AMD64 Technology

AMD64 Architecture Programmer's Manual

Volume 6: 128-Bit and 256-Bit XOP and FMA4 Instructions

Publication No.	Revision	Date
43479	3.04	November 2009

Advanced Micro Devices

Contents

Tables

Revision History

Date	Revision	Description
November 2009	3.04	Removed #UD CR0.EM exception from all exception tables. Added VPERMIL2PD and VPERMIL2PS instructions. Corrected many small factual errors and typos.

Preface

About This Book

The instructions described in this book are part of a multivolume work entitled the *AMD64 Architecture Programmer's Manual*. The following table lists each volume and its order number.

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 Media Instructions	26568
Volume 5: 64-Bit Media and x87 Floating-Point Instructions	26569
Volume 6: 128-Bit and 256-Bit XOP and FMA4 Instructions	43479

Audience

This document is intended for all programmers writing application or system software for a processor that implements the AMD64 architecture.

Organization

Volumes 3 through 6 describe the AMD64 architecture's instruction set in detail. Together, they cover each instruction's mnemonic syntax, opcodes, functions, affected flags, and possible exceptions.

The AMD64 instruction set is divided into seven subsets:

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

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

This volume describes the 128-bit and 256-bit XOP and FMA4 instruction extensions. The index at the end cross-references topics within this volume. For other topics relating to the AMD64 architecture, and for information on instructions in other subsets, see the tables of contents and indexes of the other volumes.

Definitions

Many of the following definitions assume an in-depth knowledge of the legacy x86 architecture. See "Related Documents" on page 22 for descriptions of the legacy x86 architecture.

Terms and Notation

In addition to the notation described below, "Opcode-Syntax Notation" in Volume 3 describes notation relating specifically to opcodes.

1011b

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

F0EAh

A hexadecimal value—in this example a 2-byte value.

[1,2)

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

7–4

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

128-bit media instructions

Instructions that use the 128-bit XMM registers. These are a combination of the SSE and SSE2 instruction sets.

64-bit media instructions

Instructions that use the 64-bit MMX registers. These are primarily a combination of MMX™ and 3DNow!™ instruction sets, with some additional instructions from the SSE 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.

#GP(0)

Notation indicating a general-protection exception (#GP) with error code of 0.

absolute

Said of a displacement that references the base of a code segment rather than an instruction pointer. Contrast with *relative*.

ASID

Address space identifier.

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

To write a bit value of 0. Compare *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.

CR0–CR4

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

CR0.PE = 1

Notation indicating that the PE bit of the CR0 register has a value of 1.

direct

Referencing a memory location whose address is included in the instruction's syntax as an immediate operand. The address may be an absolute or relative address. Compare *indirect*.

dirty data

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

displacement

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

doubleword

Two words, or four bytes, or 32 bits.

double quadword

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

DS:rSI

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

EFER.LME = 0

Notation indicating that the LME bit of the EFER register has a value of 0.

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 executing an instruction. The processor's response to an exception depends on the type of the exception. For all exceptions except 128-bit media SIMD floating-point exceptions and x87 floating-point exceptions, control is transferred to the 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.

FF /0

Notation indicating that FF is the first byte of an opcode, and a subfield in the second byte has a value of 0.

flush

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

GDT

Global descriptor table.

GIF

Global interrupt flag.

IDT

Interrupt descriptor table.

IGN

Ignore. Field is ignored.

indirect

Referencing a memory location whose address is in a register or other memory location. The address may be an absolute or relative address. Compare *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. See "Related Documents" on page 22 for descriptions of the legacy x86 architecture.

legacy mode

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

long mode

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

lsb

Least-significant bit.

LSB

Least-significant byte.

main memory

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

mask

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

MBZ

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

memory

Unless otherwise specified, *main memory.*

ModRM

A byte following an instruction opcode that specifies address calculation based on mode (Mod), register (R), and memory (M) variables.

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.

multimedia instructions

A combination of *128-bit media instructions* and *64-bit media instructions*.

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 a processor's 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, or eight bytes, or 64 bits.

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.

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

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

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

RAZ

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

real-address mode

A submode of legacy mode with 16-bit addressing and operand size and a simple form of segmentation, lacking the segment and privilege protection mechanisms of protected mode. See *real mode*.

real mode

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

relative

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

REX

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

RIP-relative addressing

Addressing relative to the 64-bit RIP instruction pointer.

set

To write a bit value of 1. Compare *clear*.

SIB

A byte following an instruction opcode that specifies address calculation based on scale (S), index (I), and base (B).

SIMD

Single instruction, multiple data. See *vector*.

*SSE*n *and SSSE*n

Various extensions to the SSE instruction set. See *128-bit media instructions* and *64-bit media instructions*.

sticky bit

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

TOP

The x87 top-of-stack pointer.

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 of the 128-bit and 64-bit media instructions use vectors as operands. Vectors are also called *packed* or *SIMD* (single-instruction multiple-data) operands.

(2) An index into an interrupt descriptor table (IDT), used to access exception handlers. Compare *exception*.

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*.

Registers

In the following list of registers, the names are used to refer either to a given register or to the contents of that register:

AH–DH

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

AL–DL

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

AL–r15B

The low 8-bit AL, BL, CL, DL, SIL, DIL, BPL, SPL, and R8B–R15B registers, available in 64-bit mode.

BP

Base pointer register.

CRn

Control register number *n*.

CS

Code segment register.

eAX–eSP

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

EBP

Extended base pointer register.

EFER

Extended features enable register.

eFLAGS

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

EFLAGS

32-bit (extended) flags register.

eIP

16-bit or 32-bit instruction-pointer register. Compare *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. Compare *RFLAGS*.

RFLAGS

64-bit flags register. Compare *rFLAGS*.

rIP

16-bit, 32-bit, or 64-bit instruction-pointer register. Compare *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), a new register introduced in the AMD64 architecture to speed interrupt management.

TR

Task register.

XMM0–XMM15

The 128-bit XMM registers; each is the lower half of a corresponding 256-bit YMM register.

YMM0–YMM15

The 256-bit YMM registers; the lower half of each of these is the corresponding 128-bit XMM register.

Endian Order

The x86 and AMD64 architectures address memory using little-endian byte-ordering. Multibyte values are stored with their least-significant byte at the lowest byte address, and they are illustrated with their least significant byte at the right side. Strings are illustrated in reverse order, because the addresses of their 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 New 128-Bit and 256-Bit Instructions

This release of the AMD64 architecture covers the XOP and FMA4 instruction set extensions. These 128-bit and 256-bit instructions complement the AMD64 128-bit media instructions deescribed in detail in the *AMD64 Architecture Programmer's Manual Volume 4: 128-Bit Media Instructions*, order# 26568. This document describes new instructions that are designed to:

- Improve performance by increasing the work per instruction and
- reduce the need to copy and move around register operands.

These instruction set extensions include:

- Floating-point multiply accumulate instructions
- Floating-point fraction extract
- Integer horizontal add instructions
- Integer multiply accumulate instructions
- Byte permutation and bit granularity conditional move instructions
- Packed integer compare and individual-partition shift/rotate instructions

These instructions all use the new XOP instruction format, which takes advantage of the three- and four-operand non-destructive capability, 256-bit operand size, and instruction length efficiency provided by this encoding. These instructions operate on either the lower 128- or full 256-bits of the new YMM registers. Context handling of the YMM register set is supported by the new XSAVE/XRSTOR instructions in conjunction with the XSETBV and XGETBV instructions. Support for YMM context handling must be provided by the operating system and must be indicated by setting CR4.OSXSAVE to 1.

Support for the new instructions is indicated by use of the CPUID instruction:

- XOP—ECX bit 11 as returned by CPUID function 8000_0001h.
- FMA4—ECX bit 16 as returned by CPUID function 8000_0001h.

Attempting to execute these instructions causes a #UD exception either if they are not present in the hardware or if operating system support for YMM context switching is not indicated by setting CR4.OSXSAVE to 1.

1.1 New Instruction Format

The XOP instructions utilize a new three-byte XOP prefix preceding the opcode byte. This prefix replaces the use of the 0F, 66, F2 and F3 prefix bytes and the REX prefix and encodes additional information as well. The FMA4 instructions utilize the new AVX VEX prefix which provides similar encoding capabilities.

Figure 1-1 shows the byte order of the instruction format.

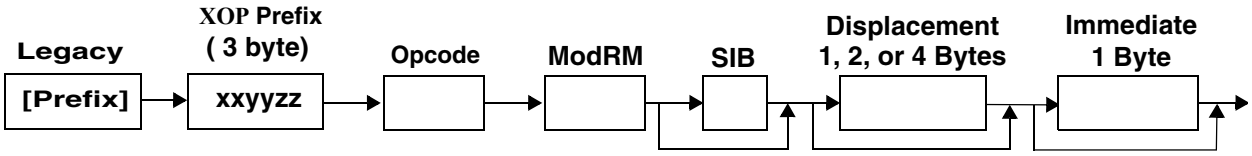


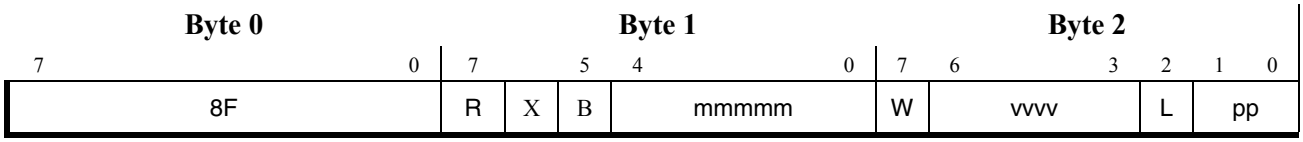
Figure 1-1. Instruction Byte-Order

1.1.1 Legacy Prefix

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

1.1.2 Three-byte Prefix Format

The format of the three-byte form of the XOP and FMA4 instruction prefixes is shown in Figure 1-2.



	Bit	Mnemonic	Description
Byte 0	7–0	8Fh	XOP Prefix Byte for 3-byte XOP Prefix
Byte 1	7	R	Inverted one bit extension to ModRM.reg field
	6	X	Inverted one bit extension of the SIB index field
	5	B	Inverted one bit extension of the ModRM r/m field or the SIB base field
	4–0	mmmmm	XOP opcode map select: 08h—instructions with immediate byte; 09h—instructions with no immediate;
Byte 2	7	W	Default operand size override for a general purpose register to 64-bit size in 64-bit mode; operand configuration specifier for certain XMM/YMM-based operations.
	6–3	vvvv	Source or destination register specifier in inverted 1's complement format.
	2	L	Vector length for XMM/YMM-based operations.
	1–0	pp	Specifies whether there's an implied 66, F2, or F3 opcode extension

Figure 1-2. Three-byte XOP Format

Prefix Byte 0

Byte 0 of the XOP prefix is set to 8Fh. This signifies an XOP prefix only in conjunction with the mmmmm field of the following byte being greater than or equal to 8; if the mmmmm field is less than 8 then these two bytes are a form of the POP instruction rather than an XOP prefix.

Prefix Byte 1

Byte 1 of the XOP prefix has four fields.

R Bit (Prefix Byte 1, Bit 7). This bit provides a one bit extension of the ModRM.reg field in 64-bit mode, permitting access to all 16 YMM/XMM and GPR registers. In 32-bit protected and compatibility modes, this bit must be set to 1. This bit is the bit-inverted equivalent of the REX.R bit.

X Bit (Prefix Byte 1, Bit 6). This bit provides a one bit extension of the SIB.index field in 64-bit mode, permitting access to 16 YMM/XMM and GPR registers. In 32-bit protected and compatibility modes, this bit must be set to 1. This bit is the bit-inverted equivalent of the REX.X bit.

B Bit (Prefix Byte 1, Bit 5). This bit provides a one-bit extension of either the ModRM.r/m field to specify a GPR or XMM register or to the SIB base field to specify a GPR. This permits access to 16 registers. In 32-bit protected and compatibility modes, this bit is ignored. This bit is the bit-inverted equivalent of the REX.B bit and is available only in the 3-byte prefix format.

mmmmm (Prefix Byte 1, Bits 4–0. A five bit field encoding a one- or two-byte opcode prefix.

Prefix Byte 2

Byte 2 of the three-byte prefix has four fields.

W Bit (Prefix Byte 2, Bit 7). The meaning of the W bit is opcode specific. This bit toggles source operand order or is ignored, depending upon the opcode.

vvvv (Prefix Byte 2, Bits 6–3). Encodes a source XMM or YMM register in inverted 1s complement form.

L (Prefix Byte 2, Bit 2). If L is 0, encodes a vector length of 128-bits or indicates scalar operands; if L is 1, the vector length is 256-bits. The register operands for a given instruction are either all 128-bit XMM registers or all 256-bit YMM registers.

pp (Prefix Byte 2, Bits 1–0). Specifies an implied 66, F2, or F3 opcode extension as defined by the following table.

Table 1-1. VEX.pp Prefix Mapping

pp	Implied Prefix
00b	None
01b	66h
10b	F3h
11b	F2h

1.2 Opcode Byte

The format of the opcode byte is shown in Figure 1-3. For most instructions, the operand element size (OES) is specified by the two least-significant opcode bits, as shown in Table 1-2.

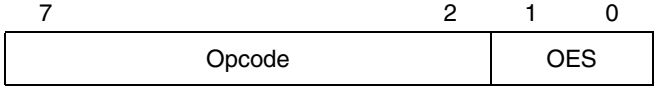


Figure 1-3. Opcode Byte Format

Table 1-2. Operand Element Size—OES

Opcode.OES	Integer Operation	Floating-Point Operation
00	Byte	PS
01	Word	PD
10	Doubleword	SS
11	Quadword	SD

1.3 Destination XMM registers

The destination of XOP and FMA4 instructions may be a 128-bit XMM register or a 256-bit YMM register. When a 128-bit result is written to a destination XMM register, the upper 128 bits of the corresponding YMM register are cleared.

1.4 Four-Operand Instructions

Some new instructions require three input operands and one destination register. This is accomplished by using the Prefix.vvvv field and *Imm8*[7:4] along with the MODRM.reg and MODRM.r/m fields.

VPCMOV is an example of a four operand instruction:

VPCMOV *dest, src1, src2, src3; dest = (src1 & src3) | (src2 & ~src3)*

The first operand is the destination operand and is an XMM or YMM register addressed by the ModRM.reg field.

The second, third and fourth operands are sources. The first source operand is an XMM register specified by the vvvv field. The second and third source operands are specified by the MODRM.r/m and Imm8[7:4] fields, respectively, when VEX.W is set to 0. The FMA4, VPCMOV and VPPERM instructions provide the option of swapping the second and third source operands by setting W to 1, as shown in Table 1-3. This allows either the second data operand or the control operand to be memory based.

Table 1-3. Operand Configurations for FMA4,V PCMOV and VPPERM Instructions

XOP.W	dest	src1	src2	src3
0	ModRM.reg	VEX/XOP.vvvv	modrm.r/m	imm8[7:4]
1	ModRM.reg	VEX/XOP.vvvv	imm8[7:4]	ModRM.r/m

1.5 Three-Operand Instructions

Some instructions have two source operands and a destination operand.

VPROTB is an example of a three operand instruction:

VPROTB dest, src, count dest = src << | >> count

The first operand is the destination operand, and is an XMM register addressed by the ModRM.reg field. The second and third operands are source operands. One source operand is an XMM register addressed by the XOP.vvvv field, the other source operand is an XMM register or memory operand addressed by the ModRM.r/m field.

For certain instructions, in the three-operand format the XOP.W bit determines which source operand is specified by which operand field, as shown in Table 1-4.

Table 1-4. Operand Configurations for Three Operand Instructions

VEX.W	dest	src	count
0	ModRM.reg	ModRM.r/m	VEX.vvvv
1	ModRM.reg	VEX.vvvv	ModRM.r/m

1.6 Two Operand Instructions

Two-operand instructions use the normal ModRM-based operand assignment. For most instructions, the first operand is the destination, addressed by the ModRM.reg field and the second operand is either an XMM or YMM register or a memory operand, as determined by the ModRM.mod field. For all of these instructions, the XOP.vvvv field is not applicable and must be set to 1111b.

The VFRCZPD instruction is an example of a two operand instruction.

VFRCZPD *xmm1*, *xmm2*/*mem128*

1.7 XOP Integer Multiply (Add) and Accumulate Instructions

The multiply and accumulate and multiply, add and accumulate instructions operate on and produce packed signed integer values. These instructions allow the accumulation of results from (possibly) many iterations of similar operations without a separate intermediate addition operation to update the accumulator register.

1.7.1 Saturation

Some instructions limit the result of an operation to the maximum or minimum value representable by the data type of the destination—an operation known as *saturation*. Many of the integer multiply and accumulate instructions saturate the cumulative results of the multiplication and addition (accumulation) operations before writing the final results to the destination (accumulator) register.

Note, however, that not all multiply and accumulate instructions saturate results. (For further discussion of saturation, see the *AMD64 Architecture Programmer's Manual Volume 1: Application Programming*, order# 24592.)

1.7.2 Multiply and Accumulate Instructions

The operation of a typical XOP integer multiply and accumulate instruction is shown in Figure 1-4 on page 32.

The multiply and accumulate instructions operate on and produce packed signed integer values. These instructions first multiply the value in the first source operand by the corresponding value in the second source operand. Each signed integer product is then added to the corresponding value in the third source operand, which is the accumulator and is identical to the destination operand. The results may or may not be saturated prior to being written to the destination register, depending on the instruction.

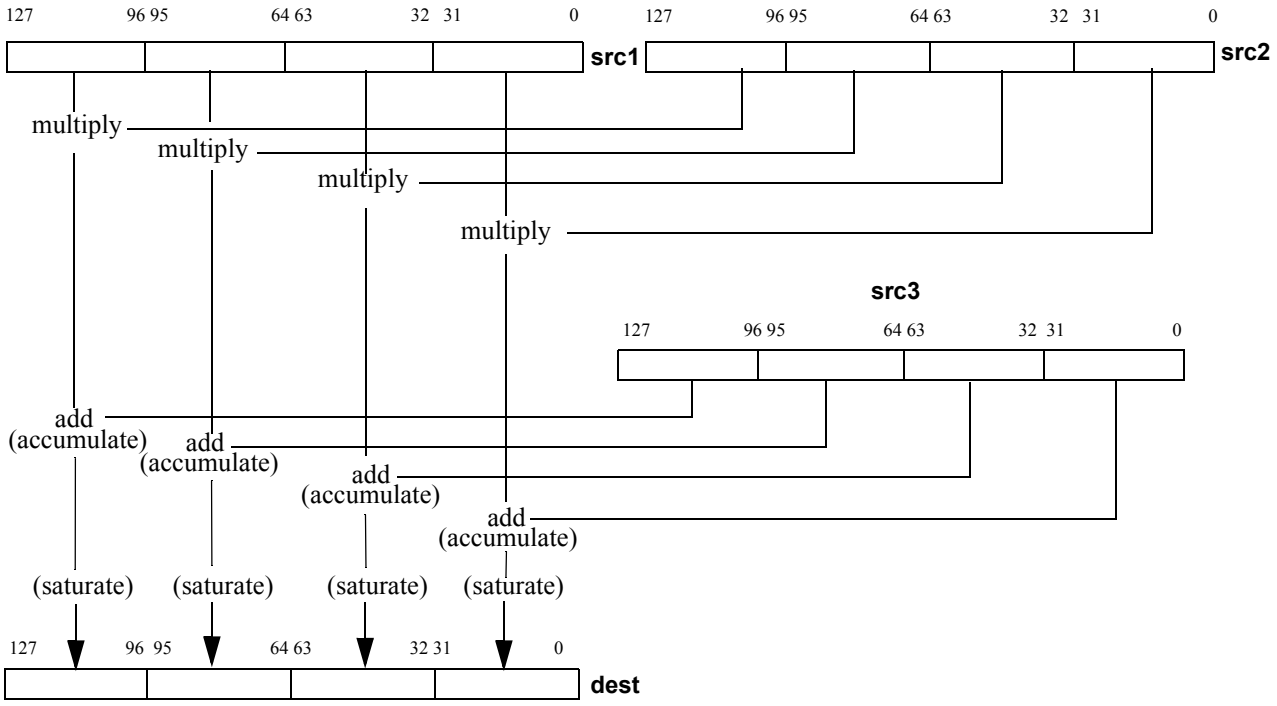


Figure 1-4. Operation of Multiply and Accumulate Instructions

The XOP instruction extensions provide the following integer multiply and accumulate instructions.

- VPMACSSWW—Packed Multiply Accumulate Signed Word to Signed Word with Saturation
- VPMACSWW—Packed Multiply Accumulate Signed Word to Signed Word
- VPMACSSWD—Packed Multiply Accumulate Signed Word to Signed Doubleword with Saturation
- VPMACSWD—Packed Multiply Accumulate Signed Word to Signed Doubleword
- VPMACSSDD—Packed Multiply Accumulate Signed Doubleword to Signed Doubleword with Saturation
- VPMACSDD—Packed Multiply Accumulate Signed Doubleword to Signed Doubleword
- VPMACSSDQL—Packed Multiply Accumulate Signed Low Doubleword to Signed Quadword with Saturation
- VPMACSSDQH—Packed Multiply Accumulate Signed High Doubleword to Signed Quadword with Saturation
- VPMACSDQL—Packed Multiply Accumulate Signed Low Doubleword to Signed Quadword
- VPMACSDQH—Packed Multiply Accumulate Signed High Doubleword to Signed Quadword

1.7.3 Integer Multiply, Add and Accumulate Instructions

The operation of the multiply, add and accumulate instructions is illustrated in Figure 1-5.

The multiply, add and accumulate instructions first multiply each packed signed integer value in the first source operand by the corresponding packed signed integer value in the second source operand. The odd and even adjacent resulting products are then added. Each resulting sum is then added to the corresponding packed signed integer value in the third source operand.

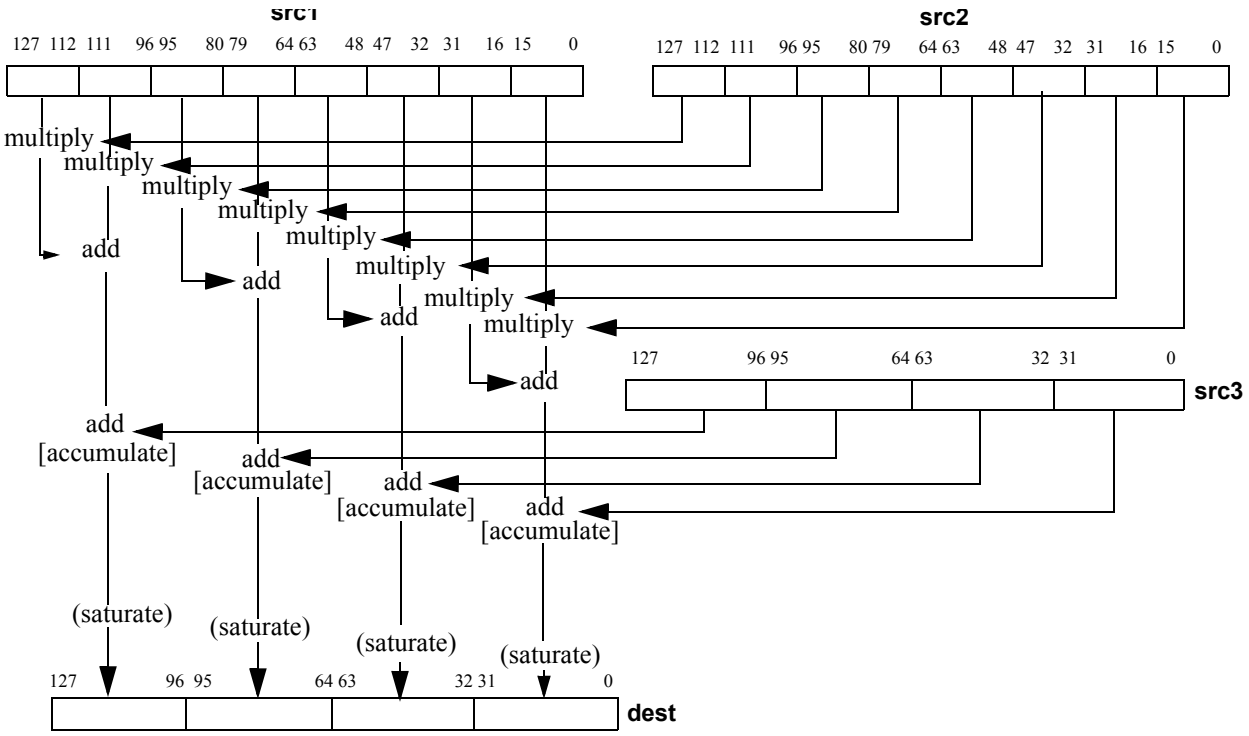


Figure 1-5. Operation of Multiply, Add and Accumulate Instructions

The XOP instruction set provides the following integer multiply, add and accumulate instructions.

- VPMADCSSWD—Packed Multiply Add and Accumulate Signed Word to Signed Doubleword with Saturation
- VPMADCSWD—Packed Multiply Add and Accumulate Signed Word to Signed Doubleword

1.8 Packed Integer Horizontal Add and Subtract

The packed horizontal add and subtract signed byte instructions successively add adjacent pairs of signed integer values from the source XMM register or 128-bit memory operand and pack the (sign-extended) integer result of each addition in the destination.

- VPHADDBW—Packed Horizontal Add Signed Byte to Signed Word

- VPHADDBD—Packed Horizontal Add Signed Byte to Signed Doubleword
- VPHADDBQ—Packed Horizontal Add Signed Byte to Signed Quadword
- VPHADDDQ—Packed Horizontal Add Signed Doubleword to Signed Quadword
- VPHADDUBW—Packed Horizontal Add Unsigned Byte to Word
- VPHADDUBD—Packed Horizontal Add Unsigned Byte to Doubleword
- VPHADDUBQ—Packed Horizontal Add Unsigned Byte to Quadword
- VPHADDUWD—Packed Horizontal Add Unsigned Word to Doubleword
- VPHADDUWQ—Packed Horizontal Add Unsigned Word to Quadword
- VPHADDUDQ—Packed Horizontal Add Unsigned Doubleword to Quadword
- VPHADDWD—Packed Horizontal Add Signed Word to Signed Doubleword
- VPHADDWQ—Packed Horizontal Add Signed Word to Signed Quadword
- VPHSUBBW—Packed Horizontal Subtract Signed Byte to Signed Word
- VPHSUBWD—Packed Horizontal Subtract Signed Word to Signed Doubleword
- VPHSUBDQ—Packed Horizontal Subtract Signed Doubleword to Signed Quadword

1.9 Vector Conditional Moves

XOP instructions include vector conditional move instructions:

- VPCMOV—Vector Conditional Moves
- VPPERM—Packed Permute Bytes

The VPCMOV instruction implements the C/C++ language ternary ‘?’ operator a bit level. This instruction operates on individual bits and requires a bitwise predicate in one XMM or YMM register and the two source operands in two more XMM or YMM registers.

The VPPERM instruction performs vector permutation on a packed array of 32 bytes composed of two 16-byte input operands. The VPPERM instruction replaces each destination byte with 00h, FFh, or one of the 32 bytes of the packed array. A byte selected from the array may have an additional operation such as NOT or bit reversal applied to it, before it is written to the destination. The action for each destination byte is determined by a corresponding control byte. The VPPERM instruction allows either the second 16-byte input array or the control array to be memory based, per the XOP.W bit.

1.10 Packed Integer Rotates and Shifts

These instructions rotate/shift the elements of the vector in the first source YMM or 128-bit memory operand by the amount specified by a control byte. The rotates and shifts differ in the way they handle the control byte.

1.10.1 Packed Integer Shifts

The packed integer shift instructions shift each element of the vector in the first source XMM or 128-bit memory operand by the amount specified by a control byte contained in the least significant byte of the corresponding element of the second source operand. The result of each shift operation is returned in the destination XMM register. This allows load-and-shift from memory operations, with either the source operand or the shift-count operand being memory-based, as indicated by the XOP.W bit. The XOP instruction set provides the following packed integer shift instructions:

- VPSHLB—Packed Shift Logical Bytes
- VPSHLW—Packed Shift Logical Words
- VPSHLD—Packed Shift Logical Doublewords
- VPSHLQ—Packed Shift Logical Quadwords
- VPSHAB—Packed Shift Arithmetic Bytes
- VPSHAW—Packed Shift Arithmetic Words
- VPSHAD—Packed Shift Arithmetic Doublewords
- VPSHAQ—Packed Shift Arithmetic Quadwords

1.10.2 Packed Integer Rotate

There are two variants of the packed integer rotate instructions. The first is identical to that described above (see “Packed Integer Shifts”). In the second variant, the control byte is supplied as an 8-bit immediate operand that specifies a single rotate amount for every element in the first source operand. The XOP instruction set provides the following packed integer rotate instructions:

- VPROTB—Packed Rotate Bytes
- VPROTW—Packed Rotate Words
- VPROTD—Packed Rotate Doublewords
- VPROTQ—Packed Rotate Quadwords

1.11 Packed Integer Comparison and Predicate Generation

The XOP comparison instructions compare packed integer values in the first source XMM register with corresponding packed integer values in the second source XMM register or 128-bit memory. The type of comparison is specified by the immediate-byte operand. The resulting predicate is placed in the destination XMM register. If the condition is true, all bits in the corresponding field in the destination register are set to 1s; otherwise all bits in the field are set to 0s.

Table 1-5. Immediate Operand Values for Unsigned Vector Comparison Operations

Immediate Operand Byte		Comparison Operation
Bits 7:3	Bits 2:0	

00000b	000b	Less Than
	001b	Less Than or Equal
	010b	Greater Than
	011b	Greater Than or Equal
	100b	Equal
	101b	Not Equal
	110b	False
	111b	True

The integer comparison and predicate generation instructions compare corresponding packed signed or unsigned bytes in the first and second source operands and write the result of each comparison in the corresponding element of the destination. The result of each comparison is a value of all 1s (TRUE) or all 0s (FALSE). The type of comparison is specified by the three low-order bits of the immediate-byte operand. The XOP instruction set provides the following integer comparison instructions.

- VPCOMUB—Compare Vector Unsigned Bytes
- VPCOMUW—Compare Vector Unsigned Words
- VPCOMUD—Compare Vector Unsigned Doublewords
- VPCOMUQ—Compare Vector Unsigned Quadwords
- VPCOMB—Compare Vector Signed Bytes
- VPCOMW—Compare Vector Signed Words
- VPCOMD—Compare Vector Signed Doublewords
- VPCOMQ—Compare Vector Signed Quadwords

1.12 Fraction Extract

The fraction extract instructions isolate the fractional portions of vector or scalar floating point operands. The result of _PD and _PS instructions is a vector of integer numbers. The result of _SD and _SS instructions is always a scalar integer number. XOP provides the following fraction extract instructions:

- VFRCZPD—Extract Fraction Packed Double-Precision Floating-Point
- VFRCZPS—Extract Fraction Packed Single-Precision Floating-Point
- VFRCZSD— Extract Fraction Scalar Double-Precision Floating-Point
- VFRCZSS— Extract Fraction Scalar Single-Precision Floating Point

The VFRCZPD and VFRCZPS instructions extract the fractional portions of a vector of double-/single-precision floating-point values in an XMM or YMM register or a 128- or 256-bit memory location and write the results in the corresponding field in the destination register.

The VFRCZSS and VFRCZSD instructions extract the fractional portion of the single-/double-precision scalar floating-point value in an XMM register or 32- or 64-bit memory location and writes the result in the lower element of the destination register. The upper elements of the destination XMM register are unaffected by the operation, while the upper 128 bits of the corresponding YMM register are cleared to zeros.

2 AMD XOP and FMA4 Instructions

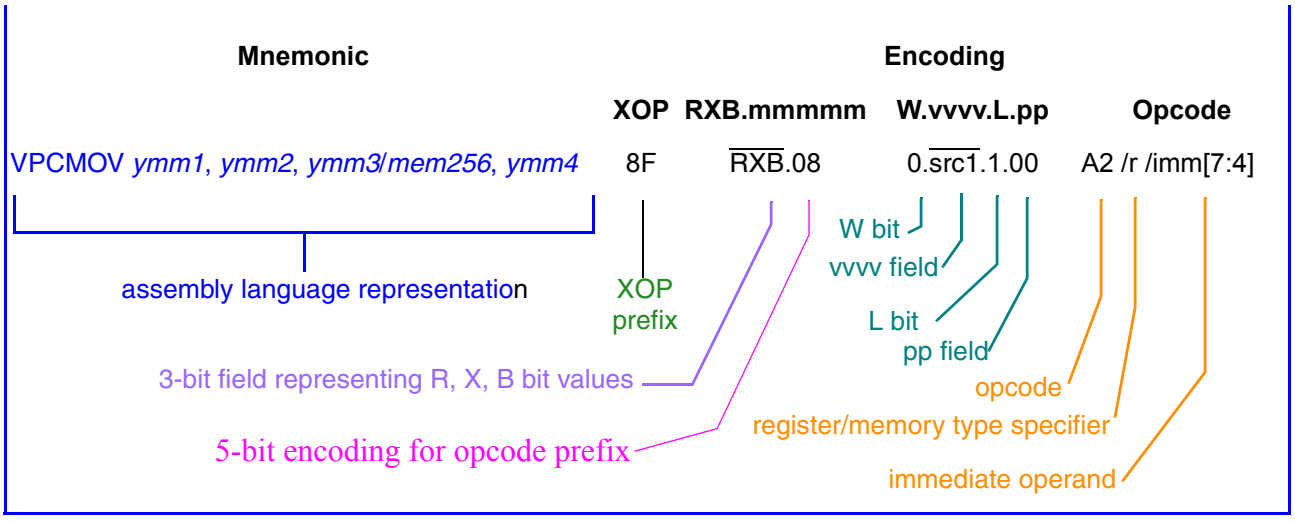
The following section describes the complete set of XOP 128-media instructions. Instructions are listed alphabetically by mnemonic.

2.1 Notation

The notation used to denote the size and type of source and destination operands in both mnemonics and opcodes is discussed in detail in Section 2.5, "Notation," on page 37 in the *AMD64 Architecture Programmer's Manual Volume 3: General Purpose and System Instructions*. Mnemonic conventions that are idiosyncratic to the XOP instruction set have been included in *Chapter 1, "New 128-Bit and 256-Bit Instructions"*, in this document.

2.1.1 Opcode Syntax

Opcode specification for the XOP and FMA4 instruction sets, with their two, three and four operand syntax, requires a slightly different approach from that used to specify the opcodes for previous generation 64- and 128-bit instructions (documented in the *AMD64 Architecture Programmer's Manual Volume 4: 128-Bit Media Instructions*, order# 26568, and *AMD64 Architecture Programmer's Manual Volume 5: 64-Bit Media and x87 Floating-Point Instructions*, order# 26569). In the following pages, opcodes are specified using the order of fields and bits as they occur in a complete opcode specification as outlined in Section 1.1, "New Instruction Format," on page 25. The following opcode specification is typical:



Most of the terms and symbols used in the following pages are defined in Section 1.1, "New Instruction Format," on page 25. The following notations and convention are used in this volume, in addition to the opcode notational conventions specified in Section 2.5.2, "Opcode Syntax," on page 39

in the *AMD64 Architecture Programmer's Manual Volume 3: General Purpose and System Instructions*:

cntr

Control bits (for comparison instructions); immediate byte bits 3–0.

is4

Destination register specifier; immediate byte bits 7:4.

$\overline{\text{RXB}}$

Bit field specifying the R, X and B bit values. Specified in one's complement form.

VEX.W

The meaning of the W bit is opcode specific. This bit toggles source operand order or is ignored, depending upon the opcode.

VEX.L

Vector length specifier

VEX.vvvv

Additional operand register specifier.

XOP

Indicates the XOP prefix byte (8Fh).

2.2 Operand Specification

The packed values in a operand are numbered starting with 0, which is considered to be even-numbered.

2.3 Instruction Reference

VFMADDPD Multiply and Add Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value in the first source by the corresponding packed double-precision floating-point value in the second source, then adds each product to the corresponding packed double-precision floating-point value in the third source and writes the rounded results to the destination register.

The VFMADDPD instruction requires four operands:

VFMADDPD dest, src1, src2, src3 *dest = (src1* src2) + src3*

The 128-bit version multiplies each of the two double-precision values in the first source XMM register by its corresponding double-precision value in the second source. It then adds each intermediate product to the corresponding double-precision value in the third source and places the result in the destination XMM register.

The 256-bit version multiplies each of the four double-precision values in the first source YMM register by its corresponding double-precision value in the second source. It then adds each product to the corresponding double-precision value in the third source and places the results in the destination YMM register.

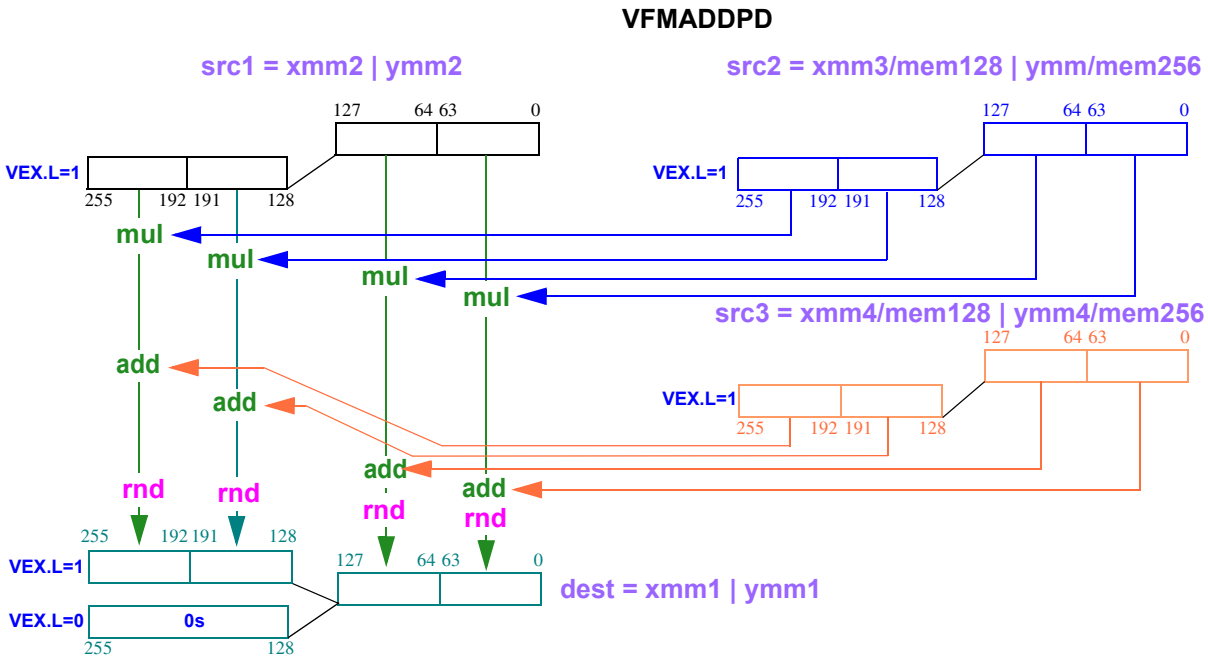
If VEX.W is 0, the second source is either a register or memory and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by the rounding mode in MXCSR.

The VFMADDPD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	69 /r /is4
VFMADDPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	69 /r /is4
VFMADDPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	69 /r /is4
VFMADDPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	69 /r /is4



Related Instructions

VFMADDPS, VFMADDSD, VFMADDSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMADDPS Multiply and Add Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value in the first source by the corresponding single-precision floating-point value in the second source, then adds each product to the corresponding packed single-precision floating-point value in the third source and writes the rounded results to the destination register.

The VFMADDPS instruction requires four operands:

VFMADDPS dest, src1, src2, src3 *dest = src1* src2 + src3*

The 128-bit version multiplies each of the four single-precision values in the first source XMM register by its corresponding single-precision value in the second source. It then adds each product to the corresponding single-precision value in the third source and places the results in the destination XMM register.

The 256-bit version multiplies each of the eight single-precision values in the first source YMM register by its corresponding double-precision value in the second source. It then adds each product to the corresponding double-precision value in the third source and places the results in the destination YMM register.

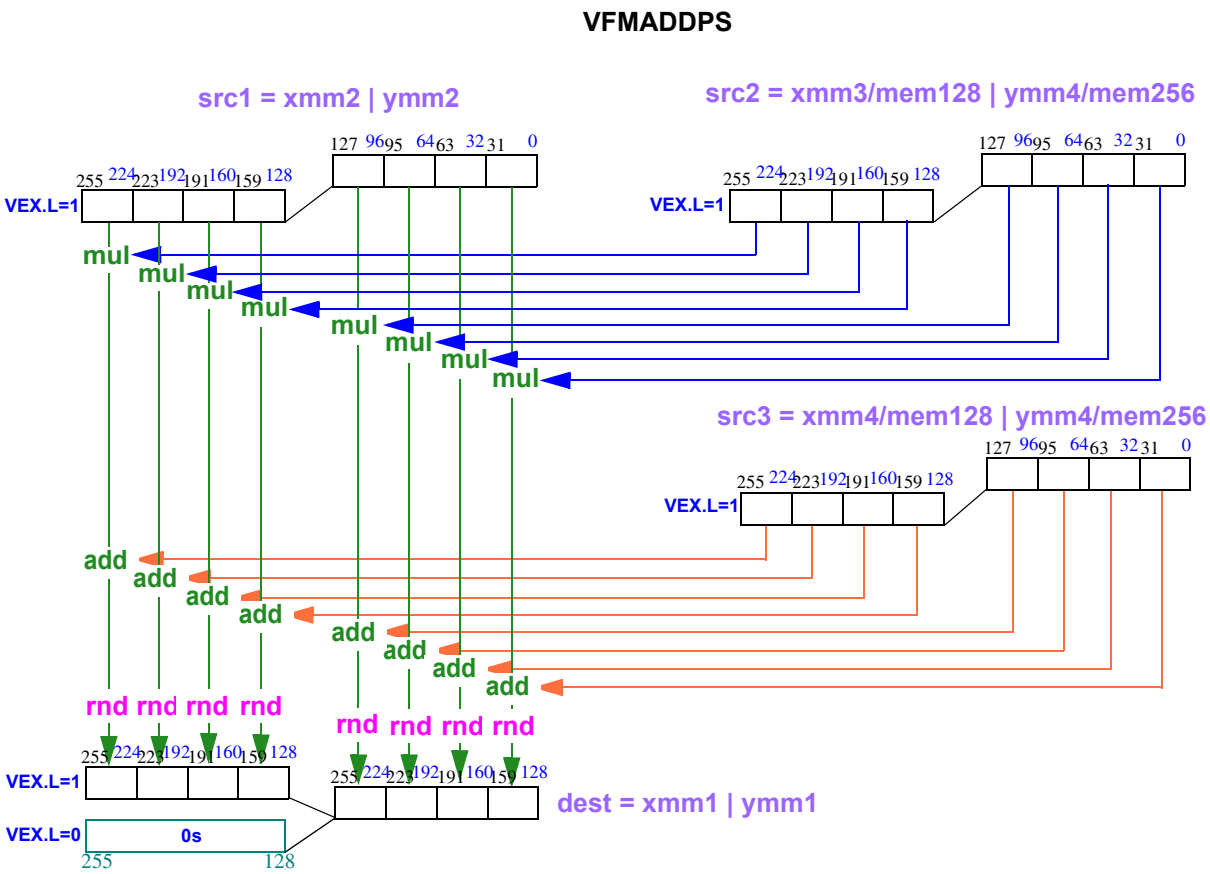
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by the rounding mode in MXCSR.

The VFMADDPS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	68 /r /is4
VFMADDPS *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	68 /r /is4
VFMADDPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	68 /r /is4
VFMADDPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	68 /r /is4



Related Instructions

VFMADDPD, VFMADDSD, VFMADDSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.
SIMD Floating-Point Exceptions				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMADDSD Multiply and Add Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value in the low-order quadword of the first source by the double-precision floating-point value in the low-order quadword of the second source, then adds the product to the double-precision floating-point value in the low-order quadword of the third source. The low-order quadword result is written to the destination.

The VFMADDSD instruction requires four operands:

VFMADDSD dest, src1, src2, src3 *dest = src1* src2 + src3*

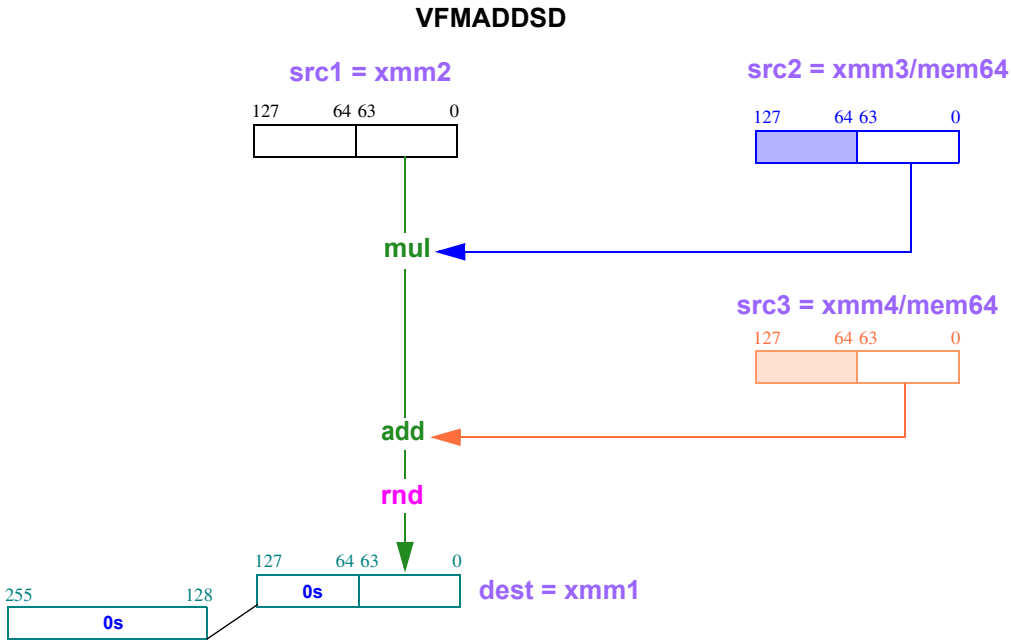
If VEX.W is 0, the second source is either a register or 64-bit memory location and the third source is a register. If VEX.W is 1, the second source is a a register and the third source is a register or 64-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, the upper quadword of the destination register (bits 64–127) and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The intermediate product is not rounded; the infinitely precise product is used in the addition. The result of the addition is rounded, as specified by the rounding mode in MXCSR.

The VFMADDSD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	6B /r /is4
VFMADDSD *xmm1, xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	6B /r /is4



Related Instructions

VFMADDPD, VFMADDPS, VFMADDSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMADDSS Multiply and Add Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value in the low-order doubleword of the first source by the low-order single-precision floating-point value in the second source, then adds the product to the low-order single-precision floating-point value in the third source. The low-order doubleword result is written to the destination.

The VFMADDSS instruction requires four operands:

VFMADDSS dest, src1, src2, src3 *dest = src1* src2 + src3*

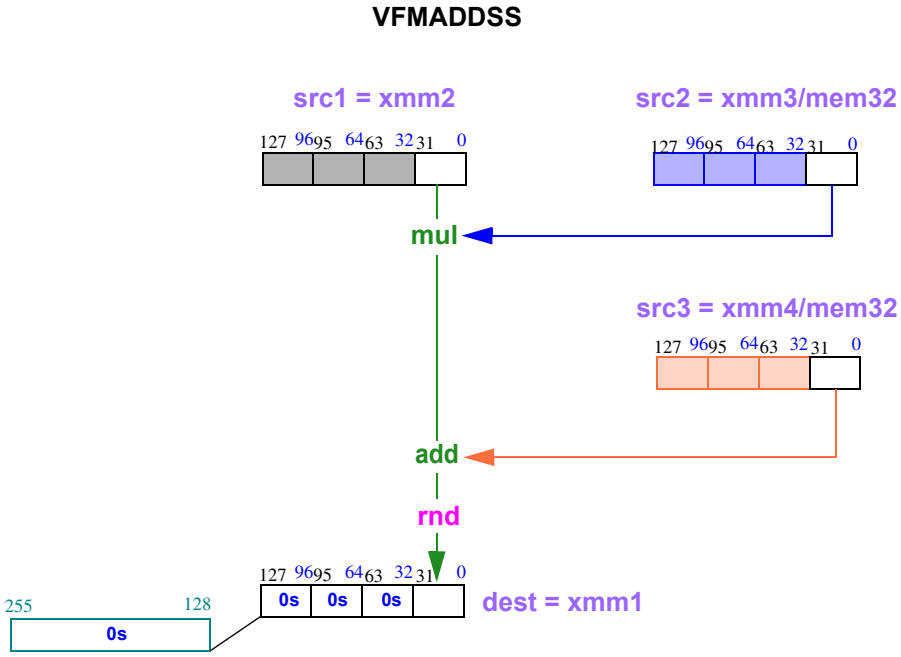
If VEX.W is 0, the second source is either a register or 32-bit memory location and the third source is a register. If VEX.W is 1, the second source is a a register and the third source is a register or 32-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, the upper three doublewords of the destination register (bits 32–127) and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The intermediate product is not rounded; the infinitely precise product is used in the addition. The result of the addition is rounded, as specified by the rounding mode in MXCSR.

The VFMADDSS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	6A /r /is4
VFMADDSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	6A /r /is4



Related Instructions

VFMADDPD, VFMADDPS, VFMADDSD

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMADDSUBPD Multiply with Alternating Add/Subtract of Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value in the first source by the corresponding packed double-precision floating-point value in the second source. Adds each odd-numbered double-precision floating-point value in the third source to the corresponding infinite-precision intermediate product; subtracts each even-numbered double-precision floating-point value in the third source from its corresponding product. Finally, writes the results to the destination.

The 128-bit version multiplies each of the two double-precision floating-point values in the first source by its corresponding value in the second source. The low-order double-precision floating-point value in the third source is subtracted from its corresponding infinite-precision product and the high-order double-precision floating-point value in the third source is added to its corresponding product. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of the four double-precision floating-point values in first source by its corresponding double-precision value in the second source. The even-numbered double-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the odd-numbered double-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is an XMM register or a YMM register, depending on the vector size, as determined by VEX.L.

If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

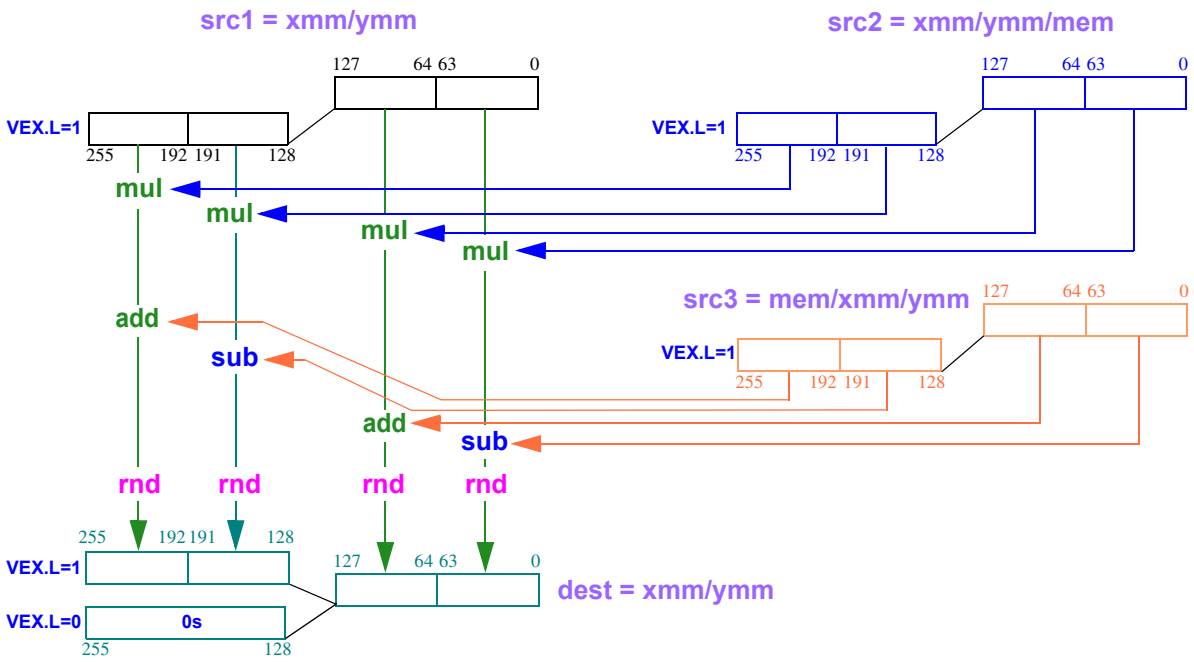
The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When writing to a 128-bit XMM destination register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the final addition and subtraction operation(s). The results of the addition and subtraction operations are rounded, as specified by the rounding mode in MXCSR.

The VFMADDSUBPD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	5D /r /is4
VFMADDSUBPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	5D /r /is4
VFMADDSUBPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	5D /r /is4
VFMADDSUBPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	5D /r /is4

VFMADDSUBPD



Related Instructions

VFMADDSUBPD, VFMSUBADDPD, VFMSUBADDPS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)			X	A source operand was an SNaN value.
			X	+/–zero was multiplied by +/–infinity
			X	+infinity was added to –infinity
			X	+infinity was subtracted from +infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMADDSUBPS Multiply with Alternating Add/Subtract of Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value in the first source by the corresponding packed single-precision floating-point value in the second source. Adds each odd-numbered single-precision floating-point value in the third source to the corresponding infinite-precision intermediate product; subtracts each even-numbered single-precision floating-point value in the third source from its corresponding product. Finally, writes the results to the destination.

The 128-bit version multiplies each of the four single-precision floating-point values in first source by its corresponding single-precision value in the second source. The even-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the odd-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of the eight single-precision floating-point values in first source by its corresponding single-precision value in the second source. The even-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the odd-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is either an XMM register or a YMM register, depending on the vector size, as determined by VEX.L.

If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

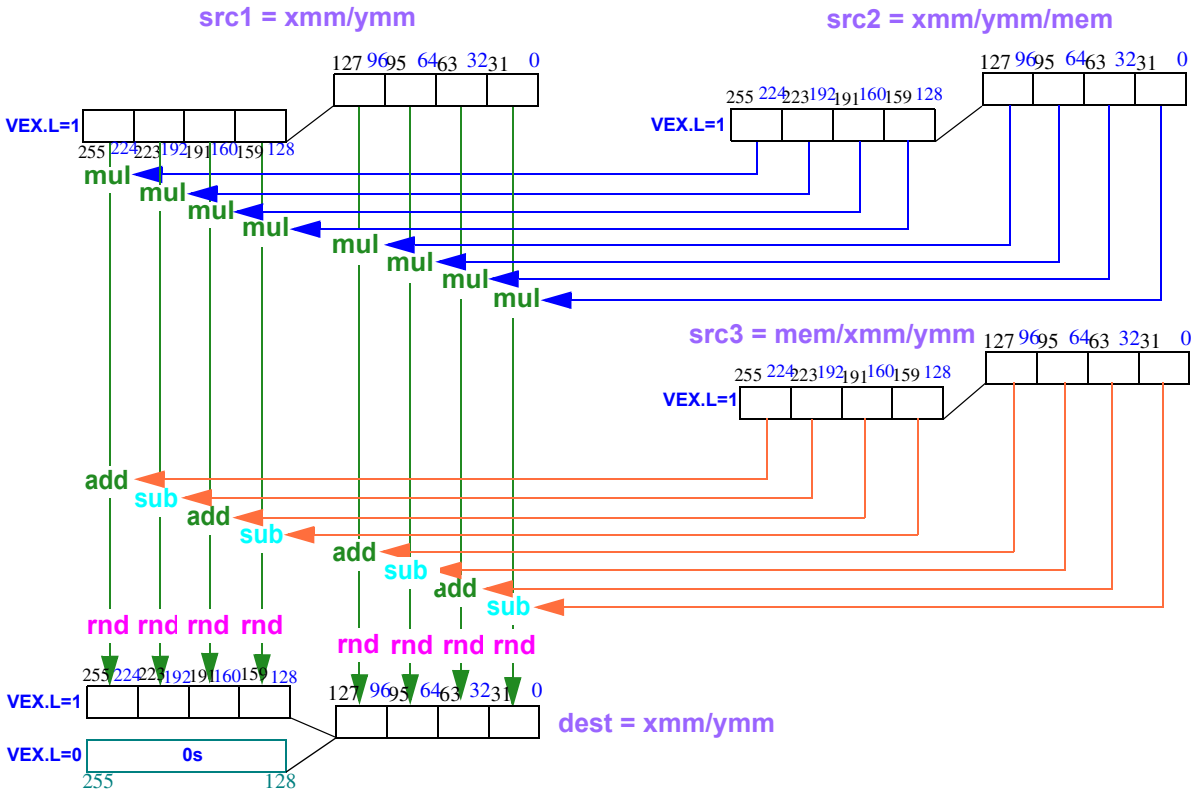
The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When writing to a 128-bit XMM destination register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise intermediate products are used in the addition and subtraction operations. The results of the addition and subtraction operations are rounded, as specified by the rounding mode in MXCSR.

The VFMADDSUBPS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMADDSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	5C /r /is4
VFMADDSUBPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	5C /r /is4
VFMADDSUBPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	5C /r /is4
VFMADDSUBPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	5C /r /is4

VFMADDSUBPS



Related Instructions

VFMADDSUBPD, VFMSUBADDPD, VFMSUBADDPS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)			X	A source operand was an SNaN value.
			X	+/–zero was multiplied by +/–infinity
			X	+infinity was added to –infinity
			X	+infinity was subtracted from +infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMSUBADDPD Multiply with Alternating Subtract/Add of Packed Double-Precision Floating-Point

Multiplies each packed double-precision floating-point value in the first source by the corresponding packed double-precision floating-point value in the second source. Adds each even-numbered double-precision floating-point value in the third source to the corresponding infinite-precision intermediate product; subtracts each odd-numbered double-precision floating-point value in the third source from its corresponding product. Finally, writes the results to the destination.

The 128-bit version multiplies each of the two double-precision floating-point values in the first source by its corresponding value in the second source. The high-order double-precision floating-point value in the third source is subtracted from its corresponding infinite-precision product and the low-order double-precision floating-point value in the third source is added to its corresponding product. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of the four double-precision floating-point values in first source by its corresponding double-precision value in the second source. The odd-numbered double-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the even-numbered double-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is either an XMM register or a YMM register, depending on the vector size, as determined by VEX.L.

If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source operand is a register or memory location.

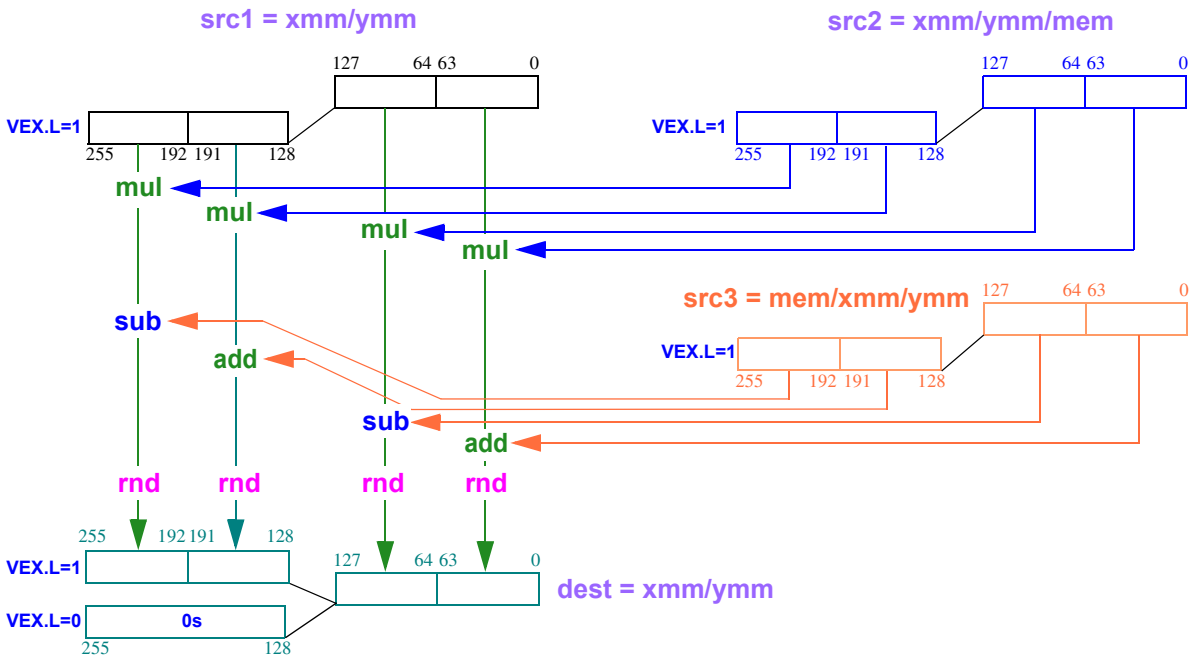
The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When writing to a 128-bit XMM destination register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the two infinitely precise intermediate products are used in the addition. The results of the addition and subtraction operations are rounded, as specified by the rounding mode in MXCSR.

The VFMSUBADDPD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	5F /r /is4
VFMSUBADDPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	5F /r /is4
VFMSUBADDPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	5F /r /is4
VFMSUBADDPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	5F /r /is4

VFMSUBADDPD



Related Instructions

VFMADDSUBPD, VFMADDSUBPS, VFMSUBADDPS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
			X	+infinity was subtracted from +infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMSUBADDPS Multiply with Alternating Subtract/Add of Packed Single-Precision Floating-Point

Multiplies each packed single-precision floating-point value in the first source by the corresponding packed single-precision floating-point value in the second source. Adds each even-numbered single-precision floating-point value in the third source to the corresponding infinite-precision intermediate product; subtracts each odd-numbered single-precision floating-point value in the third source from its corresponding product. Finally, writes the results to the destination.

The 128-bit version multiplies each of the four single-precision floating-point values in first source by its corresponding single-precision value in the second source. The odd-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the even-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The 256-bit version multiplies each of the eight single-precision floating-point values in first source by its corresponding single-precision value in the second source. The odd-numbered single-precision values in the third source are subtracted from their corresponding infinite-precision intermediate products and the even-numbered single-precision values in the third source are added to their corresponding infinite precision intermediate products. The results of these operations are placed in their corresponding positions in the destination.

The first source is either an XMM register or a YMM register, depending on the vector size, as determined by VEX.L.

If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

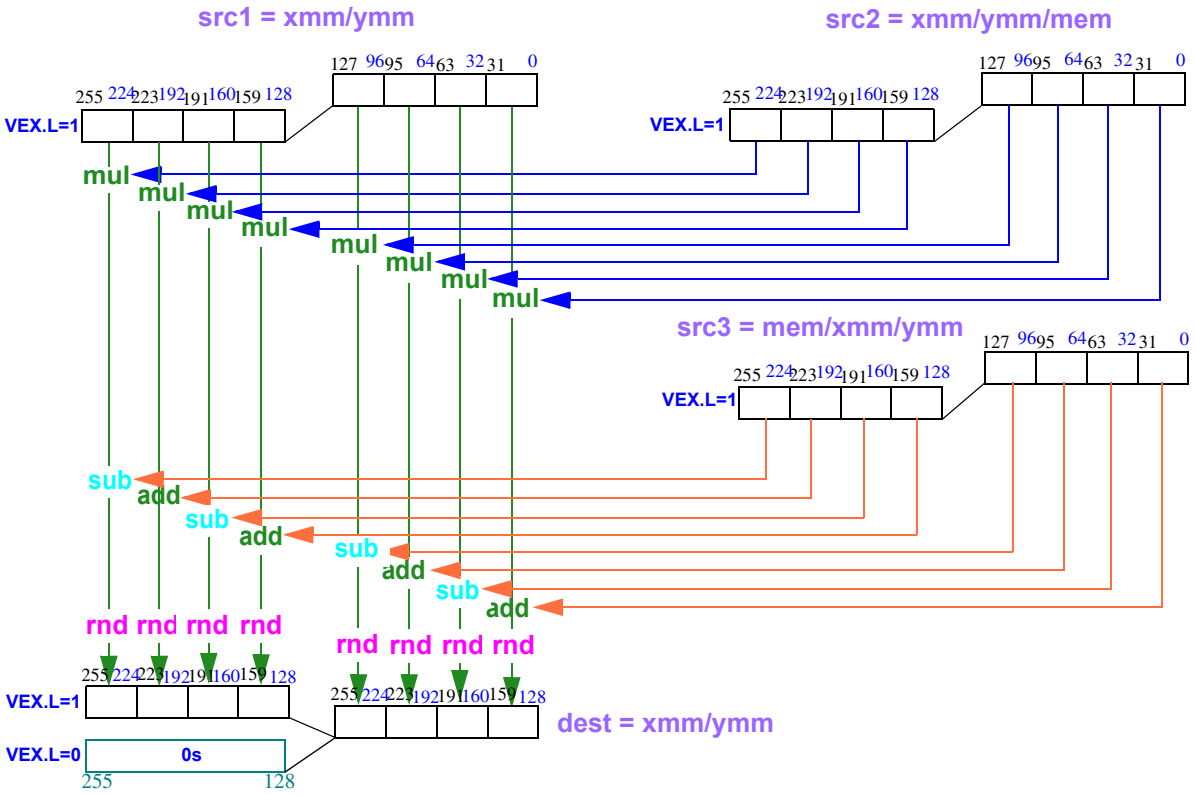
The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When writing to a 128-bit XMM destination register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the additions and subtracts are rounded, as specified by the rounding mode in MXCSR.

The VFMSUBADDPS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	5E /r /is4
VFMSUBADDPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	5E /r /is4
VFMSUBADDPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	5E /r /is4
VFMSUBADDPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	5E /r /is4

VFMSUBADDPS



Related Instructions

VFMADDSUBPD, VFMADDSUBPS, VFMSUBADDPD, VFMSUBADDPS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.

Exception	Real	Virtual 8086	Protected	Cause of Exception
SIMD Floating-Point Exceptions				
Invalid-operation exception (IE)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
			X	+infinity was subtracted from +infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMSUBPD Multiply and Subtract Packed Double-Precision Floating-Point

Multiplies each of the packed double-precision floating-point values in the first source by its corresponding packed double-precision floating-point value in the second source, then subtracts the corresponding packed double-precision floating-point values in the third source from the intermediate products of the multiplication. The results are written to the destination register.

The VFMSUBPD instruction requires four operands:

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

The 128-bit version multiplies two packed double-precision floating-point values in the first source, by their corresponding packed double-precision floating point values in the second source, producing two intermediate products. The two double precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination XMM register.

The 256-bit version multiplies four packed double-precision floating-point values in the first source by their corresponding packed double-precision floating point values in the second source, producing four intermediate products. The four double-precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination YMM register.

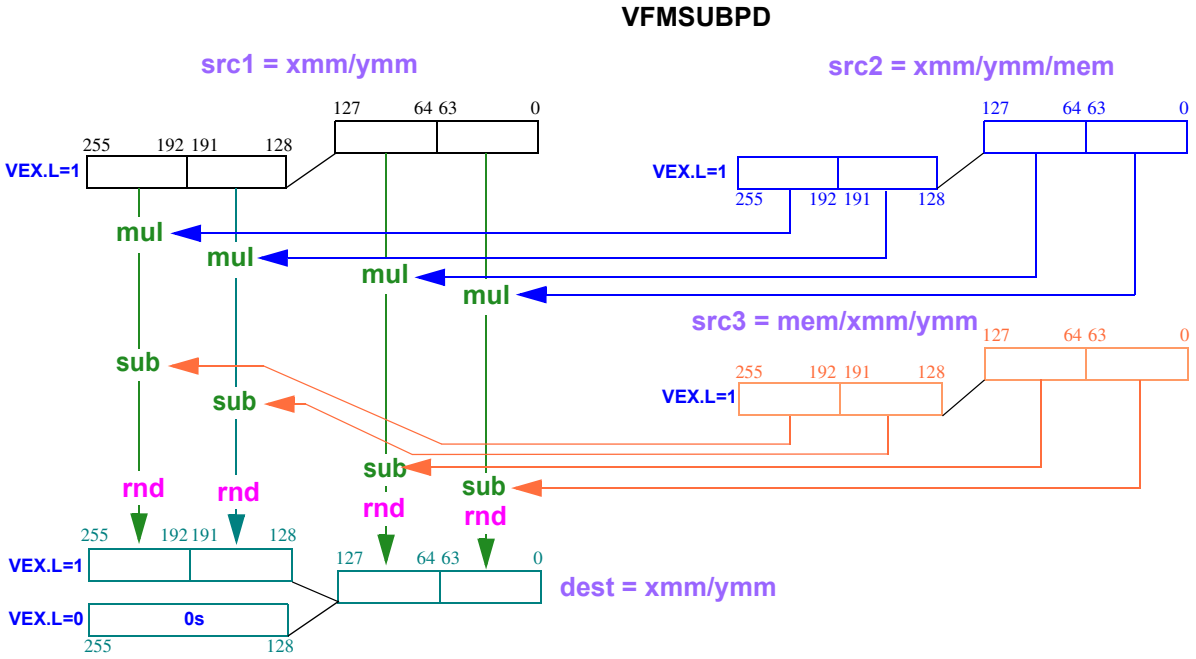
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When writing to a 128-bit XMM destination register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the two infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by the rounding mode in MXCSR.

The VFMSUBPD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	6D /r /is4
VFMSUBPD *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	6D /r /is4
VFMSUBPD *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	6D /r /is4
VFMSUBPD *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	6D /r /is4



Related Instructions

VFMSUBPS, VFMSUBSD, VFMSUBSS

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
			X	-infinity was subtracted from -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMSUBPS Multiply and Subtract Packed Single-Precision Floating-Point

Multiplies each of the packed single-precision floating-point values in the first source by its corresponding packed single-precision floating-point value in the second source, then subtracts the corresponding packed single-precision floating-point values in the third source from the products. The four results are written to the destination register.

The VFMSUBPS instruction requires four operands:

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

The 128-bit version multiplies four packed single-precision floating-point values in the first source by their corresponding packed single-precision floating point values in the second source, producing four intermediate products. The four single-precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination XMM register.

The 256-bit version multiplies eight packed single-precision floating-point values in the first source by their corresponding packed single-precision floating point values in the second source, producing eight intermediate products. The eight single-precision floating-point values in the third source are subtracted from the intermediate products of the multiplication and the remainders are placed in the destination YMM register.

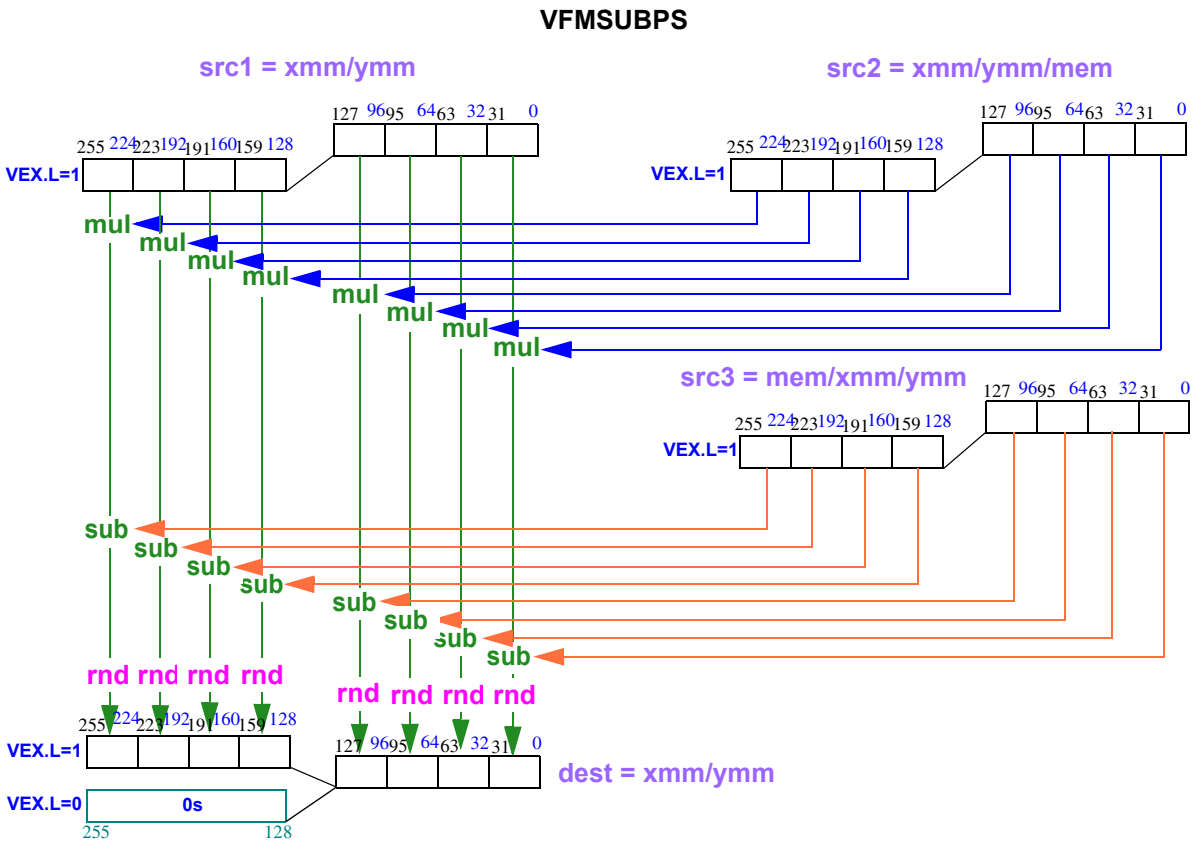
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When writing to a 128-bit XMM destination register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the two infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by the rounding mode in MXCSR.

The VFMSUBPS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	6C /r /is4
VFMSUBPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	6C /r /is4
VFMSUBPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	6C /r /is4
VFMSUBPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	6C /r /is4



Related Instructions

VFMSUBPD, VFMSUBSD, VFMSUBSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
			X	-infinity was subtracted from -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMSUBSD Multiply and Subtract Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value in the low-order quadword of the first source by the double-precision floating-point value in the low-order quadword of the second source, then subtracts the double-precision floating-point value in the low-order quadword of the third source from the intermediate product. The low-order quadword result is written to the destination.

The VFMSUBSD instruction requires four operands:

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

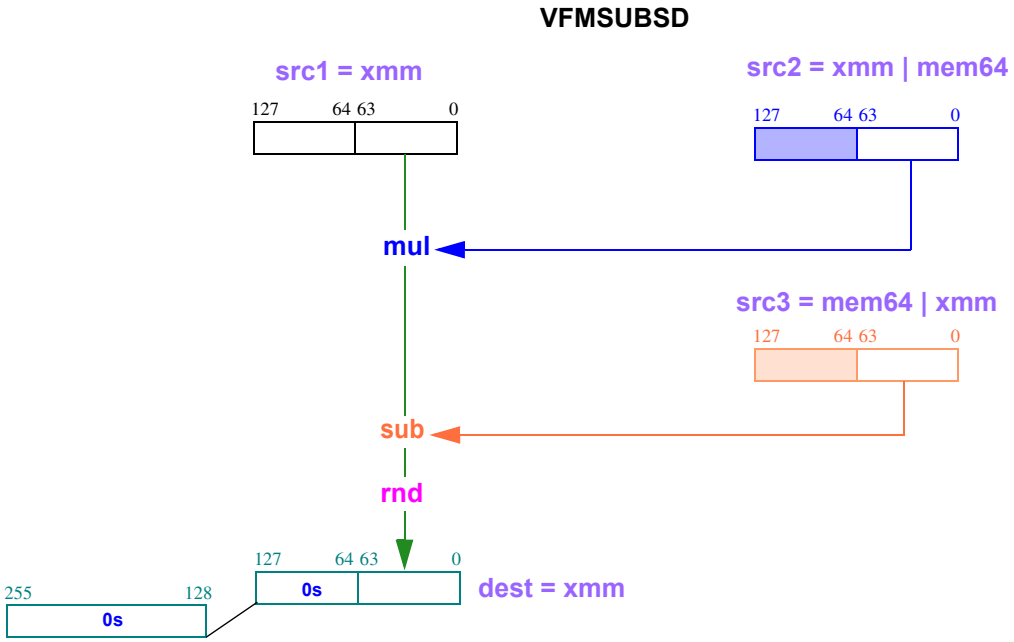
If VEX.W is 0, the second source is either a register or 64-bit memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or 64-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, the upper quadword of the destination register (bits 64–127) and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The intermediate product is not rounded; the infinitely precise product is used in the subtraction. The result of the subtraction is rounded, as specified by the rounding mode in MXCSR.

The VFMSUBSD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	6F /r /is4
VFMSUBSD *xmm1, xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	6F /r /is4



Related Instructions

VFMSUBPD, VFMSUBPS, VFMSUBSS

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
			X	-infinity was subtracted from -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFMSUBSS Multiply and Subtract Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value in the low-order doubleword of the first source by the single-precision floating-point value in the low-order doubleword of the second source, then subtracts the single-precision floating-point value in the low-order doubleword of the third source from the product. The low-order doubleword result is written to the destination.

The VFMSUBSS instruction requires four operands:

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

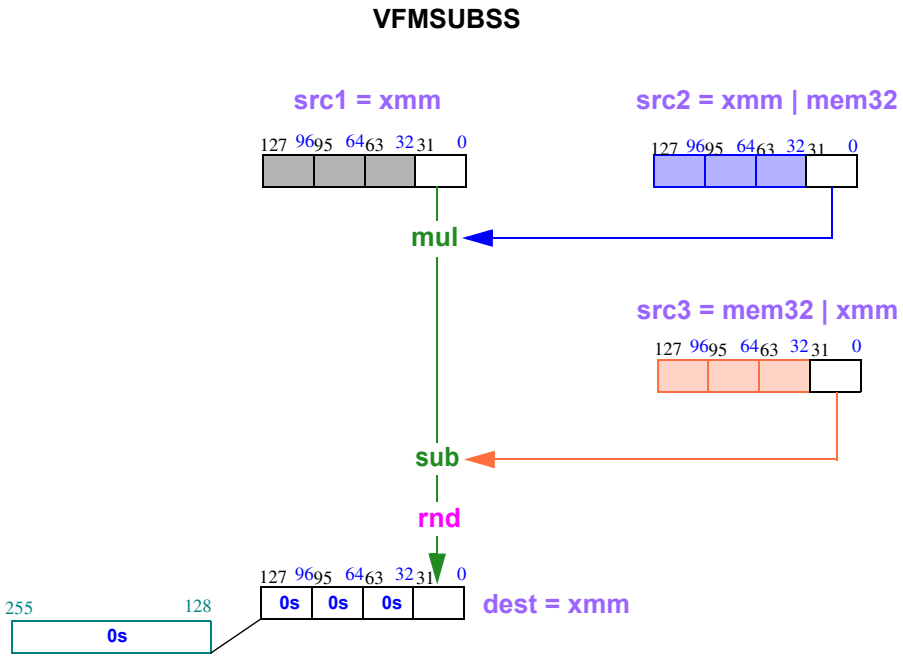
If VEX.W is 0, the second source is either a register or 32-bit memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or 32-bit memory location.

The destination is an XMM register. When the result is written to the destination XMM register, the upper three doublewords of the destination register (bits 32–127) and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The intermediate product is not rounded; the infinitely precise product is used in the subtraction. The result of the subtraction is rounded, as specified by the rounding mode in MXCSR.

The VFMSUBSS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFMSUBSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	6E /r /is4
VFMSUBSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	6E /r /is4



Related Instructions

VFMSUBPD, VFMSUBPS, VFMSUBSD

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/–zero was multiplied by +/– infinity
			X	+infinity was added to –infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMADDPD Negative Multiply and Add Packed Double-Precision Floating-Point

Multiplies each of the packed double-precision floating-point values in the first source by the corresponding packed double-precision floating-point values in the second source, then negates the products and adds them to the corresponding packed double-precision floating-point values in the third source. The results are written to the destination register.

The VFNMADDPD instruction requires four operands:

VFNMADDPD dest, src1, src2, src3 *dest = – (src1* src2) + src3*

The 128-bit version multiplies the two double-precision values in the first source XMM register by the corresponding double-precision values in the second source, which can be either an XMM register or a 128-bit memory location. It then negates each product and adds it to the corresponding double-precision value in the third source. The results are then placed in the destination XMM register.

The 256-bit version multiplies the four double-precision values in the first source YMM register by the four double-precision values in the second source, which can be either a YMM register or a 256-bit memory location. It then negates each product and adds it to the corresponding double-precision value in the third source. The results are then placed in the destination YMM register.

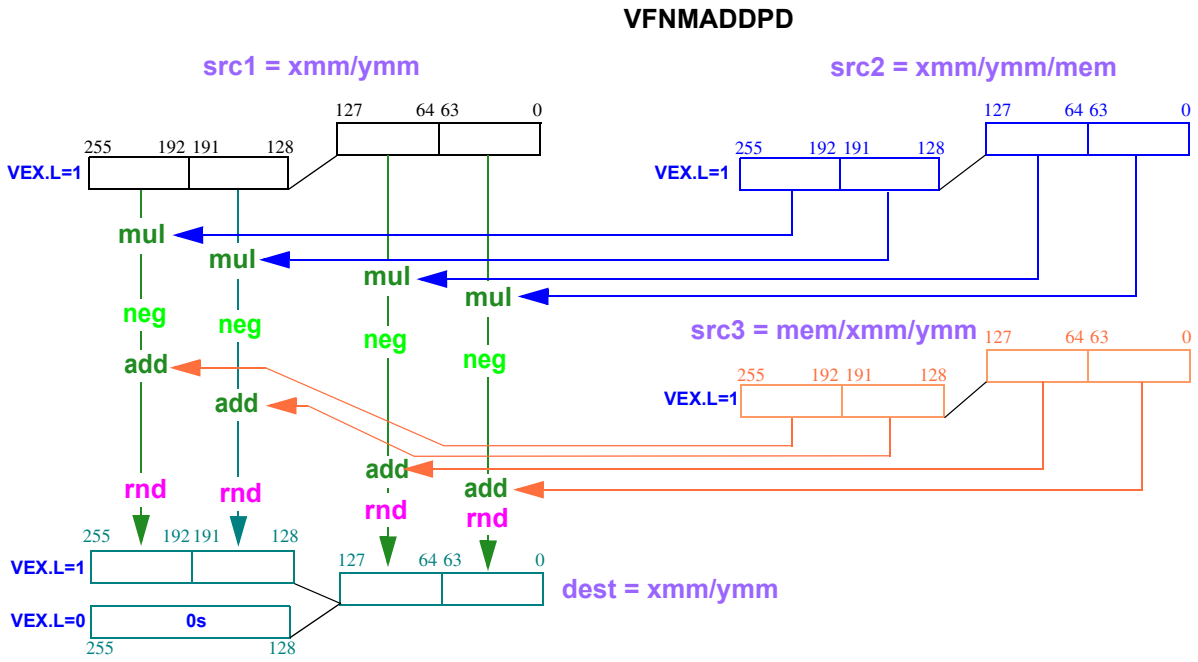
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by the rounding mode in MXCSR.

The VFNMADDPD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMADDPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	79 /r /is4
VFNMADDPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	79 /r /is4
VFNMADDPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	79 /r /is4
VFNMADDPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	79 /r /is4



Related Instructions

VFNMADDPS, VFNMADDSD, VFNMADDSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMADDPS Negative Multiply and Add Packed Single-Precision Floating-Point

Multiplies each of the packed single-precision floating-point values in first source by the corresponding packed single-precision floating-point value in the second source, then negates the products and adds them to the corresponding packed single-precision floating-point values in the third source. The results are written to the destination register.

The VFNMADDPS instruction requires four operands:

VFNMADDPS dest, src1, src2, src3 *dest = – (src1* src2) + src3*

The 128-bit version multiplies the four single-precision values in the first source XMM register by the corresponding single-precision values in the second source, which can be either an XMM register or a 128-bit memory location. It then negates each product and adds it to the corresponding single-precision value in the third source. The results are then placed in the destination XMM register.

The 256-bit version multiplies the eight single-precision values in the first source YMM register by the eight single-precision values in the second source, which can be either a YMM register or a 256-bit memory location. It then negates each product and adds it to the corresponding single-precision value in the third source. The result is then placed in the destination YMM register.

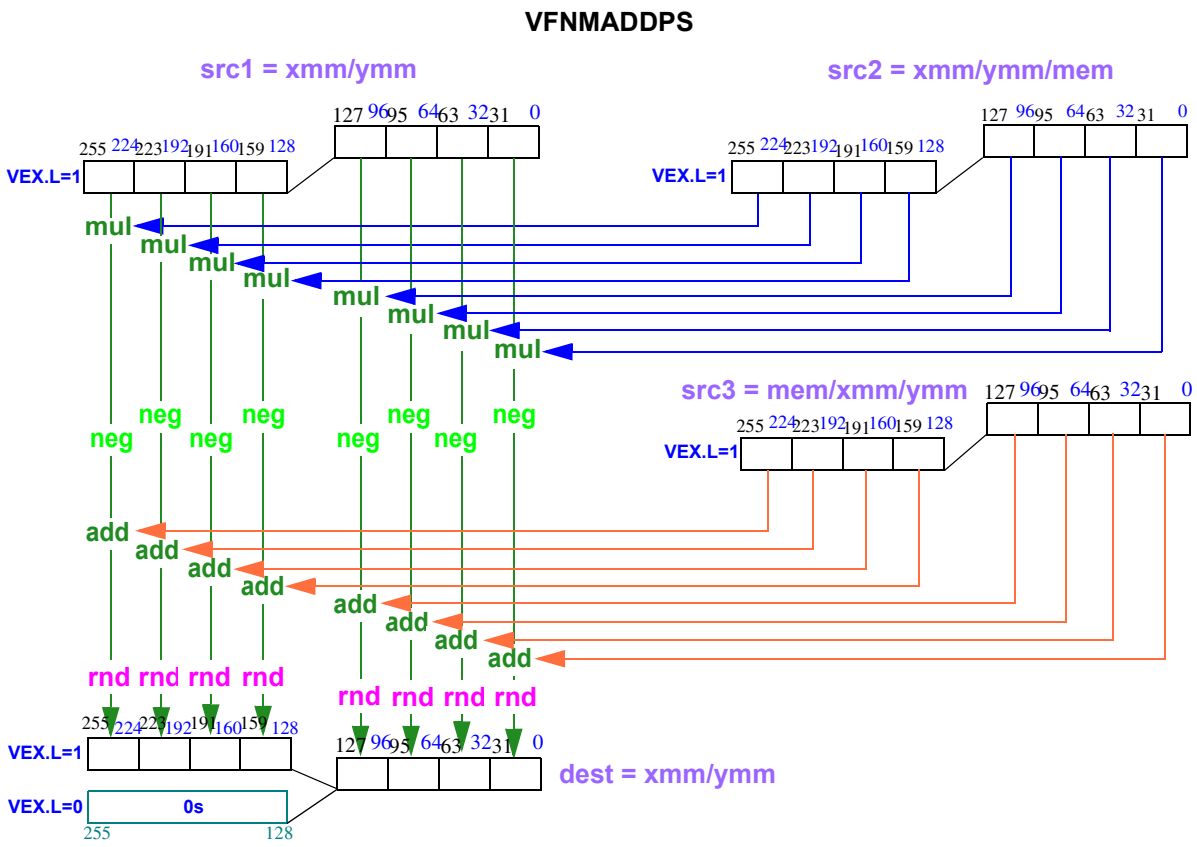
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by the rounding mode in MXCSR.

The FNMADDPS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMADDPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	78 /r /is4
VFNMADDPS *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	78 /r /is4
VFNMADDPS *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	78 /r /is4
VFNMADDPS *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	78 /r /is4



Related Instructions

VFNMADDPD, VFNMADDSD, VFNMADDSS

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMADDSD Negative Multiply and Add Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value in the low-order quadword of the first source by the double-precision floating-point value in the low-order quadword of the second source, then negates the product and adds it to the double-precision floating-point value in the low-order quadword of the third source. The low-order quadword result is written to the destination register.

The VFNMADDSD instruction requires four operands:

VFNMADDSD dest, src1, src2, src3 *dest = – (src1* src2) + src)*

The first source is an XMM register indicated by VEX.vvvv.

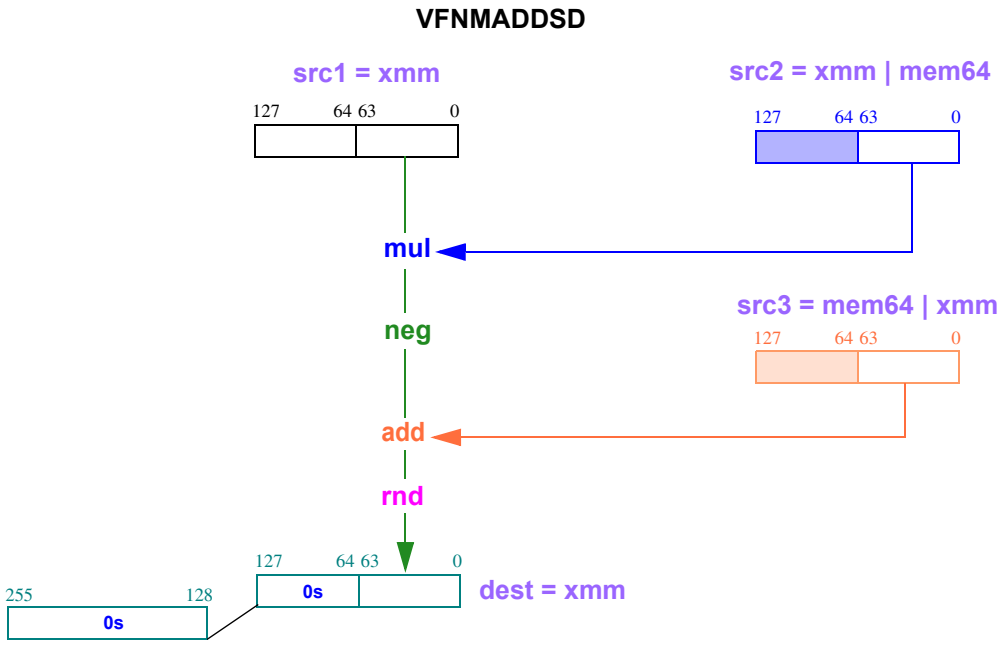
If VEX.W is 0, the second source is either a register or 64-bit memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or 64-bit memory location.

The destination is always an XMM register. When the result is written to the destination XMM register, the high quadword of the destination register (bits 64–127) and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by the rounding mode in MXCSR.

The VFNMADDSD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMADDSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	7B /r /is4
VFNMADDSD *xmm1*, *xmm2, xmm3, xmm4*/*mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	7B /r /is4



Related Instructions

VFNMADDPD, VFNMADDPS, VFNMADDSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMADDSS Negative Multiply and Add Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value in the low-order doubleword of the first source by the single-precision floating-point value in the low-order doubleword of the second source, then negates the product and adds it to the single-precision floating-point value in the low-order doubleword of the third source. The low-order doubleword result is written to the destination.

The VFNMADDSS instruction requires four operands:

VFNMADDSS dest, src1, src2, src3 *dest = - (src1* src2) + src3*

If VEX.W is 0, the second source is either a register or 32-bit memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or 32-bit memory location.

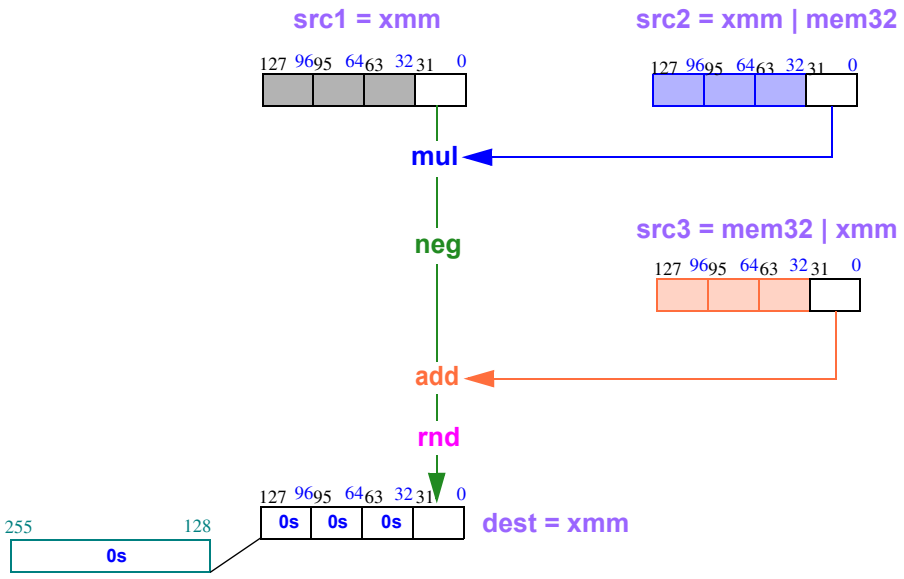
The destination is always an XMM register. When the result is written to the destination XMM register, the upper three doublewords of the destination register (bits 32–127) and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the addition. The results of the addition are rounded, as specified by the rounding mode in MXCSR.

The VFNMADDSS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMADDSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	7A /r /is4
VFNMADDSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	7A /r /is4

VFNMADDSS



Related Instructions

VFNMADDPD, VFNMADDPS, VFNMADDSS

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMSUBPD Negative Multiply and Subtract Packed Double-Precision Floating-Point

Multiplies each of the packed double-precision floating-point values in the first source by the corresponding packed double-precision floating-point value in the second source, then subtracts the corresponding packed double-precision floating-point value in the third source from the negated interim products.The results are written to the destination register.

The VFNMSUBPD instruction requires four operands:

VFNMSUBPD dest, src1, src2, src3 *dest = – (src1* src2) - src3*

The 128-bit version multiplies each of the two double-precision values in the first source XMM register by its corresponding double-precision value in the second source, which can be either an XMM register or a 128-bit memory location. It then subtracts the corresponding double-precision value in the third source from the negated interim product. The results are then placed in the destination XMM register.

The 256-bit version multiplies each of the four double-precision values in the first source YMM register by its corresponding double-precision value in the second source, which can be either a YMM register or a 256-bit memory location. It then subtracts the corresponding double-precision value in the third source from the negated interim product. The results are then placed in the destination YMM register.

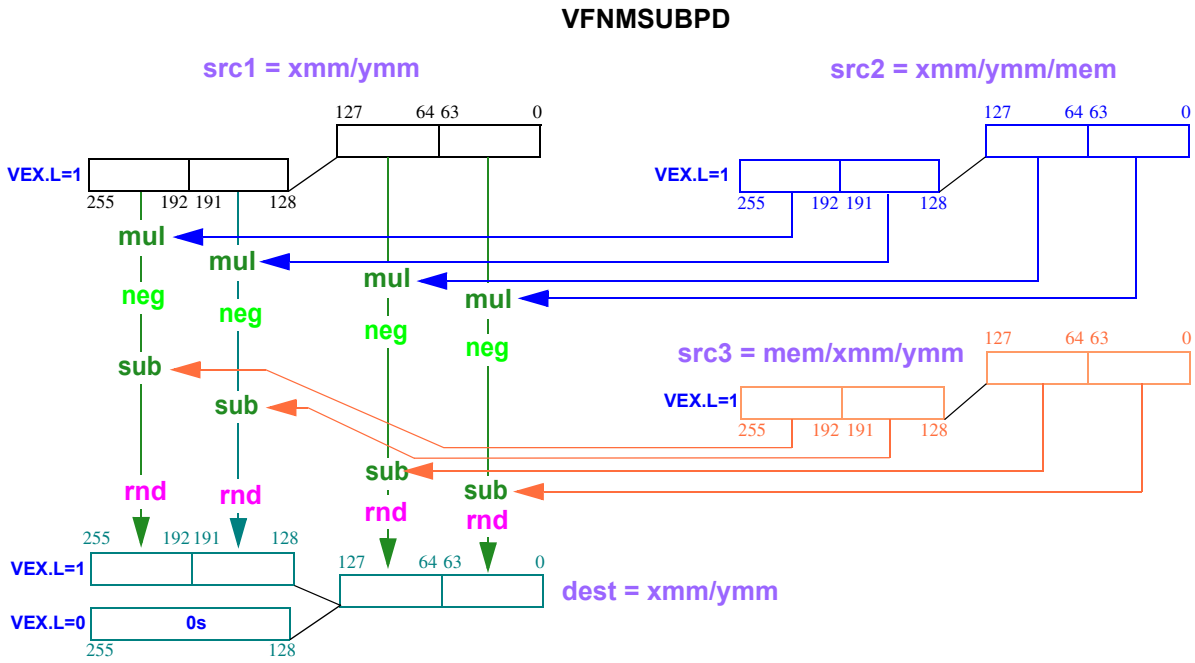
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by the rounding mode in MXCSR.

The VFNMSUBPD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMSUBPD *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	7D /r /is4
VFNMSUBPD *ymm1*, *ymm2*, *ymm3*/*mem256*, *ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	7D /r /is4
VFNMSUBPD *xmm1*, *xmm2, xmm3, xmm4*/*mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	7D /r /is4
VFNMSUBPD *ymm1*, *ymm2, ymm3, ymm4*/*mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	7D /r /is4



Related Instructions

VFNMSUBPS, VFNMSUBSD, VFNMSUBSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/–zero was multiplied by +/–infinity
			X	+infinity was added to –infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMSUBPS Negative Multiply and Subtract Packed Single-Precision Floating-Point

Multiplies each of the packed single-precision floating-point values in the first source by the corresponding packed single-precision floating-point value in the second source, then subtracts the corresponding packed single-precision floating-point values in the third source from the negated products. The results are written to the destination register.

The VFNMSUBPS instruction requires four operands:

VFNMSUBPS dest, src1, src2, src3 dest = – (src1 src2) – src3*

The 128-bit version multiplies each of the four single-precision values in the first source XMM register by its corresponding single-precision value in the second source, which can be either an XMM register or a 128-bit memory location. It then subtracts the corresponding single-precision value in the third source from the negated interim product. The results are then placed in the destination XMM register.

The 256-bit version multiplies each of the eight single-precision values in the first source YMM register by its corresponding single-precision value in the second source, which can be either a YMM register or a 256-bit memory location. It then subtracts the corresponding single-precision value in the third source from the negated interim product. The results are then placed in the destination YMM register.

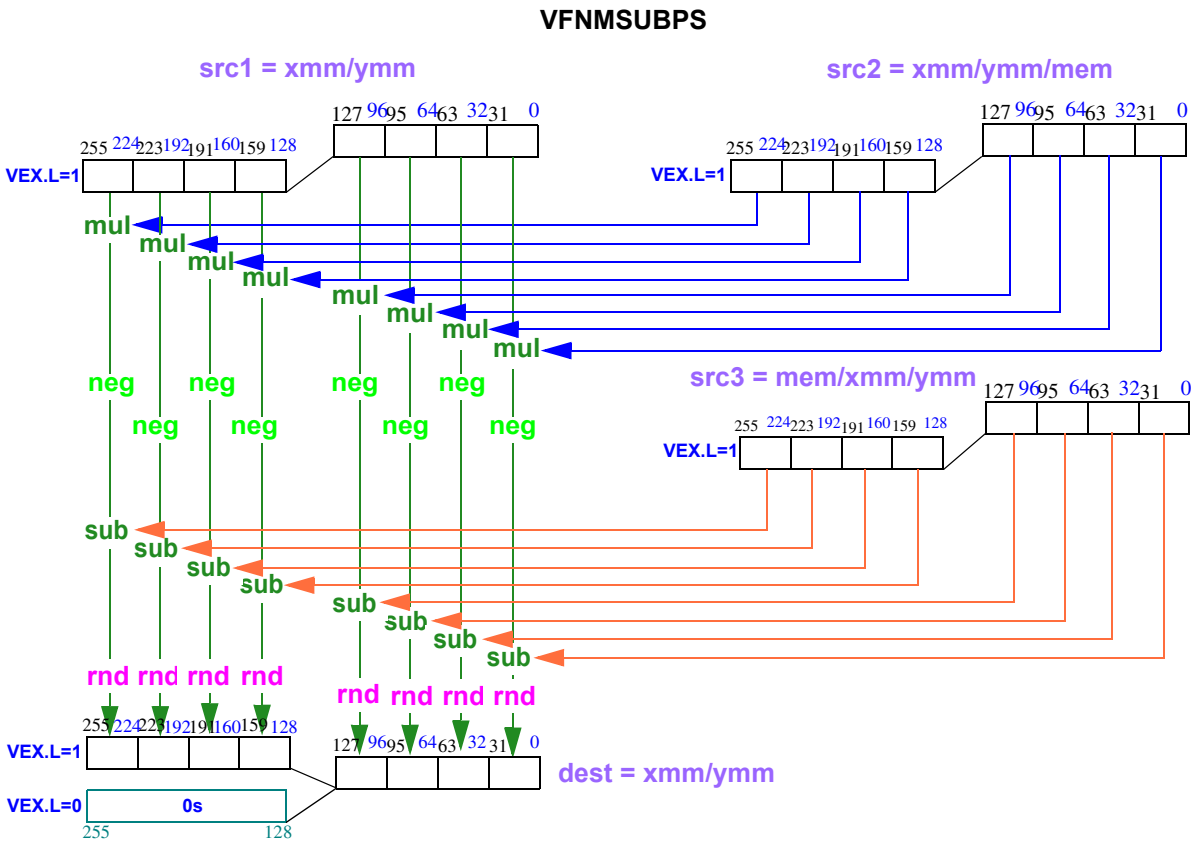
If VEX.W is 0, the second source is either a register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by the rounding mode in MXCSR.

The VFNMSUBPS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMSUBPS *xmm1, xmm2, xmm3/mem128, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	7C /r /is4
VFNMSUBPS *ymm1, ymm2, ymm3/mem256, ymm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.01	7C /r /is4
VFNMSUBPS *xmm1, xmm2, xmm3, xmm4/mem128*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	7C /r /is4
VFNMSUBPS *ymm1, ymm2, ymm3, ymm4/mem256*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.01	7C /r /is4



Related Instructions

VFNMSUBPD, VFNMSUBSD, VFNMSUBSS

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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/–zero was multiplied by +/–infinity
			X	+infinity was added to –infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMSUBSD Negative Multiply and Subtract Scalar Double-Precision Floating-Point

Multiplies the double-precision floating-point value in the low-order quadword of the first source by the double-precision floating-point value in the low-order quadword of the second source, then subtracts the double-precision floating-point value in the low-order quadword of the third source from the negated interim product.The low-order quadword result is written to the destination.

The VFNMSUBSD instruction requires four operands:

VFNMSUBSD dest, src1, src2, src3 dest = – (src1 src2) – src3*

The first source is an XMM register.

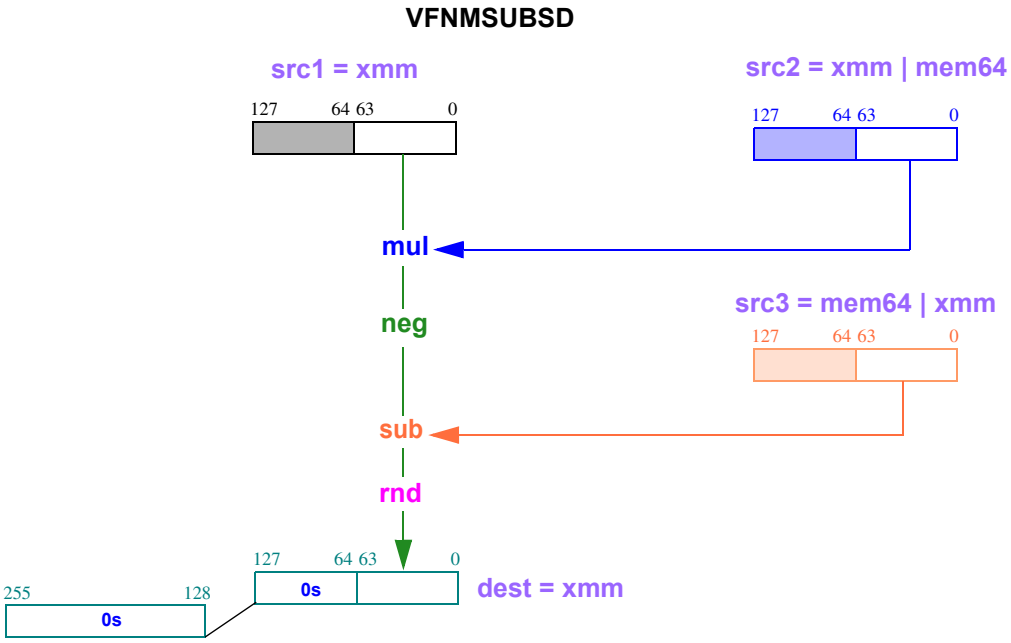
If VEX.W is 0, the second source is either a register or 64-bit memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or 64-bit memory location.

The destination is always an XMM register indicated by VEX.vvvv. All unaffected bits of the destination XMM register (bits 64–127) and its corresponding YMM register (bits 128–255) are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by the rounding mode in MXCSR.

The VFNMSUBSD instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMSUBSD *xmm1, xmm2, xmm3/mem64, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	7F /r /is4
VFNMSUBSD *xmm1, xmm2, xmm3, xmm4/mem64*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	7F /r /is4



Related Instructions

VFNMSUBPD, VFNMSUBPS, VFNMSUBSS

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/-zero was multiplied by +/- infinity
			X	+infinity was added to -infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFNMSUBSS Negative Multiply and Subtract Scalar Single-Precision Floating-Point

Multiplies the single-precision floating-point value in the low-order doubleword of the first source by the single-precision floating-point value in the low-order doubleword of the second source, then subtracts the single-precision floating-point value in the low-order doubleword of the third source from the negated product. The low-order doubleword result is written to the destination.

The VFNMSUBSS instruction requires four operands:

VFNMSUBSS dest, src1, src2, src3 *dest = - (src1* src2) - src3*

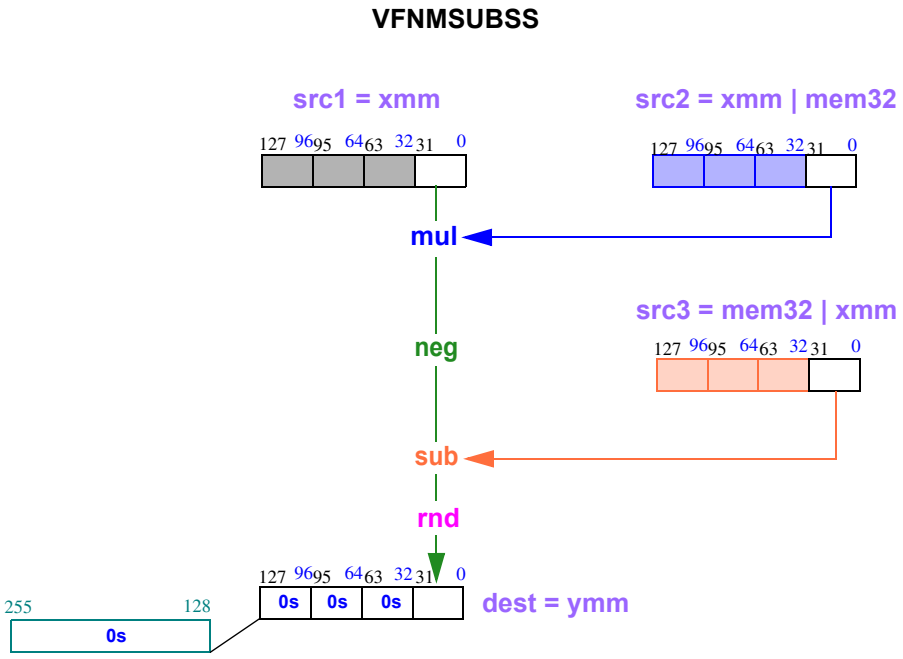
If VEX.W is 0, the second source is either a register or 32-bit memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or 32-bit memory location.

The destination is always an XMM register indicated by VEX.vvvv. All unaffected bits of the destination XMM register (bits 32–127) and its corresponding YMM register (bits 128–255) are cleared to zeros.

The intermediate products are not rounded; the infinitely precise products are used in the subtraction. The results of the subtraction are rounded, as specified by the rounding mode in MXCSR.

The VFNMSUBSS instruction is an FMA4 instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFNMSUBSS *xmm1, xmm2, xmm3/mem32, xmm4*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.01	7E /r /is4
VFNMSUBSS *xmm1*, *xmm2*, *xmm3, xmm4*/*mem32*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.01	7E /r /is4



Related Instructions

VFNMSUBPD, VFNMSUBPS, VFNMSUBSD

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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		FMA4 instructions are only recognized in protected mode.
			X	The FMA4 instructions are not supported, as indicated by ECX bit 16 of CPUID function 8000_0001h.
			X	The operating-system XSAVE support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits (YMM and XMM) of XFEATURE_ENABLED_MASK were not both set.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value.
			X	+/–zero was multiplied by +/–infinity
			X	+infinity was added to –infinity
			X	–infinity was subtracted from –infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Overflow exception (OE)			X	A rounded result was too large to fit into the format of the destination operand.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.
Precision exception (PE)			X	A result could not be represented exactly in the destination format.

VFRCZPD Extract Fraction Packed Double-Precision Floating-Point

Extracts the fractional portion of each double-precision floating-point value in a source register or memory location and writes the resulting values in the corresponding elements of the destination register. The fractional results are precise.

If XOP.L is 0, the source is an XMM register or 128-bit memory location; If XOP.L is 1, the source is a YMM register or 256-bit memory location.

The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of XOP.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The rounding mode indicated in the MXCSR is ignored unless the input is an integer, a zero, or a denormal value that is coerced to zero by MXCSR.DAZ, in which case the sign of the resultant zero is a function of MXCSR.RC:

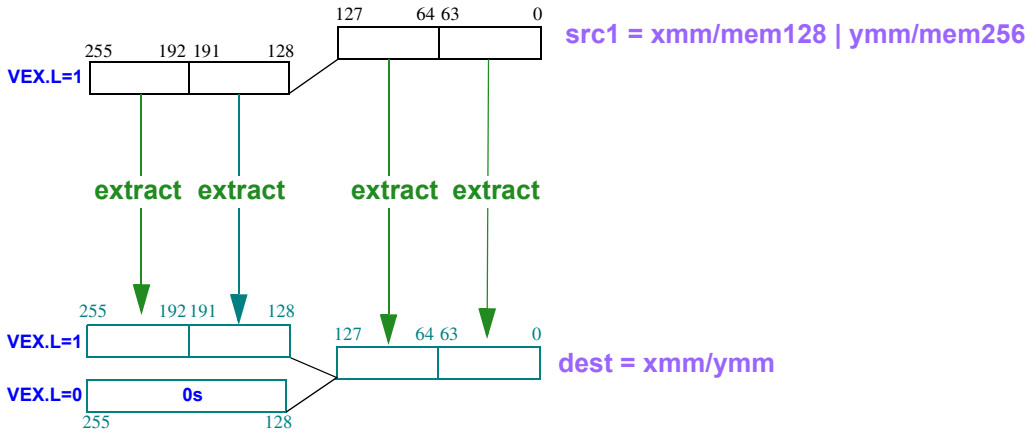
MXCSR.RC	Result
Round down	-0
Round to nearest	+0
Round up	+0
Round toward zero	+0

If the source value is QNaN, it is written to the destination with no exception generated. If the source value is infinity, the instruction returns an indefinite value when the invalid-operation exception (IE) is masked. If the source value is an integer, the instruction returns zero. The sign of the instruction result is the same as the input.

The VFRCZPD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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

VFRCZPD



Related Instructions

ROUNDPD, ROUNDPS, ROUNDSD, 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
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.*

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
			X	VEX.W was set to 1.
			X	VEX.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value or infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.

VFRCZPS Extract Fraction Packed Single-Precision Floating-Point

Extracts the fractional portion of each of the single-precision floating-point values in a source register or memory location and writes the resulting values to the corresponding elements of the destination register. The fractional results are exact.

IfXOP.L is 0, the source is an XMM register or 128-bit memory location; If XOP.L is 1, the source is a YMM register or 256-bit memory location.

The destination is always an XMM register or a YMM register, depending on the vector size, as determined by the value of XOP.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

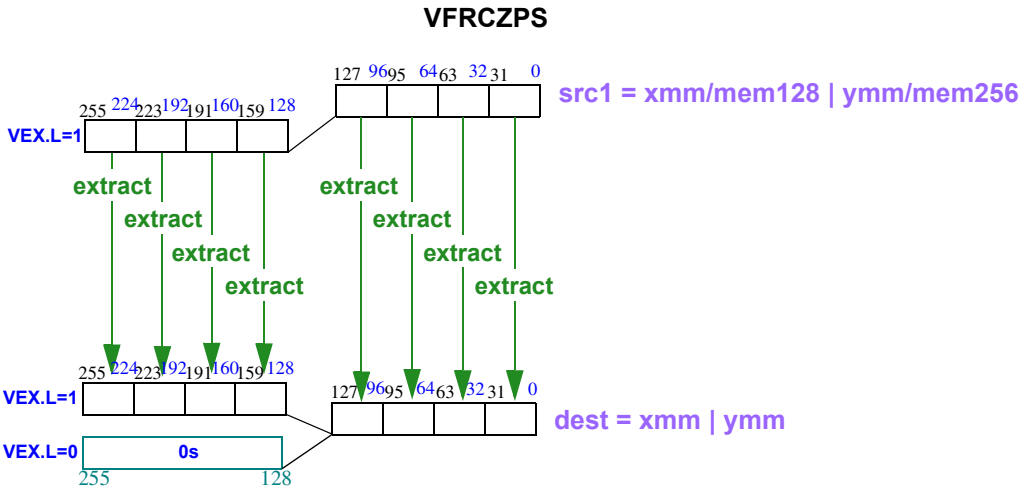
The rounding mode indicated in the MXCSR is ignored unless the input is an integer, a zero, or a denormal value that is coerced to zero by MXCSR.DAZ, in which case the sign of the resultant zero is a function of MXCSR.RC:

MXCSR.RC	Result
Round down	-0
Round to nearest	+0
Round up	+0
Round toward zero	+0

If the source value is QNaN, it is written to the destination with no exception generated. If the source value is infinity, the instruction returns an indefinite value when the invalid-operation exception (IE) is masked. If the source value is an integer, the instruction returns zero. The sign of the instruction result is the same as the input.

The VFRCZPS instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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

ROUNDPD, ROUNDPS, ROUNDSD, 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
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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
			X	VEX.W was set to 1.
			X	VEX.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an 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)			X	A source operand was an SNaN value or infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.

VFRCZSD Extract Fraction Scalar Double-Precision Floating-Point

Extracts the fractional portion of the double-precision floating-point value in the low-order quadword of an XMM register or 64-bit memory location and writes the result in the low-order quadword of the destination XMM register. The fractional results are precise.

When the result is written to the destination XMM register, the upper quadword of the destination register and the upper 128-bits of the corresponding YMM register are cleared to zeros.

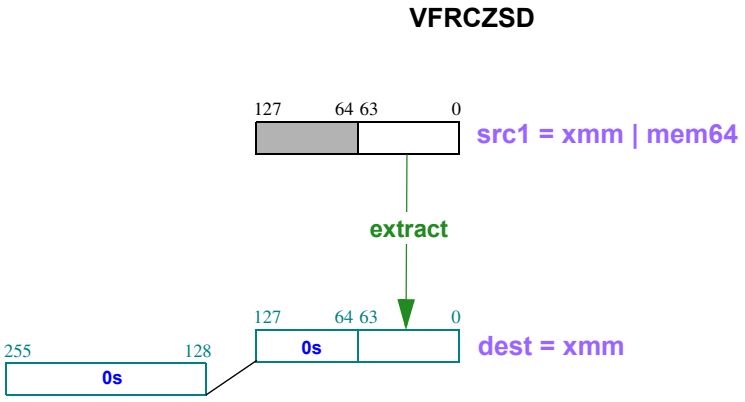
The rounding mode indicated in the MXCSR is ignored unless the input is an integer, a zero, or a denormal value that is coerced to zero by MXCSR.DAZ, in which case the sign of the resultant zero is a function of MXCSR.RC:

MXCSR.RC	Result
Round down	-0
Round to nearest	+0
Round up	+0
Round toward zero	+0

If the source value is QNaN, it is written to the destination with no exception generated. If the source value is infinity, the instruction returns an indefinite value when the invalid-operation exception (IE) is masked. If the source value is an integer, the instruction returns zero. The sign of the instruction result is the same as the input.

The VFRCZSD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VFRCZSD *xmm1*, *xmm2*/*mem64*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	83 /r



Related Instructions

ROUNDPD, ROUNDPS, ROUNDSD, 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
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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
			X	VEX.W was set to 1.
			X	VEX.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.
SIMD Floating-Point Exceptions				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)			X	A source operand was an SNaN value or infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.

VFRCZSS Extract Fraction Scalar Single-Precision Floating Point

Extracts the fractional portion of the single-precision floating-point value in the low-order doubleword of an XMM register or 32-bit memory location and writes the result in the low-order doubleword in the destination XMM register. The fractional results are precise.

When the result is written to the destination XMM register, the upper three doublewords of the destination register and the upper 128-bits of the corresponding YMM register are cleared to zeros.

The upper 224 bits of the YMM destination register are cleared to zeros.

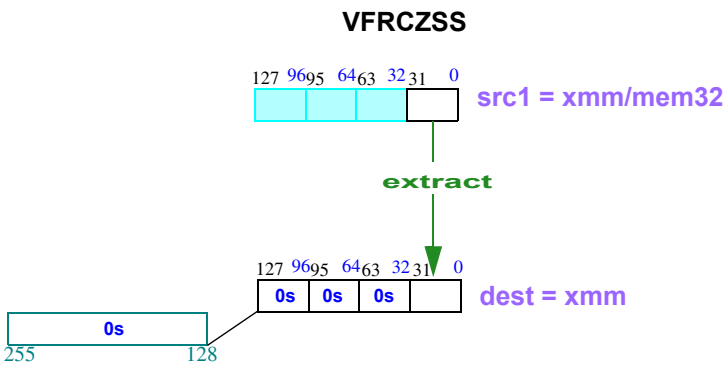
The rounding mode indicated in the MXCSR is ignored unless the input is an integer, a zero, or a denormal value that is coerced to zero by MXCSR.DAZ, in which case the sign of the resultant zero is a function of MXCSR.RC:

MXCSR.RC	Result
Round down	-0
Round to nearest	+0
Round up	+0
Round toward zero	+0

If the source value is QNaN, it is written to the destination with no exception generated. If the source value is infinity, the instruction returns an indefinite value when the invalid-operation exception (IE) is masked. If the source value is an integer, the instruction returns zero. The sign of the instruction result is the same as the input.

The VFRCZSS instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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
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.*																

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT = 0. See *SIMD Floating-Point Exceptions*, below, for details.
			X	VEX.W was set to 1.
			X	VEX.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.
SIMD Floating-Point Exception, #XF			X	There was an unmasked SIMD floating-point exception while CR4.OSXMMEXCPT=1. See *SIMD Floating-Point Exceptions*, below, for details.
SIMD Floating-Point Exceptions				

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid-operation exception (IE)			X	A source operand was an SNaN value or infinity
Denormalized-operand exception (DE)			X	A source operand was a denormal value.
Underflow exception (UE)			X	A rounded result was too small to fit into the format of the destination operand.

VPCMOV Vector Conditional Moves

Moves bits of either the first source or the second source into their corresponding positions in the destination, depending on the value of the corresponding selector bit in the selector. If the selector bit is set to 1, the corresponding bit in the first source is moved to the destination; otherwise, the corresponding bit from the second source is moved to the destination.

This instruction directly implements the C-language ternary "?" operation on each of the source bits.

Arbitrary bit-granular predicates can be constructed by any number of methods, or loaded as constants from memory. The VPCMOV 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 register 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.

The VPCMOV instruction requires four operands:

VPCMOV dest, src1, src2, selector

The vector size is determined by the value of VEX.L. All moves are 128 bits in length if XOP.L is cleared to 0 and 256 bits in length if XOP.L is set to 1. The sources are the same size as the destination.

The first source (*src1*) is always an XMM or YMM register specified by XOP.vvvv.

This instruction supports operand configuration using XOP.W. When XOP.W is 0, the second source (*src2*) is an XMM or YMM register or 128- or 256-bit memory location specified by MODRM.rm and the *selector* is an XMM or YMM register specified by imm8[7:4]. When XOP.W is 1, the second source (*src2*) is an XMM or YMM register specified by imm8[7:4] and *selector* is an XMM or YMM register or 128- or 256-bit memory location specified by MODRM.rm.

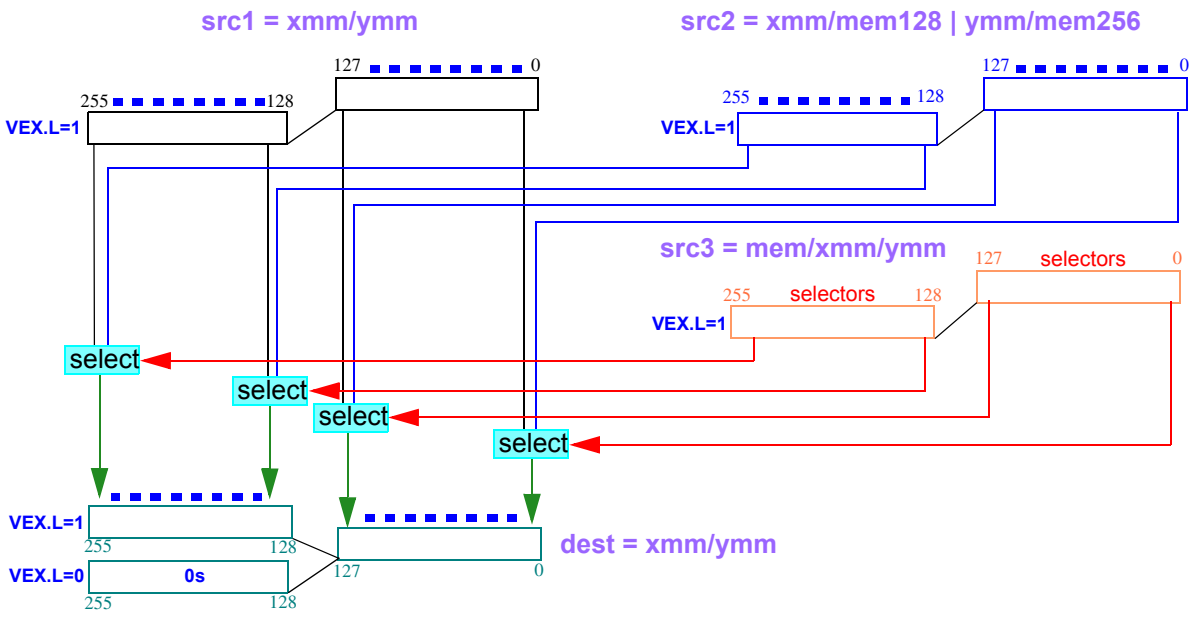
The destination (*dest*) is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPCMOV instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCMOV *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	A2 /r imm[7:4]
VPCMOV *ymm1, ymm2, ymm3/mem256, ymm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{ysrc1}}$.1.00	A2 /r imm[7:4]
VPCMOV *xmm1, xmm2, xmm3, xmm4/mem128*	8F	$\overline{\text{RXB}}$.08	1.$\overline{\text{xsrc1}}$.0.00	A2 /r imm[7:4]
VPCMOV *ymm1, ymm2, ymm3, ymm4/mem256*	8F	$\overline{\text{RXB}}$.08	1.$\overline{\text{ysrc1}}$.1.00	A2 /r imm[7:4]

VPCMOV



Related Instructions

VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMUW, VCMPPD, VCMPPS

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

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).

The VPCOMB instruction requires four operands:

VPCOMB dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the comparison results are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

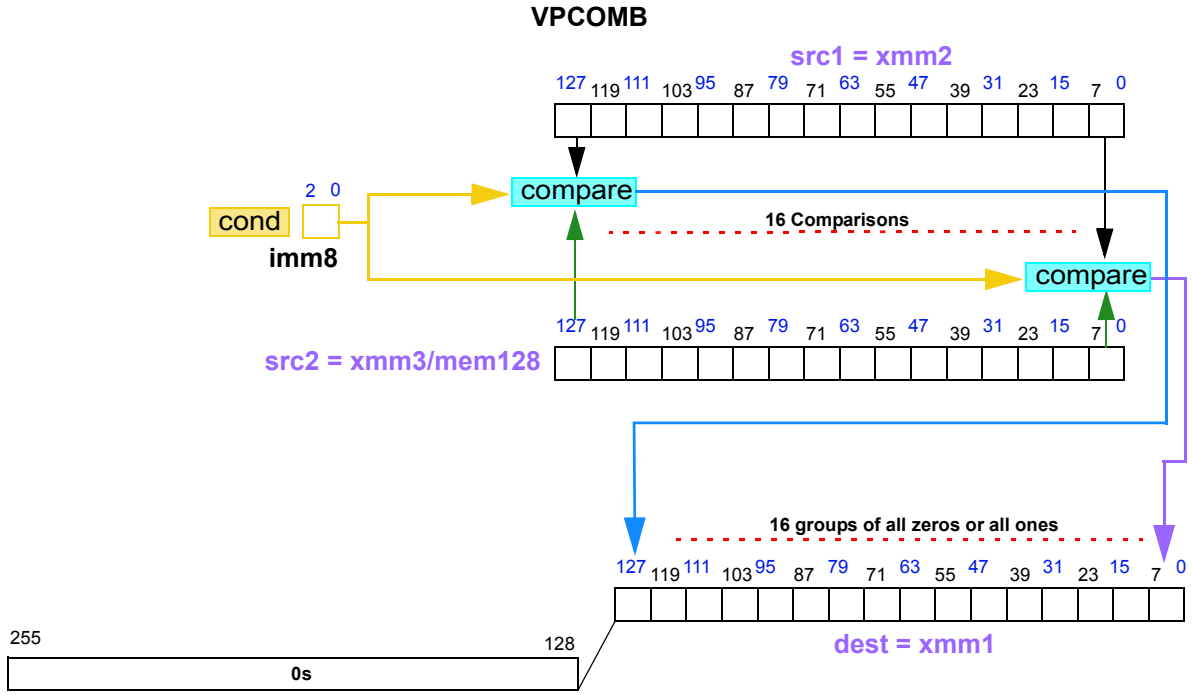
The *comp* type is specified by the three low-order bits of an immediate-byte, as shown in Table 2-1. The VPCOMB instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-1. VPCOMB Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTB	0	Less Than
VPCOMLEB	1	Less Than or Equal
VPCOMGTB	2	Greater Than
VPCOMGEB	3	Greater Than or Equal
VPCOMEQB	4	Equal
VPCOMNEQB	5	Not Equal
VPCOMFALSEB	6	False
VPCOMTRUEB	7	True

The VPCOMB instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMB *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	CC /r /imm8



Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPCOMD Compare Vector Signed Doublewords

Compares corresponding packed signed doublewords in the first and second sources and writes the result of each comparison in the corresponding doubleword of the destination. The result of each comparison is a 32-bit value of all 1s (TRUE) or all 0s (FALSE).

The VPCOMD instruction requires four operands:

VPCOMD dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the results of the comparisons are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

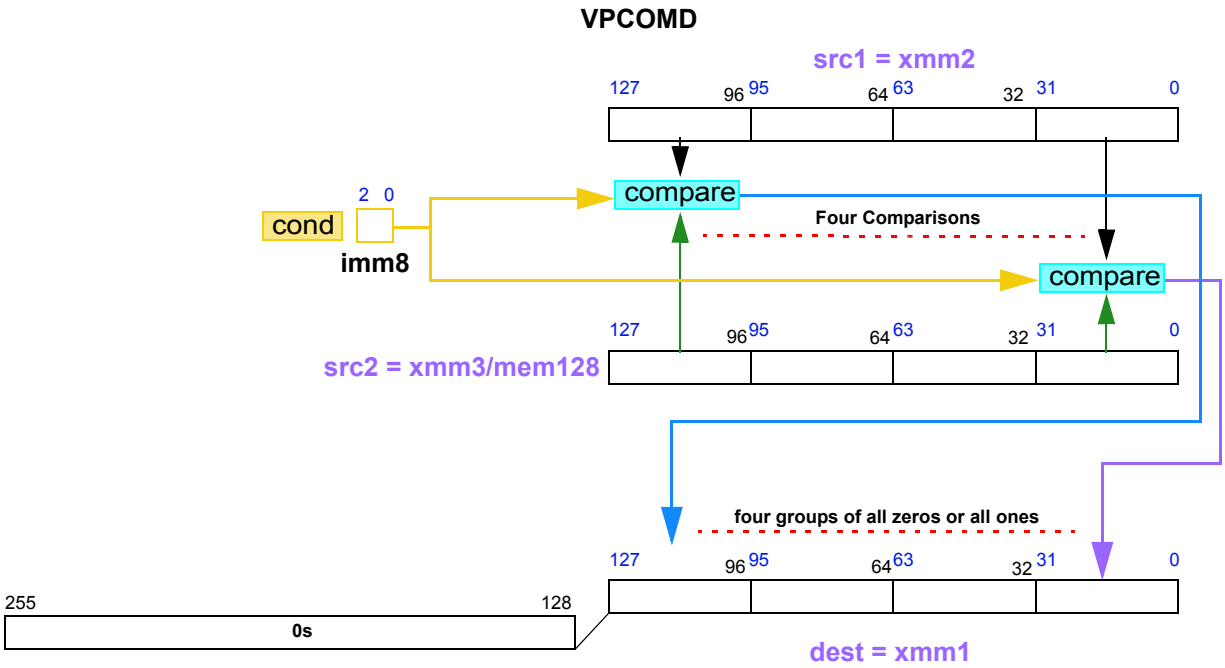
The *comp* type is specified by the three low-order bits of an immediate-byte, as shown in Table 2-2. The VPCOMD instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-2. VPCOMD Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTD	0	Less Than
VPCOMLED	1	Less Than or Equal
VPCOMGTD	2	Greater Than
VPCOMGED	3	Greater Than or Equal
VPCOMEQD	4	Equal
VPCOMNEQD	5	Not Equal
VPCOMFALSED	6	False
VPCOMTRUED	7	True

The VPCOMD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMD *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	CE /r /imm8



Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPCOMQ Compare Vector Signed Quadwords

Compares corresponding packed signed quadwords in the first and second sources and writes the result of each comparison in the corresponding quadword of the destination. The result of each comparison is a 64-bit value of all 1s (TRUE) or all 0s (FALSE).

The VPCOMQ instruction requires four operands:

VPCOMQ dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the result is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

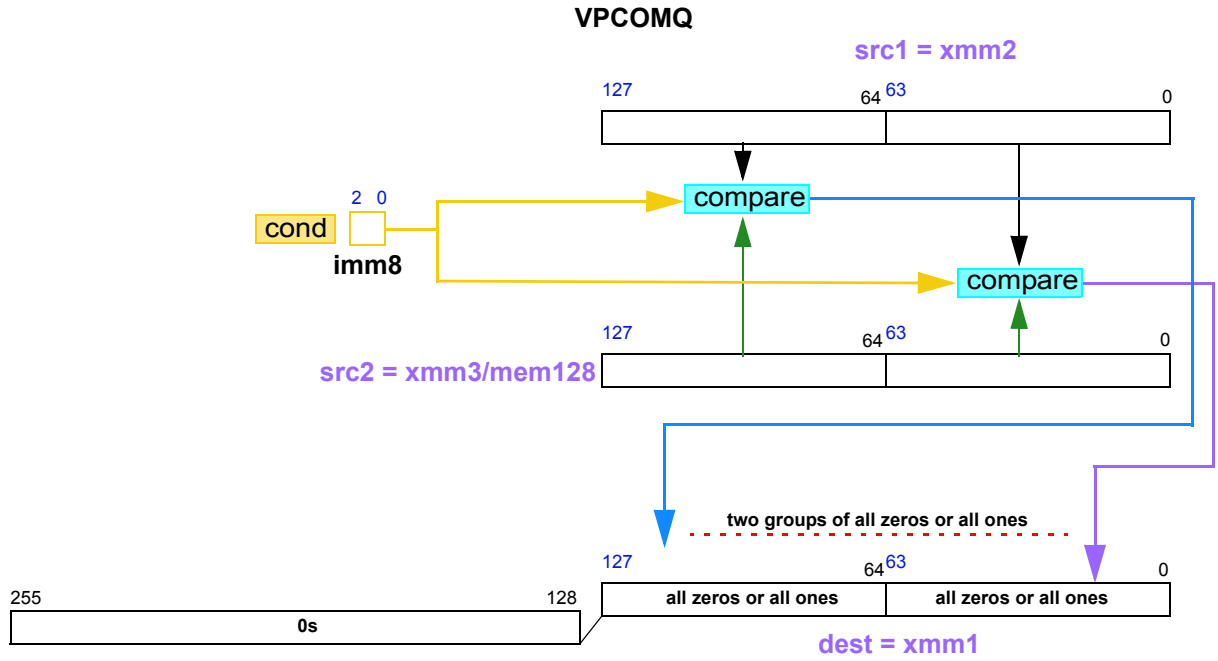
The *comp* type is specified by the three low-order bits of an immediate-byte, as shown in Table 2-3. The VPCOMQ instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-3. VPCOMQ Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTQ	0	Less Than
VPCOMLEQ	1	Less Than or Equal
VPCOMGTQ	2	Greater Than
VPCOMGEQ	3	Greater Than or Equal
VPCOMEQQ	4	Equal
VPCOMNEQQ	5	Not Equal
VPCOMFALSEQ	6	False
VPCOMTRUEQ	7	True

The VPCOMQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMQ *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	CF /r imm8



Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPCOMUB Compare Vector Unsigned Bytes

Compares corresponding packed unsigned 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).

The VPCOMUB instruction requires four operands:

VPCOMUB dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the result is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

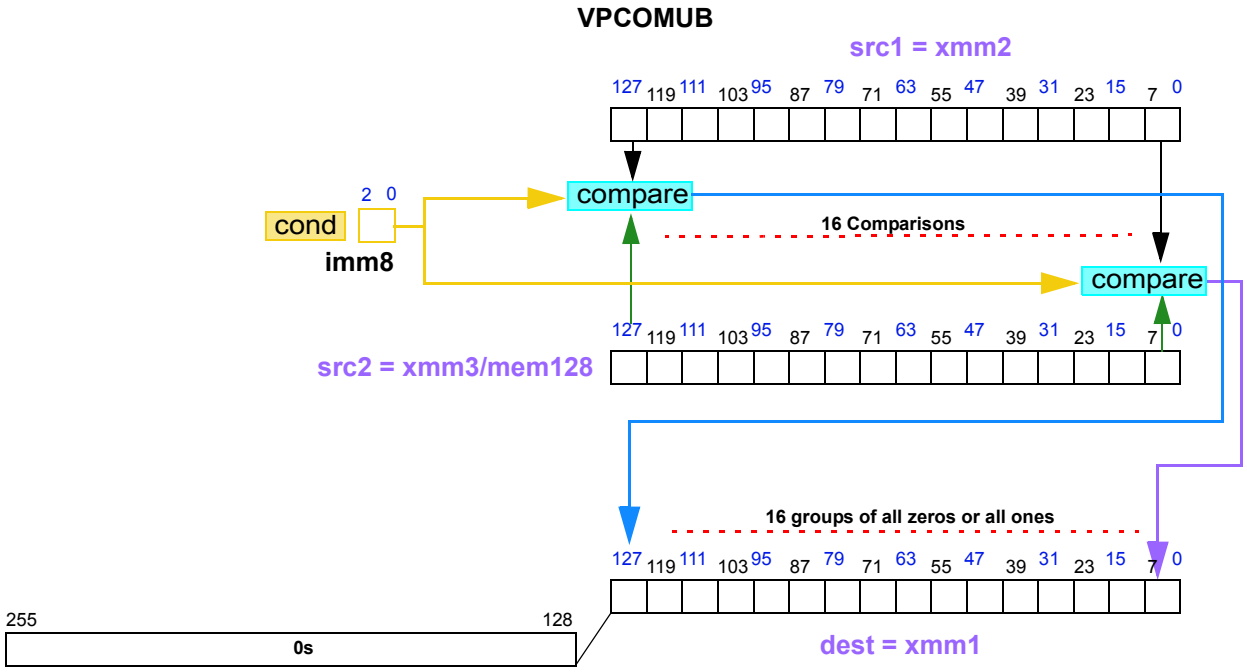
The *comp* type is specified by the three low-order bits of an immediate-byte, as shown in Table 2-4. The VPCOMUB instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-4. VPCOMUB Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTUB	0	Less Than
VPCOMLEUB	1	Less Than or Equal
VPCOMGTUB	2	Greater Than
VPCOMGEUB	3	Greater Than or Equal
VPCOMEQUB	4	Equal
VPCOMNEQUB	5	Not Equal
VPCOMFALSEUB	6	False
VPCOMTRUEUB	7	True

The VPCOMUB instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMUB *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	EC /r imm8



Related Instructions

VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPCOMUD Compare Vector Unsigned Doublewords

Compares corresponding packed unsigned doublewords in the first and second sources and writes the result of each comparison in the corresponding doubleword of the destination. The result of each comparison is a 32-bit value of all 1s (TRUE) or all 0s (FALSE).

The VPCOMUD instruction requires four operands:

VPCOMUD dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the results are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

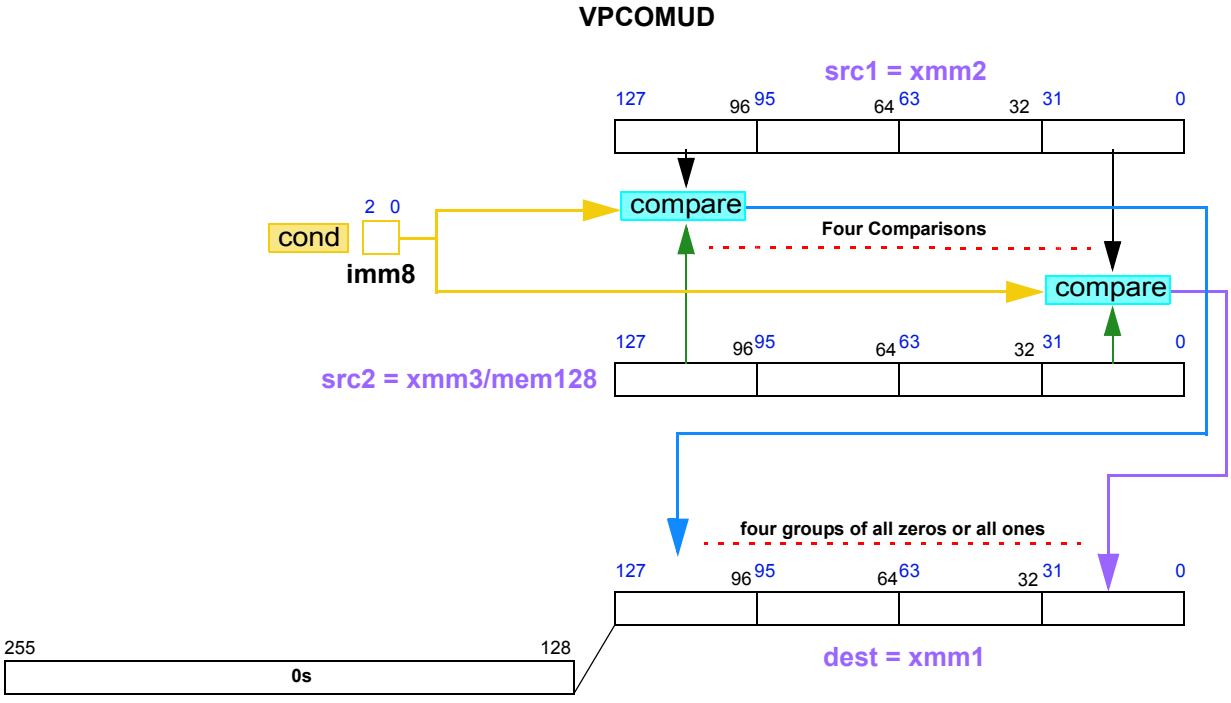
The *comp* type is specified by the three low-order bits of an immediate-byte, as shown Table 2-5. The VPCOMUD instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-5. VPCOMUD Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTUD	0	Less Than
VPCOMLEUD	1	Less Than or Equal
VPCOMGTUD	2	Greater Than
VPCOMGEUD	3	Greater Than or Equal
VPCOMEQUD	4	Equal
VPCOMNEQUD	5	Not Equal
VPCOMFALSEUD	6	False
VPCOMTRUEUD	7	True

The VPCOMUD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMUD *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	EE /r imm8



Related Instructions

VPCOMUB, VPCOMUW, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPCOMUQ Compare Vector Unsigned Quadwords

Compares corresponding packed unsigned quadwords in the first and second sources and writes the result of each comparison in the corresponding quadword of the destination. The result of each comparison is a 64-bit value of all 1s (TRUE) or all 0s (FALSE).

The VPCOMUQ instruction requires four operands:

VPCOMUQ dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the results are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

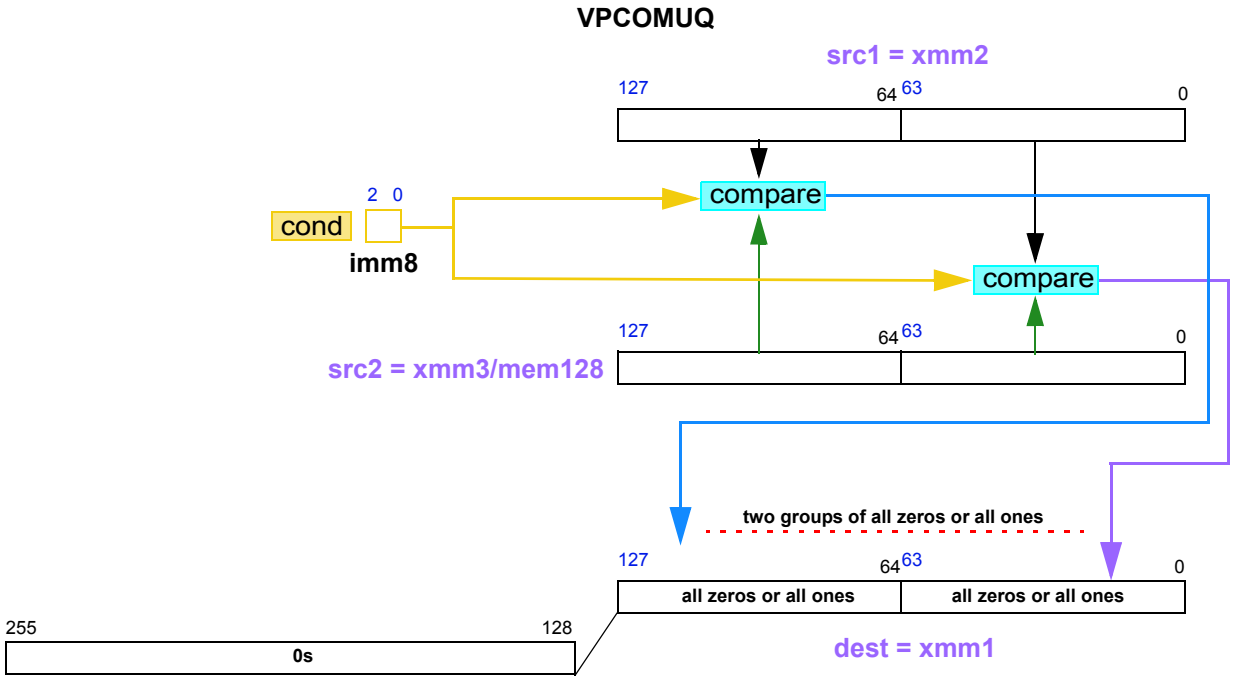
The *comp* type is specified by the three low-order bits of an immediate-byte, as shown in Table 2-6. The VPCOMUQ instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-6. VPCOMUQ Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTUQ	0	Less Than
VPCOMLEUQ	1	Less Than or Equal
VPCOMGTUQ	2	Greater Than
VPCOMGEUQ	3	Greater Than or Equal
VPCOMEQUQ	4	Equal
VPCOMNEQUQ	5	Not Equal
VPCOMFALSEUQ	6	False
VPCOMTRUEUQ	7	True

The VPCOMUQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMUQ *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	EF /r imm8



Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPCOMUW Compare Vector Unsigned Words

Compares corresponding packed unsigned 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).

The VPCOMUW instruction requires four operands:

VPCOMUW dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the results are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (src1) is an XMM register specified by the XOP.vvvv field and the second source (src2) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

The *comp* type is specified by the three low-order bits of an immediate-byte, as defined in Table 2-7. The VPCOMUW instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

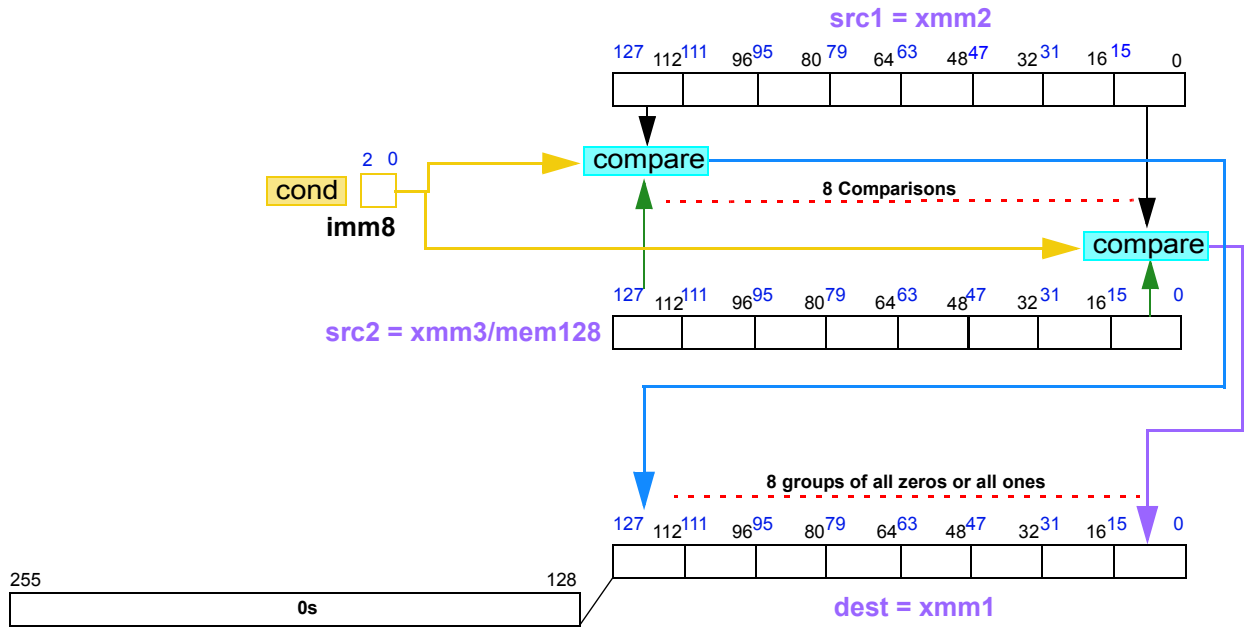
Table 2-7. VPCOMUW Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTUW	0	Less Than
VPCOMLEUW	1	Less Than or Equal
VPCOMGTUW	2	Greater Than
VPCOMGEUW	3	Greater Than or Equal
VPCOMEQUW	4	Equal
VPCOMNEQUW	5	Not Equal
VPCOMFALSEUW	6	False
VPCOMTRUEUW	7	True

The VPCOMUW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMUW *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	ED /r imm8

VPCOMUW



Related Instructions

VPCOMUB, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMW, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

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).

The VPCOMW instruction requires four operands:

VPCOMW dest, src1, src2, comp

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the results are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv field and the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field.

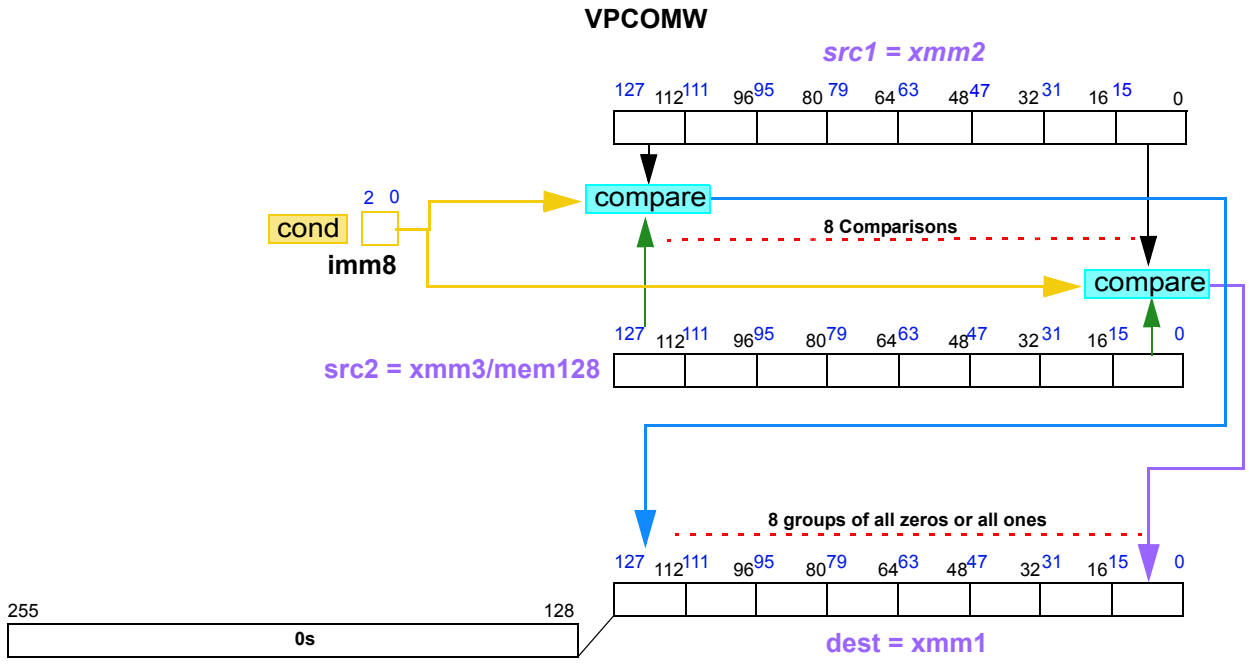
The *comp* type is specified by the three low-order bits of an immediate-byte, as defined in Table 2-8. The VPCOMW instruction with an appropriate value of *imm8* is aliased to the following mnemonics to facilitate coding.

Table 2-8. VPCOMW Comparison Operations

Mnemonic	Implied Value of *imm8*	Comparison Operation
VPCOMLTW	0	Less Than
VPCOMLEW	1	Less Than or Equal
VPCOMGTW	2	Greater Than
VPCOMGEW	3	Greater Than or Equal
VPCOMEQW	4	Equal
VPCOMNEQW	5	Not Equal
VPCOMFALSEW	6	False
VPCOMTRUEW	7	True

The VPCOMW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPCOMW *xmm1, xmm2, xmm3/mem128, imm8*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	CD /r imm8



Related Instructions

VPCOMUB, VPCOMUW, VPCOMUD, VPCOMUQ, VPCOMB, VPCOMD, VPCOMQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPERMIL2PD Permute Two-Source Double-Precision Floating-Point Values

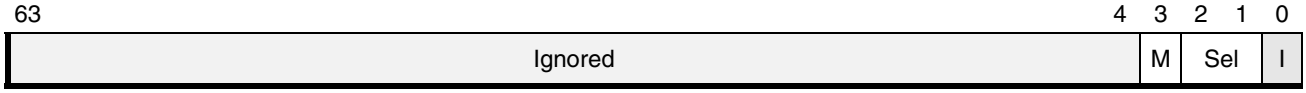
Loads each quadword element of the destination register with either a quadword value from one of the source registers, or with zero. The first and second source operands contain the quadword values to copy from, and the third source operand and an immediate byte operand control which source quadword to copy and whether each destination quadword is cleared to zero.

There are XMM and YMM versions of this instruction, as determined by the value of VEX.L. Both versions require four operands:

VPERMIL2PD dest, src1, src2, selector, imm8

Selector Operand. The selector operand is either a register or memory operand. The selector operand for the 128-bit version of this instruction is divided into two quadword selector elements; that for the 256-bit version is divided into four quadword selector elements. Each selector operand determines the value moved to its positionally corresponding element in the destination XMM or YMM register.

The bit field of each quadword selector element determines the source element selected and the value actually written to the destination register element. See Figure 2-1.



Bits	Mnemonic	Description
63-4	Ignored	
3	M	Match
2-1	Sel	Select
0	Ignored	

Figure 2-1. 64-Bit Selector Element Format

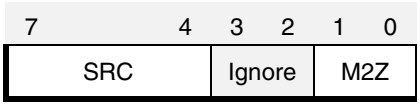
Bit 0 and bits 63–4 of the quadword selector element are ignored. Selector element fields are:

- Select—Bits 2–1 of each quadword element of the selector determine which source value is copied into the corresponding quadword element of the destination register. The source value selected by each bit pattern is defined in Table 2-9:
- Match—Bit 3. The values of the Match bit and the Match field of the immediate byte operand determine whether each quadword in the destination operand is written with zero. See table 2-4 for details. See Table 2-10 for details.

Table 2-9. Selector and Source Selected

Selector	Source Selected for Quadwords 0 and 1	Source Selected for Quadwords 2 and 3
00b	*src1*[63:0]	*src1*[191:128]
01b	*src1*[127:64]	*src1*[255:192]
10b	*src2*[63:0]	*src2*[191:128]
11b	*src2*[127:64]	*src2*[255:192]

Immediate Operand. The immediate byte layout is illustrated in Figure 2-2.



Bits	Mnemonic	Description
7–4	SRC	Source Register Select
3-2	Ignore	
1–0	M2Z	Match to Zero

Figure 2-2. 8-Bit Immediate Operand Format

- Source Register Select—Bits 7–4. In 64-bit mode, if VEX.W is 0, the 4-bit *src3* field identifies the third source register; if VEX.W is 1, SRC identifies the second source register. (See source register discussion below.)

  In non-64-bit mode, only bits [6:4] identify the register, and imm8[7] is ignored.
- Match to Zero—Bits 1–0. The two-bit M2Z field interacts with the Match bit of the selector element to determine the double-precision floating-point value that is written to the corresponding doubleword element in the destination operand. For details on the interaction between the immediate operand Match to Zero field and the Selector element Match bit, see Table 2-10 below.

Interaction between Selector Match Bit and Immediate Operand Match Field. The results of combining the selector match bit and the immediate operand match field are shown in Table 2-10.

Table 2-10. Interaction of Selector Match Bit and Immediate Operand Match Field

Immediate M2Z Field	Selector Match Bit	Value Loaded into Destination Quadword
0Xb	X	Operand value according to Table 2-9.
10b	0	Operand value according to Table 2-9.
10b	1	Zero
11b	0	Zero
11b	1	Operand value according to Table 2-9.

- Bits 7-4. If VEX.W is 0, this field identifies the third source register; if VEX.W is 1, this field identifies the second source register.

  In non-64-bit mode, only bits [6:4] identify the register, and imm8[7] is ignored.
- Bits 1–0. The two-bit M2Z field interacts with the Match bit of the selector element to determine the double-precision floating-point value that is written to the corresponding quadword element in the destination operand. For details on the interaction between the immediate operand Match to Zero field and the Selector element Match bit, see Table 2-10 above.

Source Operands. The 128-bit version of this instruction forms a single source operand by concatenation of the first source XMM register and the second source, which can be either an XMM register or a 128-bit memory location, to form a single operand partition consisting of four 64-bit double-precision floating point values.

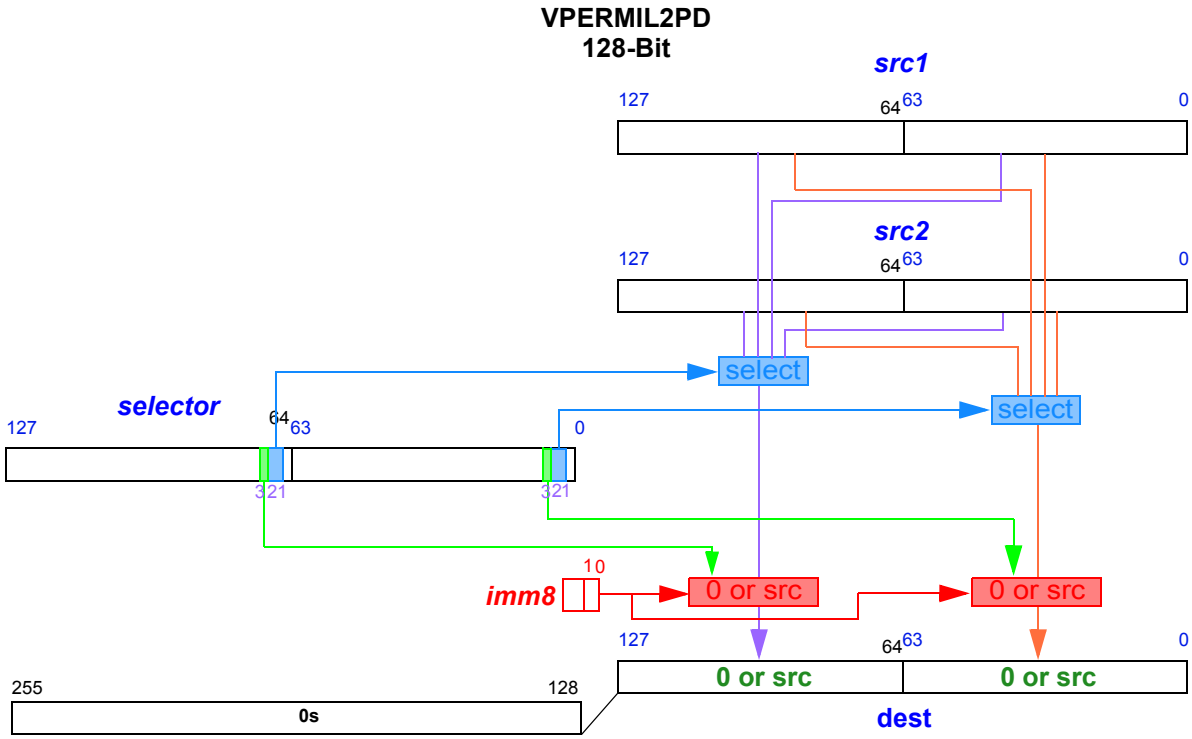
The 256-bit version forms two source operands by concatenation of the first source YMM register and the second source, which can be either an XMM register or a 128-bit memory location, to form two operand partitions each containing four 64-bit double-precision floating point values.

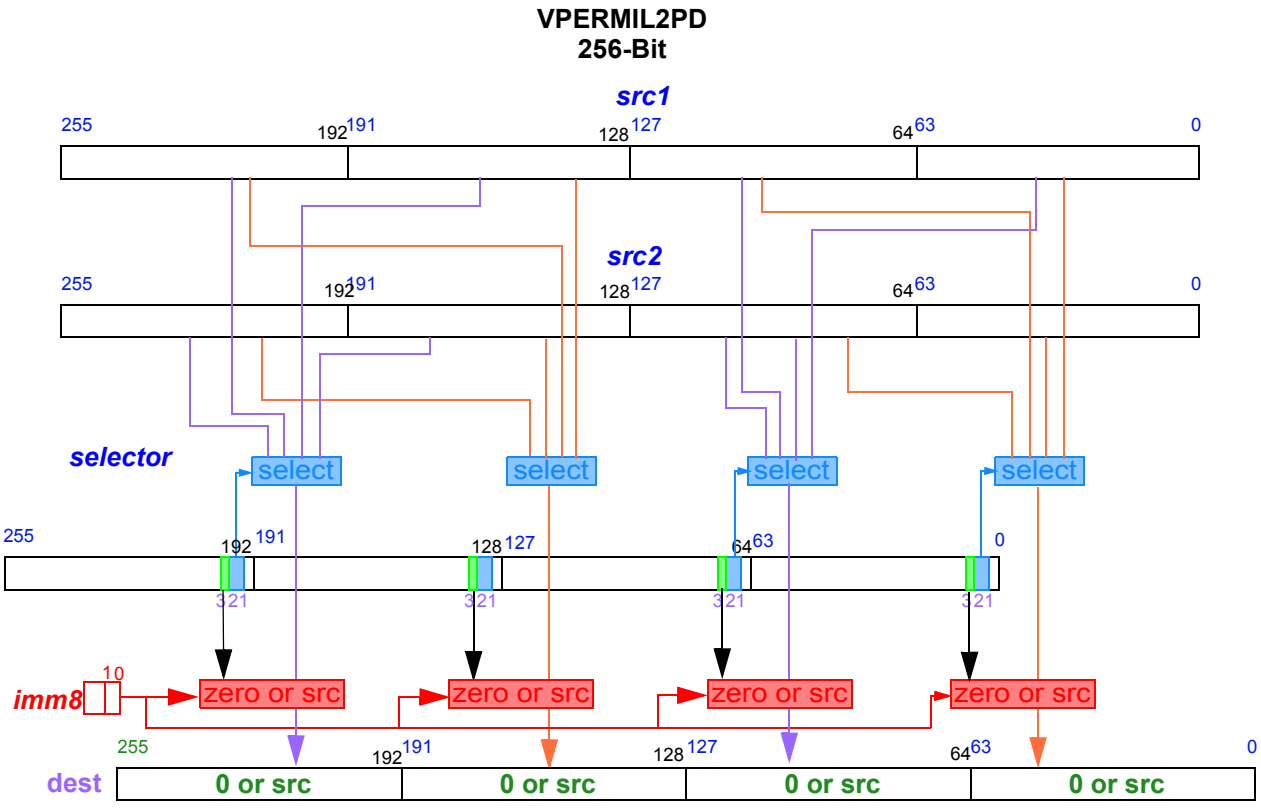
If VEX.W is 0, the second source is either an XMM or a YMM register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or a memory location. (See “Source Register Select,” above for a discussion of the use of the immediate byte to select the second or third source register.)

Destination Register. The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPERMIL2PD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	VEX	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPERMIL2PD *xmm1*, *xmm2*, *xmm3/mem128*, *xmm4*, *imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.00	49 /r imm8
VPERMIL2PD *xmm1*, *xmm2*, *xmm3*, *xmm4/mem128*, *imm8*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.00	49 /r imm8
VPERMIL2PD *ymm1*, *ymm2*, *ymm3/mem256*, *ymm4*, *imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.00	49 /r imm8
VPERMIL2PD *ymm1*, *ymm2*, *ymm3*, *ymm4/mem256*, *imm8*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.00	49 /r imm8


VPERMIL2PD
128-Bit
src1
src2
selector
select
imm8
0 or src
0s
dest


VPERMIL2PD
256-Bit
src1
src2
selector
select
imm8
zero or src
0 or src
dest

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPD, VPERMILPS, VPPERM

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

VPERMIL2PS Permute Two-Source Single-Precision Floating-Point Values

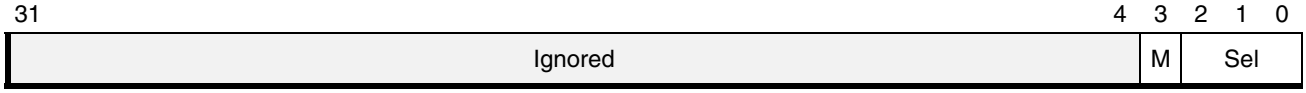
Loads each doubleword element of the destination register with either a doubleword value from one of the source registers, or with zero. The first and second source operands contain the doubleword values to copy from, and the third source operand and imm8 control which source doubleword to copy, and whether each destination doubleword is cleared to zero.

There are XMM and YMM versions of this instruction, as determined by the value of VEX.L. Both versions require four operands:

VPERMIL2PS dest, src1, src2, selector, imm8

Selector Operand. The selector operand is either a register or memory operand. The selector operand for the 128-bit version of this instruction is divided into four doubleword selector elements; that for the 256-bit version is divided into eight doubleword selector elements. Each selector operand determines the value moved to its positionally corresponding element in the destination XMM or YMM register.

The bit field of each doubleword selector element determines the source element selected and the value actually written to the destination register element. See Figure 2-3.



Bits	Mnemonic	Description
31-4	Ignored	
3	M	Match
2-0	Sel	Select

Figure 2-3. 32-Bit Selector Element Format

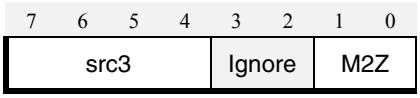
Bits 31–4 of the doubleword selector element are ignored. Selector element fields are:

- Select—Bits 2–0 of each doubleword element of the selector determine which source value is copied into the corresponding doubleword element of the destination register. The source value selected by each bit pattern is defined in Table 2-11:
- Match—Bit 3. The values of the Match bit and the Match field of the immediate byte operand determine whether each doubleword in the destination operand is written with zero. See table 2-4 for details

Table 2-11. Selector and Source Selected

Selector	Source Selected for Doublewords 0, 1, 2, 3	Source Selected for Doublewords 4, 5, 6, 7
000b	*src1*[31:0]	*src1*[159:128]
001b	*src1*[63:32]	*src1*[191:160]
010b	*src1*[95:64]	*src1*[223:192]
011b	*src1*[127:96]	*src1*[255:224]
100b	*src2*[31:0]	*src2*[159:128]
101b	*src2*[63:32]	*src2*[191:160]
110b	*src2*[95:64]	*src2*[223:192]
111b	*src2*[127:96]	*src2*[255:224]

Immediate Operand. The immediate byte layout is illustrated in Figure 2-4.



Bits	Mnemonic	Description
7–4	SRC	Source Register Select
3-2	Ignore	
1–0	M2Z	Match to Zero

Figure 2-4. 8-Bit Immediate Operand Format

- Source Register Select—Bits 7–4. In 64-bit mode, if VEX.W is 0, the 4-bit *src3* field identifies the third source register; if VEX.W is 1, SRC identifies the second source register. (See source register discussion below.)

  In non-64-bit mode, only bits [6:4] identify the register, and imm8[7] is ignored.
- Match to Zero—Bits 1–0. The two-bit M2Z field interacts with the Match bit of the selector element to determine the single-precision floating-point value that is written to the corresponding doubleword element in the destination operand. For details on the interaction between the immediate operand Match to Zero field and the Selector element Match bit, see Table 2-12 below.

Interaction between Selector Match Bit and Immediate Operand Match Field. The results of combining the selector match bit and the immediate operand match field are shown in Table 2-12.

Table 2-12. Interaction of Selector Match Bit and Immediate Operand Match Field

Immediate M2Z Field	Selector Match Bit	Value Loaded into Destination Quadword
0Xb	X	Operand value according to Table 2-11.
10b	0	Operand value according to Table 2-11.
10b	1	Zero
11b	0	Zero
11b	1	Operand value according to Table 2-11.

Source Operands. The 128-bit version of this instruction forms a single source operand by concatenation of the first source XMM register and the second source, which can be either an XMM register or a 128-bit memory location, to form a single operand partition consisting of eight 32-bit single-precision floating-point values.

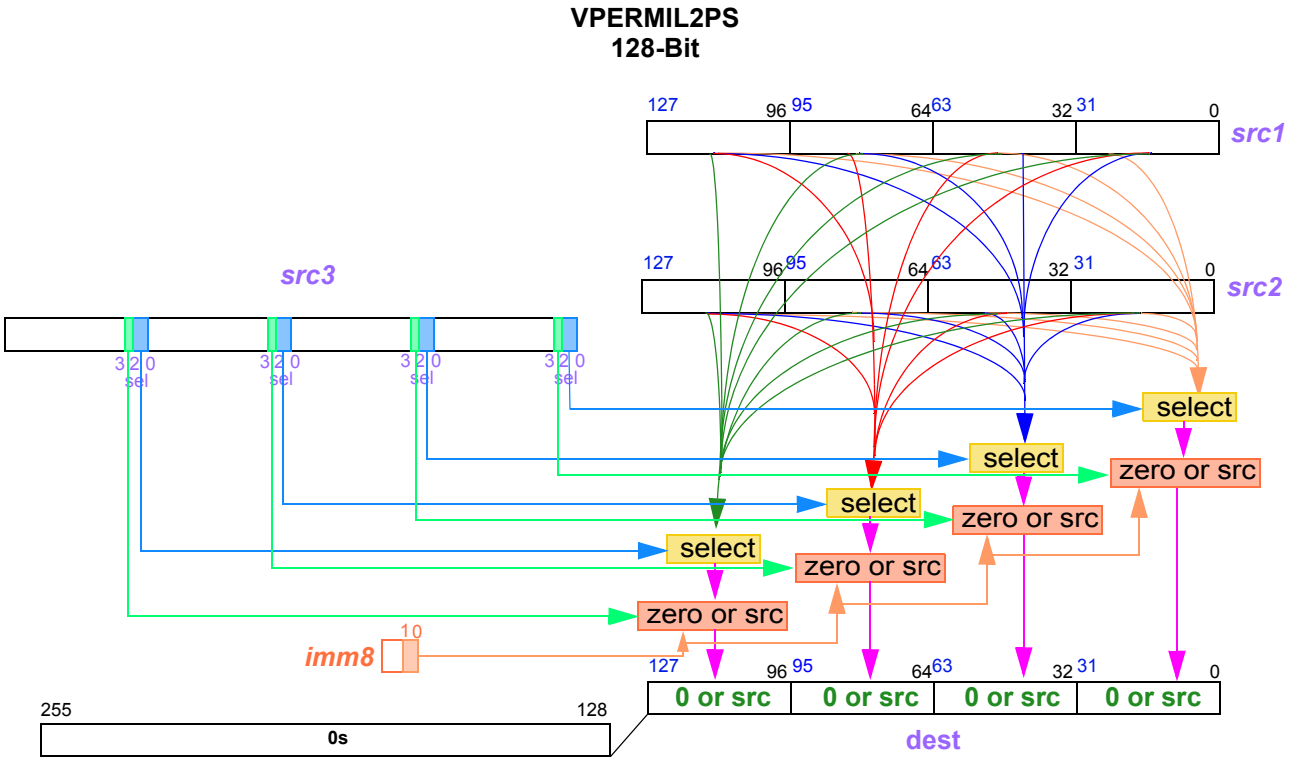
The 256-bit version forms two source operands by concatenation of the lower 128-bits of the first source with the lower 128-bits of the second source and concatenation of the upper 128-bits of the first source with the upper 128-bits of the second source, which forms two operand partitions each consisting of eight 32-bit single precision floating-point values. The first source is a YMM register and the second source is either a YMM register or a 256-bit memory location.

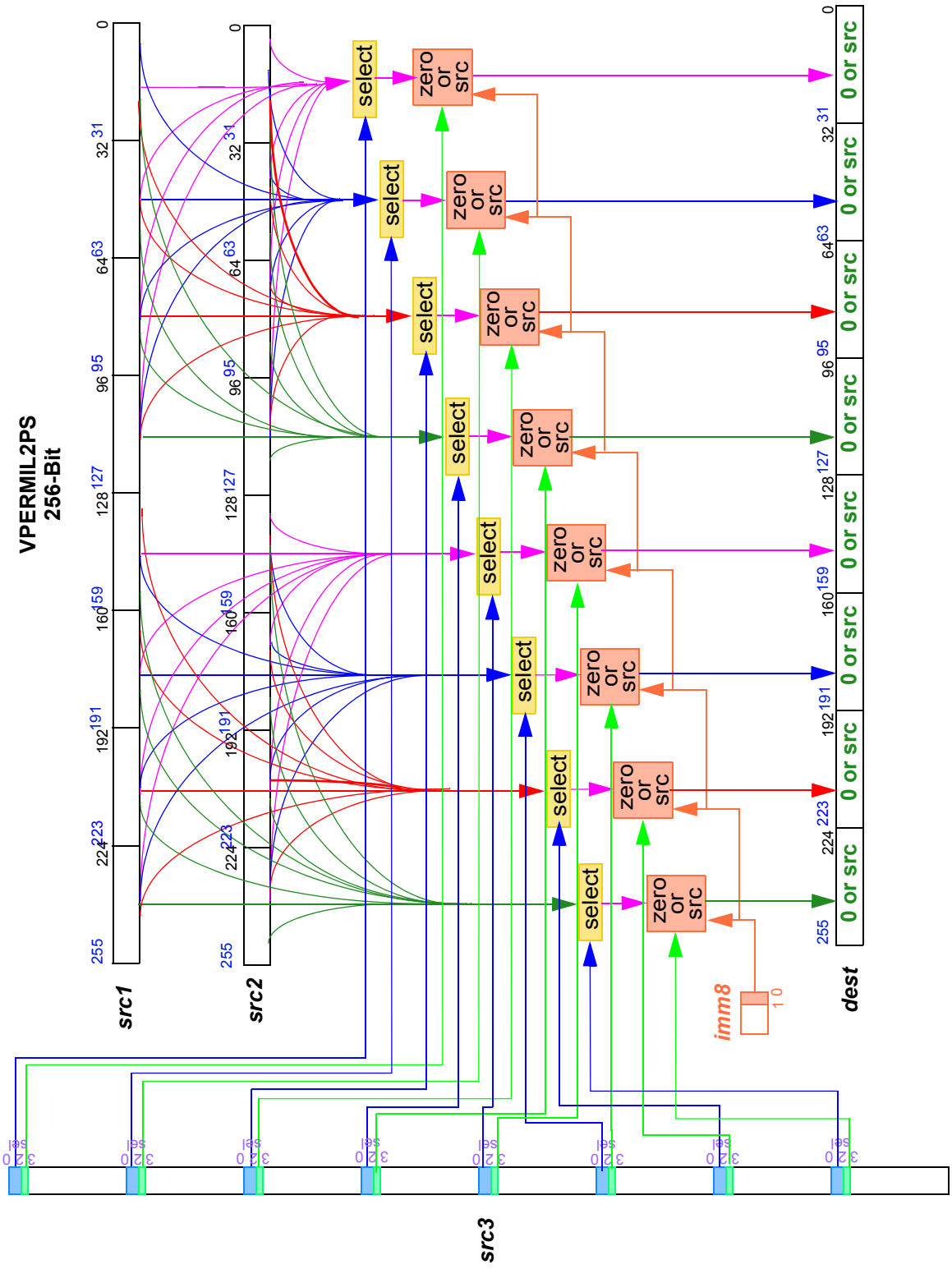
If VEX.W is 0, the second source is either an XMM or a YMM register or memory location and the third source is a register. If VEX.W is 1, the second source is a register and the third source is a register or a memory location. (See "Source Register Select," above for a discussion of the use of the immediate byte to select the second or third source register.)

Destination Register. The destination is always either an XMM register or a YMM register, depending on the vector size, as determined by the value of VEX.L. When the destination is a 128-bit XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPERMIL2PS instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPERMIL2PS *xmm1*, *xmm2, xmm3*/*mem128, xmm4, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{xsrc1}}$.0.00	48 /r imm8
VPERMIL2PS *xmm1*, *xmm2, xmm3, xmm4*/*mem128, imm8*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{xsrc1}}$.0.00	48 /r imm8
VPERMIL2PS *ymm1*, *ymm2, ymm3*/*mem256, ymm4, imm8*	C4	$\overline{\text{RXB}}$.03	0.$\overline{\text{ysrc1}}$.1.00	48 /r imm8
VPERMIL2PS *ymm1*, *ymm2, ymm3, ymm4*/*mem256, imm8*	C4	$\overline{\text{RXB}}$.03	1.$\overline{\text{ysrc1}}$.1.00	48 /r imm8


VPERMIL2PS
128-Bit
127
96
95
64
63
32
31
0
src1
src2
src3
3 2 0
sel
select
zero or src
imm8
1 0
255
128
0s
0 or src
dest


VPERMIL2PS
256-Bit
src1
src2
src3
dest
imm8
select
zero or src
0 or src

Related Instructions

VPERM2F128, VPERMIL2PD, VPERMIL2PS, VPERMILPD, VPERMILPS, VPPERM

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.

VPHADDBD Packed Horizontal Add Signed Byte to Signed Doubleword

Adds four successive 8-bit signed integer values from the source and packs the sign-extended results of the additions in the corresponding doubleword in the destination.

This instruction takes two operands:

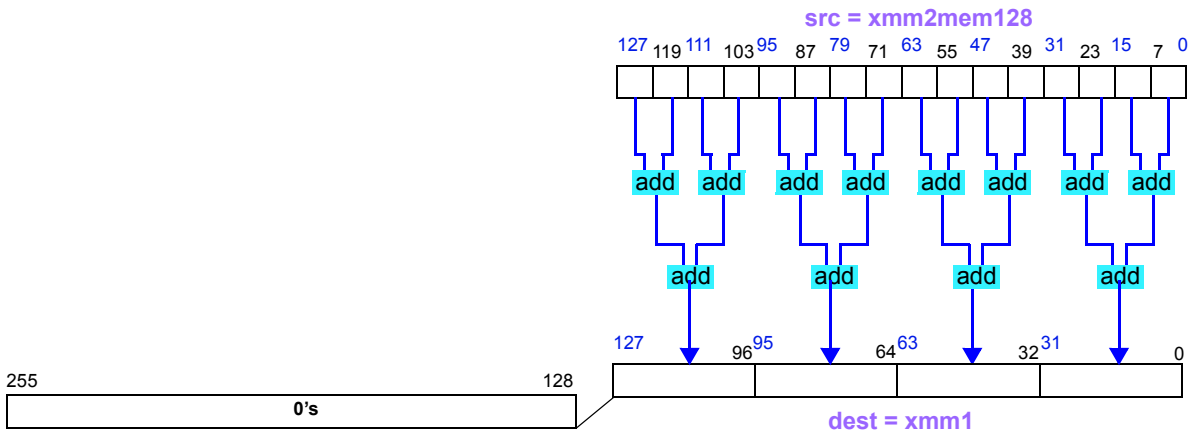
VPHADDBD *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDBD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDBD *xmm1*, *xmm2*/*mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C2 /r

VPHADDBD



Related Instructions

VPHADDBW, VPHADDBQ, VPHADDWD, VPHADDWQ, VPHADDDQ

rFLAGS Affected

None

MXCSR FLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDBQ Packed Horizontal Add Signed Byte to Signed Quadword

Adds eight successive 8-bit signed integer values from the source and packs the sign-extended results of the additions in the corresponding quadword in the destination.

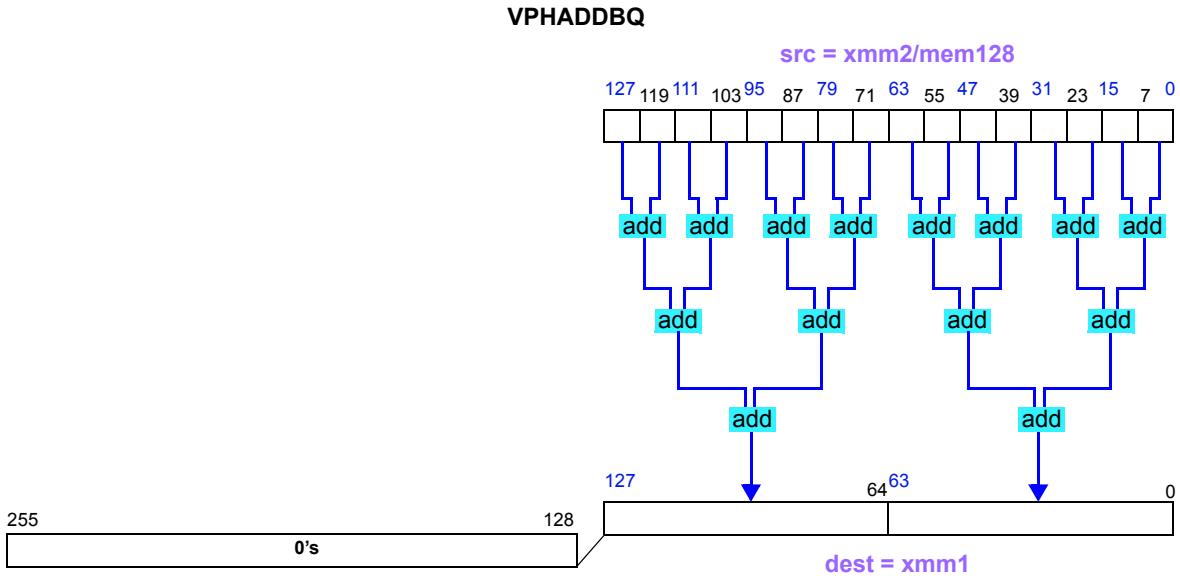
This instruction takes two operands:

VPHADDBQ *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDBQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDBQ *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	C3 /r



Related Instructions

VPHADDBW, VPHADDBD, VPHADDWD, VPHADDWQ, VPHADDDQ

rFLAGS Affected

None

MXCSR FLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDBW Packed Horizontal Add Signed Byte to Signed Word

Adds each adjacent pair of 8-bit signed integer values from the source and packs the sign-extended 16-bit integer result of each addition in the corrresponding word element of the destination.

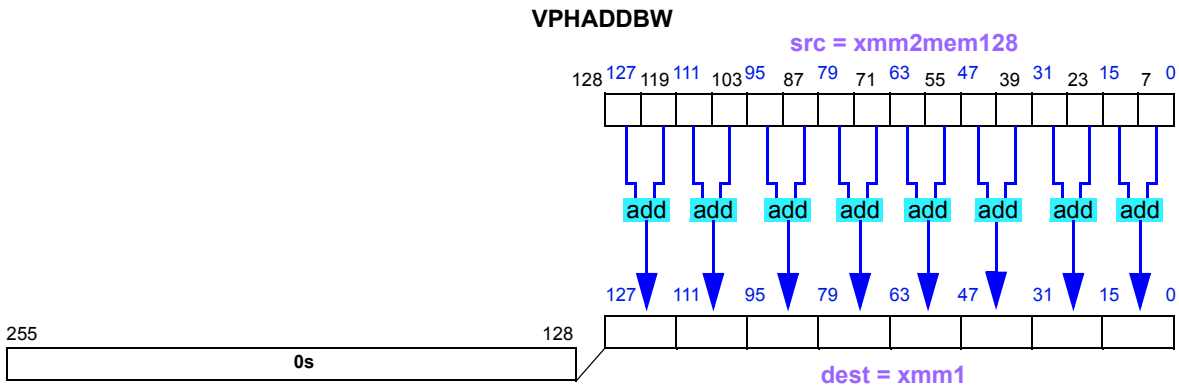
This instruction takes two operands:

VPHADDBW *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination XMM register is written, the upper 128 bits are cleared to zeros.

The PHADDBW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDDQ Packed Horizontal Add Signed Doubleword to Signed Quadword

Adds each adjacent pair of signed doubleword integer values in the source and packs the sign-extended sums of each additions in the corresponding quadword in the destination register.

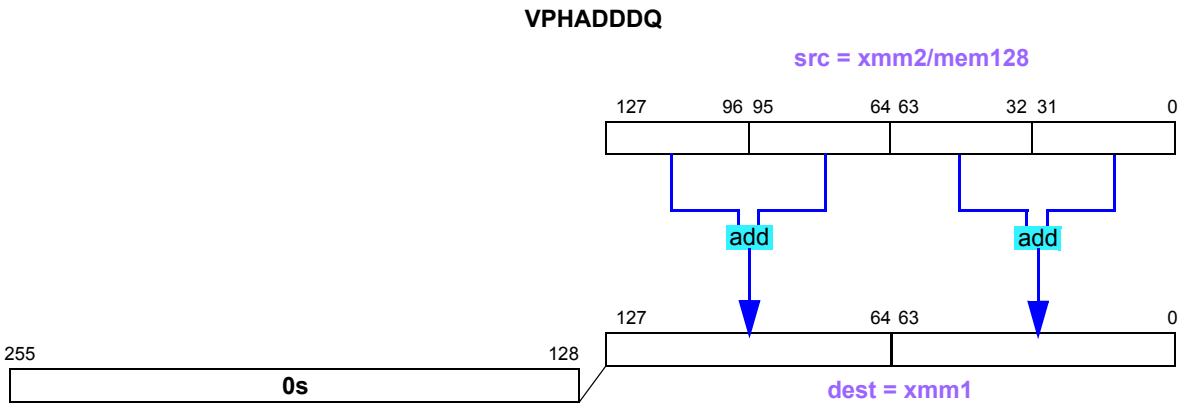
This instruction takes two operands:

VPHADDDQ *dest*, *src*

The source is an XMM register or 128-bit memory location and the destination is an XMM register. . When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDDQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDUBD Packed Horizontal Add Unsigned Byte to Doubleword

Adds four successive 8-bit unsigned integer values from the source and packs the results of the additions in the corresponding doubleword in the destination.

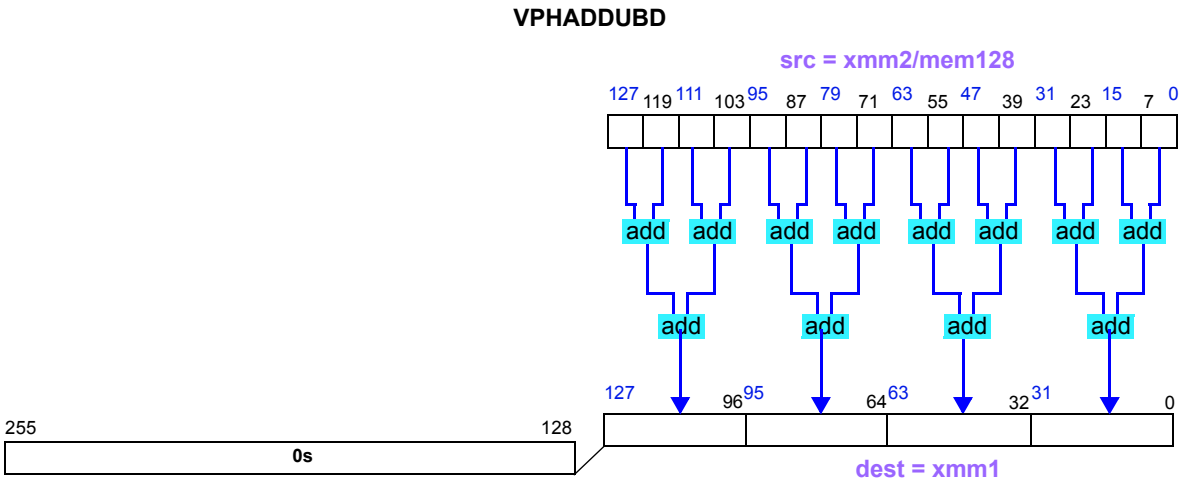
This instruction takes two operands:

VPHADDUBD *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDUBD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDUBQ Packed Horizontal Add Unsigned Byte to Quadword

Adds eight successive 8-bit unsigned integer values from the second source and packs the results of the additions in the corresponding quadword in the destination.

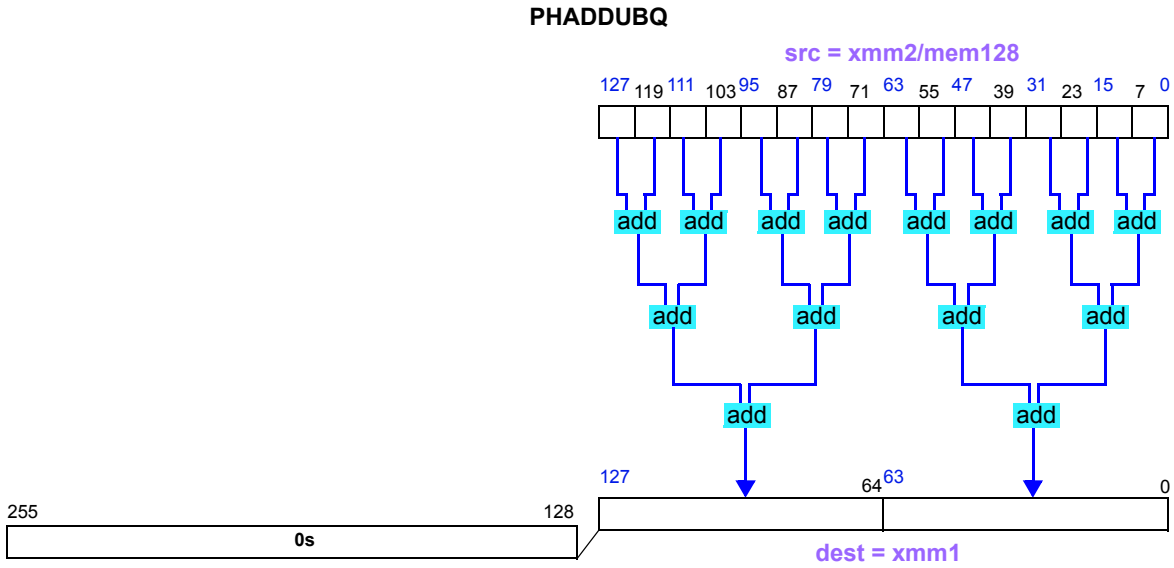
This instruction takes two operands:

VPHADDUBQ *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The PHADDUBQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDUBW Packed Horizontal Add Unsigned Byte to Word

Adds each adjacent pair of 8-bit unsigned integer values from the source and packs the 16-bit integer results of each addition in the corresponding word in the destination.

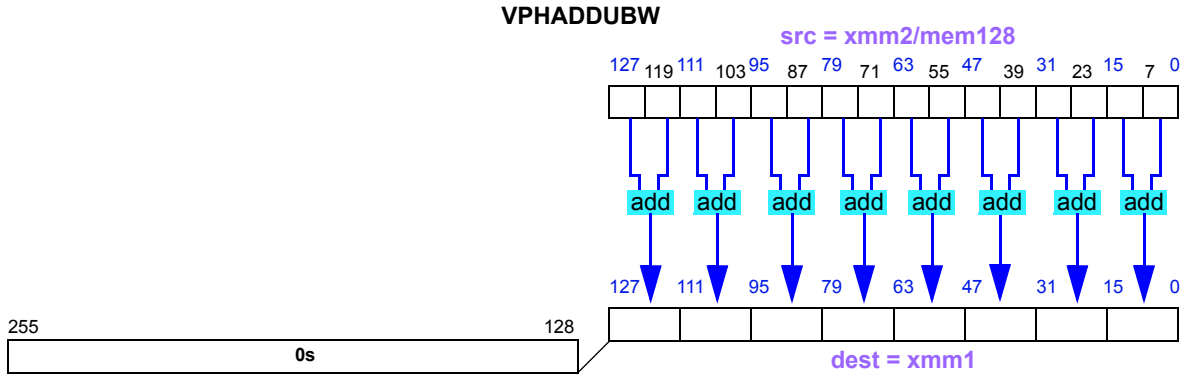
This instruction takes two operands:

VPHADDUBW *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDUBW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDUBWD *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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDUDQ Packed Horizontal Add Unsigned Doubleword to Quadword

Adds each adjacent pair of 32-bit unsigned integer values from the source and packs the results of each addition in the corresponding quadword in the destination.

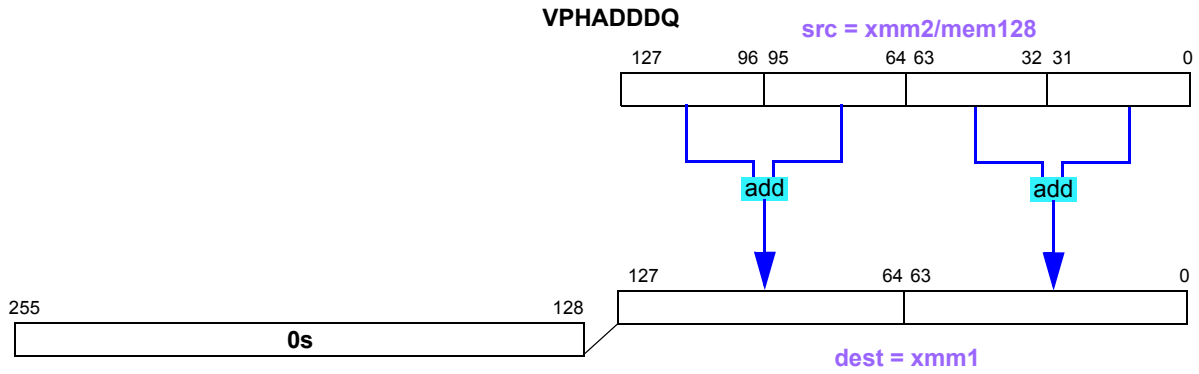
This instruction takes two operands:

VPHADDUDQ *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDUDQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDUWD Packed Horizontal Add Unsigned Word to Doubleword

Adds each adjacent pair of 16-bit unsigned integer values from the source and packs the results of each addition in the corresponding doubleword in the destination.

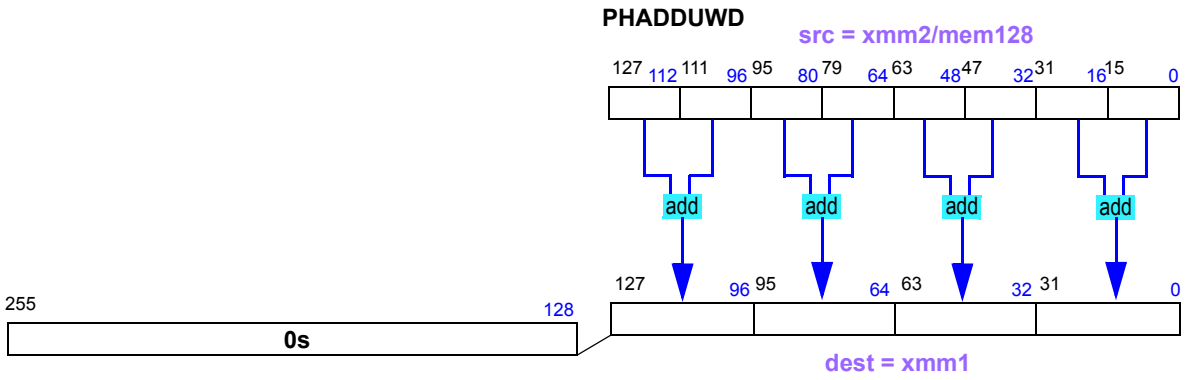
This instruction takes two operands:

VPHADDUWD *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDUWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHADDUWD *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	D6 /r



Related Instructions

VPHADDUBW, VPHADDUBD, VPHADDUBQ, VPHADDUWQ, VPHADDUDQ

rFLAGS Affected

None

MXCSR FLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDUWQ Packed Horizontal Add Unsigned Word to Quadword

Adds four successive 16-bit unsigned integer values from the source and packs the results of the additions in the corresponding quadword element in the destination.

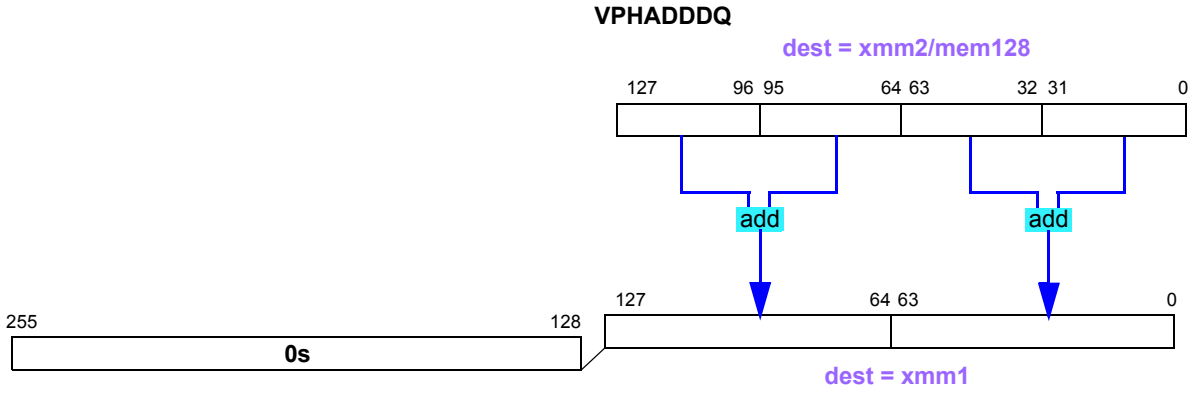
This instruction takes two operands:

VPHADDUWQ *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHADDUWQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPHADDWD Packed Horizontal Add Signed Word to Signed Doubleword

Adds each adjacent pair of 16-bit signed integer values from the source and packs the sign-extended results of the addition in the corresponding doubleword in the destination).

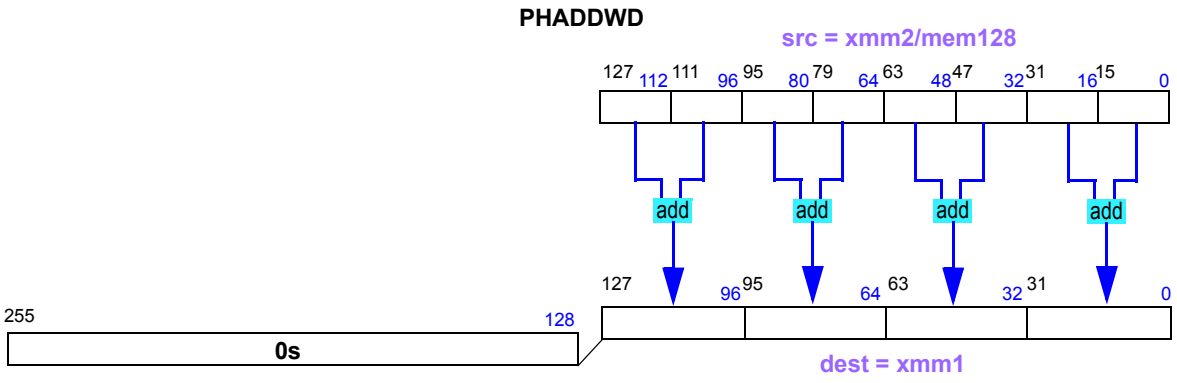
This instruction takes two operands:

VPHADDWD *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination XMM register is written, the upper 128 bits or the corresponding YMM register are cleared to zeros.

The VPHADDWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

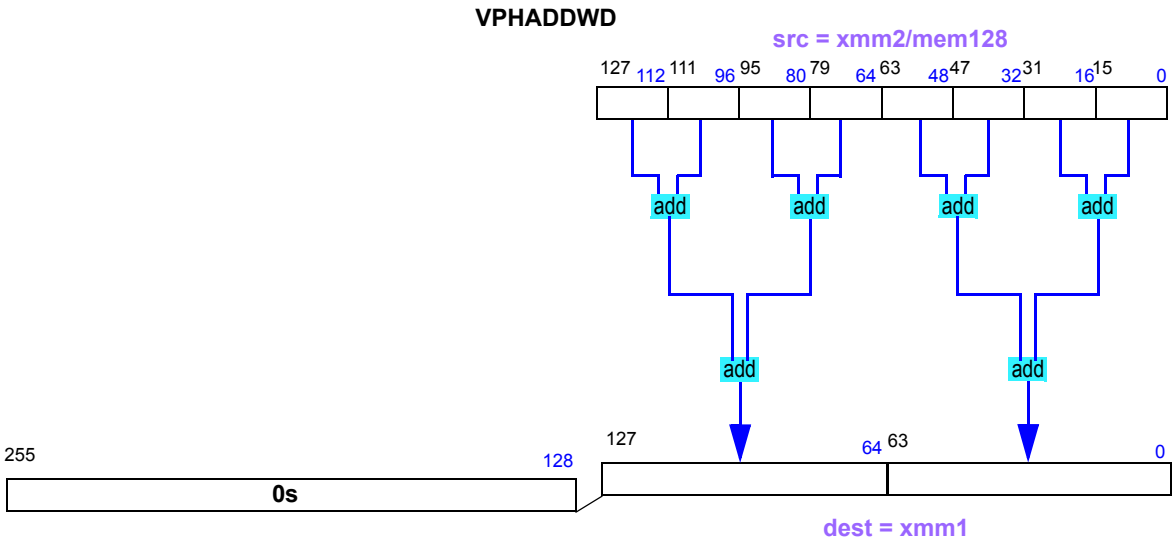
VPHADDWQ Packed Horizontal Add Signed Word to Signed Quadword

Adds four successive 16-bit signed integer values from the second source and packs the sign-extended results of each addition in the corresponding quadword in the destination.

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeroes.

The VPHADDWQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

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 results of each subtraction in the destination.

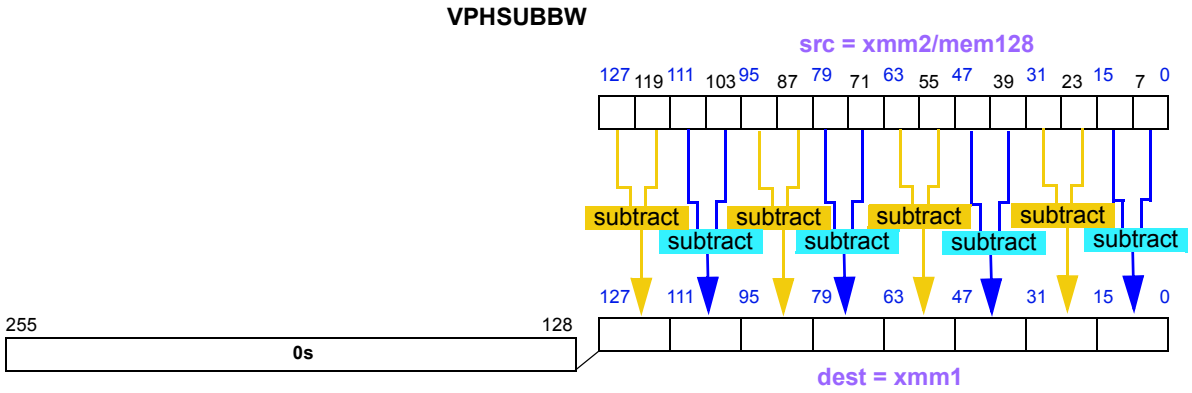
This instruction takes two operands:

VPHSUBBW *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHSUBBW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBBW *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	E1 /r



Related Instructions

VPHSUBWD, VPHSUBDQ

rFLAGS Affected

None

MXCSR FLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

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 result of each subtraction in the corresonding quadword element of the destination.

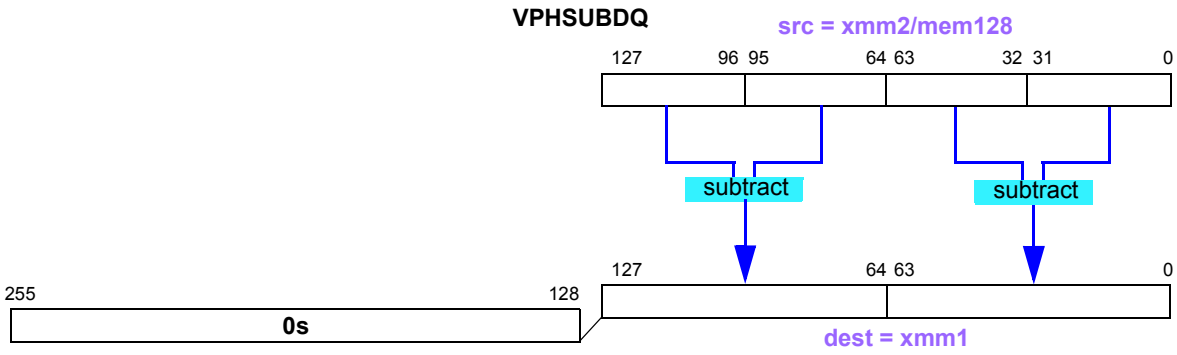
This instruction takes two operands:

VPHSUBDQ *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHSUBDQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBDQ *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	E3 /r



Related Instructions

VPHSUBBW, VPHSUBWD

rFLAGS Affected

None

MXCSR FLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

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 from the source and packs the sign-extended 32-bit integer result of each subtraction in the destination.

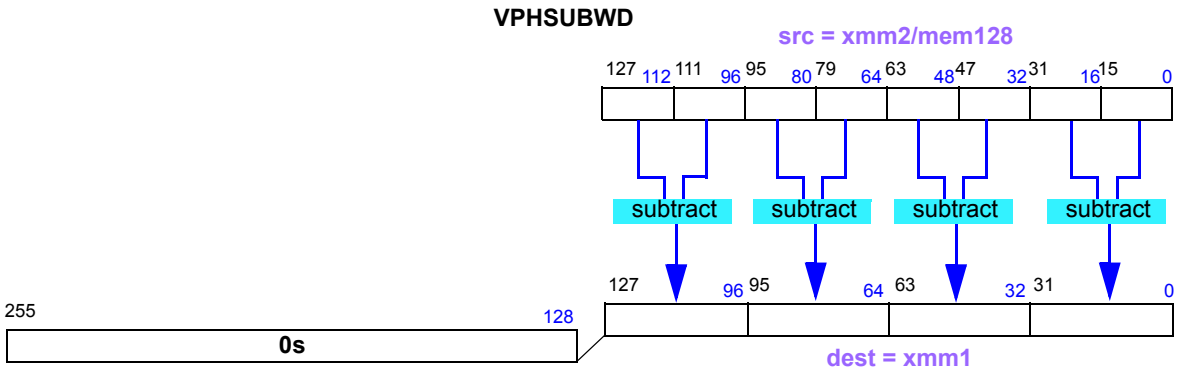
This instruction takes two operands:

VPHSUBWD *dest*, *src*

The destination is an XMM register and the source is an XMM register or 128-bit memory location. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The VPHSUBWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPHSUBWD *xmm1*, *xmm2/mem128*	8F	$\overline{\text{RXB}}$.09	0.1111.0.00	E2 /r



Related Instructions

VPHSUBBW, VPHSUBDQ

rFLAGS Affected

None

MXCSR FLAGS Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSDD Packed Multiply Accumulate Signed Doubleword to Signed Doubleword

Multiplies each packed 32-bit signed integer value in the first source by the corresponding packed 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the corresponding packed 32-bit signed integer value in the third source. The four resulting 32-bit sums are stored in the destination.

The VPMACSDD instruction requires four operands:

VPMACSDD dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

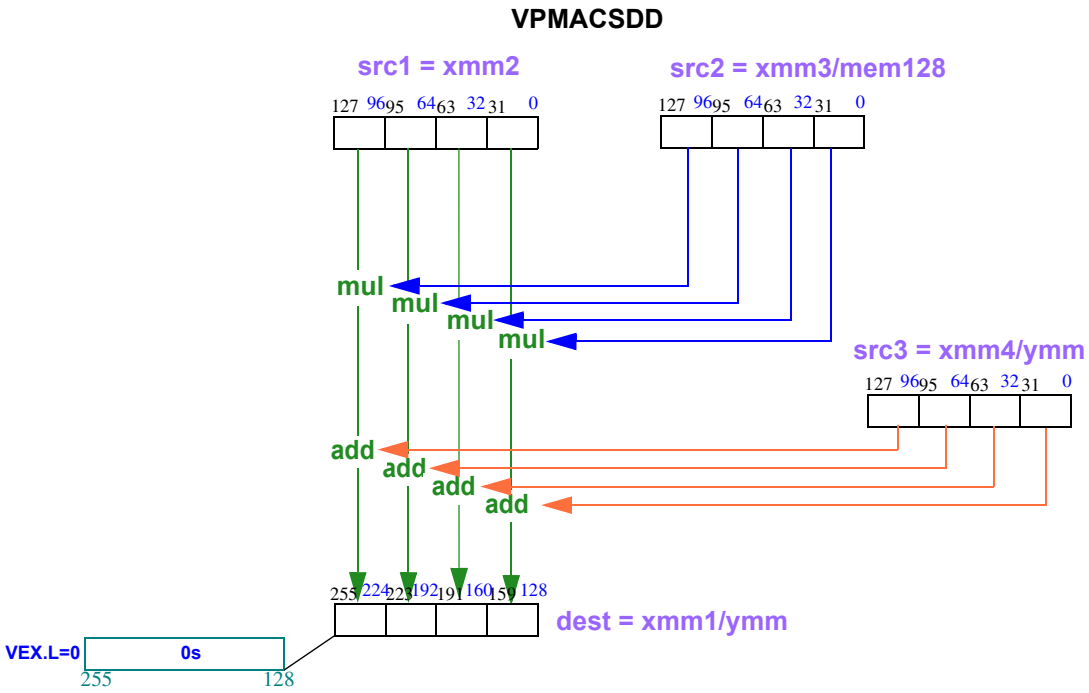
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When the third source designates the same XMM register as the destination, the XMM register behaves as an accumulator.

No saturation is performed on the sum. If 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. If 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.

The VPMACSDD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSDD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	9E /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSDQH Packed Multiply Accumulate Signed High Doubleword to Signed Quadword

Multiplies the second 32-bit signed integer value of the first source by the second 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the low-order 64-bit signed integer value in the third source. Simultaneously, multiplies the fourth 32-bit signed integer value of the first source by the fourth 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the second 64-bit signed integer value in the third source.The results are written to the destination register.

The VPMACSDQH instruction requires four operands:

VPMACSDQH dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

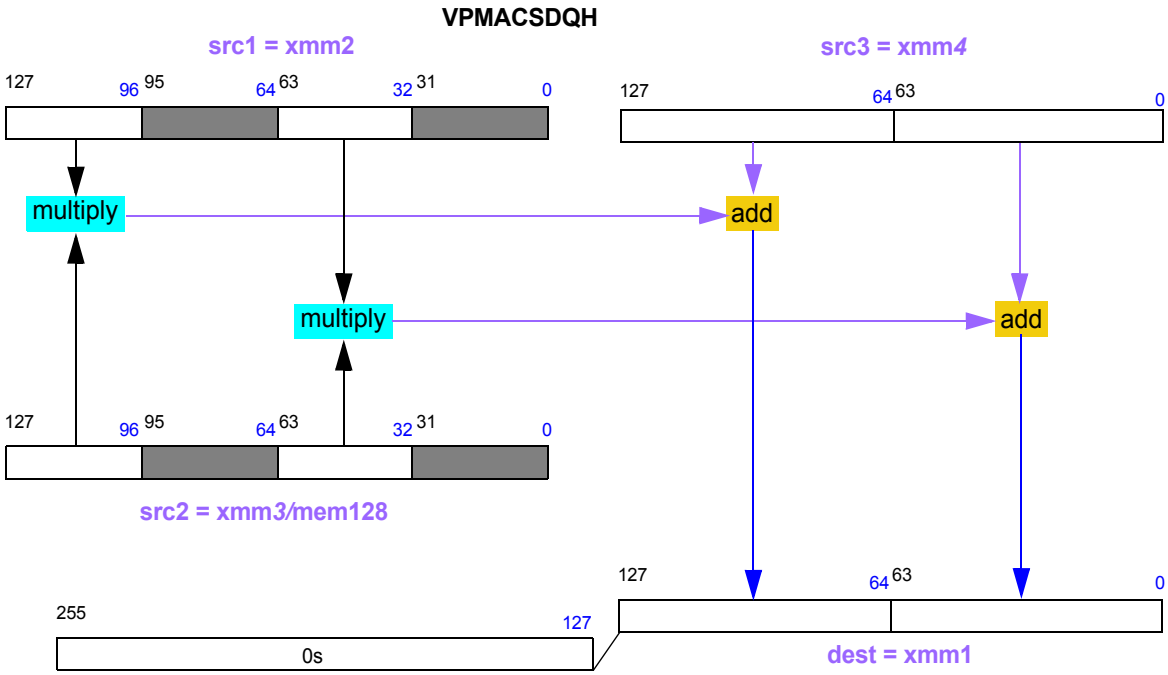
The first source (*src1*) is an XMM register specified by the XOP.vvvv field; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When the third source designates the same XMM register as th destination register, the XMM register behaves as an accumulator.

No saturation is performed on the sum. If the result of the add overflows, the carry is ignored (neither the overflow nor carry bit in rFLAGS is set).

The VPMACSDQH instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSDQH *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	9F /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

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 low-order 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the low-order 64-bit signed integer value in the third source. Simultaneously, multiplies the third 32-bit signed integer value of the first source by the corresponding 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the second 64-bit signed integer value in the third source. The results are written to the destination (register.

TheVPMACSDQL instruction requires four operands:

VPMACSDQL dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination register is a YMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

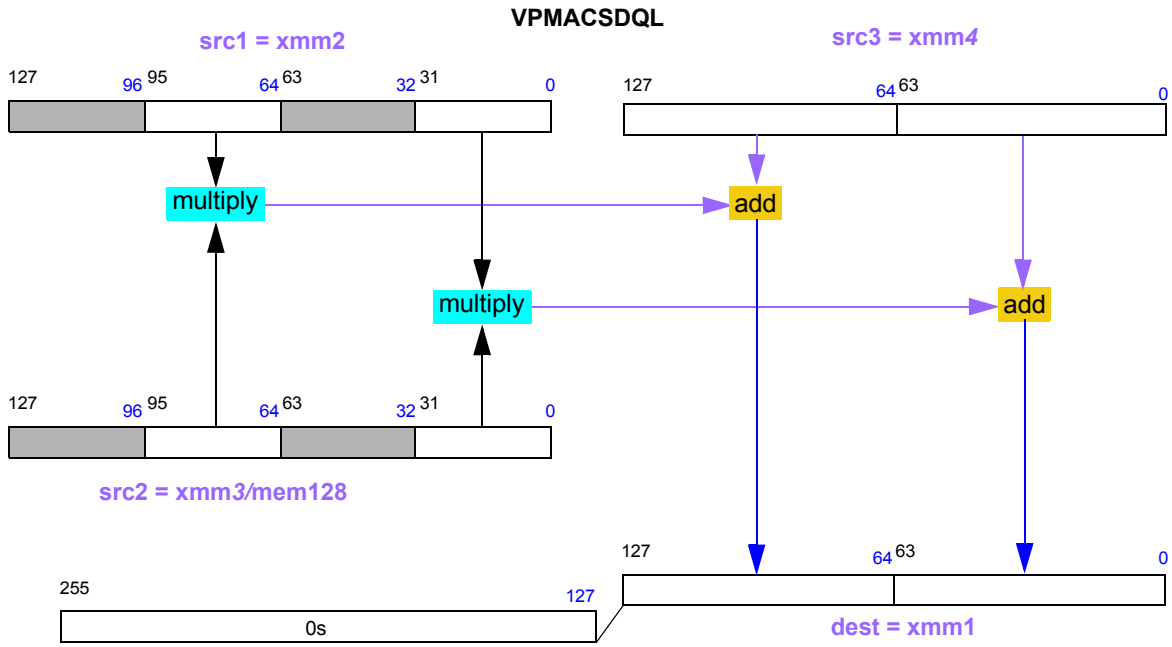
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

No saturation is performed on the sum. If 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 in the destination.

The VPMACSDQL instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSDQL *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	97 /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSSDD Packed Multiply Accumulate Signed Doubleword to Signed Doubleword with Saturation

Multiplies each packed 32-bit signed integer value in the first source by the corresponding packed 32-bit signed integer value in the second source, then adds each 64-bit signed integer product to the corresponding packed 32-bit signed integer value in the third source. The saturated results are written to the destination register.

The VPMACSSDD instruction requires four operands:

VPMACSSDD dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

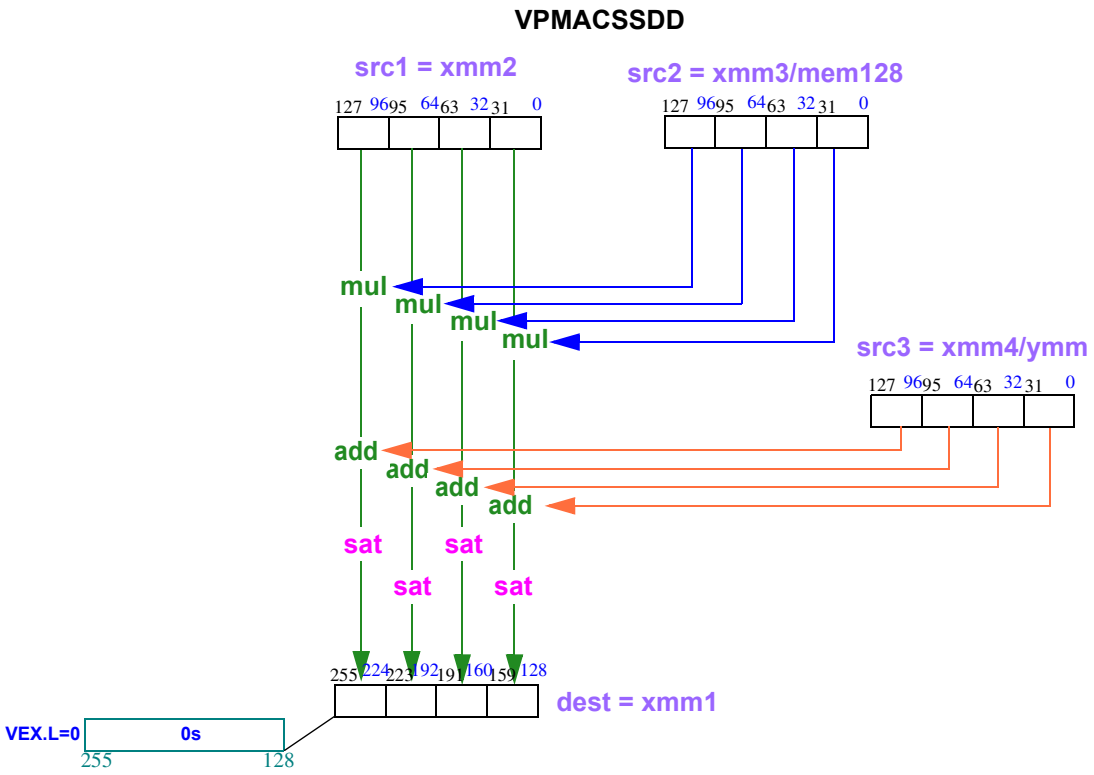
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

Out of range results of the addition are saturated to fit into a signed 32-bit integer. For each packed value in the destination, if the value is larger than the largest signed 32-bit integer, it is saturated to 7FFF_FFFFh, and if the value is smaller than the smallest signed 32-bit integer, it is saturated to 8000_0000h.

The VPMACSSDD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSSDD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	8E /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSSDQH Packed Multiply Accumulate Signed High Doubleword to Signed Quadword with Saturation

Multiplies the second 32-bit signed integer value of the first source by the second 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the low-order 64-bit signed integer value in the third source. Simultaneously, multiplies the fourth 32-bit signed integer value of the first source by the fourth 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the high-order 64-bit signed integer value in the third source. The saturated results are written to the destination register.

The PMACSSDQH instruction requires four operands:

VPMACSSDQH dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

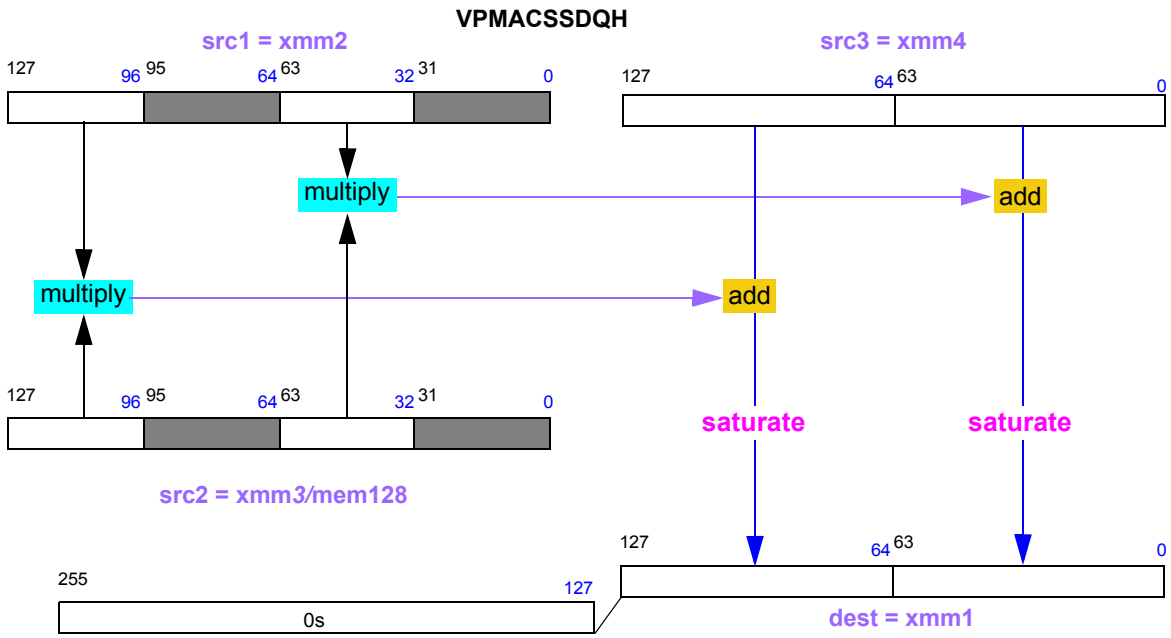
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

Out of range results of the addition are saturated to fit into a signed 64-bit integer. For each packed value in the destination, if the value is larger than the largest signed 64-bit integer, it is saturated to 7FFF_FFFF_FFFF_FFFFh, and if the value is smaller than the smallest signed 64-bit integer, it is saturated to 8000_0000_0000_0000h.

The VPMACSSDQH instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSSDQH xmm1, xmm2, xmm3/mem128, xmm4	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	8F /r is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSSDQL Packed Multiply Accumulate Signed Low Doubleword to Signed Quadword with Saturation

Multiplies the low-order 32-bit signed integer value of the first source by the low-order 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the low-order 64-bit signed integer value in the third source. Simultaneously, multiplies the third 32-bit signed integer value of the first source by the third 32-bit signed integer value in the second source, then adds the 64-bit signed integer product to the high-order 64-bit signed integer value in the third source. The saturated results are written to the destination register.

The VPMACSSDQL instruction requires four operands:

VPMACSSDQL dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) register is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

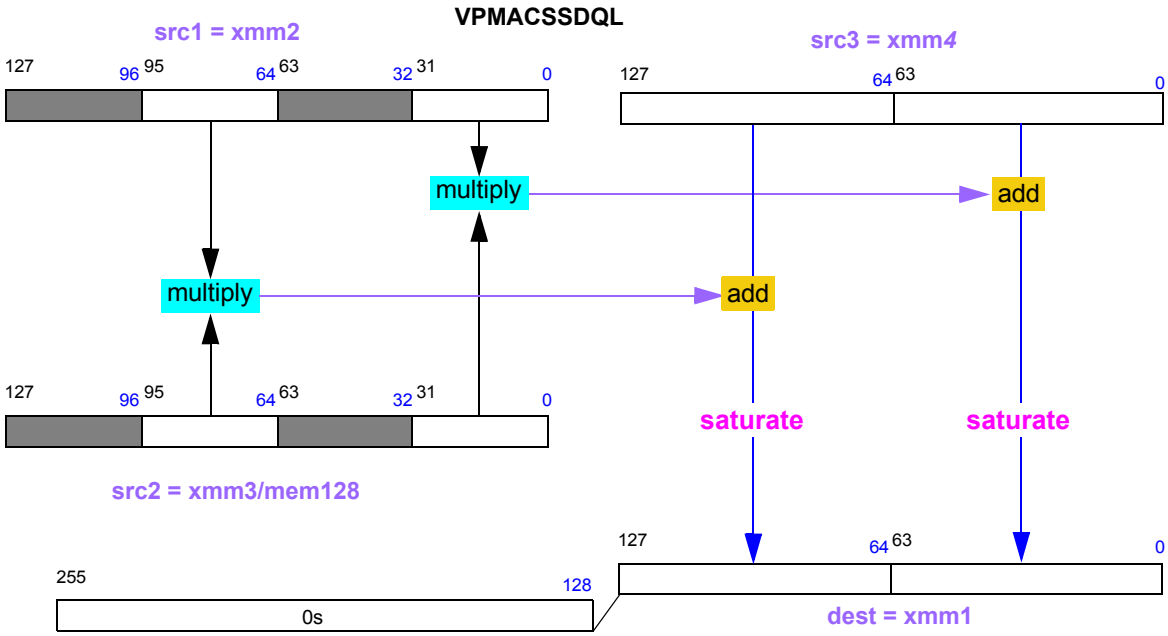
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

Out of range results of the addition are saturated to fit into a signed 64-bit integer. For each packed value in the destination, if the value is larger than the largest signed 64-bit integer, it is saturated to 7FFF_FFFF_FFFF_FFFFh, and if the value is smaller than the smallest signed 64-bit integer, it is saturated to 8000_0000_0000_0000h.

The VPMACSSDQL instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
PMACSSDQL *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	87 /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSSWD Packed Multiply Accumulate Signed Word to Signed Doubleword with Saturation

Multiplies the odd-numbered packed 16-bit signed integer values in the first source by the corresponding packed 16-bit signed integer values in the second source, then adds the 32-bit signed integer products to the corresponding packed 32-bit signed integer values in the third source. The saturated results are written to the destination register.

The VPMACSSWD instruction requires four operands:

VPMACSSWD dest, src1, src2, src3 *dest = src1* src2 + src3*

The destinationa (*dest*) is an XMM register addressed by the MODRM.reg field. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

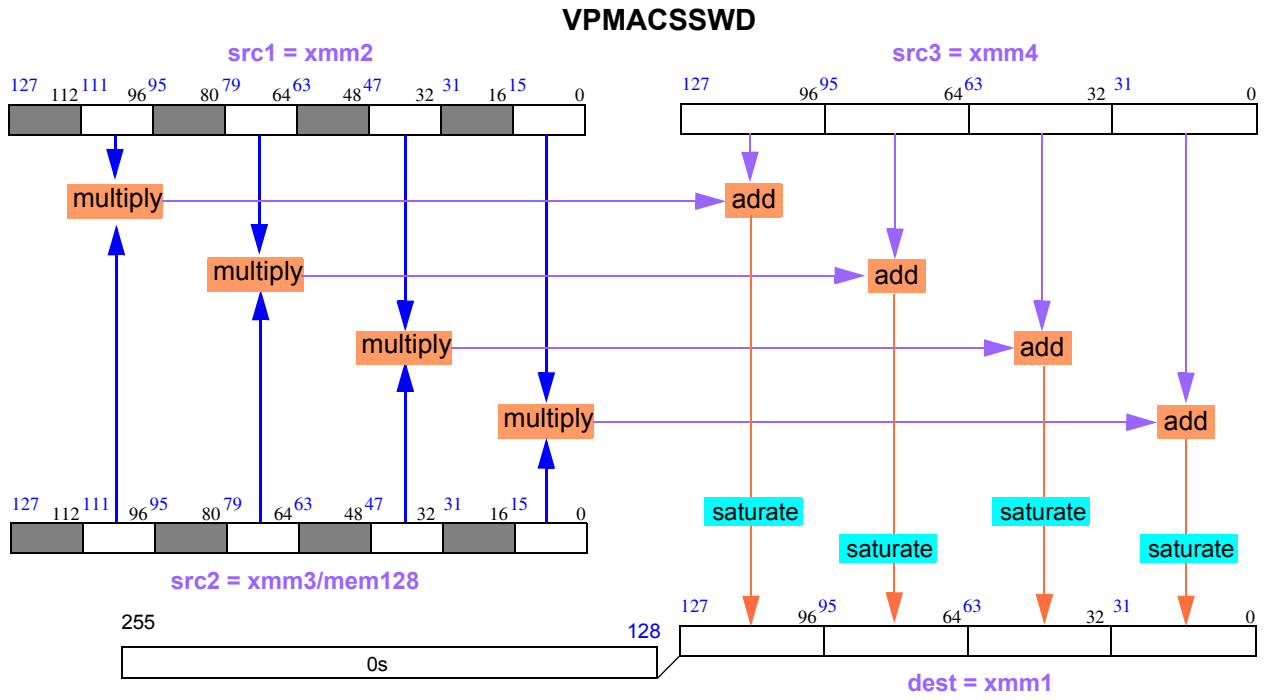
The first source (*src1*) is an XMM register specified by the XOP.vvvv field; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

Out of range results of the addition are saturated to fit into a signed 32-bit integer. For each packed value in the destination, if the value is larger than the largest signed 32-bit integer, it is saturated to 7FFF_FFFFh, and if the value is smaller than the smallest signed 32-bit integer, it is saturated to 8000_0000h.

The VPMACSSWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	86 /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSSWW Packed Multiply Accumulate Signed Word to Signed Word with Saturation

Multiplies each packed 16-bit signed integer value in the first source by its corresponding packed 16-bit signed integer value in the second source, then adds the 32-bit signed integer products to the corresponding packed 16-bit signed integer value in the third source. The saturated results are written to the destination register.

The VPMACSSWW instruction requires four operands:

VPMACSSWW dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination register is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

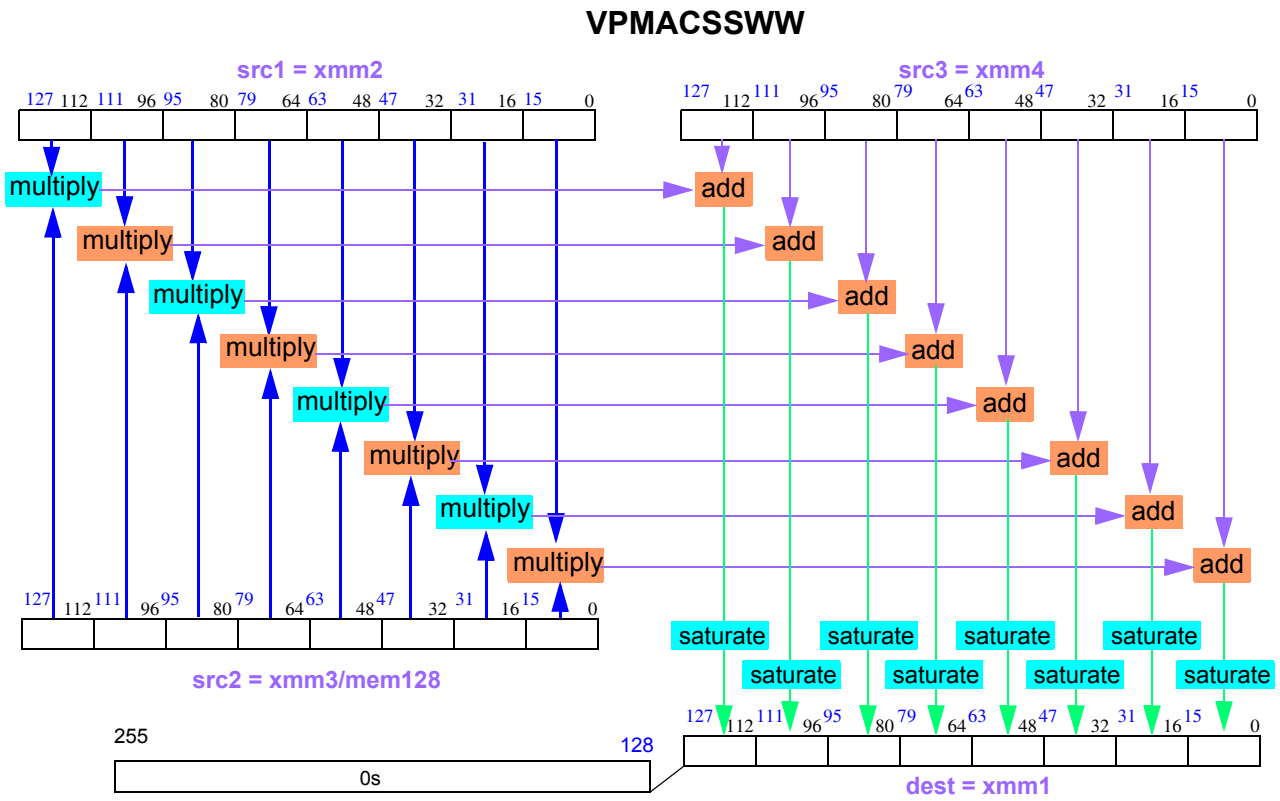
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* and *dest* designate the same XMM register, this register behaves as an accumulator.

Out of range results of the addition are saturated to fit into a signed 16-bit integer. For each packed value in the destination, if the value is larger than the largest signed 16-bit integer, it is saturated to 7FFFh, and if the value is smaller than the smallest signed 16-bit integer, it is saturated to 8000h.

The VPMACSSWW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
PMACSSWW *xmm1*, *xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	85 /r /is4



Related Instructions

VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL,VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSWD Packed Multiply Accumulate Signed Word to Signed Doubleword

Multiplies each odd-numbered packed 16-bit signed integer value in the first source by the corresponding packed 16-bit signed integer value in the second source, then adds the 32-bit signed integer products to the corresponding packed 32-bit signed integer value in the third source. The four results are written to the destination register.

The VPMACSWD instruction requires four operands:

VPMACSWD dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) register is an XMM register addressed by the MODRM.reg field. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

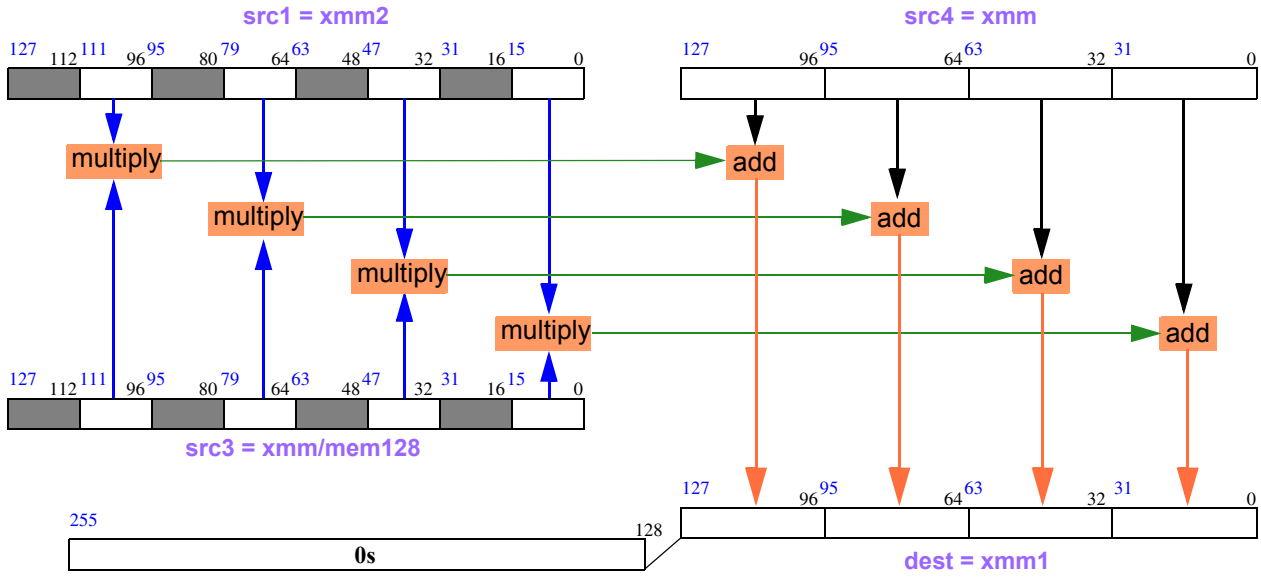
When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

If 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 in the destination.

The VPMACSWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSWD *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	96 /r /is4

VPMACSWD



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSSDD, VPMACSDO, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMACSWW Packed Multiply Accumulate Signed Word to Signed Word

Multiplies each packed 16-bit signed integer value in the first source by the corresponding packed 16-bit signed integer value in the second source, then adds each 32-bit signed integer product to the corresponding packed 16-bit signed integer value in the third source. The eight results are written to the destination register.

The VPMACSWW instruction requires four operands:

VPMACSWW dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the destination XMM register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

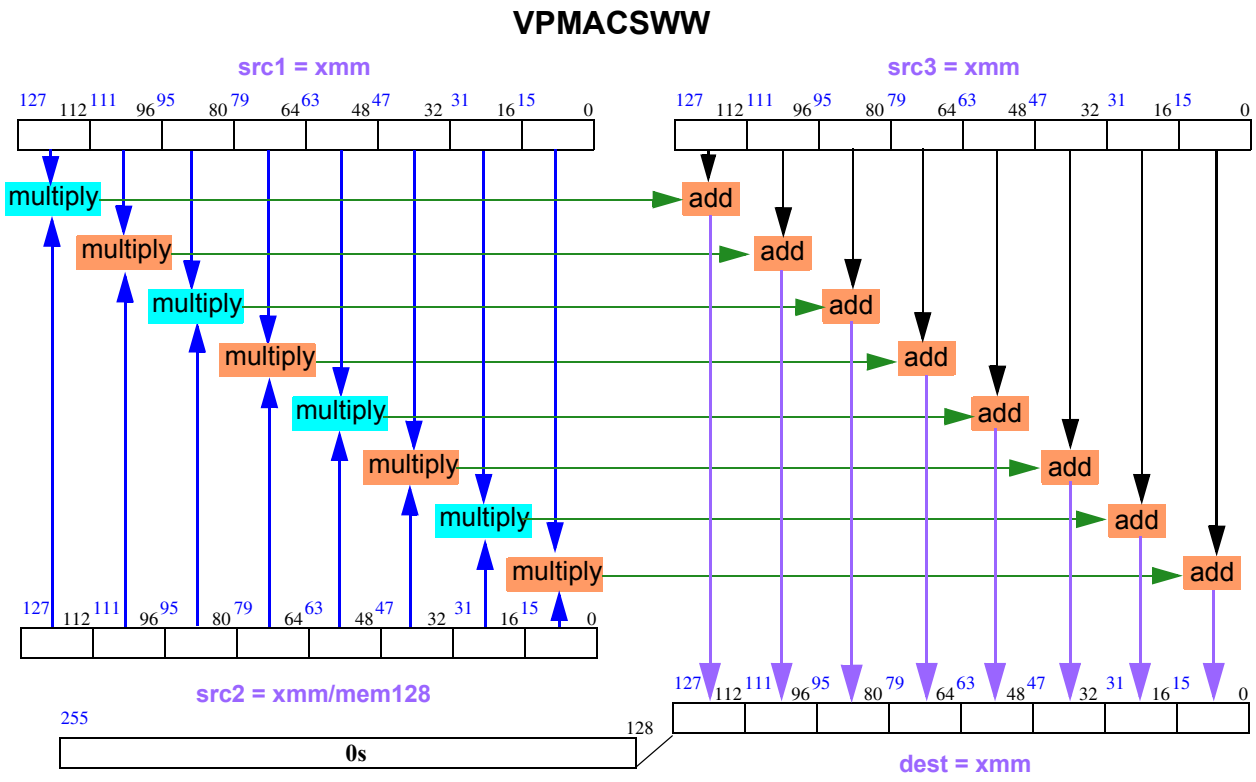
The first source (*src1*) is an XMM register specified by the XOP.vvvv fields; the second source (*src2*) is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source (*src3*) is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

No saturation is performed on the sum. If the result of the multipliplication causes non-zero values to be set in the upper 16 bits of the 32 bit result, they are ignored. If 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 in the destination.

The VPMACSWW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMACSWW *xmm1, xmm2, xmm3/mem128, xmm4*	8F	RXB.08	0.$\overline{\text{xsrc1}}$.0.00	95 /r /is4



Related Instructions

VPMACSSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMADCSSWD Packed Multiply, Add and Accumulate Signed Word to Signed Doubleword with Saturation

Multiplies each packed 16-bit signed integer value in the first source by the corresponding packed 16-bit signed integer value in 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 in the third source. The four results are written to the destination (accumulator) register.

The VPMADCSSWD instruction requires four operands:

VPMADCSSWD dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination register is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

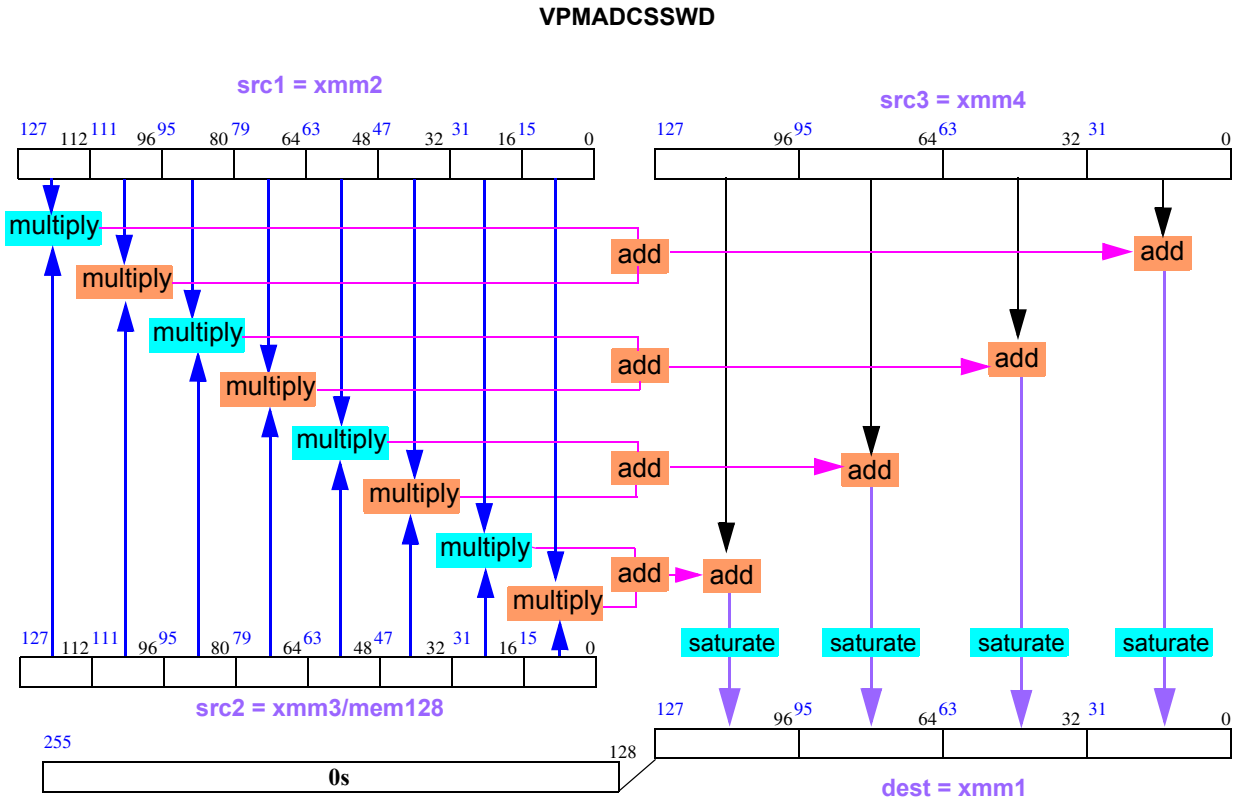
The first source is an XMM register specified by the XOP.vvvv fields; the second source is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

Out of range results of the addition are saturated to fit into a signed 32-bit integer. For each packed value in the destination, if the value is larger than the largest signed 32-bit integer, it is saturated to 7FFF_FFFFh, and if the value is smaller than the smallest signed 32-bit integer, it is saturated to 8000_0000h.

The VPMADCSSWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPMADCSSWD *xmm1*, *xmm2*, *xmm3*/*mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	A6 /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPMADCSWD Packed Multiply Add and Accumulate Signed Word to Signed Doubleword

Multiplies each packed 16-bit signed integer value in the first source by the corresponding packed 16-bit signed integer value in the second source, then adds the 32-bit signed integer products of the even-odd adjacent words together and adds their sum to the corresponding packed 32-bit signed integer values in the third source. The four results are written to the destination register.

The VPMADCSWD instruction requires four operands:

VPMADCSWD dest, src1, src2, src3 *dest = src1* src2 + src3*

The destination register is an XMM register addressed by the MODRM.reg field. When the destination register is written, the upper 128 bits of the corresponding YMM register are cleared to zeros.

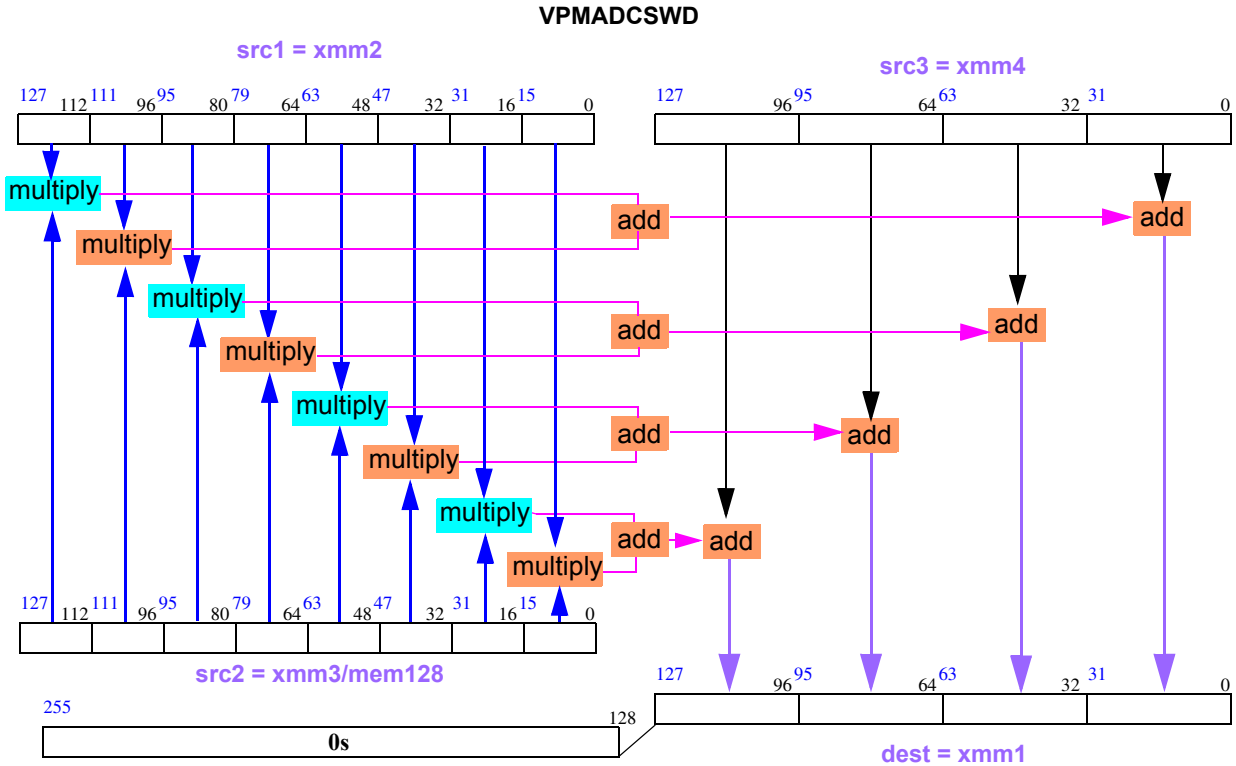
The first source is an XMM register specified by the XOP.vvvv fields, the second source is an XMM register or 128-bit memory location specified by the MODRM.rm field; and the third source is an XMM register specified by imm8[7:4].

When *src3* designates the same XMM register as the *dest* register, the XMM register behaves as an accumulator.

No saturation is performed on the sum. If 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.

The VPMADCSWD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
PMADCSWD *xmm1*, *xmm2*, *xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.08	0.$\overline{\text{xsrc1}}$.0.00	B6 /r /is4



Related Instructions

VPMACSSWW, VPMACSWW, VPMACSSWD, VPMACSWD, VPMACSSDD, VPMACSDD, VPMACSSDQL, VPMACSSDQH, VPMACSDQL, VPMACSDQH, VPMADCSSWD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.W was set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPPERM Packed Permute Bytes

Selects 16 of the 32-packed bytes in the two sources and optionally applies a logical transformation to each selected byte before it is stored to its specified position in the destination XMM register.

The VPPERM instruction requires four operands:

VPPERM dest, src1, src2, selector

The 32-byte source consists of the concatenation of the second source (*src2*) and the first source (*src1*).

The third source operand (*selector*) contains 16 control bytes, each specifying the source byte and the logical operation to perform on that byte before the result is stored in the destination register. The order of the bytes in the destination register is the same as that of the selector bytes in the selector.

The *src1* operand is always an XMM register specified by XOP.vvvv

This instruction supports operand source configuration using XOP.W. When XOP.W is 0, *src2* is an XMM register or 128-bit memory location specified by MODRM.rm and *selector* is an XMM register specified by imm8[7:4]. When XOP.W is 1, *src2* is an XMM register specified by imm8[7:4] and *selector* is an XMM register or 128-bit memory location specified by MODRM.rm.

The destination (*dest*) is always an XMM register specified by MODRM.reg. When the result operand is written to the *dest* XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

For each byte of the 16-byte result, the corresponding *selector* byte is used as follows:

• Bits 4:0 of the *selector* selects the source byte to move from the 32 bytes from *src2*:*src1*.

• Bits 7:5 of the *selector* selects the logical operation to perform on the selected operand.

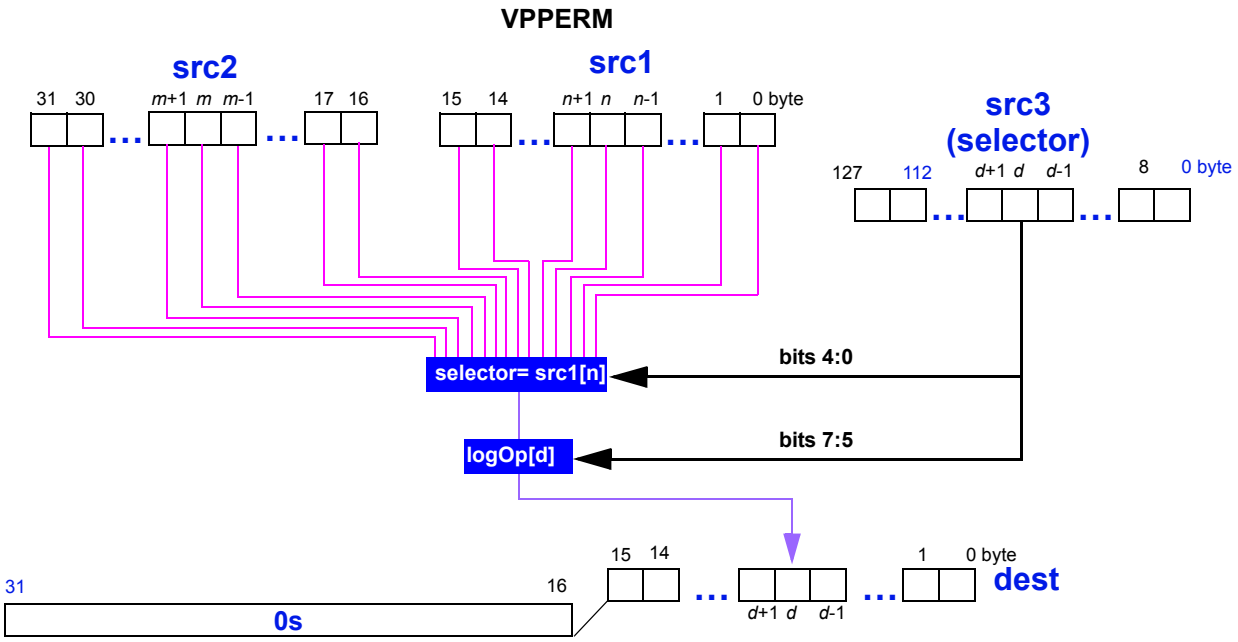
Table 2-13. VPPERM Control Byte

Bits	Description			
7:5	Defines the logical operation performed on the selected operand.			
	OP	**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		
	101	FFh		
	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.		
4:0	Selects the source byte to permute			
	Selector	**Source Selected**	**Selector**	**Source Selected**
	00000	*src1*[7:0]	10000	*src2*[7:0]
	00001	*src1*[15:8]	10001	*src2*[15:8]
	00010	*src1*[23:16]	10010	*src2*[23:16]
	00011	*src1*[31:24]	10011	*src2*[31:24]
	00100	*src1*[39:32]	10100	*src2*[39:32]
	00101	*src1*[47:40]	10101	*src2*[47:40]
	00110	*src1*[55:48]	10110	*src2*[55:48]
	00111	*src1*[63:56]	10111	*src2*[63:56]
	01000	*src1*[71:64]	11000	*src2*[71:64]
	01001	*src1*[79:72]	11001	*src2*[79:72]
	01010	*src1*[87:80]	11010	*src2*[87:80]
	01011	*src1*[95:88]	11011	*src2*[95:88]
	01100	*src1*[103:96]	11100	*src2*[103:96]
	01101	*src1*[111:104]	11101	*src2*[111:104]
	01110	*src1*[119:112]	11110	*src2*[119:112]
	01111	*src1*[127:120]	11111	*src2*[127:120]

TheVPPERM instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic **Encoding**

	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPPERM *xmm1, xmm2, xmm3, xmm4/mem128*	8F	$\overline{\text{RXB}}$.8	1.$\overline{\text{xsrc1}}$.0.00	A3 /r is4
VPPERM *xmm1, xmm2, xmm3/mem128, xmm4*	8F	$\overline{\text{RXB}}$.8	0.$\overline{\text{xsrc1}}$.0.00	A3 /r is4



Related Instructions

VPSHUFHW, VPSHUFD, VPSHUFLW, VPSHUFW, VPERMIL2PS, VPERMIL2PD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled while MXCSR.MM=1.

VPROTB Packed Rotate Bytes

Rotates each byte of the source by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding byte of the destination.

There are two versions of the instruction, depending on the source of the count byte used for each 8-bit shift:

- *VPROTB dest, src, fixed-count*
- *VPROTB dest, src, variable-count-src*

The destination (*dest*) operand of both versions of this instruction is an XMM register addressed by the MODRM.reg field. When the result of the rotation is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *fixed-count* version of this instruction rotates each byte element of the source (*src*) by the number of bits specified by the immediate *fixed-count* byte. All byte elements of the source are rotated by the same number of bits. The source is a 128-bit XMM register or memory location addressed by the MODRM.rm field.

The *variable-count-src* version of this instruction rotates each byte of the source by the amount specified in the corresponding byte element in the *variable-count-src*, which is an XMM register or 128-bit memory location.

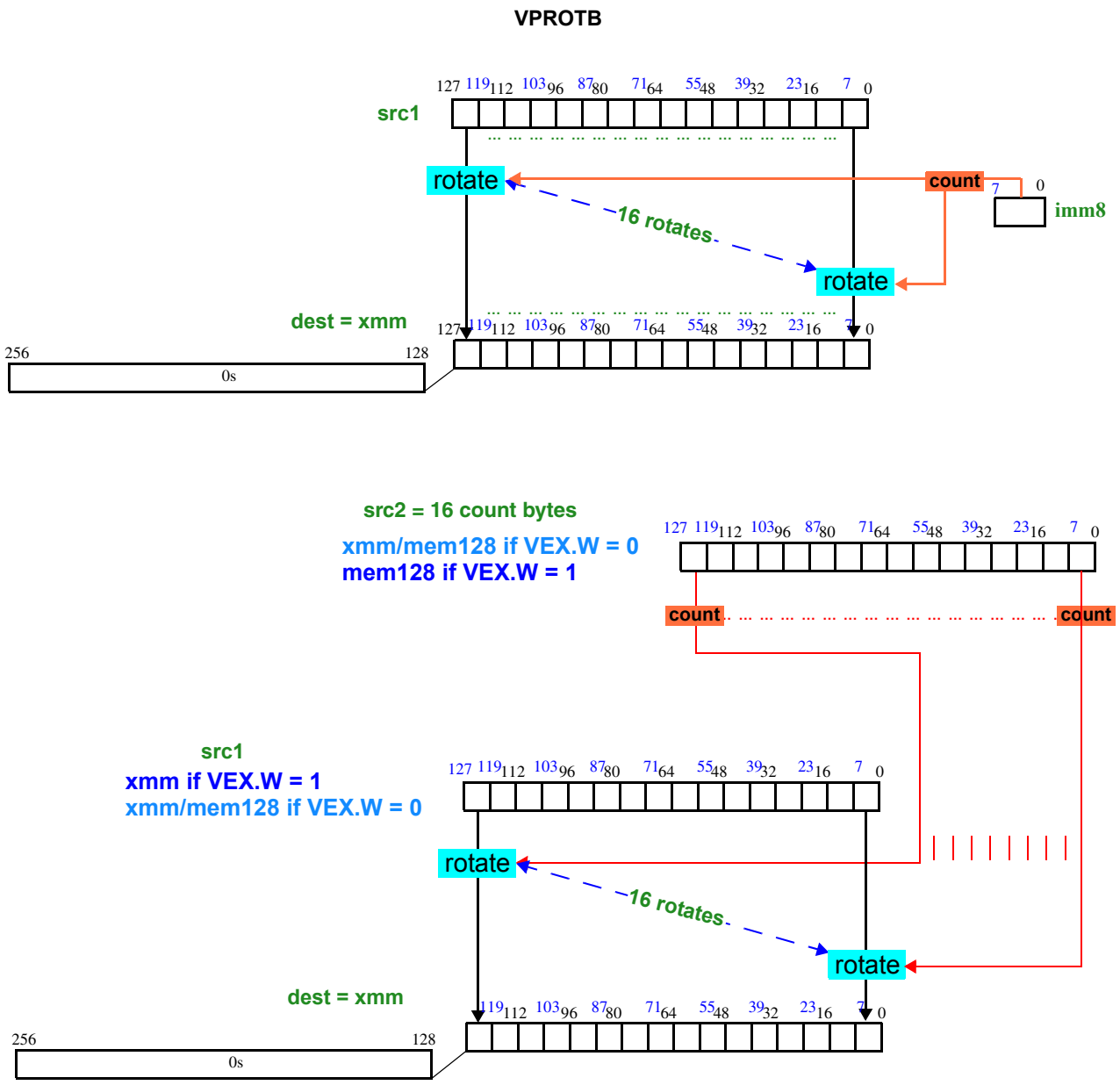
The *src* and *variable-count-src* are configurable through XOP.W. If XOP.W is 0, the *variable-count-src* is an XMM register specified by XOP.vvvv and the *src* operand is an XMM register or 128-bit memory location specified by MODRM.rm. If XOP.W is 1, the *variable-count-src* is an XMM register or 128-bit memory location specified by MODRM.rm and the *src* operand is an XMM register specified by XOP.vvvv.

If 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.

If 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.

The VPROTB instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTB *xmm1*, *xmm2*/*mem128, xmm8*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	90 /r
VPROTB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b for immediate count form of instruction (opcode C0h).
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPROTD Packed Rotate Doublewords

Rotates each of the four doublewords of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding doubleword of the destination.

There are two variants of this instruction, depending on the source of the count byte used for each doubleword shift:

- *VPROTD dest, src, fixed-count*
- *VPROTD dest, src, variable-count*

The *dest* operand of both versions of this instruction is an XMM register addressed by the MODRM.reg field. When the 128-bit result operand is written to the *dest* register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The fixed count version of this instruction rotates each doubleword of the source operand by the number of bits specified by the immediate *fixed-count* byte operand. All doubleword elements of the source operand are rotated by the same number of bits. The *src* is anXMM register or memory location addressed by the MODRM.rm field.

The variable count version of this 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.

The *src* and *variable-count* operand vector are configurable through XOP.W. If XOP.W is 0, the *src* is an XMM register or 128-bit memory location specified by the MODRM.rm field and the *variable-count* operand vector is an XMM register specified by XOP.vvvv. If XOP.W is 1, the *src* operand is an XMM register specified by XOP.vvvv and the variable-count operand is an XMM register or 128-bit memory location specified by the MODRM.rm field.

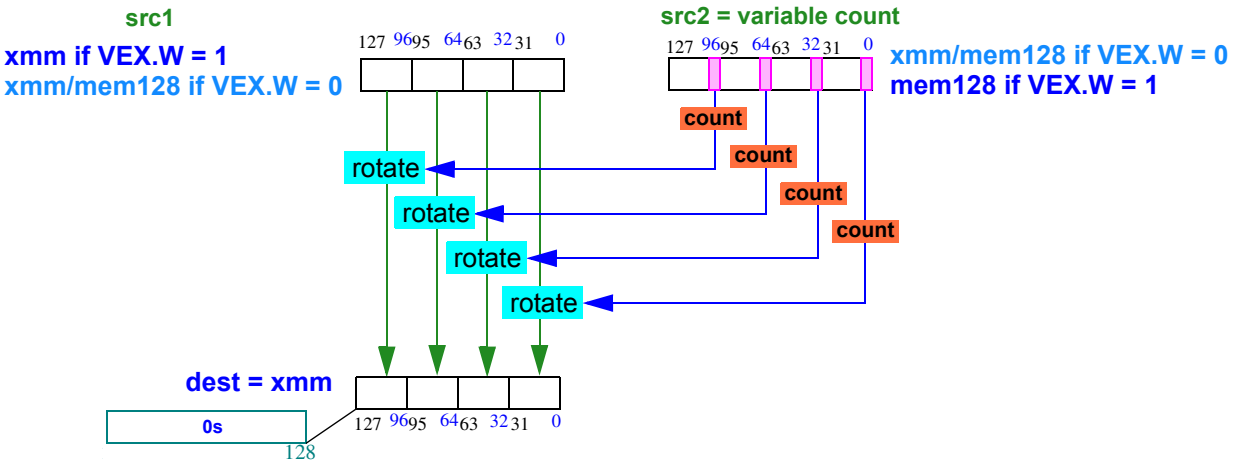
If the count value is positive, bits are rotated to the left (toward the more significant bit positions). The bits rotated out to the left of the most significant bit of each source doubleword operand are rotated back in at the right end (least-significant bit) of the doubleword.

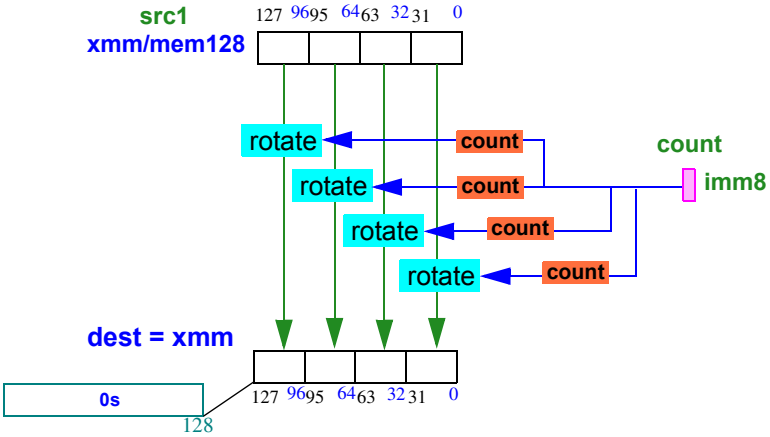
If the count value is negative, bits are rotated to the right (toward the least significant bit positions). The bits rotated to the right out of the least significant bit of each source doubleword operand are rotated back in at the left end (most-significant bit) of the doubleword.

The VPROTD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTD *xmm1*, *xmm2/mem128*, *xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	92 /r
VPROTD *xmm1*, *xmm2*, *xmm3/mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.0.00	92 /r
VPROTD *xmm1*, *xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C2 /ib

VPROTD





Related Instructions

VPROTB, VPROTW, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD, VPSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b for immediate count form of instruction (opcode C2h).
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPROTQ Packed Rotate Quadwords

Rotates each of the quadwords of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding quadword of the destination.

There are two variants of this instruction, depending on the source of the count byte used for each quadword shift:

- *VPROTQ dest, src, fixed-count*
- *VPROTQ dest, src, variable-count*

The *dest* operand of both versions of this instruction is an XMM register addressed by the MODRM.reg field. When the 128-bit result is written to the *dest* XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The fixed count version of this instruction rotates each quadword in the source by the number of bits specified by the immediate *fixed-count* byte operand. All quadword elements of the source are rotated by the same number of bits. The *src* is a 128-bit XMM register or memory location addressed by the MODRM.rm field.

The variable count version of this instruction rotates each quadword of the source by the amount specified in the low order byte of the corresponding quadword of the *variable-count* operand.

The *src* and *variable-count* are configurable through XOP.W. If XOP.W is 0, the *src* is an XMM register or 128-bit memory location specified by MODRM.rm and the *count* is an XMM register specified by XOP.vvvv. If XOP.W is 1, *src* is an XMM register specified by XOP.vvvv and the *variable-count* is an XMM register or 128-bit memory location specified by MODRM.rm.

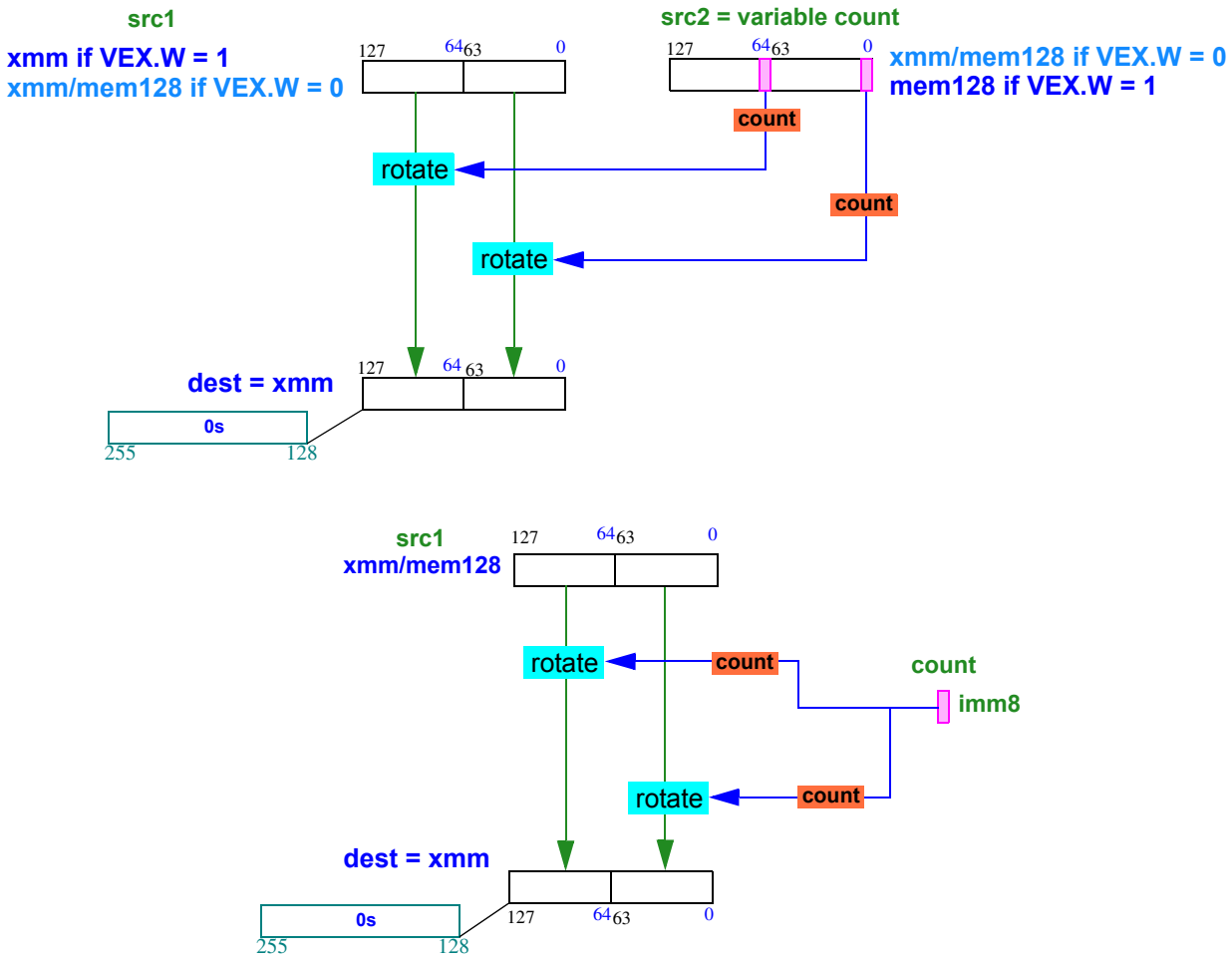
If the count value is positive, bits are rotated to the left (toward the more significant bit positions) of the operand element. The bits rotated out to the left of the most significant bit of the word element are rotated back in at the right end (least-significant bit).

If 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.

The VPROTQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTQ *xmm1*, *xmm2/mem128*, *xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	93 /r
VPROTQ *xmm1*, *xmm2*, *xmm3/mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.0.00	93 /r
VPROTQ *xmm1*, *xmm2/mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C3 /ib

VPROTQ



Related Instructions

PROTB, PROTW, PROTD, PSHLB, PSHLW, PSHLD, PSHLQ, PSHAB, PSHAW, PSHAD, PSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b for immediate count form of instruction (opcode C3h).
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPROTW Packed Rotate Words

Rotates each of the eight words of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding word of the destination.

There are two variants of this instruction, depending on the source of the count byte used for each word shift:

- *VPROTW dest, src, fixed-count*
- *VPROTW dest, src, variable-count*

The *dest* operand of both versions of this instruction is a YMM register addressed by the MODRM.reg field. When the 128-bit result operand is written to the *dest* XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The fixed count version of this instruction rotates each word of the source operand by the number of bits specified by the immediate *fixed-count* byte operand. All word elements of the source operand are rotated by the same number of bits. The *src* operand is a 128-bit YMM register or memory location addressed by the MODRM.rm field.

The variable count version of this instruction rotates each word of the source operand by the amount specified in the low order byte of the corresponding word of the *variable-count* operand.

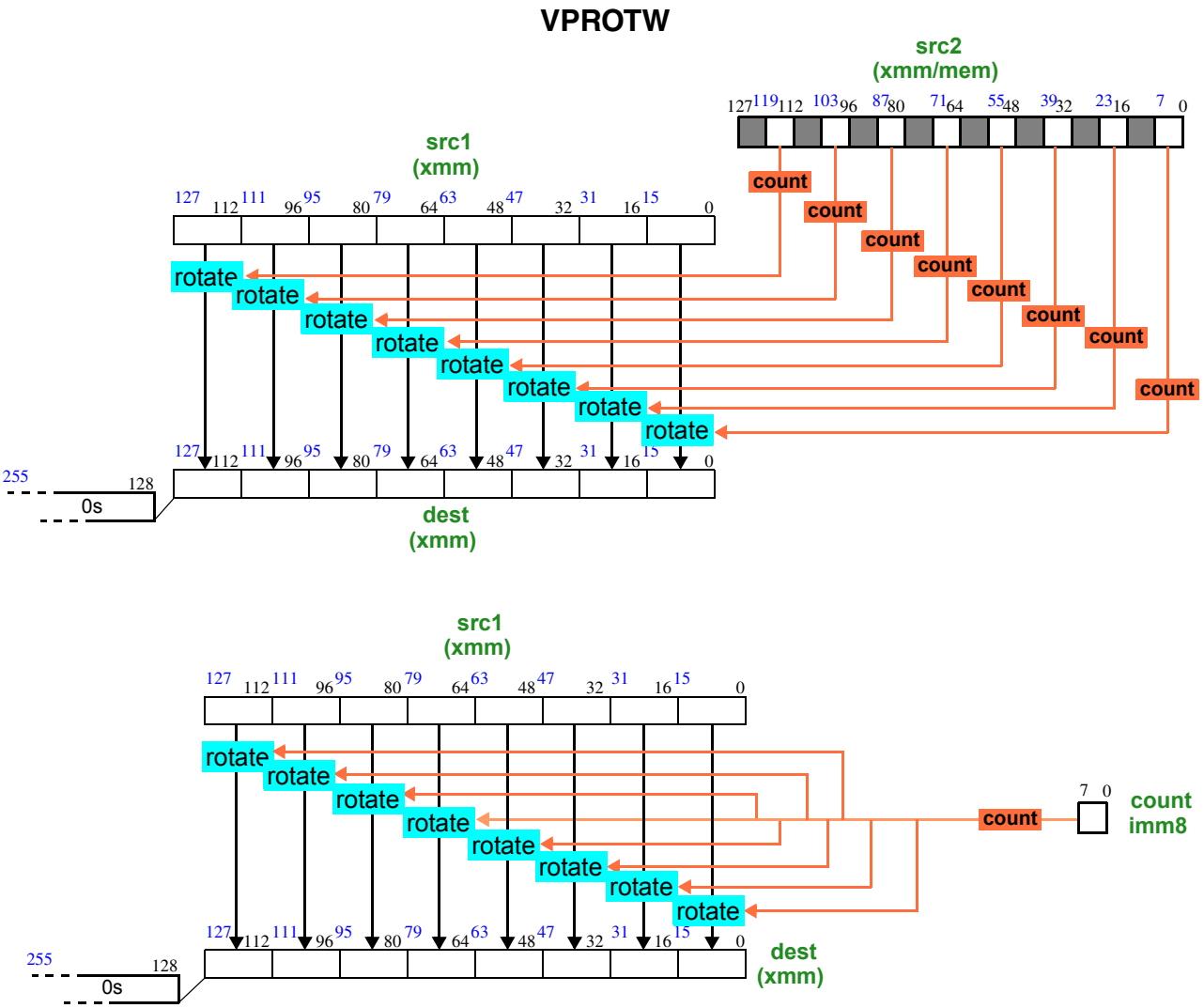
The *src* and *count* operands are configurable through XOP.W. If XOP.W is 0, the *src* operand is an XMM register or 128-bit memory location specified by MODRM.rm and the *count* operand is an XMM register specified by XOP.vvvv. If XOP.W is 1, the *src* operand is an XMM register specified by XOP.vvvv and the *variable-count* operand is an XMM register or 128-bit memory location specified by MODRM.rm.

If 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.

If 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.

The PROTW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPROTW *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	91 /r
VPROTW *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.0.00	91 /r
VPROTW *xmm1, xmm2*/*mem128*, *imm8*	8F	$\overline{\text{RXB}}$.08	0.1111.0.00	C1 /r /ib



Related Instructions

PROTB, PROTD, PROTQ, PSHLB, PSHLW, PSHLD, PSHLQ, PSHAB, PSHAW, PSHAD, PSHAQ

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
			X	XOP.vvvv was not 1111b for immediate count form of instruction (opcode C1h).
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHAB Packed Shift Arithmetic Bytes

Shifts each signed byte of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding byte of the destination.

The count byte for each 8-bit shift is an 8-bit signed two's-complement value in the corresponding byte element of the count *operand*.

If 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.

If 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.

The VPSHAB instruction requires three operands:

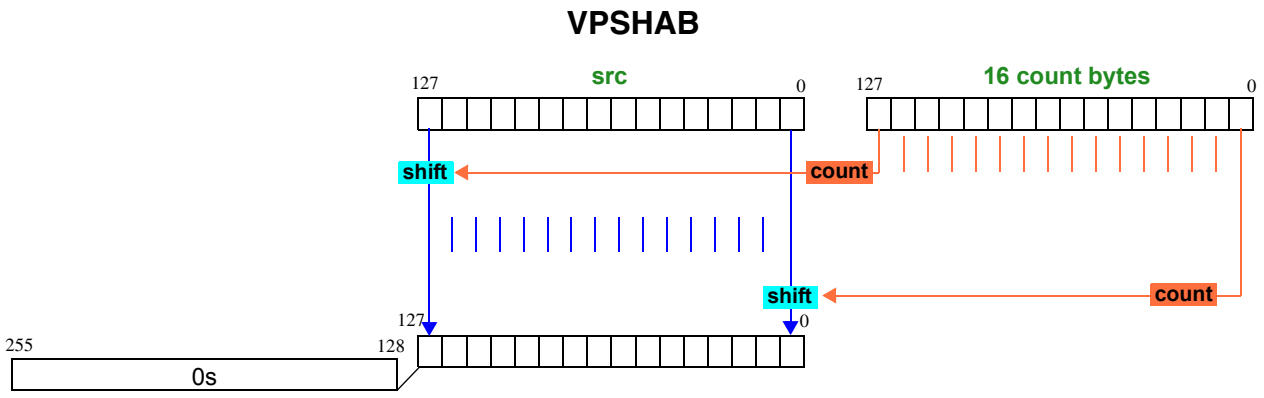
VPSHAB dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the results are written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

If XOP.W is 0, the *count* is an XMM register specified by XOP.vvvv and the *src* is an XMM register or 128-bit memory location specified by MODRM.rm. If XOP.W is 1, the *count* is an XMM register or 128-bit memory location specified by MODRM.rm and the *src* operand is an XMM register specified by XOP.vvvv.

The VPSHAB instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHAB x*mm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	98 /r
VPSHAB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.

Exception	Real	Virtual 8086	Protected	Cause of Exception
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHAD Packed Shift Arithmetic Doublewords

Shifts each of the four signed doublewords of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding doubleword of the destination.

The count byte for each doubleword shift is an 8-bit signed two's-complement value located in the low-order byte of the corresponding doubleword element of the *count* operand.

If 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.

If 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.

The VPSHAD instruction requires three operands:

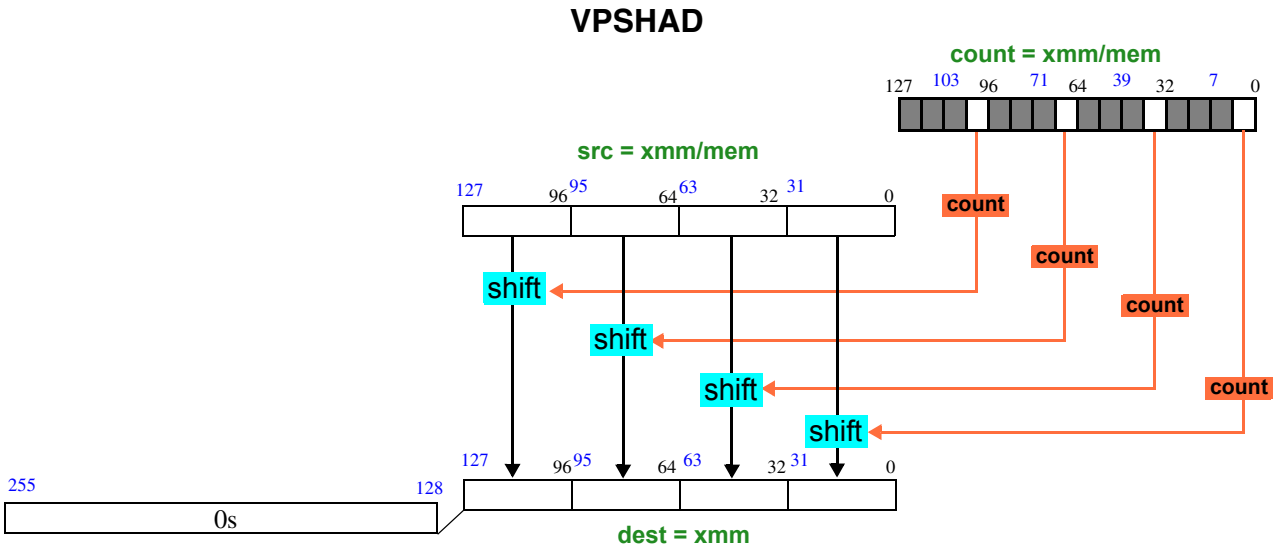
VPSHAD dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the 128-bit result is written to the *dest* XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* is an XMM register specified by XOP.vvvv and the *src* is an XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* is an XMM register or memory location specified by MODRM.rm and the *src* is an XMM register specified by XOP.vvvv.

The VPSHAD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHAD *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	9A /r
VPSHAD *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHAQ Packed Shift Arithmetic Quadwords

Shifts the two quadwords of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding quadword of the destination.

The count byte for each quadword shift is an 8-bit signed two's-complement value located in the low-order byte of the corresponding quadword element of the *count* operand.

If 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.

If 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.

The VPSHAQ instruction requires three operands:

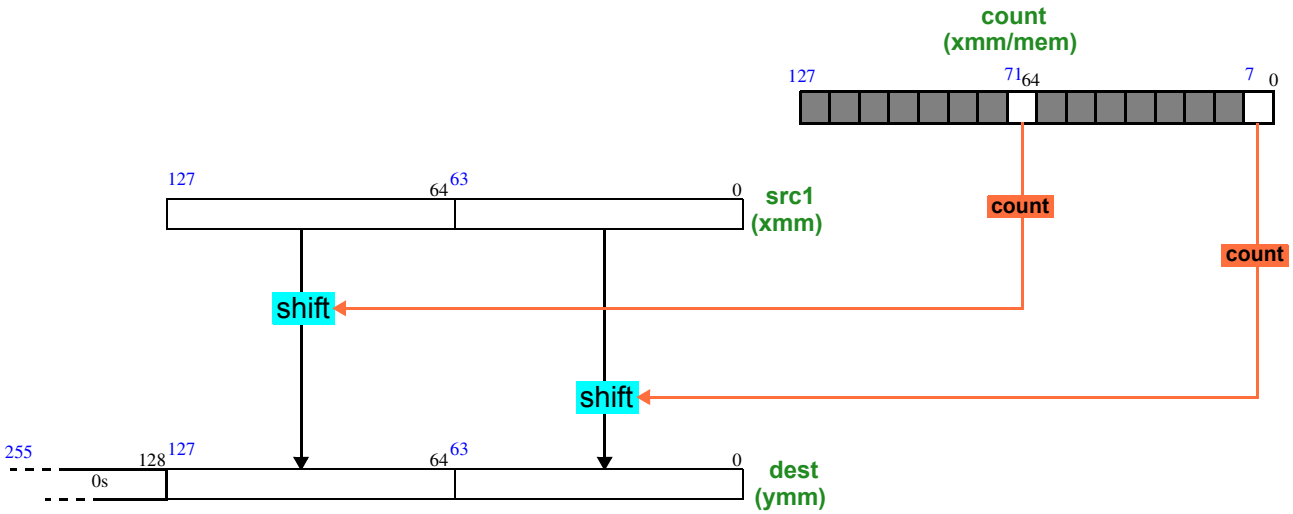
VPSHAQ dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the 128-bit result operand is written to the *dest* XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* is a 128-bit XMM register specified by XOP.vvvv and the *src* is a 128-bit XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* is a 128-bit XMM register or memory location specified by MODRM.rm and the *src* is a 128-bit XMM register specified by XOP.vvvv.

The VPSHAQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHAQ *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	9B /r
VPSHAQ *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.0.00	9B /r



Related Instructions

VPROTB, VPROTW, VPROTD, VPROTQ, VPSHLB, VPSHLW, VPSHLD, VPSHLQ, VPSHAB, VPSHAW, VPSHAD

rFLAGS Affected

None

MXCSR Flags Affected

None

Exceptions

Exception	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enable.

VPSHAW Packed Shift Arithmetic Words

Shifts each of the eight words of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding signed word of the destination.

The count byte for each word shift is an 8-bit signed two's-complement value located in the low-order byte of the corresponding word element of the *count* operand.

If 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.

If 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.

The VPSHAW instruction requires three operands:

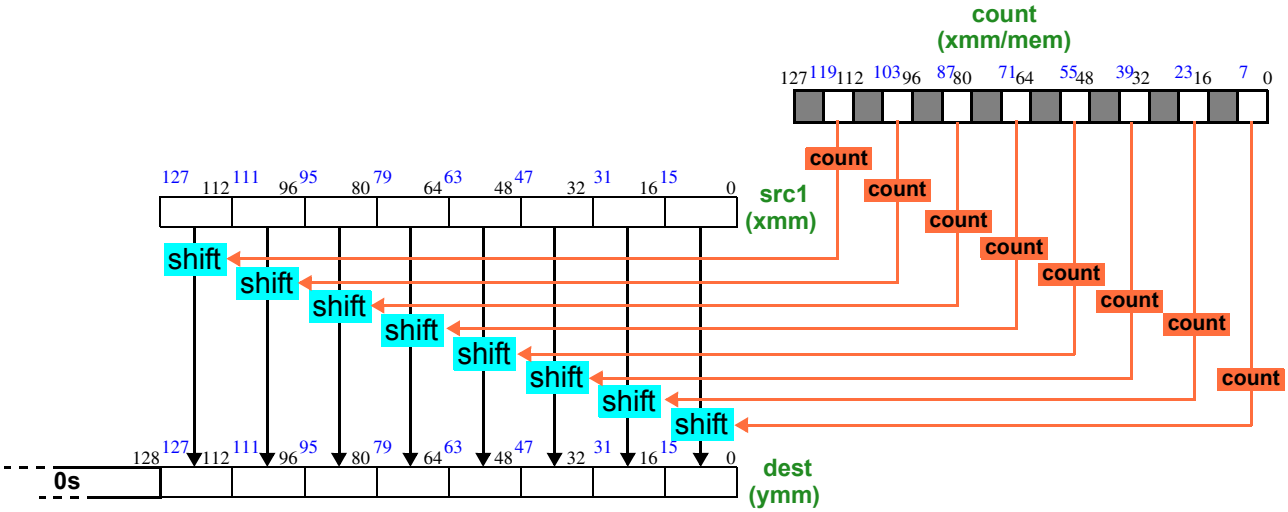
VPSHAW dest, src, count

The destination (*dest*) is a YMM register addressed by the MODRM.reg field. When the 128-bit result operand is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* is a 128-bit XMM register specified by XOP.vvvv and the *src* operand is a 128-bit XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* operand is a 128-bit XMM register or memory location specified by MODRM.rm and the *src* operand is a 128-bit XMM register specified by XOP.vvvv.

The VPSHAW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHAW *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	99 /r
VPSHAW *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHLB Packed Shift Logical Bytes

Shifts each byte of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding byte of the destination.

The count byte for each byte shift is an 8-bit signed two's-complement value located in the the corresponding byte element of the *count* operand.

If 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.

If 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.

The VPSHLB instruction requires three operands:

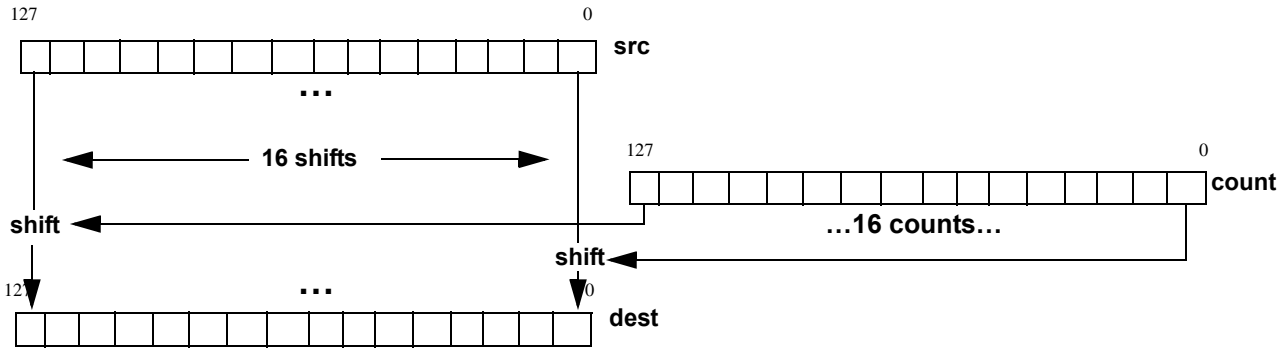
VPSHLB dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the 128-bit result is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* is a 128-bit XMM register specified by XOP.vvvv and the *src* is a 128-bit XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* is a 128-bit XMM register or memory location specified by MODRM.rm and the *src* is a 128-bit XMM register specified by XOP.vvvv.

The VPSHLB instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLB *xmm1*, *xmm2*/*mem128, xmm3*	8F	$\overline{\text{RXB}}$.9	0.$\overline{\text{xcnt}}$.0.00	94 /r
VPSHLB *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.9	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
			X	VEX.vvvv was not 1111b.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHLD Packed Shift Logical Doublewords

Shifts each doubleword of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding doubleword of the destination.

The count byte for each doubleword shift is an 8-bit signed two's-complement value located in the low-order byte of the corresponding doubleword element of the *count* operand.

If 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.

If 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.

The VPSHLD instruction requires three operands:

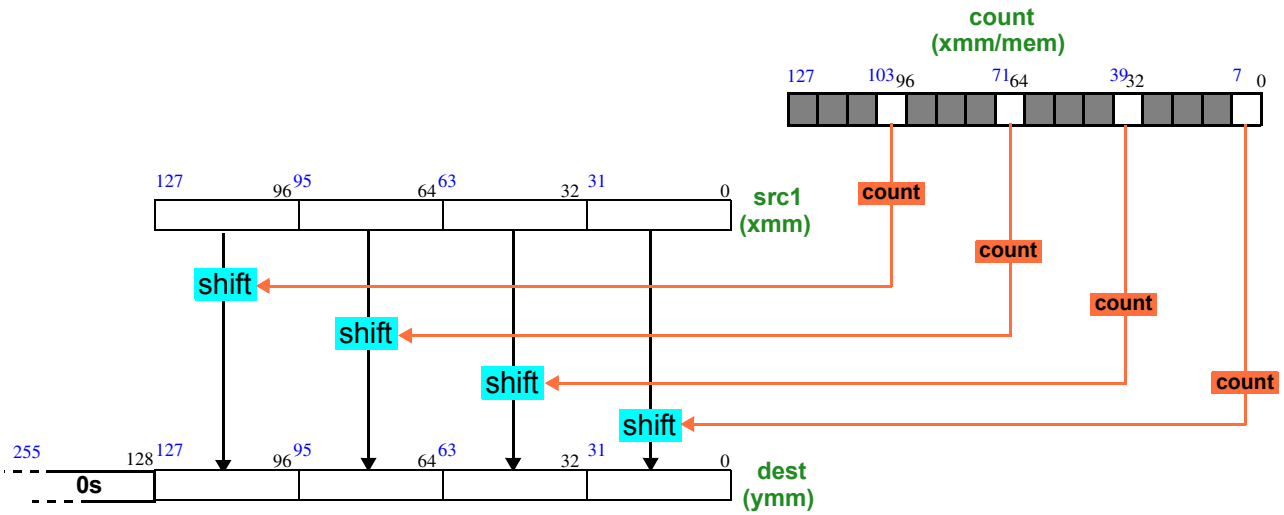
VPSHLD dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the 128-bit result is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* is a 128-bit XMM register specified by XOP.vvvv and the *src* is a 128-bit XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* is a 128-bit XMM register or memory location specified by MODRM.rm and the *src* operand is a 128-bit XMM register specified by XOP.vvvv.

The VPSHLD instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLD *xmm1*, *xmm3*/*mem128, xmm2*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	96 /r
VPSHLD *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHLQ Packed Shift Logical Quadwords

Shifts the two quadwords of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding quadword of the destination.

The count byte for each quadword shift is an 8-bit signed two's-complement value located in the low-order byte of the corresponding quadword element of the *count* operand.

Bit 6 of the count byte is ignored.

If 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.

If 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.

The VPSHLQ instruction requires three operands:

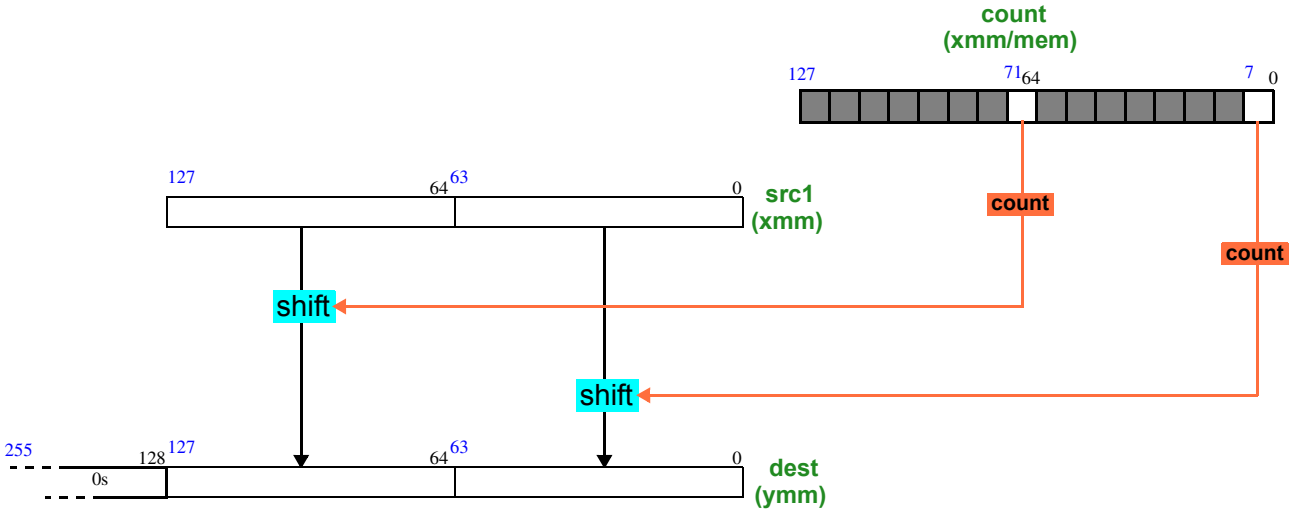
VPSHLQ dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the 128-bit result is written to the *dest* YMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* operand is a 128-bit XMM register specified by XOP.vvvv and the *src* is a 128-bit XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* is a 128-bit XMM register or memory location specified by MODRM.rm and the *src* is a 128-bit XMM register specified by XOP.vvvv.

The VPSHLQ instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLQ *xmm1*, *xmm3*/*mem128, xmm2*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	97 /r
VPSHLQ *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.

Exception	Real	Virtual 8086	Protected	Cause of Exception
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.

VPSHLW Packed Shift Logical Words

Shifts each of the eight words of the source operand by the amount specified in the signed value of the corresponding *count* byte and writes the result in the corresponding word of the destination.

The count byte for each word shift is an 8-bit signed two's-complement value located in the low-order byte of the corresponding word element of the *count* operand.

If 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.

If 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.

The VPSHLW instruction requires three operands:

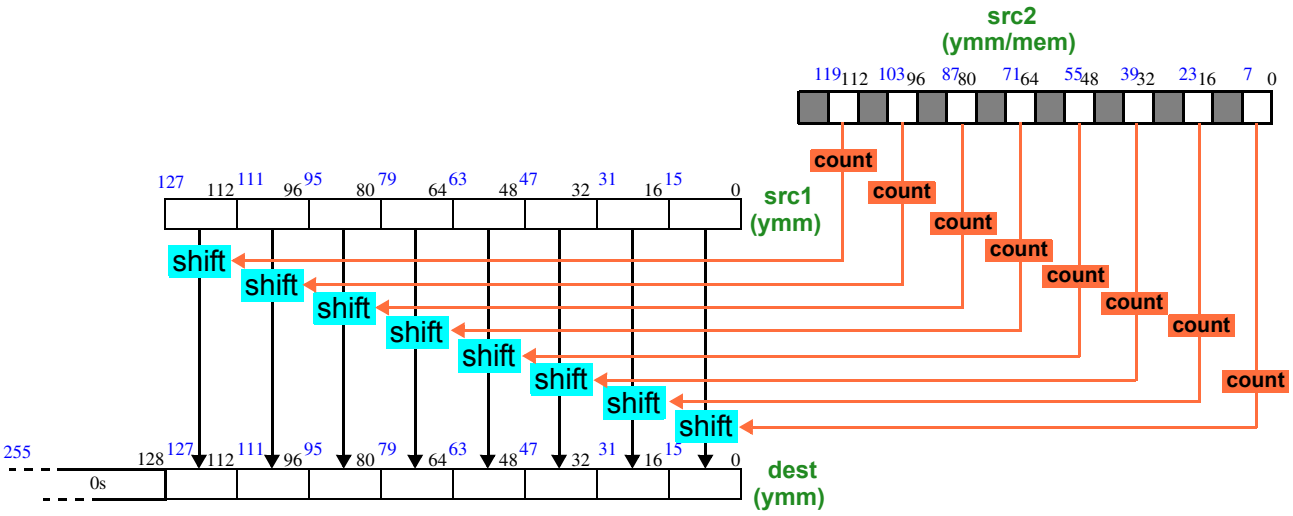
VPSHLW dest, src, count

The destination (*dest*) is an XMM register addressed by the MODRM.reg field. When the 128-bit result is written to the destination XMM register, the upper 128 bits of the corresponding YMM register are cleared to zeros.

The *src* and *count* are configurable through XOP.W. If XOP.W is 0, the *count* operand is a 128-bit XMM register specified by XOP.vvvv and the *src* is a 128-bit XMM register or memory location specified by MODRM.rm. If XOP.W is 1, the *count* is a 128-bit XMM register or memory location specified by MODRM.rm and the *src* is a 128-bit XMM register specified by XOP.vvvv.

The VPSHLW instruction is an XOP instruction. The presence of this instruction set is indicated by a CPUID feature bit. (See the *CPUID Specification*, order# 25481.)

Mnemonic	Encoding			
	XOP	RXB.mmmmm	W.vvvv.L.pp	Opcode
VPSHLW *xmm1*, *xmm3*/*mem128, xmm2*	8F	$\overline{\text{RXB}}$.09	0.$\overline{\text{xcnt}}$.0.00	95 /r
VPSHLW *xmm1*, *xmm2, xmm3*/*mem128*	8F	$\overline{\text{RXB}}$.09	1.$\overline{\text{xsrc}}$.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	Real	Virtual 8086	Protected	Cause of Exception
Invalid opcode, #UD	X	X		XOP instructions are only recognized in protected mode.
			X	The XOP instructions are not supported, as indicated by ECX bit 11 of CPUID function 8000_0001h.
			X	The operating-system XSAVE/XRSTOR support bit (OSXSAVE) of CR4 was cleared to 0, as indicated by ECX bit 27 of CPUID function 0000_0001h.
			X	The operating-system YMM support bits XFEATURE_ENABED_MASK[2:1] were were not both set to 1.
			X	XOP.L was set to 1.
Device not available, #NM			X	The task-switch bit (TS) of CR0 was set to 1.
Stack, #SS			X	A memory address exceeded the stack segment limit or was non-canonical.
General protection, #GP			X	A memory address exceeded a data segment limit or was non-canonical.
			X	A null data segment was used to reference memory.
Page fault, #PF			X	A page fault resulted from the execution of the instruction.
Alignment Check, #AC			X	An unaligned memory reference was performed while alignment checking was enabled.