift count array is specified by either a second 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

The first source operand is a YMM register. The shift count array is specified by either a second YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPSRLVD	AVX2	CPUID Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPSRLVD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.0.01	45 /r
VPSRLVD *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	0.$\overline{src1}$.1.01	45 /r

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VPSRLVQ Variable Shift Right Logical Quadwords

Right-shifts each quadword in the first source operand by a count specified in the corresponding quadword of a second source operand and writes the shifted values to the destination.

The second source operand is treated as an array of unsigned 64-bit integers. Each integer specifies the shift count of the corresponding quadword of the first source operand. Each quadword is shifted independently.

Zero is shifted into the most-significant bit of the element on each right-shift. Low-order bits shifted out of each element are discarded. If the shift count for any quadword is greater than 63, that quadword is cleared in the destination.

This instruction has 128-bit and 256-bit encodings:

XMM Encoding

The first source operand is an XMM register. The shift count array is specified by either a second 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

The first source operand is a YMM register. The shift count array is specified by either a second YMM register or a 256-bit memory location. The destination is a YMM register.

Instruction Support

Form	Subset	Feature Flag
VPSRLVQ	AVX2	CPUID Fn0000_00007_EBX[AVX2]_x0 (bit 5)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VPSRLVQ *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src1}$.0.01	45 /r
VPSRLVQ *ymm1*, *ymm2*, *ymm3/mem256*	C4	$\overline{\text{RXB}}$.02	1.$\overline{src1}$.1.01	45 /r

Related Instructions

(V)PSLLD, (V)PSLLDQ, (V)PSLLQ, (V)PSLLW, (V)PSRAD, (V)PSRAW, (V)PSRLD, (V)PSRLDQ, (V)PSRLQ, (V)PSRLW, VPSLLVD, VPSLLVQ, VPSRAVD, VPSRLVD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

VTESTPD Packed Bit Test

Performs two different logical operations on the sign bits of the first and second packed floating-point operands and updates the ZF and CF flags based on the results.

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 complements (NOT) of the sign bits of each double-precision floating-point element of the first source with 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.

Instruction Support

Form	Subset	Feature Flag
VTESTPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VTESTPD *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	0F /r
VTESTPD *ymm1*, *ymm2*/*mem256*	C4	$\overline{\text{RXB}}$.02	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.*

MXCSR Flags Affected

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.
	X	X		AVX instructions are only recognized in protected mode.
	X	X	X	CR0.EM = 1.
	X	X	X	CR4.OSFXSR = 0.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			X	VEX.W = 1.
			X	VEX.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding VEX prefix.
	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	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		X	X	Instruction execution caused a page fault.

X — AVX exception

VTESTPS Packed Bit Test

Performs two different logical operations on the sign bits of the first and second packed floating-point operands and updates the ZF and CF flags based on the results.

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 complements (NOT) of the sign bits of each single-precision floating-point element of the first source with 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.

Instruction Support

Form	Subset	Feature Flag
VTESTPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VTESTPS *xmm1*, *xmm2/mem128*	C4	$\overline{\text{RXB}}$.02	0.1111.0.01	0E /r
VTESTPS *ymm1*, *ymm2/mem256*	C4	$\overline{\text{RXB}}$.02	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.*

MXCSR Flags Affected

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.
	X	X		AVX instructions are only recognized in protected mode.
	X	X	X	CR0.EM = 1.
	X	X	X	CR4.OSFXSR = 0.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			X	VEX.W = 1.
			X	VEX.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding VEX prefix.
	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	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		X	X	Instruction execution caused a page fault.

X — AVX exception

VZEROALL — Zero All YMM Registers

Clears all YMM registers.

In 64-bit mode, YMM0–15 are all cleared (set to all zeros). In legacy and compatibility modes, only YMM0–7 are cleared. The contents of the MXCSR is unaffected.

Instruction Support

Form	Subset	Feature Flag
VZEROALL	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VZEROALL	C4	$\overline{\text{RXB}}$.01	X.1111.1.00	77

Related Instructions

VZEROUPPER

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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 Zero All YMM Registers Upper

Clears the upper octword of all YMM registers. The corresponding XMM registers (lower octword of each YMM register) are not affected.

In 64-bit mode, the instruction operates on registers YMM0–15. In legacy and compatibility mode, the instruction operates on YMM0–7. The contents of the MXCSR is unaffected.

Instruction Support

Form	Subset	Feature Flag
VZEROUPPER	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Encoding			
	VEX	RXB.map_select	W.vvvv.L.pp	Opcode
VZEROUPPER	C4	$\overline{\text{RXB}}$.01	X.1111.0.00	77

Related Instructions

VZEROUPPER

rFLAGS Affected

None

MXCSR Flags Affected

None

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	XFEATURE_ENABLED_MASK[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.				

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 the XSAVE instruction description 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. If CPUID reports support for ECX=1 (see table below), then the XGETBV instruction supports an ECX value of 1. When ECX=1, XGETBV returns the logical and of XCR0 and the current value of the XINUSE state-component bitmap.

Instruction Support

Form	Subset	Feature Flag
XGETBV	XSAVE/XRSTOR	CPUID Fn0000_0001_ECX[XSAVE] (bit 26)
XGETBV	ECX=1 support	CPUID Fn0000_000D_EAX_x1[2] = 1

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XGETBV	0F 01 D0	Copies content of the XCR specified by ECX into EDX:EAX.

Related Instructions

RDMSR, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

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.
	X	X	X	Lock prefix (F0h) preceding opcode.
	X	X	X	CR4.OSXSAVE = 0
General protection, #GP	X	X	X	ECX specifies a reserved or unimplemented XCR address.
X — exception generated				

XORPD XOR
VXORPD 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 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.

VXORPD

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
XORPD	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VXORPD	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VXORPD *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.01	57 /r
VXORPD *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.01	57 /r

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

XORPS XOR
VXORPS 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 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.

VXORPS

The extended form of the instruction has both 128-bit and 256-bit encodings:

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. 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.

Instruction Support

Form	Subset	Feature Flag
XORPS	SSE2	CPUID Fn0000_0001_EDX[SSE2] (bit 26)
VXORPS	AVX	CPUID Fn0000_0001_ECX[AVX] (bit 28)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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.map_select	W.vvvv.L.pp	Opcode
VXORPS *xmm1*, *xmm2*, *xmm3/mem128*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.0.00	57 /r
VXORPS *ymm1, ymm2, ymm3/mem256*	C4	$\overline{\text{RXB}}$.01	X.$\overline{src1}$.1.00	57 /r

Related Instructions

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

rFLAGS Affected

None

MXCSR Flags Affected

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	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

XRSTOR Restore Extended States

Restores a partial or full 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 the descriptions of XSAVE and XRSTOR instructions for basic operational details.

The XRSTOR instruction may operate on the buffer in standard form or a compact form. The compact form is indicated in the memory buffer with XCOMP_BV[63]=1.

In either form, the instruction creates a Requested Feature Bit Map (RBFM) which is the logical AND of EDX:EAX and XCR0. Then for each feature bit:

1. If RFBM = 0, XRSTOR does not update the component.
2. If RFBM = 1 but the corresponding XSTATE_BV bit is 0, the component is set to its reset state without reading anything out of the buffer.
3. IF RFBM =1 and XSTATE_BV =1, the component state is read from the buffer.
4. XRSTOR loads an internal state value XRSTOR_INFO that can be used to further optimize a subsequent XSAVEOPT or XSAVES. This reflects the current privilege level and virtualization mode as well as the save area's base address and XCOMP_BV field.
5. If RFBM=1, the corresponding XINUSE bit is set to the state of XSTATE_BV.

For standard mode, MXCSR is loaded if RFBM[1]=1 or RFBM[2]=1. It is never initialized.

For compact mode, MXCSR is associated with RFBM[1].

In some generations, the FP error pointers were only restored if there was a Floating point error logged. In newer generations, the FP error pointers are always restored. This is indicated by CPUID Fn8000_0008_EBX[2].

Instruction Support

Form	Subset	Feature Flag
XRSTOR	XRSTOR	CPUID Fn0000_00001_ECX[XSAVE] (bit 26)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XRSTOR *mem*	0F AE /5	Restores user-specified processor state from memory.

Related Instructions

XGETBV, XRSTORS, XSAVE, XSAVEC, XSAVES, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

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.
	X	X	X	CR4.OSXSAVE = 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.
	X	X	X	XCOMP_BV[i] = 0 & XSTATE_BV[i] = 1
	X	X	X	XCOMP_BV[I] = 1 & XCR0[i] = 0
	X	X	X	Bytes 63:16 of header are non-zero
Page fault, #PF	X	X	X	Instruction execution caused a page fault.
X — exception generated				

XRSTORS Restore extended states supervisor

Restores processor state from memory.

XRSTORS is very similar to the XRSTOR instruction in compacted form with the following differences:

1. XRSTORS must be executed at CPL=0
2. XRSTORS must read XCOMP_BV[63]=1, otherwise it will cause a #GP(0) exception
3. XRSTORS is able to restore state enabled from the IA32_XSS MSR.

All other behavior is the same as XRSTOR with the compact form.

Instruction Support

Form	Subset	Feature Flag
XRSTOR	XRSTOR	CPUID Fn0000_00001_ECX_X1[XSAVES] (bit 3)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XRSTOR *mem*	0F C7 /3	Saves user-specified processor state to memory

Related Instructions

XGETBV, XRSTOR, XSAVE, XSAVEC, XSAVES, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

None

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.OSXSAVE = 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.
	X	X	X	CPL <> 0
	X	X	X	(XSTATE_BV[i] & ~IA321_XSS[i]) = 1
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.

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. This bit-wise logical AND of EDX:EAX and XCR0 is known as the Requested Feature Bit Map (RFBM). A component is not saved when its corresponding RFBM bit is zero.

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.

The MXCSR register is saved if either of RFBM bits 0 or 1 are set to 1. If there is no floating point error present, some generations would not write out any of the FP error pointers. On newer generations, these fields are written to zeros. This is indicated by CPUID Fn8000_0008_EBX[2].

Instruction Support

Form	Subset	Feature Flag
XSAVE	XSAVE/XRSTOR	CPUID Fn0000_0001_ECX[XSAVE] (bit 26)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XSAVE *mem*	0F AE /4	Saves user-specified processor state to memory.

Related Instructions

XGETBV, XRSTOR, XSAVEOPT, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

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.
	X	X	X	CR4.OSXSAVE = 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				

XSAVEC Save extended states in compacted form

Saves a user-defined subset of enabled processor state data to a specified memory address, possibly in a compacted form.

This instruction and associated data structures extend the FXSAVE/FXRSTOR memory image used to manage processor states and provides compaction functionality for more efficient context switching. See the XSAVE and XRSTOR instruction descriptions for basic operational details..

XSAVEC is very similar to XSAVE but provides the following alternate functionality:

1. XSAVEC differs from XSAVE by using the init optimization and compaction.
2. XSAVEC differs by only saving a component if its RFBM=1 and its XINUSE=1. XINUSE is a means by which the processor determines whether the feature is in its Initial state.
3. XSAVEC never writes bytes 511:464 of the legacy XSAVE data structure.
4. XSAVEC calculates XSTATE_BV by performing the logical AND of the RFBM and XINUSE bitmaps and writes it to the XSAVE area.
5. XSAVEC calculates XCOMP_BV as [63]=1 and 62:0 = RFBM, and writes it to the XSAVE area.
6. XSAVEC does not modify any other parts of the header except as indicated in 4 and 5.
7. XSAVEC uses the compacted format of the XSAVE extended region while saving state.

Instruction Support

Form	Subset	Feature Flag
XSAVE mem	XSAVEC	CPUID Fn0000_0000D_EAX_x1[XSAVEC] (bit 1)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XSAVEOPT *mem*	0F C7 /4	Saves user-specified processor state to memory.

Related Instructions

XGETBV, XRSTOR, XRSTORS, XSAVE, XSAVES, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

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.
	X	X	X	CR4.OSXSAVE = 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				

XSAVEOPT Save Extended States Performance Optimized

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 the XSAVE and XRSTOR instruction descriptions for basic operational details.

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, XSAVEOPT sets the corresponding bit in the XSTATE_BV field of the save area header. XSAVEOPT 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, XSAVEOPT may clear the corresponding XSTATE_BV bit instead of setting it. This optimization is implementation-dependent.

XSAVEOPT may provide other implementation-specific optimizations, such as the *modified* optimization described for XSAVES.

Instruction Support

Form	Subset	Feature Flag
XSAVEOPT	XSAVEOPT	CPUID Fn0000_0000D_EAX_x1[XSAVEOPT] (bit 0)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XSAVEOPT *mem*	0F AE /6	Saves user-specified processor state to memory.

Related Instructions

XGETBV, XRSTOR, XSAVE, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

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.
	X	X	X	CR4.OSXSAVE = 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				

XSAVES Save Extended States Supervisor

Saves a user-defined subset of enabled processor state data to a specified memory address, possibly in a compacted form.

This instruction and associated data structures extend the XSAVE/XRSTOR memory image used to manage processor states and provides compaction functionality. See the XSAVE and XRSTOR instruction descriptions for basic operational details.

The XSAVES is very similar to XSAVEC but provides the following alternate functionality:

1. XSAVES must be executed at CPL=0
2. XSAVES can save state enabled in the IA32_XSS MSR. The specific state elements saved are determined by the logical AND of EDX:EAX with the logical OR of XCR0 with the IA32_XSS MSR.
3. XSAVES can use the *modified* optimization to not save components, even if RFBM=1 and XINUSE=1 for the stated component. If the component state has not been modified internally since the last execution of XRSTOR or XRSTORS and the XRSTOR_INFO state (an execution environment signature created by the last XRSTOR) matches the current execution state of this XSAVES, the state save can be skipped.

Instruction Support

Form	Subset	Feature Flag
XSAVES	XSAVES	CPUID Fn0000_0000D_EAX_x1[XSAVES] (bit 3)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

Mnemonic	Opcode	Description
XSAVES mem	0F C7 /5	Saves user-specified processor state to memory

Related Instructions

XGETBV, XRSTOR, XRSTORS, XSAVE, XSAVEC, XSETBV

rFLAGS Affected

None

MXCSR Flags Affected

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.
	X	X	X	CR4.OSXSAVE = 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 the XSAVE instruction description 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.

Instruction Support

Form	Subset	Feature Flag
XSETBV	XSAVE/XRSTOR	CPUID Fn0000_0001_ECX[XSAVE] (bit 26)

For more on using the CPUID instruction to obtain processor feature support information, see Appendix E of Volume 3.

Instruction Encoding

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, XSAVEOPT

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X		X	Instruction not supported, as indicated by CPUID feature identifier.
	X		X	CR4.OSXSAVE = 0.
	X		X	Lock prefix (F0h) preceding opcode.
General protection, #GP		X	X	CPL != 0.
	X		X	ECX specifies a reserved or unimplemented XCR address.
	X		X	Any must be zero (MBZ) bits in the XCR were set.
	X		X	Setting XCR0[2:1] to 10b.
	X		X	Writing 0 to XCR[0].
X — exception generated				

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).

Hyperlinks in the table point to the exception tables which follow.

Table 3-1. Instructions By Exception Class

Mnemonic	Extended Type	Legacy Type
Class 1 — AVX / SSE Vector Aligned (VEX.vvvv != 1111)		
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 1X — SSE / AXV / AVX2 Vector (VEX.vvvv != 1111b or VEX.L=1 && !AVX2)		
MOVNTDQA VMOVNTDQA	AVX, AVX2	SSE4.1
Class 2 — AVX / SSE Vector (SIMD 111111)		
DIVPD VDIVPD	AVX	SSE2
DIVPS VDIVPS	AVX	SSE
Class 2-1 — AVX / SSE Vector (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 Vector (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 Vector (SIMD 100001)		
(unused)	—	—

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
Class 2A — AVX / SSE Vector (SIMD 111111, VEX.L = 1)		
(unused)	—	—
Class 2A-1 — AVX / SSE Vector (SIMD 111011, VEX.L = 1)		
DPPD VDPPD	AVX	SSE4.1
Class 2B — AVX / SSE Vector (SIMD 111111, VEX.vvvv != 1111b)		
(unused)	—	—
Class 2B-1 — AVX / SSE Vector (SIMD 100000, VEX.vvvv != 1111b)		
CVTDQ2PS VCVTDQ2PS	AVX	SSE2
Class 2B-2 — AVX / SSE Vector (SIMD 100001, VEX.vvvv != 1111b)		
CVTPD2DQ VCVTPD2DQ	AVX	SSE2
CVTPS2DQ VCVTPS2DQ	AVX	SSE2
CVTTPS2DQ VCVTTPS2DQ	AVX	SSE2
CVTTPD2DQ VCVTTPD2DQ	AVX	SSE2
ROUNDPD, VROUNDPD	AVX	SSE4.1
ROUNDPS, VROUNDPS	AVX	SSE4.1
Class 2B-3 — AVX / SSE Vector (SIMD 111011, VEX.vvvv != 1111b)		
CVTPD2PS VCVTPD2PS	AVX	SSE2
Class 2B-4 — AVX / SSE Vector (SIMD 100011, VEX.vvvv != 1111b)		
SQRTPD VSQRTPD	AVX	SSE2
SQRTPS VSQRTPS	AVX	SSE
Class 3 — AVX / SSE Scalar (SIMD 111111)		
DIVSD VDIVSD	AVX	SSE2
DIVSS VDIVSS	AVX	SSE
Class 3-1 — AVX / SSE Scalar (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 Scalar (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

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
Class 3-3 — AVX / SSE Scalar (SIMD 100000)		
CVTSI2SD VCVTSI2SD	AVX	SSE2
CVTSI2SS VCVTSI2SS	AVX	SSE
Class 3-4 — AVX / SSE Scalar (SIMD 100001)		
ROUNDSD, VROUNDSD	AVX	SSE4.1
ROUNDSS, VROUNDSS	AVX	SSE4.1
Class 3-5 — AVX / SSE Scalar (SIMD 100011)		
SQRTSD VSQRTSD	AVX	SSE2
SQRTSS VSQRTSS	AVX	SSE
Class 3A — AVX / SSE Scalar (SIMD 111111, VEX.vvvv != 1111b)		
(unused)	—	—
Class 3A-1 — AVX / SSE Scalar (SIMD 000011, VEX.vvvv != 1111b)		
COMISD VCOMISD	AVX	SSE2
COMISS VCOMISS	AVX	SSE
CVTPS2PD VCVTPS2PD	AVX	SSE2
Class 3A-2 — AVX / SSE Scalar (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 Vector		
AESDEC VAESDEC	AVX	AES
AESDECLAST VAESDECLAST	AVX	AES
AESENC VAESENC	AVX	AES
AESENCLAST VAESENCLAST	AVX	AES
AESIMC VAESIMC	AVX	AES
AESKEYGENASSIST VAESKEYGENASSIST	AVX	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 VPCLMULQDQ	AVX	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

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
XORPD VXORPD	AVX	SSE2
XORPS VXORPS	AVX	SSE
Class 4A — AVX / SSE Vector (VEX.W = 1)		
BLENDVPD VBLENDVPD	AVX	SSE4.1
BLENDVPS VBLENDVPS	AVX	SSE4.1
Class 4B — AVX / SSE Vector (VEX.L = 1)		
(unused)	—	—
Class 4B-X — SSE / AVX / AVX2 (VEX.L = 1 && !AVX2)		
MPSADBW VMPSADBW	AVX, AVX2	SSE4.1
PACKSSDW VPACKSSDW	AVX, AVX2	SSE2
PACKSSWB VPACKSSWB	AVX, AVX2	SSE2
PACKUSDW VPACKUSDW	AVX, AVX2	SSE4.1
PACKUSWB VPACKUSWB	AVX, AVX2	SSE2
PADDB VPADDB	AVX, AVX2	SSE2
PADDD VPADDD	AVX, AVX2	SSE2
PADDQ VPADDQ	AVX, AVX2	SSE2
PADDSB VPADDSB	AVX, AVX2	SSE2
PADDSW VPADDSW	AVX, AVX2	SSE2
PADDUSB VPADDUSB	AVX, AVX2	SSE2
PADDUSW VPADDUSW	AVX, AVX2	SSE2
PADDW VPADDW	AVX, AVX2	SSE2
PALIGNR VPALIGNR	AVX, AVX2	SSSE3
PAND VPAND	AVX, AVX2	SSE2
PANDN VPANDN	AVX, AVX2	SSE2
PAVGB VPAVGB	AVX, AVX2	SSE
PAVGW VPAVGW	AVX, AVX2	SSE
PBLENDW VPBLENDW	AVX, AVX2	SSE4.1
PCMPEQB VPCMPEQB	AVX, AVX2	SSE2
PCMPEQD VPCMPEQD	AVX, AVX2	SSE2
PCMPEQQ VPCMPEQQ	AVX, AVX2	SSE4.1
PCMPEQW VPCMPEQW	AVX, AVX2	SSE2
PCMPGTB VPCMPGTB	AVX, AVX2	SSE2
PCMPGTD VPCMPGTD	AVX, AVX2	SSE2
PCMPGTQ VPCMPGTQ	AVX, AVX2	SSE4.2
PCMPGTW VPCMPGTW	AVX, AVX2	SSE2
PHADDD VPHADDD	AVX, AVX2	SSSE3
PHADDSW VPHADDSW	AVX, AVX2	SSSE3
PHADDW VPHADDW	AVX, AVX2	SSSE3
PHSUBD VPHSUBD	AVX, AVX2	SSSE3
PHSUBW VPHSUBW	AVX, AVX2	SSSE3
PHSUBSW VPHSUBSW	AVX, AVX2	SSSE3
PMADDUBSW VPMADDUBSW	AVX, AVX2	SSSE3

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
PMADDWD VPMADDWD	AVX, AVX2	SSE2
PMAXSB VPMAXSB	AVX, AVX2	SSE4.1
PMAXSD VPMAXSD	AVX, AVX2	SSE4.1
PMAXSW VPMAXSW	AVX, AVX2	SSE
PMAXUB VPMAXUB	AVX, AVX2	SSE
PMAXUD VPMAXUD	AVX, AVX2	SSE4.1
PMAXUW VPMAXUW	AVX, AVX2	SSE4.1
PMINSB VPMINSB	AVX, AVX2	SSE4.1
PMINSD VPMINSD	AVX, AVX2	SSE4.1
PMINSW VPMINSW	AVX, AVX2	SSE
PMINUB VPMINUB	AVX, AVX2	SSE
PMINUD VPMINUD	AVX, AVX2	SSE4.1
PMINUW VPMINUW	AVX, AVX2	SSE4.1
PMULDQ VPMULDQ	AVX, AVX2	SSE4.1
PMULHRSW VPMULHRSW	AVX, AVX2	SSSE3
PMULHUW VPMULHUW	AVX, AVX2	SSE2
PMULHW VPMULHW	AVX, AVX2	SSE2
PMULLD VPMULLD	AVX, AVX2	SSE4.1
PMULLW VPMULLW	AVX, AVX2	SSE2
PMULUDQ VPMULUDQ	AVX, AVX2	SSE2
POR VPOR	AVX, AVX2	SSE2
PSADBW VPSADBW	AVX, AVX2	SSE
PSHUFB VPSHUFB	AVX, AVX2	SSSE3
PSIGNB VPSIGNB	AVX, AVX2	SSSE3
PSIGND VPSIGND	AVX, AVX2	SSSE3
PSIGNW VPSIGNW	AVX, AVX2	SSSE3
PSUBB VPSUBB	AVX, AVX2	SSE2
PSUBD VPSUBD	AVX, AVX2	SSE2
PSUBQ VPSUBQ	AVX, AVX2	SSE2
PSUBSB VPSUBSB	AVX, AVX2	SSE2
PSUBSW VPSUBSW	AVX, AVX2	SSE2
PSUBUSB VPSUBUSB	AVX, AVX2	SSE2
PSUBUSW VPSUBUSW	AVX, AVX2	SSE2
PSUBW VPSUBW	AVX, AVX2	SSE2
PUNPCKHBW VPUNPCKHBW	AVX, AVX2	SSE2
PUNPCKHDQ VPUNPCKHDQ	AVX, AVX2	SSE2
PUNPCKHQDQ VPUNPCKHQDQ	AVX, AVX2	SSE2
PUNPCKHWD VPUNPCKHWD	AVX, AVX2	SSE2
PUNPCKLBW VPUNPCKLBW	AVX, AVX2	SSE2
PUNPCKLDQ VPUNPCKLDQ	AVX, AVX2	SSE2
PUNPCKLQDQ VPUNPCKLQDQ	AVX, AVX2	SSE2
PUNPCKLWD VPUNPCKLWD	AVX, AVX2	SSE2

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
PXOR VPXOR	AVX, AVX2	SSE2
Class 4C — AVX / SSE Vector (VEX.vvvv != 1111b)		
MOVSHDUP VMOVSHDUP	AVX	SSE3
MOVSLDUP VMOVSLDUP	AVX	SSE3
PTEST VPTEST	AVX	SSE4.1
RCPPS VRCPPS	AVX	SSE
RSQRTPS VRSQRTPS	AVX	SSE
Class 4C-1 — AVX / SSE Vector (write to RO memory, VEX.vvvv != 1111b)		
LDDQU VLDDQU	AVX	SSE3
MOVDQU VMOVDQU	AVX	SSE2
MOVUPD VMOVUPD	AVX	SSE2
MOVUPS VMOVUPS	AVX	SSE
Class 4D — AVX / SSE Vector (VEX.vvvv != 1111b, VEX.L = 1)		
MASKMOVDQU VMASKMOVDQU	AVX	SSE2
PCMPESTRI VPCMPESTRI	AVX	SSE4.2
PCMPESTRM VPCMPESTRM	AVX	SSE4.2
PCMPISTRI VPCMPISTRI	AVX	SSE4.2
PCMPISTRM VPCMPISTRM	AVX	SSE4.2
PHMINPOSUW VPHMINPOSUW	AVX	SSE4.1
Class 4D-X — SSE / AVX / AVX2 Vector (VEX.vvvv != 1111b, (VEX.L = 1 && !AVX2))		
PABSB VPABSB	AVX, AVX2	SSSE3
PABSD VPABSD	AVX, AVX2	SSSE3
PABSW VPABSW	AVX, AVX2	SSSE3
PSHUFD VPSHUFD	AVX, AVX2	SSE2
PSHUFHW VPSHUFHW	AVX, AVX2	SSE2
PSHUFLW VPSHUFLW	AVX, AVX2	SSE2
Class 4E — AVX / SSE Vector (VEX.W = 1, VEX.L = 1)		
(unused)	—	—
Class 4E-X — SSE / AVX / AVX2 Vector (VEX.W = 1, (VEX.L = 1 && !AVX2))		
PBLENDVB VPBLENDVB	AVX	SSE4.1
Class 4F — AVX / SSE (VEX.L = 1)		
(unused)	—	—
Class 4F-X — SSE / AVX / AVX2 Vector (VEX.L = 1 && !AVX2)		
PSLLD VPSLLD	AVX, AVX2	SSE2
PSLLQ VPSLLQ	AVX, AVX2	SSE2
PSLLW VPSLLW	AVX, AVX2	SSE2
PSRAD VPSRAD	AVX, AVX2	SSE2
PSRAW VPSRAW	AVX, AVX2	SSE2
PSRLD VPSRLD	AVX, AVX2	SSE2
PSRLQ VPSRLQ	AVX, AVX2	SSE2
PSRLW VPSRLW	AVX, AVX2	SSE2
Class 4G — AVX Vector (VEX.W = 1, VEX.vvvv != 1111b)		

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
VTESTPD	AVX	—
VTESTPS	AVX	—
Class 4H — AVX, 256-bit only (VEX.L = 0; No SIMD Exceptions)		
VPERMD	AVX2	—
VPERMPS	AVX2	—
Class 4H-1 — AVX2, 256-bit only (VEX.L = 0, VEX.vvvv != 1111b)		
VPERMPD	AVX2	—
VPERMQ	AVX2	—
Class 4J — AVX2 (VEX.W = 1)		
VPBLENDD	AVX2	—
VPSRAVD	AVX2	—
Class 4K — AVX2		
VPMASKMOVD	AVX2	—
VPMASKMOVQ	AVX2	—
VPSLLVD	AVX2	—
VPSLLVQ	AVX2	—
VPSRLVD	AVX2	—
VPSRLVQ	AVX2	—
Class 5 — AVX / SSE Scalar		
RCPSS VRCPSS	AVX	SSE
RSQRTSS VRSQRTSS	AVX	SSE
Class 5A — AVX / SSE Scalar (VEX.L = 1)		
INSERTPS VINSERTPS	AVX	SSE4.1
Class 5B — AVX / SSE Scalar (VEX.vvvv != 1111b)		
CVTDQ2PD VCVTDQ2PD	AVX	SSE2
MOVDDUP VMOVDDUP	AVX	SSE3
Class 5C — AVX /SSE Scalar (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
Class 5C-X — SSE / AVX / AVX2 Scalar (VEX.vvvv != 1111b, (VEX.L = 1 && !AVX2))		
PMOVSXBD VPMOVSXBD	AVX, AVX2	SSE4.1
PMOVSXBQ VPMOVSXBQ	AVX, AVX2	SSE4.1
PMOVSXBW VPMOVSXBW	AVX, AVX2	SSE4.1
PMOVSXDQ VPMOVSXDQ	AVX, AVX2	SSE4.1
PMOVSXWD VPMOVSXWD	AVX, AVX2	SSE4.1
PMOVSXWQ VPMOVSXWQ	AVX, AVX2	SSE4.1
PMOVZXBD VPMOVZXBD	AVX, AVX2	SSE4.1
PMOVZXBQ VPMOVZXBQ	AVX, AVX2	SSE4.1
PMOVZXBW VPMOVZXBW	AVX, AVX2	SSE4.1
PMOVZXDQ VPMOVZXDQ	AVX, AVX2	SSE4.1

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
PMOVZXWD VPMOVZXWD	AVX, AVX2	SSE4.1
PMOVZXWQ VPMOVZXWQ	AVX, AVX2	SSE4.1
Class 5C-1 — AVX / SSE Scalar (write to RO memory, 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 Scalar (write to RO memory, VEX.vvvv != 1111b (variant))		
MOVSD VMOVSD	AVX	SSE2
MOVSS VMOVSS	AVX	SSE
Class 5E — AVX / SSE Scalar (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		
(unused)	—	—
Class 6A — AVX Mixed Memory Argument (VEX.W = 1)		
(unused)	—	—
Class 6A-1 — AVX Mixed Memory Argument (write to RO memory, VEX.W = 1)		
VMASKMOVPD	AVX	—
VMASKMOVPS	AVX	—
Class 6B — AVX Mixed Memory Argument (VEX.W = 1, VEX.L = 0)		
VINSERTF128	AVX	—
VINSERTI128	AVX2	—
VPERM2F128	AVX	—
VPERM2I128	AVX2	—
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.L = 0, VEX.vvvv != 1111b)		
VBROADCASTF128	AVX	—
VBROADCASTI128	AVX2	—
VEXTRACTI128	AVX2	—
Class 6C-X — AVX / AVX2 (W=1, vvvv!=1111b, L=0, (reg src op specified && !AVX2))		
VBROADCASTSD	AVX, AVX2	—
Class 6D — AVX Mixed Memory Argument (VEX.W = 1, VEX.vvvv != 1111b)		
VPBROADCASTB	AVX2	—
VPBROADCASTD	AVX2	—
VPBROADCASTQ	AVX2	—

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
VPBROADCASTW	AVX2	—
Class 6D-X — AVX / AVX2 (W = 1, vvvv != 1111b, (ModRM.mod = 11b && !AVX2))		
VBROADCASTSS	AVX, AVX2	—
Class 6E — AVX Mixed Memory Argument (VEX.W = 1, VEX.vvvv != 1111b (variant))		
VPERMILPD	AVX	—
VPERMILPS	AVX	—
Class 6F — AVX2 (VEX.W = 1, VEX.vvvv != 1111b, VEX.L = 0, ModRM.mod = 11b)		
VBROADCASTI128	AVX2	—
Class 7 — AVX / SSE No Memory Argument		
(unused)	—	—
Class 7A — AVX /SSE No Memory Argument (VEX.L = 1)		
MOVHLPS VMOVHLPS	AVX	SSE
MOVLHPS VMOVLHPS	AVX	SSE
Class 7A-X SSE / AVX / AVX2 Vector (VEX.L = 1 && !AVX2)		
PSLLDQ VPSLLDQ	AVX, AVX2	SSE2
PSRLDQ VPSRLDQ	AVX, AVX2	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)		
(not used)	—	—
Class 7C-X SSE / AVX / AVX2 Vector (VEX.vvvv != 1111b, (VEX.L = 1 && !AVX2))		
PMOVMSKB VPMOVMSKB	AVX, AVX2	SSE2
Class 8 — AVX No Memory Argument (VEX.vvvv != 1111b, VEX.W = 1)		
VZEROALL	AVX	—
VZEROUPPER	AVX	—
Class 9 — AVX 4-byte Argument (write to RO memory, VEX.vvvv != 1111b, VEX.L = 1)		
STMXCSR VSTMXCSR	AVX	SSE
Class 9A — AVX 4-byte argument (reserved MBZ = 1, VEX.vvvv != 1111b, VEX.L = 1)		
LDMXCSR VLDMXCSR	AVX	SSE

Table 3-1. Instructions By Exception Class (continued)

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 (continued)

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 (SIMD 110011, XOP.vvvv != 1111b, XOP.W = 1)		
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 — F16C Instructions		
VCVTPH2PS	F16C	—
VCVTPS2PH	F16C	—
Class 12 — AVX2 VSID (ModRM.mod = 11b, ModRM.rm != 100b)		
VGATHERDPD	AVX2	—
VGATHERDPS	AVX2	—
VGATHERQPD	AVX2	—
VGATHERQPS	AVX2	—
VPGATHERDD	AVX2	—
VPGATHERDQ	AVX2	—
VPGATHERQD	AVX2	—
VPGATHERQQ	AVX2	—
Class FMA-2 — FMA / FMA4 Vector (SIMD Exceptions PE, UE, OE, DE, IE)		
VFMADDPD	FMA4	—
VFMADDPS	FMA4	—
VFMADDSUBPD	FMA4	—
VFMADDSUBPS	FMA4	—
VFMSUBADDPD	FMA4	—
VFMSUBADDPS	FMA4	—
VFMSUBPD	FMA4	—
VFMSUBPS	FMA4	—
VFNMADDPD	FMA4	—

Table 3-1. Instructions By Exception Class (continued)

Mnemonic	Extended Type	Legacy Type
VFNMADDPS	FMA4	—
VFNMSUBPD	FMA4	—
VFNMSUBPS	FMA4	—
Class FMA-3 — FMA / FMA4 Scalar (SIMD Exceptions PE, UE, OE, DE, IE)		
VFMADDSD	FMA4	—
VFMADDSS	FMA4	—
VFMSUBSD	FMA4	—
VFMSUBSS	FMA4	—
VFNMADDSD	FMA4	—
VFNMADDSS	FMA4	—
VFNMSUBSD	FMA4	—
VFNMSUBSS	FMA4	—
Unique Cases		
XGETBV	—	—
XRSTOR	—	—
XSAVE/XSAVEOPT	—	—
XSETBV	—	—

Class 1 — AVX / SSE Vector Aligned (VEX.vvvv != 1111)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[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 a 16-byte boundary.
	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 1X — SSE / AXV / AVX2 Vector (VEX.vvvv != 1111b or VEX.L=1 && !AVX2)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 a 16-byte boundary.
	S	S	X	Write to a read-only data segment.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit 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, AVX2, and SSE exception
A — AVX, AVX2 exception
S — SSE exception

Class 2 — AVX / SSE Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Page fault, #PF		S	X	Instruction execution caused a page fault.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when 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 2A-1 — AVX / SSE Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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	Non-aligned memory operand while MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Scalar (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	XFEATURE_ENABLED_MASK[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 when 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 Vector

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[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.
	S	S	S	Memory operand not 16-byte aligned and MXCSR.MM = 0.
			X	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

Class 4B-X — SSE / AVX / AVX2 (VEX.L = 1 && !AVX2)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

Class 4C — AVX / SSE Vector (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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (write to RO memory, 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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	X	Memory operand not 16-byte aligned when alignment checking enabled.
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 Vector (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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

Class 4D-X — SSE / AVX / AVX2 Vector (VEX.vvvv != 1111b, (VEX.L = 1 && !AVX2))

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

Class 4E — AVX / SSE Vector (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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		S	X	Instruction execution caused a page fault.

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

Class 4E-X — SSE / AVX / AVX2 Vector (VEX.W = 1, (VEX.L = 1 && !AVX2))

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		S	X	Instruction execution caused a page fault.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

Class 4F — AVX / SSE (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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF			A	Instruction execution caused a page fault.

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

Class 4F-X — SSE / AVX / AVX2 Vector (VEX.L = 1 && !AVX2)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	When alignment checking enabled: • 128-bit memory operand not 16-byte aligned. • 256-bit memory operand not 32-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.

X — AVX, AVX2, and SSE exception
A — AVX and AVX2 exception
S — SSE exception

Class 4G — AVX Vector (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.
	X	X		AVX instructions are only recognized in protected mode.
	X	X	X	CR0.EM = 1.
	X	X	X	CR4.OSFXSR = 0.
			X	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			X	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			X	VEX.W = 1.
			X	VEX.vvvv ! = 1111b.
			X	REX, F2, F3, or 66 prefix preceding VEX prefix.
	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	Null data segment used to reference memory.
Alignment check, #AC	S	S	S	Memory operand not 16-byte aligned when alignment checking enabled and MXCSR.MM = 1.
			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		X	X	Instruction execution caused a page fault.
X — AVX exception				

Class 4H — AVX, 256-bit only (VEX.L = 0; No SIMD Exceptions)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	A	A	A	CR0.EM = 1.
	A	A	A	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L= 0.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	A	A	A	Lock prefix (F0h) preceding opcode.
Device not available, #NM	A	A	A	CR0.TS = 1.
Stack, #SS	A	A	A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	A	A	A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Alignment check, #AC			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

Class 4H-1 — AVX2, 256-bit only (VEX.L = 0, VEX.vvvv != 1111b)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	A	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	A	A	A	CR0.EM = 1.
	A	A	A	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L= 0.
			A	VEX.vvvv ! = 1111b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
	A	A	A	Lock prefix (F0h) preceding opcode.
Device not available, #NM	A	A	A	CR0.TS = 1.
Stack, #SS	A	A	A	Memory address exceeding stack segment limit or non-canonical.
General protection, #GP	A	A	A	Memory address exceeding data segment limit or non-canonical.
			A	Null data segment used to reference memory.
Alignment check, #AC			A	Memory operand not 16-byte aligned when alignment checking enabled.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

Class 4J — AVX2 (VEX.W = 1)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

Class 4K — AVX2

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF			A	Instruction execution caused a page fault.
A — AVX2 exception				

Class 5 — AVX / SSE Scalar

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

Class 5A — AVX / SSE Scalar (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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

Class 5B — AVX / SSE Scalar (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	XFEATURE_ENABLED_MASK[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.

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

Class 5C — AVX /SSE Scalar (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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

Class 5C-X — SSE / AVX / AVX2 Scalar (VEX.vvvv != 1111b, (VEX.L = 1 && !AVX2))

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 1 when AVX2 not supported.
			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 when alignment checking enabled.

X — SSE, AVX, and AVX2 exception
A — AVX, AVX2 exception
S — SSE exception

Class 5C-1 — AVX / SSE Scalar (write to RO memory, 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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

Class 5D — AVX / SSE Scalar (write to RO memory, 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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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

Class 5E — AVX / SSE Scalar (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	XFEATURE_ENABLED_MASK[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 when alignment checking enabled.

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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

Class 6A-1 — AVX Mixed Memory Argument (write to RO memory, 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	XFEATURE_ENABLED_MASK[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.
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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
A — AVX exception.				

Class 6C — AVX Mixed Memory Argument (VEX.W = 1, VEX.L = 0, 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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

Class 6C-X — AVX / AVX2 (W=1, vvvv!=1111b, L=0, (reg src op specified && !AVX2))

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	Register-based source operand specified when AVX2 not supported.
			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	Unaligned memory reference when alignment checking enabled.
A — AVX, AVX2 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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

Class 6D-X — AVX / AVX2 (W = 1, vvvv != 1111b, (ModRM.mod = 11b && !AVX2))

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	MODRM.mod = 11b when AVX2 not supported.
			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	Unaligned memory reference when alignment checking enabled.
A — AVX, AVX2 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	XFEATURE_ENABLED_MASK[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	Unaligned memory reference when alignment checking enabled.
A — AVX exception.				

Class 6F — AVX2 (VEX.W = 1, VEX.vvvv != 1111b, VEX.L = 0, ModRM.mod = 11b)

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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.W = 1.
			A	VEX.vvvv ! = 1111b.
			A	VEX.L = 0.
			A	Register-based source operand specified (MODRM.mod = 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	Unaligned memory reference when alignment checking enabled.
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	XFEATURE_ENABLED_MASK[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	XFEATURE_ENABLED_MASK[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 7A-X SSE / AVX / AVX2 Vector (VEX.L = 1 && !AVX2)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.L = 1 when AVX2 not supported.
			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 — SSE, AVX, and AVX2 exception
A — AVX, AVX2 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	XFEATURE_ENABLED_MASK[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	XFEATURE_ENABLED_MASK[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 7C-X SSE / AVX / AVX2 Vector (VEX.vvvv != 1111b, (VEX.L = 1 && !AVX2))

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	X	X	X	Instruction not supported, as indicated by CPUID feature identifier.
	A	A		AVX instructions are only recognized in protected mode.
	S	S	S	CR0.EM = 1.
	S	S	S	CR4.OSFXSR = 0.
			A	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			A	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	VEX.vvvv field ! = 1111b.
			A	VEX.L field = 1 when AVX2 not supported.
			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 — SSE, AVX and AVX2 exception
A — AVX, AVX2exception
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	XFEATURE_ENABLED_MASK[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 memory, 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	XFEATURE_ENABLED_MASK[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 when 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	XFEATURE_ENABLED_MASK[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 when 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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
X — XOP exception				

Class 10D — XOP Base (SIMD 110011, 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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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			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	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when alignment checking enabled.
X — XOP exception				

Class 11 — F16C Instructions

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		AVX instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			F	VEX.W field = 1.
			A	VEX.vvvv ! = 1111b.
			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.
Alignment check, #AC			F	Unaligned memory reference when alignment checking enabled.
Page fault, #PF			F	Instruction execution caused a page fault.
SIMD Floating-Point Exception, #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 exception (IE)			F	A source operand was an SNaN value.
			F	Undefined operation.
Denormalized-operand exception (DE)			F	A source operand was a denormal value.
Overflow exception (OE)			F	Rounded result too large to fit into the format of the destination operand.
Underflow exception (UE)			F	Rounded result too small to fit into the format of the destination operand.
Precision exception (PE)			F	A result could not be represented exactly in the destination format.

F — F16C exception.

Class 12 — AVX2 VSID (ModRM.mod = 11b, ModRM.rm != 100b)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD	A	A	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	XFEATURE_ENABLED_MASK[2:1] ! = 11b.
			A	REX, F2, F3, or 66 prefix preceding VEX prefix.
			A	Lock prefix (F0h) preceding opcode.
			A	MODRM.mod = 11b
			A	MODRM.rm ! = 100b
			A	YMM/XMM registers specified for destination, mask, and index not unique.
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.
Alignment check, #AC			A	Alignment checking enabled and: 256-bit memory operand not 32-byte aligned or 128-bit memory operand not 16-byte aligned.
Page fault, #PF		A	A	Instruction execution caused a page fault.
A — AVX2 exception				

Class FMA-2 — FMA / FMA4 Vector (SIMD Exceptions PE, UE, OE, DE, IE)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Memory operand not 16-byte aligned when 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 — FMA, FMA4 exception				

Class FMA-3 — FMA / FMA4 Scalar (SIMD Exceptions PE, UE, OE, DE, IE)

Exceptions

Exception	Mode			Cause of Exception
	Real	Virt	Prot	
Invalid opcode, #UD			F	Instruction not supported, as indicated by CPUID feature identifier.
	F	F		FMA instructions are only recognized in protected mode.
			F	CR4.OSXSAVE = 0, indicated by CPUID Fn0000_0001_ECX[OSXSAVE].
			F	XFEATURE_ENABLED_MASK[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	Non-aligned memory reference when 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 — FMA, 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/XSAVEOPT

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.

Appendix A AES Instructions

This appendix gives background information concerning the use of the AES instruction subset in the implementation of encryption compliant to the Advanced Encryption Standard (AES).

A.1 AES Overview

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.

A.2 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)

A.3 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 A-1 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 A-1. GFMatrix Representation of 16-byte Block

Figure A-2 shows the AMD64 AES format, with the corresponding mapping of FIPS 197 AES "words" to operand bytes.

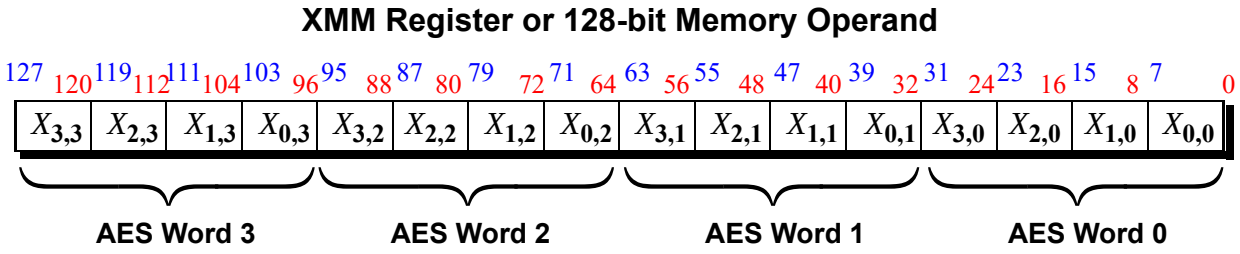


Figure A-2. GFMatrix to Operand Byte Mappings

A.4 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

$$a_nX^n + \ldots + a_1X + a_0 \rightarrow 2^n a_n + \ldots + 2a_1 + a_0 \rightarrow a_n \ldots a_1a_0\text{b}$$

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 $x \in GF$,

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

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.

A.4.1 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 A.11, "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.

A.4.2 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;
}
```

A.5 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*.

A.5.1 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**.

A.6 Initializing the Sbox and InvSBox Matrices

The AES makes use of a bijective mapping σ : 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,

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

and

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

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]];
}
```

A.6.1 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*:

$$A[i] = a_{i\mathbf{8}}\, a_{i\mathbf{7}} \ldots a_{i\mathbf{0}}$$

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

Table A-1. SBox Definition

	S[3:0]																
		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
S[7:4]	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

A.6.2 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 A-2 shows the resulting **InvSBox[ ]**, as defined in the FIPS 197.

Table A-2. 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

A.7 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

A.7.1 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 A.8, “The Cipher Function” and Section A.9, “The InvCipher Function” for detailed information.

A.7.2 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 A-3 shows the relationship between cipher key length, round sequence length, and round key length.

Table A-3. 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 key K_i comprises the four *GF* words $w[4i], \ldots, w[4i + 3]$. 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.

A.8 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);
}
```

A.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;
}
```

A.8.2 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 A.9, “The InvCipher Function” for more information.

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;
}
```

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;
}
```

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;
```

RotWord( ) Function

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

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

MixColumns( ) Function

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

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

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

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);
}
```

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;
}
```

A.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;
}
```

A.9 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);
}
```

A.9.1 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;
}
```

A.9.2 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

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

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;
```

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;
}
```

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;
}
```

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);
}
```

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;
```

```
}
```

A.9.3 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;
}
```

A.10 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 $i = 1,\ldots,N_r - 1$, it replaces W_i with the matrix product $W_i \odot Q$, where

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

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 $i = 1,\ldots,N_r - 1$.

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

C is a linear transformation and therefore, so is 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 $i = N_r - 1,\ldots,1$, 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,

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

A.11 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 x1 and x2 is added to the other. If x1 ≤ x2, then the values of x2 and a2 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;
}
```

Index

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X