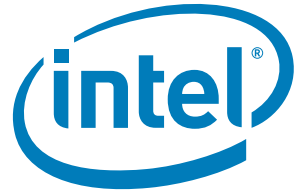

Intel® SSE4 Programming Reference

Reference Number: D91561-002
May 2007

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 5937
Denver, CO 80217-9808

or call 1-800-548-4725
or visit Intel's website at http://www.intel.com

FIGURES

TABLES

CHAPTER 1
STREAMING SIMD EXTENSIONS 4

1.1 INTRODUCTION

Intel® Streaming SIMD Extensions 4 (SSE4) introduces 54 new instructions in Intel 64 processors made from 45 nm process technology.

- 47 of the SSE4 instructions are available in 45 nm Intel processors based on the successor of Intel Core™ microarchitecture (code named Penryn). This subset of 47 SSE4 instruction is referred to as SSE4.1 in this document.
- SSE4.1 and seven other new SSE4 instructions are supported in 45 nm Intel processors based on a new microarchitecture (code named Nehalem). The subset of the 7 new SSE4 instructions available to Intel processors based on the Nehalem microarchitecture is referred to as SSE4.2 in this document.

1.2 SSE4 OVERVIEW

SSE4.1 is targeted to improve the performance of media, imaging, and 3D workloads. SSE4.1 adds instructions that improve compiler vectorization and significantly increase support for packed dword computation. The technology also provides a hint that can improve memory throughput when reading from uncacheable WC memory type.

The 47 SSE4.1 instructions (see Appendix A, “Instruction Summary and Encodings”) include:

- Two instructions perform packed dword multiplies.
- Two instructions perform floating-point dot products with input/output selects.
- One instruction performs a load with a streaming hint.
- Six instructions simplify packed blending.
- Eight instructions expand support for packed integer MIN/MAX.
- Four instructions support floating-point round with selectable rounding mode and precision exception override.
- Seven instructions improve data insertion and extractions from XMM registers
- Twelve instructions improve packed integer format conversions (sign and zero extensions).
- One instruction improves SAD (sum absolute difference) generation for small block sizes.
- One instruction aids horizontal searching operations.
- One instruction improves masked comparisons.

- One instruction adds qword packed equality comparisons.
- One instruction adds dword packing with unsigned saturation.

The seven SSE4.2 instructions improve performance in the following areas:

- String and text processing that can take advantage of single-instruction multiple-data programming techniques.
- Application-targeted accelerator (ATA) instructions.
- A SIMD integer instruction that enhances the capability of the 128-bit integer SIMD capability in SSE4.1.

SSE4 requires no new OS support to save and restore the register state beyond what is required by Streaming SIMD Extensions (SSE). There are six SSE4.1 instructions which generate numeric (SIMD floating-point) exceptions, thus requiring the OS to provide IEEE-754 compliant event handlers for post-compute exception (similar to SSE/SSE2/SSE3 instructions). SSE4.2 instructions do not generate SIMD floating-point exceptions. See *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1* , Appendix E.

SSE4 is fully compatible with software written for previous generations of Intel 64 and IA-32 architecture microprocessors. All existing software continues to run correctly without modification on microprocessors that incorporate SSE4, as well as in the presence of existing and new applications that incorporate SSE4.

CHAPTER 2
SSE4 FEATURES

2.1 NEW DATA TYPES

SSE4 does not introduce any new data types.

2.2 SSE4.1 INSTRUCTION SET

SSE4.1 instructions can use an XMM register as a source or destination. Programming SSE4.1 is similar to programming 128-bit Integer SIMD and floating-point SIMD instructions in SSE/SSE2/SSE3/SSSE3. SSE4.1 does not provide any 64-bit integer SIMD instructions.

2.2.1 Dword Multiply Instructions

SSE4.1 adds two dword multiply instructions that aid vectorization. They allow four simultaneous 32 bit by 32 bit multiplies. PMULLD returns a low 32-bits of the result and PMULDQ returns a 64-bit signed result. These represent the most common integer multiply operation. See Table 2-1.

Table 2-1. Enhanced 32-bit SIMD Multiply Supported by SSE4.1

		32 bit Integer Operation	
		unsigned x unsigned	signed x signed
Result	**Low 32-bit**	(not available)	PMULLD
	High 32-bit	(not available)	(not available)
	64-bit	PMULUDQ*	PMULDQ

NOTE:

* Available prior to SSE4.1.

2.2.2 Floating-Point Dot Product Instructions

SSE4.1 adds double-precision (for 2 elements; DPPD) and single-precision dot products (for up to 4 elements; DPPS).

These dot-product instructions include source select and destination broadcast which generally improves the usability. For example, a single DPPS instruction can be used for a 2, 3, or 4 element dot product.

2.2.3 Streaming Load Hint Instruction

Historically, CPU read accesses of WC memory type regions have significantly lower throughput than accesses to cacheable memory.

The streaming load instruction in SSE4.1, MOVNTDQA, provides a non-temporal hint that can cause adjacent 16-byte items within an aligned 64-byte region (a streaming line) to be fetched and held in a small set of temporary buffers ("streaming load buffers"). Subsequent streaming loads to other aligned 16-byte items in the same streaming line may be supplied from the streaming load buffer and can improve throughput.

Programmers are advised to use the following practices to improve the efficiency of MOVNTDQA streaming loads from WC memory:

- Streaming loads must be 16-byte aligned.
- Temporally group streaming loads of the same streaming cache line for effective use of the streaming load buffers. Loads issued much later may cause the streaming line to be refetched from memory.
- Temporally group streaming loads from at most a few streaming lines together. The number of streaming load buffers is small; grouping a modest number of streams will avoid running out of streaming load buffers and the resultant refetching of streaming lines from memory.
- Avoid writing to a streaming line until all reads to 16-byte items have occurred. Reading a 16-byte item from a streaming line that has been written, may cause the streaming line to be refetched.
- Avoid reading a given 16-byte item within a streaming line more than once; repeated loads of a particular 16-byte item are likely to cause the streaming line to be refetched.
- The streaming load buffers, reflecting the WC memory type characteristics, are not required to be snooped by operations from other agents. Software should not rely upon such coherency actions to provide any data coherency with respect to other logical processors or bus agents. Rather, software must employ memory fences (i.e. the MFENCE instruction) to insure the consistency of WC memory accesses between producers and consumers.

2.2.4 Packed Blending Instructions

SSE4.1 adds 6 instructions used for blending (BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW).

Blending conditionally copies a field in a source operand to the same field in the destination. SSE4.1 instructions improve blending operations for most field sizes. A single new SSE4.1 instruction can generally replace a sequence of 2 to 4 operations using previous architectures.

The variable blend instructions (BLENDVPS, PBLENDVPD, PBLENDW) introduce the use of control bits stored in an implicit XMM register (XMM0). The most significant bit in each field (the sign bit, for 2's compliment integer or floating-point) is used as

a selector. See Table 2-2.

Table 2-2. Blend Field Size and Control Modes Supported by SSE4.1

Instructions	Packed Double FP	Packed Single FP	Packed QWord	Packed DWord	Packed Word	Packed Byte	Blend Control
BLENDPS		X					Imm8
BLENDPD	X						Imm8
BLENDVPS		X		X(1)			XMM0
BLENDVPD	X		X(1)				XMM0
PBLENDVB			(2)	(2)	(2)	X	XMM0
PBLENDW			X	X	X		Imm8

NOTE:

1. Use of floating-point SIMD instructions on integer data types may incur performance penalties.
2. Byte variable blend can be used for larger sized fields by reformatting (or shuffling) the blend control.

2.2.5 Packed Integer MIN/MAX Instructions

SSE4.1 adds 8 packed integer MIN and MAX instructions (PMINUW, PMINUD, PMINSB, PMINSD; PMAXUW, PMAXUD, PMAXSB, PMAXSD).

Four 32-bit integer packed MIN and MAX instructions operate on unsigned and signed dwords. Two instructions operate on signed bytes. Two instructions operate on unsigned words. See Table 2-3.

Table 2-3. Enhanced SIMD Integer MIN/MAX Instructions Supported by SSE4.1

		Integer Width		
		Byte	Word	DWord
Integer Format	Unsigned	PMINUB* PMAXUB*	PMINUW PMAXUW	PMINUD PMAXUD
	Signed	PMINSB PMAXSB	PMINSW* PMAXSW*	PMINSD PMAXSD

NOTE:

* Available prior to SSE4.1.

2.2.6 Floating-Point Round Instructions with Selectable Rounding Mode

High level languages and libraries often expose rounding operations having a variety of numeric rounding and exception behaviors. Using SSE/SSE2/SSE3 instructions to mitigate the rounding-mode-related problem is sometimes not straight forward.

SSE4.1 introduces four rounding instructions (ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD) that cover scalar and packed single- and double-precision floating-point operands. The rounding mode can be selected using an immediate from one of the IEEE-754 modes (Nearest, -Inf, +Inf, and Truncate) without changing the current rounding mode; or the the instruction can be forced to use the current rounding mode. Another bit in the immediate is used to suppress inexact precision exceptions.

Rounding instructions in SSE4.1 generally permit single-instruction solutions to C99 functions ceil(), floor(), trunc(), rint(), nearbyint(). These instructions simplify the implementations of half-way-away-from-zero rounding modes as used by C99 round() and F90's nint().

2.2.7 Insertion and Extractions from XMM Registers

SSE4.1 adds 7 instructions (corresponding to 9 assembly instruction mnemonics) that simplify data insertion and extraction between general-purpose register (GPR) and XMM registers (EXTRACTPS, INSERTPS, PINSRB, PINSRD, PINSRQ, PEXTRB, PEXTRW, PEXTRD, and PEXTRQ). When accessing memory, no alignment is required for any of these instructions (unless alignment checking is enabled).

EXTRACTPS extracts a single-precision floating-point value from any offset in an XMM register and stores the result to memory or a general-purpose register. INSERTPS inserts a single floating-point value from either a 32-bit memory location or from an XMM register. In addition, INSERTPS allows the insertion of +0.0f into destination fields.

PINSRB, PINSRD, and PINSRQ insert byte, dword, or qword values from 32/64 registers or memory into an XMM register. Word values were already supported by SSE2 (PINSRW).

PEXTRB, PEXTRW, PEXTRD, and PEXTRQ extract byte, word, dword, and qword from an XMM register and insert the values into a general-purpose register or memory.

2.2.8 Packed Integer Format Conversions

A common type of operation on packed integers is the conversion by zero- or sign-extension of packed integers into wider data types. SSE4.1 adds 12 instructions that convert from a smaller packed integer type to a larger integer type (PMOVSXBW, PMOVZXBW, PMOVSXBD, PMOVZXBD, PMOVSXWD, PMOVZXWD, PMOVSXBQ, PMOVZXBQ, PMOVSXWQ, PMOVZXWQ, PMOVSXDQ, PMOVZXDQ).

The source operand is from either an XMM register or memory; the destination is an XMM register. See Table 2-4.

When accessing memory, no alignment is required for any of the instructions unless alignment checking is enabled. In which case, all conversions must be aligned to the width of the memory reference. The number of elements converted (and width of memory reference) is illustrated in Table 2-5. The alignment requirement is shown in parenthesis.

Table 2-4. New SIMD Integer conversions supported by SSE4.1

		Source Type		
		Byte	**Word**	**Dword**
Destination Type	Signed Word Unsigned Word	PMOVSXBW PMOVZXBW		
	Signed Dword Unsigned Dword	PMOVSXBD PMOVZXBD	PMOVSXWD PMOVZXWD	
	Signed Qword Unsigned Qword	PMOVSXBQ PMOVZXBQ	PMOVSXWQ PMOVZXWQ	PMOVSXDQ PMOVZXDQ

Table 2-5. New SIMD Integer Conversions Supported by SSE4.1

		Source Type		
		Byte	**Word**	**Dword**
Destination Type	Word	8 (64 bits)		
	Dword	4 (32 bits)	4 (64 bits)	
	Qword	2 (16 bits)	2 (32 bits)	2 (64 bits)

2.2.9 Improved Sums of Absolute Differences (SAD) for 4-Byte Blocks

SSE4.1 adds an instruction (MPSADBW) that performs eight 4-byte wide SAD operations per instruction. Compared to PSADBW, MPSADBW operates on smaller chunks (4-byte instead of 8-byte chunks); this makes the instruction better suited to video coding standards such as VC.1 and H.264. MPSADBW performs four times the number of absolute difference operations than that of PSADBW (per instruction). This can improve performance for dense motion searches.

MPSADBW uses a 4-byte wide field from a source operand; the offset of the 4-byte field within the 128-bit source operand is specified by two immediate control bits. MPSADBW produces eight 16-bit SAD results. Each 16-bit SAD result is formed from overlapping pairs of 4 bytes in the destination with the 4-byte field from the source operand. MPSADBW uses eleven consecutive bytes in the destination operand, its offset is specified by a control bit in the immediate byte (i.e. the offset can be from byte 0 or from byte 4). Figure 2-1 illustrates the operation of MPSADBW. MPSADBW can simplify coding of dense motion estimation by providing source and destination offset control, higher throughput of SAD operations, and the smaller chunk size.

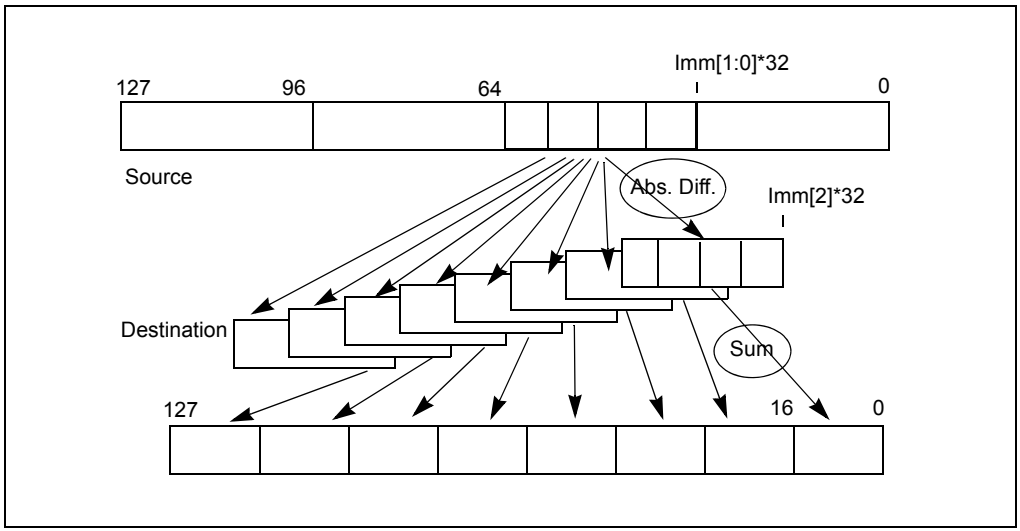

Figure 2-1. MPSADBW Operation

2.2.10 Horizontal Search

SSE4.1 adds a search instruction (PHMINPOSUW) that finds the value and location of the minimum unsigned word from one of 8 horizontally packed unsigned words. The resulting value and location (offset within the source) are packed into the low dword of the destination XMM register.

Rapid search is often a significant component of motion estimation. MPSADBW and PHMINPOSUW can be used together to improve video encode.

2.2.11 Packed Test

The packed test instruction PTEST is similar to a 128-bit equivalent to the legacy instruction TEST. With PTEST, the source argument is typically used like a bit mask.

PTEST performs a logical AND between the destination with this mask and sets the ZF flag if the result is zero. The CF flag (zero for TEST) is set if the inverted mask AND'd with the destination is all zero. Because the destination is not modified, PTEST simplifies branching operations (such as branching on signs of packed floating-point numbers, or branching on zero fields).

2.2.12 Packed Qword Equality Comparisons

SSE4.1 adds a 128-bit packed qword equality test. The new instruction (PCMPEQQ) is identical to PCMPEQD, but has qword granularity.

2.2.13 Dword Packing With Unsigned Saturation

SSE4.1 adds a new instruction PACKUSDW to complete the set of small integer pack instructions in the family of SIMD instruction extensions. PACKUSDW packs dword to word with unsigned saturation. See Table 2-6 for the complete set of packing instructions for small integers.

Table 2-6. Enhanced SIMD Pack support by SSE4.1

		Pack Type	
		DWord -> word	Word -> Byte
Saturation Type	Unsigned	PACKUSDW (new!)	PACKUSWB
	Signed	PACKSSDW	PACKSSWB

2.2.14 IEEE 754 Compliance

The six SSE4.1 instructions that perform floating-point arithmetic are:

- DPPS
- DPPD
- ROUNDPS
- ROUNDPD
- ROUNDSS
- ROUNDSD

Dot Product operations are not specified in IEEE-754. When neither FTZ nor DAZ are enabled, the dot product instructions resemble sequences of IEEE-754 multiplies and adds (with rounding at each stage), except that the treatment of input NaN's is implementation specific (there will be at least one NaN in the output). The input select fields (bits imm8[4:7]) force input elements to +0.0f prior to the first multiply and will suppress input exceptions that would otherwise have been be generated.

As a convenience to the exception handler, any exceptions signaled from DPPS or DPPD leave the destination unmodified.

Round operations signal invalid and precision only.

Table 2-7. SIMD numeric exceptions signaled by SSE4.1

	DPPS	DPPD	ROUNDPS ROUNDSS	ROUNDPD ROUNDSD
Overflow	X	X		
Underflow	X	X		
Invalid	X	X	X (1)	X (1)
Inexact Precision	X	X	X (2)	X (2)
Denormal	X	X		

NOTE:

1. Invalid is signaled only if Src = SNaN.
2. Precision is ignored (regardless of the MXCSR precision mask) if if imm8[3] = '1'.

The other SSE4.1 instructions with floating-point arguments (BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, INSERTPS, EXTRACTPS) do not signal any SIMD numeric exceptions.

2.3 SSE4.2 INSTRUCTION SET

Five of the seven SSE4.2 instructions can use an XMM register as a source or destination. These include four text/string processing instructions and one packed quadword compare SIMD instruction. Programming these five SSE4.2 instructions is similar to programming 128-bit Integer SIMD in SSE2/SSSE3. SSE4.2 does not provide any 64-bit integer SIMD instructions.

The remaining two SSE4.2 instructions uses general-purpose registers to perform accelerated processing functions in specific application areas.

2.3.1 String and Text Processing Instructions

String and text processing instructions in SSE4.2 allocates 4 opcodes to provide a rich set of string and text processing capabilities that traditionally required many more opcodes. These 4 instructions use XMM registers to process string or text elements of up to 128-bits (16 bytes or 8 words). Each instruction uses an immediate byte to support a rich set of programmable controls. A string-processing SSE4.2 instruction returns the result of processing each pair of string elements using either an index or a mask.

The capabilities of the string/text processing instructions include:

- Handling string/text fragments consisting of bytes or words, either signed or unsigned

- Support for partial string or fragments less than 16 bytes in length, using either explicit length or implicit null-termination
- Four types of string compare operations on word/byte elements
- Up to 256 compare operations performed in a single instruction on all string/text element pairs
- Built-in aggregation of intermediate results from comparisons
- Programmable control of processing on intermediate results
- Programmable control of output formats in terms of an index or mask
- Bi-directional support for the index format
- Support for two mask formats: bit or natural element width
- Not requiring 16-byte alignment for memory operand

The four SSE4.2 instructions that process text/string fragments are:

- PCMPESTRI — Packed compare explict-length strings, return index in ECX/RCX
- PCMPESTRM — Packed compare explict-length strings, return mask in XMM0
- PCMPISTRI — Packed compare implict-length strings, return index in ECX/RCX
- PCMPISTRM — Packed compare implict-length strings, return mask in XMM0

All four require the use of an immediate byte to control operation. The two source operands can be XMM registers or a combination of XMM register and memory address. The immediate byte provides programmable control with the following attributes:

- Input data format
- Compare operation mode
- Intermediate result processing
- Output selection

Depending on the output format associated with the instruction, the text/string processing instructions implicitly uses either a general-purpose register (ECX/RCX) or an XMM register (XMM0) to return the final result.

Two of the four text-string processing instructions specify string length explicitly. They use two general-purpose registers (EDX, EAX) to specify the number of valid data elements (either word or byte) in the source operands. The other two instructions specify valid string elements using null termination. A data element is considered valid only if it has a lower index than the least significant null data element.

2.3.1.1 Memory Operand Alignment

The text and string processing instructions in SSE4.2 do not perform alignment checking on memory operands. This is different from most other 128-bit SIMD instructions accessing the XMM registers. The absence of an alignment check for

these four instructions does not imply any modification to the existing definitions of other instructions.

2.3.2 Packed Comparison SIMD integer Instruction

SSE4.2 also provides a 128-bit integer SIMD instruction PCMPGTQ that performs logical compare of greater-than on packed integer quadwords.

2.3.3 Application-Targeted Accelerator Instructions

There are two application-targeted accelerator instructions in SSE4.2:

- CRC32 — Provides hardware acceleration to calculate cyclic redundancy checks for fast and efficient implementation of data integrity protocols.
- POPCNT — Accelerates software performance in the searching of bit patterns.

CHAPTER 3
APPLICATION PROGRAMMING MODEL

The application programming environment for SSE4 is similar to that of Streaming SIMD Extensions (SSE), SSE2, SSE3, and SSSE3.

3.1 CPUID

The CPUID instruction is extended to provide additional information, including three feature flags that indicate support for SSE4 instructions. CPUID's output is dependent on the contents of the EAX register upon execution. For example, the following pseudocode loads EAX with 00H and causes CPUID to return a Maximum Return Value and the Vendor Identification String in the appropriate registers:

```
MOV EAX, 00H
CPUID
```

Table 3-1 shows information returned, depending on the initial value loaded into the EAX register. Table 3-2 shows the maximum CPUID input value recognized for each family of Intel 64 and IA-32 processors on which CPUID is implemented.

Two types of information are returned: basic and extended function information. If a value is entered for CPUID.EAX is invalid for a particular processor, the data for the highest basic information leaf is returned. For example, using the Intel® Core™2 Duo processor, the following is true:

```
CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* INVALID: Returns the same information as CPUID.EAX = 0AH. *)
CPUID.EAX = 80000008H (* Returns virtual/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0AH. *)
```

CPUID can be executed at any privilege level to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed.

See also:

"Serializing Instructions" in Chapter 7, "Multiple-Processor Management," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*

AP-485, Intel Processor Identification and the CPUID Instruction (Order Number 241618)

Table 3-1. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
	Basic CPUID Information	
0H	EAX	Maximum Input Value for Basic CPUID Information (see Table 3-2)
	EBX	"Genu"
	ECX	"ntel"
	EDX	"ineI"
01H	EAX	Version Information: Type, Family, Extended Family, Model, Extended Model, and Stepping ID (see Figure 3-1)
	EBX	Bits 7-0: Brand Index Bits 15-8: CLFLUSH line size (Value ∗ 8 = cache line size in bytes) Bits 23-16: Maximum number of logical processors in this physical package. Bits 31-24: Initial APIC ID
	ECX	Extended Feature Information (see Figure 3-2 and Table 3-4)
	EDX	Feature Information (see Figure 3-3 and Table 3-5)
02H	EAX	Cache and TLB Information (see Table 3-6)
	EBX	Cache and TLB Information
	ECX	Cache and TLB Information
	EDX	Cache and TLB Information
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.) NOTE: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature. See AP-485, *Intel Processor Identification and the CPUID Instruction* (Order Number 241618) for more information on PSN.

Table 3-1. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor
	CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLES.BOOT_NT4[bit 22] = 0 (default).
	Deterministic Cache Parameters Leaf
04H	NOTE: 04H output depends on the initial value in ECX. See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level on page 1-33. EAX Bits 4-0: Cache Type* Bits 7-5: Cache Level (starts at 1) Bit 8: Self Initializing cache level (does not need SW initialization) Bit 9: Fully Associative cache Bit 10: Write-Back Invalidate 0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache. Bit 11: 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bits 13-12: Reserved Bits 25-14: Maximum number of threads sharing this cache in a physical package (see note)** Bits 31-26: Maximum number of processor cores in the physical package. **, *** EBX Bits 11-00: L = System Coherency Line Size** Bits 21-12: P = Physical Line partitions** Bits 31-22: W = Ways of associativity**

Table 3-1. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 31-00: S = Number of Sets**
	EDX	Reserved = 0 NOTES: * Cache Type fields: 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved ** Add one to the return value to get the result. *** The returned value is constant for valid initial values in ECX. Valid ECX values start from 0.
	MONITOR/MWAIT Leaf	
5H	EAX	Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	EBX	Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0
	ECX	Bits 00: Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported Bits 01: Supports treating interrupts as break-event for MWAIT, even when interrupts disabled Bits 31 - 02: Reserved
	EDX	Bits 03 - 00: Number of C0* sub C-states supported using MWait Bits 07 - 04: Number of C1* sub C-states supported using MWAIT Bits 11 - 08: Number of C2* sub C-states supported using MWAIT Bits 15 - 12: Number of C3* sub C-states supported using MWAIT Bits 19 - 16: Number of C4* sub C-states supported using MWAIT Bits 31 - 20: Reserved = 0 * The definition of C0 through C4 states for MWAIT extension are processor-specific C-states, not ACPI C-states.

Table 3-1. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Thermal and Power Management Leaf	
6H	EAX	Bits 00: Digital temperature sensor is supported if set Bits 31 - 01: Reserved
	EBX	Bits 03 - 00: Number of Interrupt Thresholds in Digital Thermal Sensor Bits 31 - 04: Reserved
	ECX	Bits 00: ACNT/MCNT. The capability to provide a measure of delivered processor performance (since last reset of the counters), as a percentage of expected processor performance at frequency specified in CPUID Brand String Bits 31 - 01: Reserved = 0
	EDX	Reserved = 0
	Architectural Performance Monitoring Leaf	
0AH	EAX	Bits 07 - 00: Version ID of architectural performance monitoring Bits 15- 08: Number of general-purpose performance monitoring counter per logical processor Bits 23 - 16: Bit width of general-purpose, performance monitoring counter Bits 31 - 24: Length of EBX bit vector to enumerate architectural performance monitoring events
	EBX	Bit 0: Core cycle event not available if 1 Bit 1: Instruction retired event not available if 1 Bit 2: Reference cycles event not available if 1 Bit 3: Last-level cache reference event not available if 1 Bit 4: Last-level cache misses event not available if 1 Bit 5: Branch instruction retired event not available if 1 Bit 6: Branch mispredict retired event not available if 1 Bits 31- 07: Reserved = 0
	ECX	Reserved = 0
	EDX	Bits 04 - 00: Number of fixed-function performance counters (if Version ID > 1) Bits 12- 05: Bit width of fixed-function performance counters (if Version ID > 1) Reserved = 0

Table 3-1. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
	Extended Function CPUID Information	
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 3-2).
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved
80000001H	EAX	Extended Processor Signature and Extended Feature Bits.
	EBX	Reserved
	ECX	Bit 0: LAHF/SAHF available in 64-bit mode Bits 31-1 Reserved
	EDX	Bits 10-0: Reserved Bit 11: SYSCALL/SYSRET available (when in 64-bit mode) Bits 19-12: Reserved = 0 Bit 20: Execute Disable Bit available Bits 28-21: Reserved = 0 Bit 29: Intel® 64 Technology available = 1 Bits 31-30: Reserved = 0
80000002H	EAX	Processor Brand String
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000003H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000004H	EAX	Processor Brand String Continued
	EBX	Processor Brand String Continued
	ECX	Processor Brand String Continued
	EDX	Processor Brand String Continued
80000005H	EAX	Reserved = 0
	EBX	Reserved = 0
	ECX	Reserved = 0
	EDX	Reserved = 0

Table 3-1. Information Returned by CPUID Instruction (Contd.)

Initial EAX Value	Information Provided about the Processor	
80000006H	EAX	Reserved = 0
	EBX	Reserved = 0
	ECX	Bits 7-0: Cache Line size in bytes Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units
	EDX	Reserved = 0
		NOTES: *L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative
80000007H	EAX	Reserved = 0
	EBX	Reserved = 0
	ECX	Reserved = 0
	EDX	Reserved = 0
80000008H	EAX	Virtual/Physical Address size Bits 7-0: #Physical Address Bits* Bits 15-8: #Virtual Address Bits Bits 31-16: Reserved = 0
	EBX	Reserved = 0
	ECX	Reserved = 0
	EDX	Reserved = 0
		NOTES: * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

INPUT EAX = 0: Returns CPUID's Highest Value for Basic Processor Information and the Vendor Identification String

When CPUID executes with EAX set to 0, the processor returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register (see Table 3-2) and is processor specific.

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is “GenuineIntel” and is expressed:

```
EBX ← 756e6547h (* "Genu", with G in the low nibble of BL *)
EDX ← 49656e69h (* "ineI", with i in the low nibble of DL *)
ECX ← 6c65746eh (* "ntel", with n in the low nibble of CL *)
```

INPUT EAX = 80000000H: Returns CPUID’s Highest Value for Extended Processor Information

When CPUID executes with EAX set to 0, the processor returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register (see Table 3-2) and is processor specific.

Table 3-2. Highest CPUID Source Operand for Intel 64 and IA-32 Processors

Intel 64 or IA-32 Processors	Highest Value in EAX	
	Basic Information	Extended Function Information
Earlier Intel486 Processors	CPUID Not Implemented	CPUID Not Implemented
Later Intel486 Processors and Pentium Processors	01H	Not Implemented
Pentium Pro and Pentium II Processors, Intel® Celeron® Processors	02H	Not Implemented
Pentium III Processors	03H	Not Implemented
Pentium 4 Processors	02H	80000004H
Intel Xeon Processors	02H	80000004H
Pentium M Processor	02H	80000004H
Pentium 4 Processor supporting Hyper-Threading Technology	05H	80000008H
Pentium D Processor (8xx)	05H	80000008H
Pentium D Processor (9xx)	06H	80000008H
Intel Core Duo Processor	0AH	80000008H
Intel Core 2 Duo Processor	0AH	80000008H
Intel Xeon Processor 3000, 3200, 5100, 5300 Series	0AH	80000008H

IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CPUID executes. The signature is returned in the upper dword. For details, see Chapter 9 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

INPUT EAX = 1: Returns Model, Family, Stepping Information

When CPUID executes with EAX set to 1, version information is returned in EAX (see Figure 3-1). For example: extended family, extended model, model, family, and processor type for the processor code-named Penryn is as follows:

- Extended Model — 0001B
- Extended Family — 0000_0000B
- Model — 0111B
- Family — 0110B
- Processor Type — 00B

See Table 3-3 for available processor type values. Stepping IDs are provided as needed.

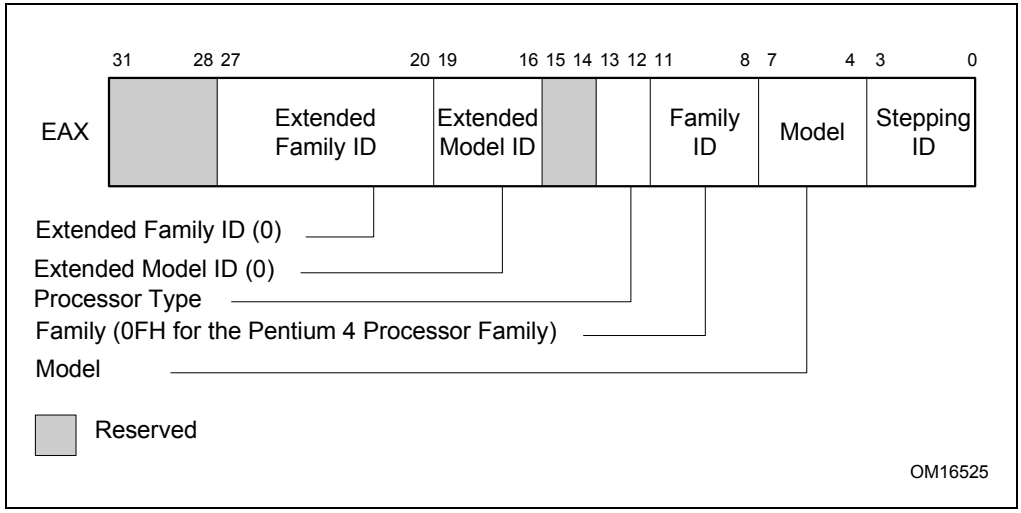

Figure 3-1. Version Information Returned by CPUID in EAX

Table 3-3. Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

Extended family, extended model, model, family, and processor type for the processor code-named Nehalem is as follows:

- Extended Model — 0001B
- Extended Family — 0000_0000B
- Model — 1010B
- Family — 0110B
- Processor Type — 00B

NOTE

See *AP-485, Intel Processor Identification and the CPUID Instruction* (Order Number 241618) and Chapter 14 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for information on identifying earlier IA-32 processors.

The Extended Family ID needs to be examined only when the Family ID is 0FH. Integrate the fields into a display using the following rule:

```
IF Family_ID ≠ 0FH
    THEN Displayed_Family = Family_ID;
    ELSE Displayed_Family = Extended_Family_ID + Family_ID;
    (* Right justify and zero-extend 4-bit field. *)
FI;
(* Show Display_Family as HEX field. *)
```

The Extended Model ID needs to be examined only when the Family ID is 06H or 0FH. Integrate the field into a display using the following rule:

```
IF (Family_ID = 06H or Family_ID = 0FH)
    THEN Displayed_Model = (Extended_Model_ID << 4) + Model_ID;
    (* Right justify and zero-extend Extended_Model_ID and Model_ID. *)
    ELSE Displayed_Model = Model_ID;
FI;
(* Show Display_Model as HEX field. *)
```

INPUT EAX = 1: Returns Additional Information in EBX

When CPUID executes with EAX set to 1, additional information is returned to the EBX register:

- Brand index (low byte of EBX) — this number provides an entry into a brand string table that contains brand strings for IA-32 processors. More information about this field is provided later in this section.
- CLFLUSH instruction cache line size (second byte of EBX) — this number indicates the size of the cache line flushed with CLFLUSH instruction in 8-byte increments. This field was introduced in the Pentium 4 processor.
- Local APIC ID (high byte of EBX) — this number is the 8-bit ID that is assigned to the local APIC on the processor during power up. This field was introduced in the Pentium 4 processor.

INPUT EAX = 1: Returns Feature Information in ECX and EDX

When CPUID executes with EAX set to 1, feature information is returned in ECX and EDX.

- Figure 3-2 and Table 3-4 show encodings for ECX.
- Figure 3-3 and Table 3-5 show encodings for EDX.

For all feature flags, a 1 indicates that the feature is supported. Use Intel to properly interpret feature flags.

NOTE

Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

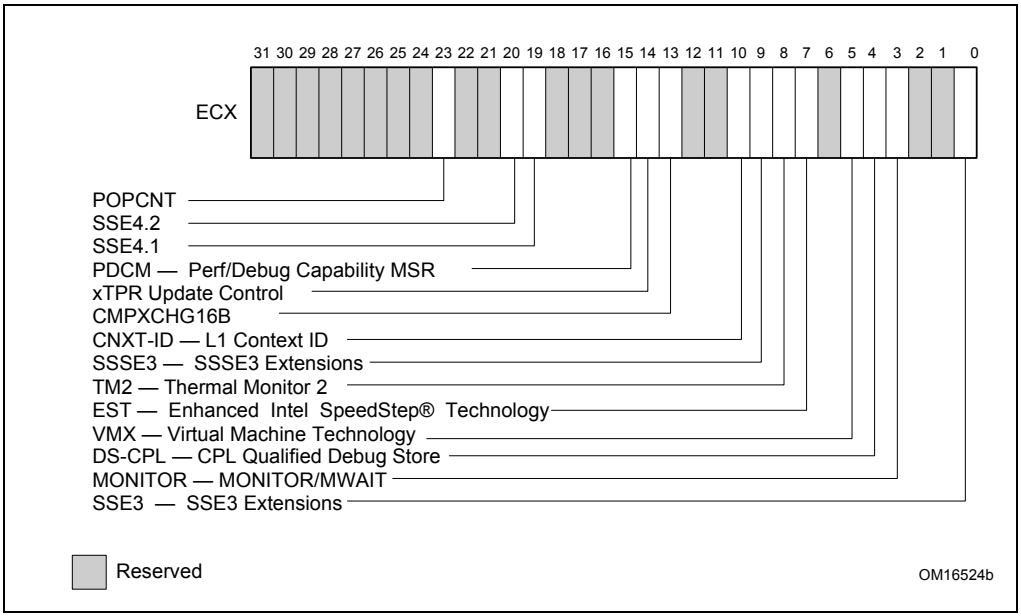

Figure 3-2. Extended Feature Information Returned in the ECX Register

Table 3-4. More on Extended Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Streaming SIMD Extensions 3 (SSE3)**. A value of 1 indicates the processor supports this technology.
1-2	Reserved	Reserved
3	MONITOR	**MONITOR/MWAIT**. A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store**. A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	Virtual Machine Extensions. A value of 1 indicates that the processor supports this technology.
6	Reserved	Reserved
7	EST	**Enhanced Intel SpeedStep® technology**. A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2**. A value of 1 indicates whether the processor supports this technology.

Table 3-4. More on Extended Feature Information Returned in the ECX Register (Contd.)

Bit #	Mnemonic	Description
9	SSSE3	**Supplemental Streaming SIMD Extensions 3 (SSSE3)**. A value of 1 indicates the processor supports this technology.
10	CNXT-ID	**L1 Context ID**. A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11-12	Reserved	Reserved
13	CMPXCHG16B	**CMPXCHG16B Available**. A value of 1 indicates that the feature is available. See the "CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes" section in Volume 2A.
14	xTPR Update Control	**xTPR Update Control**. A value of 1 indicates that the processor supports changing IA32_MISC_ENABLES[bit 23].
15	PDCM	**Perf/Debug Capability MSR**. A value of 1 indicates that the processor supports the performance and debug feature indication MSR
18 - 16	Reserved	Reserved
19	SSE4.1	**Streaming SIMD Extensions 4.1 (SSE4.1)**. A value of 1 indicates the processor supports this technology.
20	SSE4.2	**Streaming SIMD Extensions 4.2 (SSE4.2)**. A value of 1 indicates the processor supports this technology.
22 - 21	Reserved	Reserved
23	POPCNT	**POPCNT**. A value of 1 indicates the processor supports the POPCNT instruction.
31 - 24	Reserved	Reserved

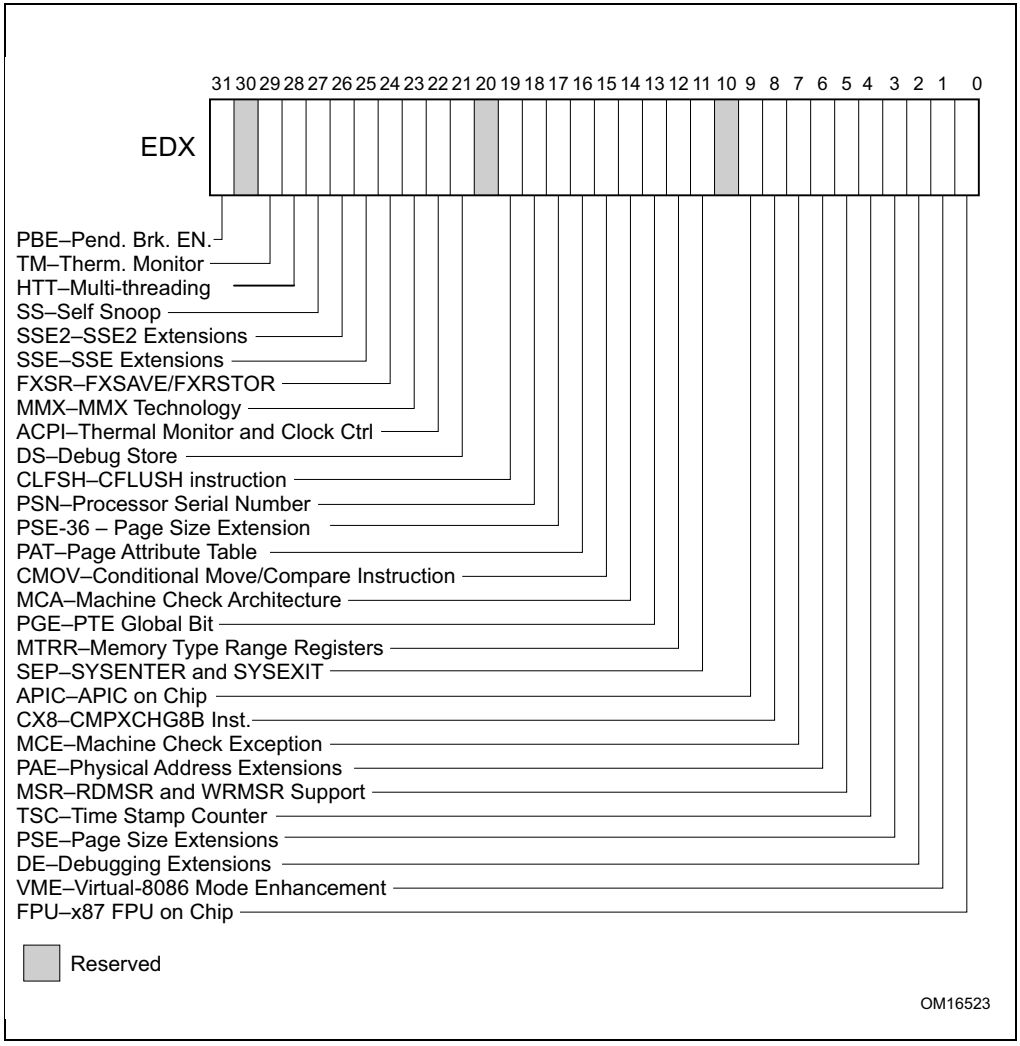

Figure 3-3. Feature Information Returned in the EDX Register

Table 3-5. More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.

Table 3-5. More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1. The actual number of address bits beyond 32 is not defined, and is implementation specific.
7	MCE	**Machine Check Exception**. Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	**PTE Global Bit.** The global bit in page directory entries (PDEs) and page table entries (PTEs) is supported, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.

Table 3-5. More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
14	MCA	**Machine Check Architecture.** The Machine Check Architecture, which provides a compatible mechanism for error reporting in P6 family, Pentium 4, Intel Xeon processors, and future processors, is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory on a 4K granularity through a linear address.
17	PSE-36	**36-Bit Page Size Extension.** Extended 4-MByte pages that are capable of addressing physical memory beyond 4 GBytes are supported. This feature indicates that the upper four bits of the physical address of the 4-MByte page is encoded by bits 13-16 of the page directory entry.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 18, "Debugging and Performance Monitoring," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*).
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.

Table 3-5. More on Feature Information Returned in the EDX Register (Contd.)

Bit #	Mnemonic	Description
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Multi-Threading.** The physical processor package is capable of supporting more than one logical processor.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	**Pending Break Enable.** The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

INPUT EAX = 2: Cache and TLB Information Returned in EAX, EBX, ECX, EDX

When CPUID executes with EAX set to 2, the processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers.

The encoding is as follows:

- The least-significant byte in register EAX (register AL) indicates the number of times the CPUID instruction must be executed with an input value of 2 to get a complete description of the processor's caches and TLBs. The first member of the family of Pentium 4 processors will return a 1.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. Table 3-6 shows the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache or TLB types. The descriptors may appear in any order.

Table 3-6. Encoding of Cache and TLB Descriptors

Descriptor Value	Cache or TLB Description
00H	Null descriptor
01H	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries
02H	Instruction TLB: 4 MByte pages, 4-way set associative, 2 entries
03H	Data TLB: 4 KByte pages, 4-way set associative, 64 entries
04H	Data TLB: 4 MByte pages, 4-way set associative, 8 entries
05H	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries
06H	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size
08H	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size
0AH	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size
0BH	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries
0CH	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size
22H	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector
23H	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
25H	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
29H	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector
2CH	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size
30H	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size
40H	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache
41H	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size
42H	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size
43H	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size
44H	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size
45H	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size
46H	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size
47H	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size
48H	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size
49H	2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size

Table 3-6. Encoding of Cache and TLB Descriptors (Contd.)

Descriptor Value	Cache or TLB Description
4AH	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size
4BH	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size
4DH	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size
4EH	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size
50H	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries
51H	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries
52H	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries
56H	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries
57H	Data TLB0: 4 KByte pages, 4-way associative, 16 entries
5BH	Data TLB: 4 KByte and 4 MByte pages, 64 entries
5CH	Data TLB: 4 KByte and 4 MByte pages,128 entries
5DH	Data TLB: 4 KByte and 4 MByte pages,256 entries
60H	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size
66H	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size
67H	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size
68H	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size
70H	Trace cache: 12 K-µop, 8-way set associative
71H	Trace cache: 16 K-µop, 8-way set associative
72H	Trace cache: 32 K-µop, 8-way set associative
78H	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size
79H	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7AH	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7BH	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector
7CH	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector
7DH	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size
7FH	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size
82H	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size
83H	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size

Table 3-6. Encoding of Cache and TLB Descriptors (Contd.)

Descriptor Value	Cache or TLB Description
84H	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size
85H	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size
86H	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size
87H	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size
B0H	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries
B3H	Data TLB: 4 KByte pages, 4-way set associative, 128 entries
B4H	Data TLB1: 4 KByte pages, 4-way associative, 256 entries
F0H	64-Byte prefetching
F1H	128-Byte prefetching

Example 3-1. Example of Cache and TLB Interpretation

The first member of the family of Pentium 4 processors returns the following information about caches and TLBs when the CPUID executes with an input value of 2:

```
EAX     66 5B 50 01H
EBX     0H
ECX     0H
EDX     00 7A 70 00H
```

Which means:

- The least-significant byte (byte 0) of register EAX is set to 01H. This indicates that CPUID needs to be executed once with an input value of 2 to retrieve complete information about caches and TLBs.
- The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.
- Bytes 1, 2, and 3 of register EAX indicate that the processor has:
  - 50H - a 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.
  - 5BH - a 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.
  - 66H - an 8-KByte 1st level data cache, 4-way set associative, with a 64-Byte cache line size.
- The descriptors in registers EBX and ECX are valid, but contain NULL descriptors.
- Bytes 0, 1, 2, and 3 of register EDX indicate that the processor has:
  - 00H - NULL descriptor.
  - 70H - Trace cache: 12 K-μop, 8-way set associative.

— 7AH - a 256-KByte 2nd level cache, 8-way set associative, with a sectored, 64-byte cache line size.

— 00H - NULL descriptor.

INPUT EAX = 4: Returns Deterministic Cache Parameters for Each Level

When CPUID executes with EAX set to 4 and ECX contains an index value, the processor returns encoded data that describe a set of deterministic cache parameters (for the cache level associated with the input in ECX). Valid index values start from 0.

Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0. The architecturally defined fields reported by deterministic cache parameters are documented in Table 3-1.

The CPUID leaf 4 also reports information about maximum number of cores in a physical package. This information is constant for all valid index values. Software can query maximum number of cores per physical package by executing CPUID with EAX=4 and ECX=0.

INPUT EAX = 5: Returns MONITOR and MWAIT Features

When CPUID executes with EAX set to 5, the processor returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management. See Table 3-1.

INPUT EAX = 6: Returns Thermal and Power Management Features

When CPUID executes with EAX set to 6, the processor returns information about thermal and power management features. See Table 3-1.

INPUT EAX = 10: Returns Architectural Performance Monitoring Features

When CPUID executes with EAX set to 10, the processor returns information about support for architectural performance monitoring capabilities. Architectural performance monitoring is supported if the version ID (see Table 3-1) is greater than Pn 0. See Table 3-1.

For each version of architectural performance monitoring capability, software must enumerate this leaf to discover the programming facilities and the architectural performance events available in the processor. The details are described in Chapter 18, “Debugging and Performance Monitoring,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*.

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

1. Processor brand string method; this method also returns the processor's maximum operating frequency
2. Processor brand index; this method uses a software supplied brand string table.

These two methods are discussed in the following sections. For methods that are available in early processors, see Section: "Identification of Earlier IA-32 Processors" in Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*.

The Processor Brand String Method

Figure 3-4 describes the algorithm used for detection of the brand string. Processor brand identification software should execute this algorithm on all Intel 64 and IA-32 processors.

This method (introduced with Pentium 4 processors) returns an ASCII brand identification string and the maximum operating frequency of the processor to the EAX, EBX, ECX, and EDX registers.

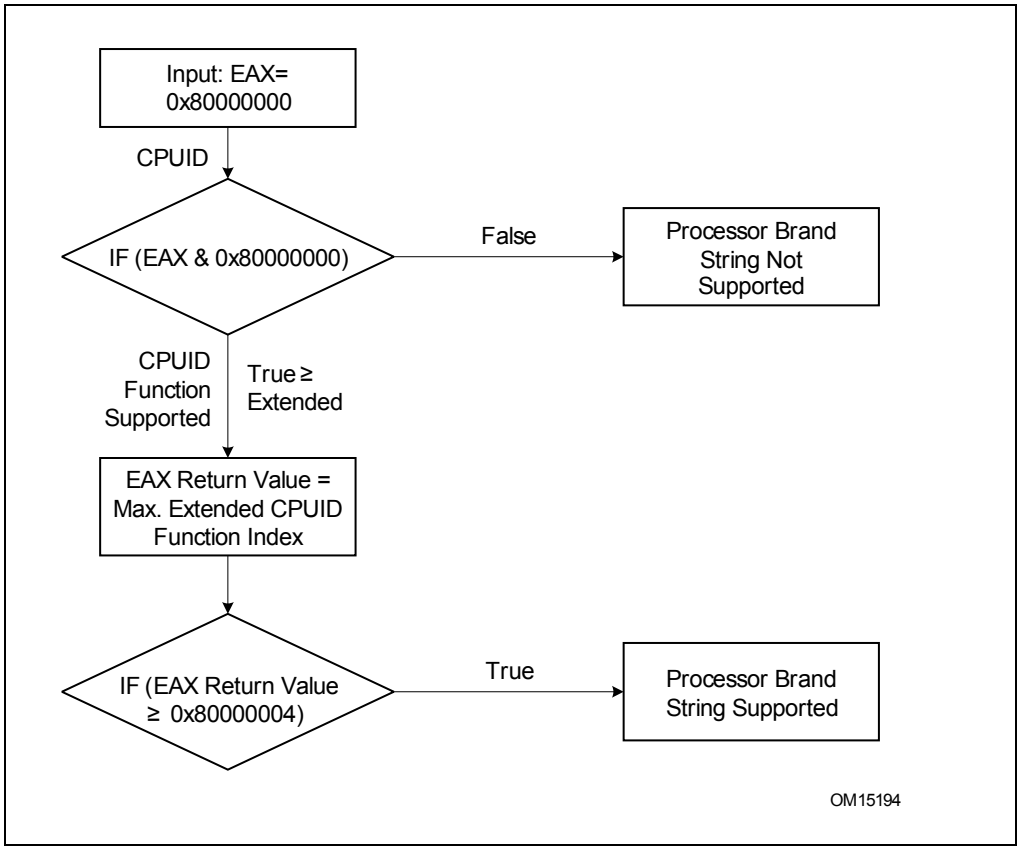

Figure 3-4. Determination of Support for the Processor Brand String

How Brand Strings Work

To use the brand string method, execute CPUID with EAX input of 8000002H through 80000004H. For each input value, CPUID returns 16 ASCII characters using EAX, EBX, ECX, and EDX. The returned string will be NULL-terminated.

Table 3-7 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 3-7. Processor Brand String Returned with Pentium 4 Processor

EAX Input Value	Return Values	ASCII Equivalent
80000002H	EAX = 20202020H EBX = 20202020H ECX = 20202020H EDX = 6E492020H	“ ” “ ” “ ” “nI ”
80000003H	EAX = 286C6574H EBX = 50202952H ECX = 69746E65H EDX = 52286D75H	“(let” “P)R” “itne” “R(mu”
80000004H	EAX = 20342029H EBX = 20555043H ECX = 30303531H EDX = 007A484DH	“ 4)” “ UPC” “0051” “\0zHM”

Extracting the Maximum Processor Frequency from Brand Strings

Figure 3-5 provides an algorithm which software can use to extract the maximum processor operating frequency from the processor brand string.

NOTE

When a frequency is given in a brand string, it is the maximum qualified frequency of the processor, not the frequency at which the processor is currently running.

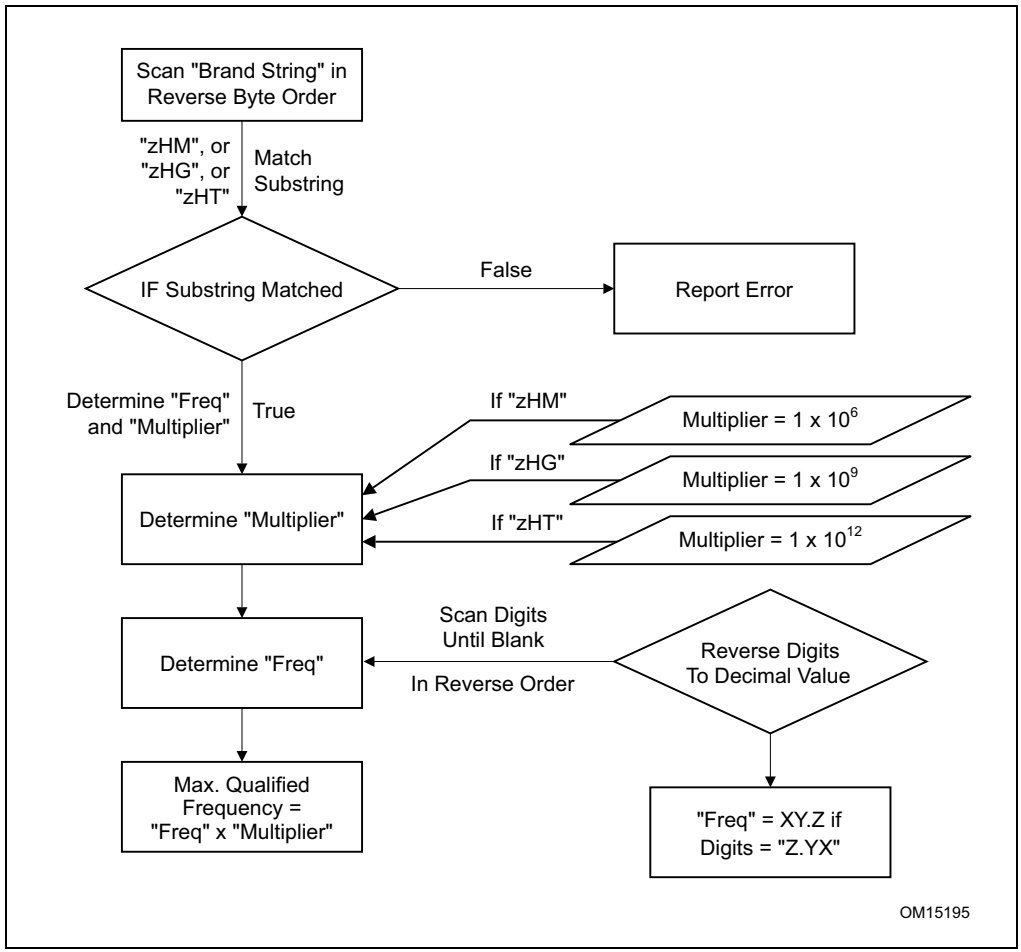

Figure 3-5. Algorithm for Extracting Maximum Processor Frequency

The Processor Brand Index Method

The brand index method (introduced with Pentium® III Xeon® processors) provides an entry point into a brand identification table that is maintained in memory by system software and is accessible from system- and user-level code. In this table, each brand index is associate with an ASCII brand identification string that identifies the official Intel family and model number of a processor.

When CPUID executes with EAX set to 1, the processor returns a brand index to the low byte in EBX. Software can then use this index to locate the brand identification string for the processor in the brand identification table. The first entry (brand index 0) in this table is reserved, allowing for backward compatibility with processors that do not support the brand identification feature. Starting with processor signature

family ID = 0FH, model = 03H, brand index method is no longer supported. Use brand string method instead.

Table 3-8 shows brand indices that have identification strings associated with them.

Table 3-8. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

Brand Index	Brand String
00H	This processor does not support the brand identification feature
01H	Intel(R) Celeron(R) processor[1]
02H	Intel(R) Pentium(R) III processor[1]
03H	Intel(R) Pentium(R) III Xeon(R) processor; If processor signature = 000006B1h, then Intel(R) Celeron(R) processor
04H	Intel(R) Pentium(R) III processor
06H	Mobile Intel(R) Pentium(R) III processor-M
07H	Mobile Intel(R) Celeron(R) processor[1]
08H	Intel(R) Pentium(R) 4 processor
09H	Intel(R) Pentium(R) 4 processor
0AH	Intel(R) Celeron(R) processor[1]
0BH	Intel(R) Xeon(R) processor; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor MP
0CH	Intel(R) Xeon(R) processor MP
0EH	Mobile Intel(R) Pentium(R) 4 processor-M; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor
0FH	Mobile Intel(R) Celeron(R) processor[1]
11H	Mobile Genuine Intel(R) processor
12H	Intel(R) Celeron(R) M processor
13H	Mobile Intel(R) Celeron(R) processor[1]
14H	Intel(R) Celeron(R) processor
15H	Mobile Genuine Intel(R) processor
16H	Intel(R) Pentium(R) M processor
17H	Mobile Intel(R) Celeron(R) processor[1]
18H – 0FFH	RESERVED

NOTES:

1. Indicates versions of these processors that were introduced after the Pentium III

3.2 DETECTING SSE4 INSTRUCTIONS

3.2.1 Detecting SSE4.1 Instructions Using CPUID

In order for an application to use SSE4.1, the following conditions must exist. Otherwise, an invalid opcode exception (Int 6) is generated:

- CR0.EM = 0 (emulation disabled)
- CR4.OSFXSR = 1(OS supports saving Streaming SIMD Extensions state during context switches)
- CPUID.01H:ECX.SSE4_1 [bit 19]= 1 (processor supports SSE4.1)

An application can determine whether SSE4.1 is supported by checking the CPUID feature flag at CPUID.01H:ECX[Bit 19]. The essential steps are illustrated in the pseudo code below.

Checking for SSE4.1 Support

```
unsigned RegECX;

boolean SSE4_1_instructions_work = TRUE;
asm{     // pseudo operation illustrating
    eax <- 1   // which CPUID feature flag to check
    cpuid
    RegECX <- ecx
    }
    if (RegECX[bit 19] ) SSE4_1_instructions_work = TRUE;
    // Add appropriate code as needed to ensure
    // OS providing adequate support for context switching, etc...
    return SSE4_1_instructions_work;
```

3.2.2 Detecting SSE4.2 Instructions Using CPUID

In order for an application to use PCMPGTQ and the text/string search instructions in SSE4.2, the following conditions must exist. Otherwise, an invalid opcode exception (Int 6) is generated:

- CR0.EM = 0 (emulation disabled)
- CR4.OSFXSR = 1(OS supports saving SSE state during context switches)
- CPUID.01H:ECX.SSE4_2 [bit 20]= 1 (processor supports SSE4.2)

An application can determine whether the desired SSE4.2 instructions are supported by checking the CPUID feature flag at CPUID.01H:ECX[Bit 20]. The essential steps are illustrated in the pseudo code below.

Example 3-2. Detecting SSE4.2 using CPUID

```
unsigned RegECX;

boolean SSE4_2_instructions_work = TRUE;
asm{     // pseudo operation illustrating
    eax <- 1   // which CPUID feature flag to check
    cpuid
    RegECX <- ecx
    }
    if (RegECX[bit 20] ) SSE4_2_instructions_work = TRUE;
    // Add appropriate code as needed to ensure
    // OS providing adequate support for context switching, etc...
    return SSE4_2_instructions_work;
```

In order for an application to use CRC32 instruction, the following condition must exist. Otherwise, an invalid opcode exception (INT 6) is generated:

CPUID.01H:ECX.SSE4_2 [bit 20]= 1 (processor supports SSE4.2)

In order for an application to use POPCNT instruction, the following condition must exist. Otherwise, an invalid opcode exception (INT 6) is generated:

CPUID.01H:ECX.SSE4_2 [bit 23]= 1 (processor supports POPCNT)

3.3 EXCEPTIONS AND SSE4

The SSE4.1 and SSE4.2 instruction sets do not introduce new types of exceptions.

CHAPTER 4
SYSTEM PROGRAMMING MODEL

This chapter describes the interface of the SSE4 to the operating system.

4.1 ENABLING SSE4

SSE4.1 and SSE4.2 are extensions of SSE, SSE2, SSE3, and SSSE3.

To check if the processor supports SSE4.1, execute CPUID with EAX =1 as input. If bit 19 of ECX is set, then the processor supports SSE4.1. If the bit is cleared, the processor does not support SSE4.1.

To check if the processor supports SSE4.2 instructions for string/text processing, PCMPGTQ, and CRC32, execute CPUID with EAX =1 as input. If bit 20 of ECX is set, then the processor supports these SSE4.2 instructions. If the bit is cleared, the processor does not support them.

Enabling OS support for SSE4.1, PCMPGTQ, string/text processing instructions of SSE4.2 has the same requirements as for SSE. See Chapter 12, “System Programming for Streaming SIMD Instruction Sets” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

To check if the processor supports the POPCNT instruction, execute CPUID with EAX =1 as input. If bit 23 of ECX is set, then the processor supports the POPCNT instruction. If the bit is cleared, the processor does not support it.

Operating system does not require special support to enable CRC32 or POPCNT beyond normal requirements of Intel 64 architecture.

4.2 DEVICE NOT AVAILABLE (DNA) EXCEPTIONS

If CR0.TS is set, attempts to execute an SSE4.1 instruction will cause a DNA exception (#NM). Likewise, an attempt to execute PCMPGTQ or a string/text processing instruction of SSE4.2 will cause a DNA exception (#NM).

If CPUID.01H:ECX.SSE4.1 [bit 19] is clear, execution of any SSE4.1 instruction causes an invalid opcode fault regardless of the state of CR0.TS.

If CPUID.01H:ECX.SSE4.2 [bit 20] is clear, execution of PCMPGTQ or a string/text processing instruction of SSE4.2 causes an invalid opcode fault regardless of the state of CR0.TS.

4.3 SSE4 EMULATION

The CR0.EM bit enables emulation of x87 floating-point instructions. It cannot be used to emulate SSE4.1. Likewise, the bit cannot be used to emulate PCMPGTQ or any of the string text processing instructions of SSE4.2.

If an SSE4.1 instruction is executed when CR0.EM is set, an Invalid Opcode exception (Int 6) is generated instead of a Device Not Available exception (INT 7).

If PCMPGTQ or an SSE4.2 string text processing instruction is executed while CR0.EM = 1, an Invalid Opcode exception (INT 6) is generated instead of a Device Not Available exception (INT 7).

CRC32 and POPCNT are not impacted by CR0.TS or CR0.EM.

CHAPTER 5
SSE4 INSTRUCTION SET

5.1 INSTRUCTION FORMATS

SSE4 uses existing instruction formats. Instructions use the ModR/M format and, in general, operations are not duplicated to provide two directions (i.e. separate load and store variants).

5.2 NOTATIONS

Besides opcodes, the following notation describes information in the ModR/M byte:

- **/digit:** (digit between 0 and 7) indicates that the instruction uses only the r/m (register and memory) operand. The reg field contains the digit that provides an extension to the instruction's opcode.
- **/digitR:** (digit between 0 and 7) indicates that the instruction uses only the register operand (ie, mod=11). The reg field contains the digit that provides an extension to the instruction's opcode.
- **/r**: indicates that the ModR/M byte of an instruction contains both a register operand and an r/m operand.

In addition, these abbreviations are used:

- **r32**: Intel Architecture 32-bit integer register
- **xmm/m128**: indicates a 128-bit Streaming SIMD Extensions/Streaming SIMD Extensions 2 register or a 128-bit memory location.
- **xmm/m64**: indicates a 128-bit Streaming SIMD Extensions/Streaming SIMD Extensions 2 register or a 64-bit memory location.
- **xmm/m32**: indicates a 128-bit Streaming SIMD Extensions/Streaming SIMD Extensions 2 register or a 32-bit memory location.
- **mm/m64**: indicates a 64-bit integer MMX™ multimedia register or a 64-bit memory location.
- **imm8**: indicates an immediate 8-bit operand.
- **ib**: indicates that an immediate byte operand follows the opcode, ModR/M byte or scaled-indexing byte.
- **<XMM0>:** indicates implied use of the XMM0 register.

When there is ambiguity, xmm1 indicates the first source operand using an XMM register and xmm2 the second source operand using an XMM register.

Some instructions use the XMM0 register as the third source operand, indicated by <XMM0>. The use of the third XMM register operand is implicit in the instruction encoding and does not affect the ModR/M encoding.

5.3 IMM8 CONTROL BYTE OPERATION FOR PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM

The operation of the immediate control byte (see Section 2.3.1) is common to the four string text processing instructions of SSE4.2. This section describes these common operations. Some of the notations introduced in this section are referenced in the reference pages of each instruction.

5.3.1 General Description

The operation of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM is defined by the combination of the respective opcode and the interpretation of an immediate control byte that is part of the instruction encoding.

The opcode controls the relationship of input bytes/words to each other (determines whether the inputs terminated strings or whether lengths are expressed explicitly) as well as the desired output (index or mask).

The Imm8 Control Byte for PCMPESTRM/PCMPESTRI/PCMPISTRM/PCMPISTRI encodes a significant amount of programmable control over the functionality of those instructions. Some functionality is unique to each instruction while some is common across some or all of the four instructions. This section describes functionality which is common across the four instructions.

The arithmetic flags (ZF, CF, SF, OF, AF, PF) are set as a result of these instructions. However, the meanings of the flags have been overloaded from their typical meanings in order to provide additional information regarding the relationships of the two inputs.

PCMPxSTRx instructions perform arithmetic comparisons between all possible pairs of bytes or words, one from each packed input source operand. The boolean results of those comparisons are then aggregated in order to produce meaningful results. The Imm8 Control Byte is used to affect the interpretation of individual input elements as well as control the arithmetic comparisons used and the specific aggregation scheme.

Specifically, the Imm8 Control Byte consists of bit fields that control the following attributes:

- **Source data format —** Byte/word data element granularity, signed or unsigned elements

- **Aggregation operation —** Encodes the mode of per-element comparison operation and the aggregation of per-element comparisons into an intermediate result
- **Polarity —** Specifies intermediate processing to be performed on the intermediate result
- **Output selection —** Specifies final operation to produce the output (depending on index or mask) from the intermediate result

5.3.1.1 Source Data Format

Table 5-1. Source Data Format

Imm8[1:0]	Meaning	Description
00b	Unsigned bytes	Both 128-bit sources are treated as packed, unsigned bytes.
01b	Unsigned words	Both 128-bit sources are treated as packed, unsigned words.
10b	Signed bytes	Both 128-bit sources are treated as packed, signed bytes.
11b	Signed words	Both 128-bit sources are treated as packed, signed words.

If the Imm8 Control Byte has bit[0] cleared, each source contains 16 packed bytes. If the bit is set each source contains 8 packed words. If the Imm8 Control Byte has bit[1] cleared, each input contains unsigned data. If the bit is set each source contains signed data.

5.3.1.2 Aggregation Operation

Table 5-2. Aggregation Operation

Imm8[3:2]	Mode	Comparison
00b	Equal any	The arithmetic comparison is “equal.”
01b	Ranges	Arithmetic comparison is “greater than or equal” between even indexed bytes/words of reg and each byte/word of reg/mem. Arithmetic comparison is “less than or equal” between odd indexed bytes/words of reg and each byte/word of reg/mem. (reg/mem[m] >= reg[n] for n = even, reg/mem[m] <= reg[n] for n = odd)
10b	Equal each	The arithmetic comparison is “equal.”
11b	Equal ordered	The arithmetic comparison is “equal.”

All 256 (64) possible comparisons are always performed. The individual Boolean results of those comparisons are referred by “BoolRes[*Reg/Mem element index, Reg element index*].” Comparisons evaluating to “True” are represented with a 1, False with a 0 (positive logic). The initial results are then aggregated into a 16-bit (8-bit) intermediate result (IntRes1) using one of the modes described in the table below, as determined by Imm8 Control Byte bit[3:2].

See Section 5.3.1.5 for a description of the overrideIfDataInvalid() function used in Table 5-3.

Table 5-3. Aggregation Operation

Mode	Pseudocode
Equal any (find characters from a set)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i++ IntRes1[j] OR= overrideIfDataInvalid(BoolRes[j,i])

Table 5-3. Aggregation Operation (Contd.)

Ranges (find characters from ranges)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i+=2 IntRes1[j] OR= (overrideIfDataInvalid(BoolRes[j,i]) AND overrideIfDataInvalid(BoolRes[j,i+1]))
Equal each (string compare)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For i = 0 to UpperBound, i++ IntRes1[i] = overrideIfDataInvalid(BoolRes[i,i])
Equal ordered (substring search)	UpperBound = imm8[0] ? 7 :15; IntRes1 = imm8[0] ? 0xFF : 0xFFFF For j = 0 to UpperBound, j++ For i = 0 to UpperBound-j, k=j to UpperBound, k++, i++ IntRes1[j] AND= overrideIfDataInvalid(BoolRes[k,i])

5.3.1.3 Polarity

Table 5-4. Polarity

Imm8[5:4]	Operation	Description
00b	Positive Polarity (+)	IntRes2 = IntRes1
01b	Negative Polarity (-)	IntRes2 = -1 XOR IntRes1
10b	Masked (+)	IntRes2 = IntRes1
11b	Masked (-)	IntRes2[i] = IntRes1[i] if reg/mem[i] invalid, else = ~IntRes1[i]

IntRes1 may then be further modified by performing a 1's compliment, according to the value of the Imm8 Control Byte bit[4]. Optionally, a mask may be used such that only those IntRes1 bits which correspond to "valid" reg/mem input elements are complimented (note that the definition of a valid input element is dependant on the specific opcode and is defined in each opcode's description). The result of the possible negation is referred to as IntRes2.

5.3.1.4 Output Selection

Table 5-5. Ouput Selection

Imm8[6]	Operation	Description
0b	Least significant index	The index returned to ECX is of the least significant set bit in IntRes2.
1b	Most significant index	The index returned to ECX is of the most significant set bit in IntRes2.

For PCMPESTRI/PCMPISTRI, the Imm8 Control Byte bit[6] is used to determine if the index is of the least significant or most significant bit of IntRes2.

Table 5-6. Output Selection

Imm8[6]	Operation	Description
0b	Bit mask	IntRes2 is returned as the mask to the least significant bits of XMM0 with zero extension to 128 bits.
1b	Byte/word mask	IntRes2 is expanded into a byte/word mask (based on imm8[1]) and placed in XMM0. The expansion is performed by replicating each bit into all of the bits of the byte/word of the same index.

Specifically for PCMPESTRM/PCMPISTRM, the Imm8 Control Byte bit[6] is used to determine if the mask is a 16 (8) bit mask or a 128 bit byte/word mask.

5.3.1.5 Valid/Invalid Override of Comparisons

PCMPxSTRx instructions allow for the possibility that an end-of-string (EOS) situation may occur within the 128-bit packed data value (see the instruction descriptions below for details). Any data elements on either source that are determined to be past the EOS are considered to be invalid, and the treatment of invalid data within a comparison pair varies depending on the aggregation function being performed.

In general, the individual comparison result for each element pair BoolRes[i.j] can be forced true or false if one or more elements in the pair are invalid. See Table 5-7.

Table 5-7. Comparison Result for Each Element Pair BoolRes[i.j]

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

5.3.1.6 Summary of Im8 Control byte

Table 5-8. Summary of Imm8 Control Byte

Imm8	Description
-------0b	128-bit sources treated as 16 packed bytes.
-------1b	128-bit sources treated as 8 packed words.
------0-b	Packed bytes/words are unsigned.
------1-b	Packed bytes/words are signed.
----00--b	Mode is equal any.
----01--b	Mode is ranges.
----10--b	Mode is equal each.
----11--b	Mode is equal ordered.
---0----b	IntRes1 is unmodified.
---1----b	IntRes1 is negated (1's compliment).
--0-----b	Negation of IntRes1 is for all 16 (8) bits.
--1-----b	Negation of IntRes1 is masked by reg/mem validity.
-0------b	Index of the least significant, set, bit is used (regardless of corresponding input element validity). IntRes2 is returned in least significant bits of XMM0.
-1------b	Index of the most significant, set, bit is used (regardless of corresponding input element validity). Each bit of IntRes2 is expanded to byte/word.
0-------b	This bit currently has no defined effect, should be 0.
1-------b	This bit currently has no defined effect, should be 0.

5.3.1.7 Diagram Comparison and Aggregation Process

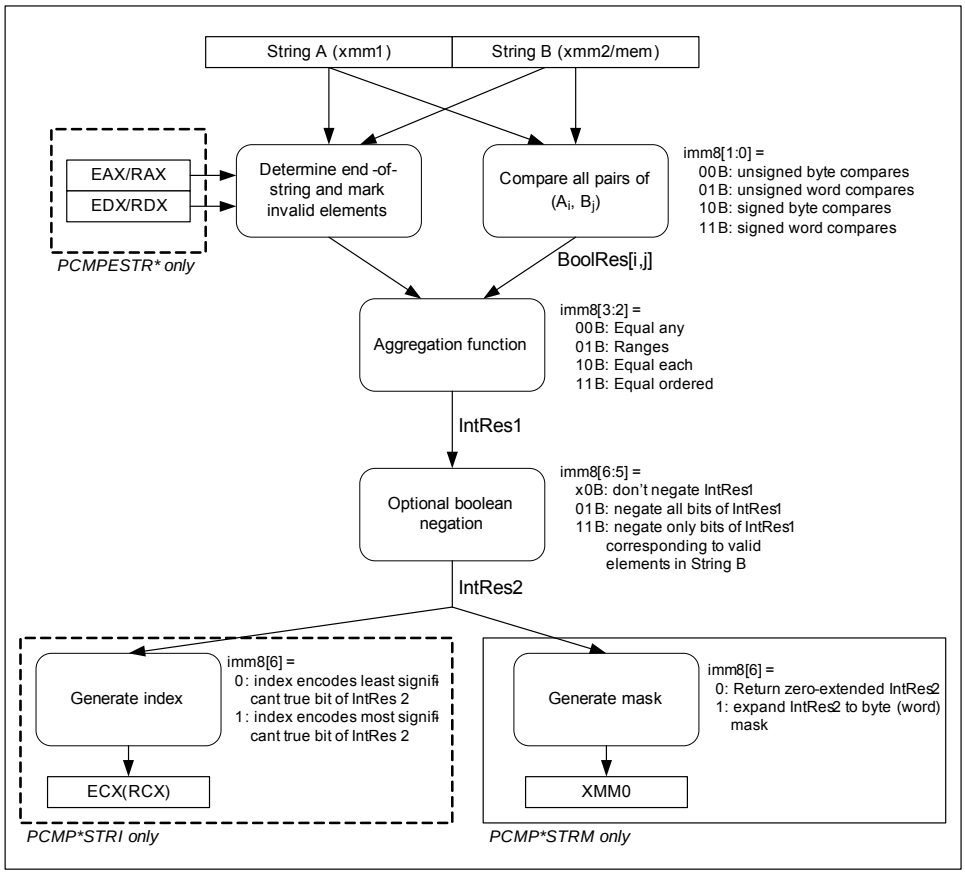

Figure 5-1. Operation of PCMPSTRx and PCMPESTRx

5.4 INSTRUCTION REFERENCE

The remainder of this chapter provides detailed descriptions of SSE4.1 and SSE4.2 instructions.

BLENDPD — Blend Packed Double Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 0D /r ib	BLENDPD *xmm1, xmm2/m128, imm8*	Valid	Valid	Select packed DP-FP values from *xmm1* and *xmm2/m128* from mask specified in imm8 and store the values into *xmm1*.

Description

Double Precision Floating-Point values from the source operand (second operand) are conditionally written to the destination operand depending on bits in the immediate operand. The immediate bits 0-1 (third operand) determine whether the corresponding DP-FP value in the destination is copied from the source (second argument).

If a bit in the mask, corresponding to a DP-FP value, is “1", then the DP-FP value is copied, else the value is left is unchanged.

Operation

BLENDPD

```
IF (imm8[0] == 1) THEN DEST[63:0] ← SRC[63:0];
    ELSE DEST[63:0] ← DEST[63:0];
IF (imm8[1] == 1) THEN DEST[127:64] ← SRC[127:64];
    ELSE DEST[127:64] ← DEST[127:64];
```

Intel C/C++ Compiler Intrinsic Equivalent

BLENDPD __m128d _mm_blend_pd (__m128d v1, __m128d v2, const int mask);

SIMD Floating-Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

BLENDPS — Blend Packed Single Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 0C /r ib	BLENDPS *xmm1, xmm2/m128, imm8*	Valid	Valid	Select packed single precision floating-point values from *xmm1* and *xmm2/m128* from mask specified in *imm8* and store the values into *xmm1*.

Description

Single Precision Floating-Point values from the source operand (second operand) are conditionally written to the destination operand (first operand) depending on mask bits in the immediate operand. The immediate bits 0-3 (third operand) determine whether the corresponding single precision floating-point value in the destination is copied from the source. If a bit in the mask, corresponding to a single precision floating-point value, is "1", then the single precision floating-point value is copied, else it is unchanged.

Operation

BLENDPS

```
IF (imm8[0] == 1) THEN DEST[31:0] ← SRC[31:0];
    ELSE DEST[31:0] ← DEST[31:0];
IF (imm8[1] == 1) THEN DEST[63:32] ← SRC[63:32];
    ELSE DEST[63:32] ← DEST[63:32];
IF (imm8[2] == 1) THEN DEST[95:64] ← SRC[95:64];
    ELSE DEST[95:64] ← DEST[95:64];
IF (imm8[3] == 1) THEN DEST[127:96] ← SRC[127:96];
    ELSE DEST[127:96] ← DEST[127:96];
```

Intel C/C++ Compiler Intrinsic Equivalent

BLENDPS __m128 _mm_blend_ps (__m128 v1, __m128 v2, const int mask);

SIMD Floating-Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.

	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If not aligned on 16-byte boundary, regardless of segment
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If not aligned on 16-byte boundary, regardless of segment
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

BLENDVPD — Variable Blend Packed Double Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 15 /r	BLENDVPD *xmm1, xmm2/m128 , <XMM0>*	Valid	Valid	Select packed DP FP values from *xmm1* and *xmm2* from mask specified in *XMM0* and store the values in *xmm1*.

Description

Double-precision floating-point values from the source operand (second argument) are conditionally written to the destination operand (first argument) depending on bits in the implicit third register argument. The most significant bit in the corresponding qword of XMM0 determines whether the destination DP FP value is copied from the source . The presence of a "1" in the mask bit indicates that the DP FP value is copied; otherwise it is left unchanged. The register assignment of the third operand is defined to be the architectural register XMM0.

Operation

BLENDPVD with implicit XMM0 register operand

```
MASK ← XMM0;
IF (MASK[63] == 1) THEN DEST[63:0] ← SRC[63:0];
    ELSE DEST[63:0] ← DEST[63:0];
IF (MASK[127] == 1) THEN DEST[127:64] ← SRC[127:64];
    ELSE DEST[127:64] ← DEST[127:64];
```

Intel C/C++ Compiler Intrinsic Equivalent

BLENDVPD __m128d _mm_blendv_pd(__m128d v1, __m128d v2, __m128d v3);

SIMD Floating-Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

BLENDVPS — Variable Blend Packed Single Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 14 /r	BLENDVPS *xmm1, xmm2/m128, <XMM0>*	Valid	Valid	Select packed single precision floating-point values from *xmm1* and *xmm2/m128* from mask specified in *XMM0* and store the values into *xmm1*.

Description

Single-precision floating-point values from the source operand (second argument) are conditionally written to the destination operand (first argument) depending on bits in the third register argument. The most significant bit in the corresponding dword in the third register determines whether the destination single precision floating-point value is copied from the source dword. The presence of a "1" in the mask bit indicates that the single precision floating-point value is copied; otherwise it is not copied. The register assignment of the third operand is defined to be the architectural register XMM0.

Operation

BLENDVPS with implicit XMM0 register operand

```
MASK ← XMM0;
IF (MASK[31] == 1) THEN DEST[31:0] ← SRC[31:0];
    ELSE DEST[31:0] ← DEST[31:0]);
IF (MASK[63] == 1) THEN DEST[63:32] ← SRC[63:32]);
    ELSE DEST[63:32] ← DEST[63:32]);
IF (MASK[95] == 1) THEN DEST[95:64] ← SRC[95:64]);
    ELSE DEST[95:64] ← DEST[95:64]);
IF (MASK[127] == 1) THEN DEST[127:96] ← SRC[127:96]);
    ELSE DEST[127:96] ← DEST[127:96]);
```

Intel C/C++ Compiler Intrinsic Equivalent

BLENDVPS __m128 _mm_blendv_ps(__m128 v1, __m128 v2, __m128 v3);

SIMD Floating-Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

CRC32 — Accumulate CRC32 Value

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
F2 0F 38 F0 /r	CRC32 *r32, r/m8*	Valid	Valid	Accumulate CRC32 on *r/m8*.
F2 REX 0F 38 F0 /r	CRC32 *r32, r/m8**	Valid	N.E.	Accumulate CRC32 on *r/m8*.
F2 0F 38 F1 /r	CRC32 *r32, r/m16*	Valid	Valid	Accumulate CRC32 on *r/m16*.
F2 0F 38 F1 /r	CRC32 *r32, r/m32*	Valid	Valid	Accumulate CRC32 on *r/m32*.
F2 REX.W 0F 38 F0 /r	CRC32 *r64, r/m8*	Valid	N.E.	Accumulate CRC32 on *r/m8*.
F2 REX.W 0F 38 F1 /r	CRC32 *r64, r/m64*	Valid	N.E.	Accumulate CRC32 on *r/m64*.

NOTES:

* In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Description

Starting with an initial value in the first operand (destination operand), accumulates a CRC32 (polynomial 0x11EDC6F41) value for the second operand (source operand) and stores the result in the destination operand. The source operand can be a register or a memory location. The destination operand must be an r32 or r64 register. If the destination is an r64 register, then the 32-bit result is stored in the least significant double word and 00000000H is stored in the most significant double word of the r64 register.

The initial value supplied in the destination operand is a double word integer stored in the r32 register or the least significant double word of the r64 register. To incrementally accumulate a CRC32 value, software retains the result of the previous CRC32 operation in the destination operand, then executes the CRC32 instruction again with new input data in the source operand. Data contained in the source operand is processed in reflected bit order. This means that the most significant bit of the source operand is treated as the least significant bit of the quotient, and so on, for all the bits of the source operand. Likewise, the result of the CRC operation is stored in the destination operand in reflected bit order. This means that the most significant bit of the resulting CRC (bit 31) is stored in the least significant bit of the destination operand (bit 0), and so on, for all the bits of the CRC.

Operation

In the pseudocode below, BIT_REFLECT on an N-bit wide operand is the bit-by-bit reflect operation from the most-significant bit to least-significant bit, as described in the paragraph above.

CRC32 instruction for 64-bit source operand and 64-bit destination operand:

```
TEMP1[63-0] ← BIT_REFLECT64 (SRC[63-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[95-0] ← TEMP1[63-0] << 32
TEMP4[95-0] ← TEMP2[31-0] << 64
TEMP5[95-0] ← TEMP3[95-0] XOR TEMP4[95-0]
TEMP6[31-0] ← TEMP5[95-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
DEST[63-32] ← 00000000H
```

CRC32 instruction for 32-bit source operand and 32-bit destination operand:

```
TEMP1[31-0] ← BIT_REFLECT32 (SRC[31-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[63-0] ← TEMP1[31-0] << 32
TEMP4[63-0] ← TEMP2[31-0] << 32
TEMP5[63-0] ← TEMP3[63-0] XOR TEMP4[63-0]
TEMP6[31-0] ← TEMP5[63-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
```

CRC32 instruction for 16-bit source operand and 32-bit destination operand:

```
TEMP1[15-0] ← BIT_REFLECT16 (SRC[15-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[47-0] ← TEMP1[15-0] << 32
TEMP4[47-0] ← TEMP2[31-0] << 16
TEMP5[47-0] ← TEMP3[47-0] XOR TEMP4[47-0]
TEMP6[31-0] ← TEMP5[47-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
```

CRC32 instruction for 8-bit source operand and 64-bit destination operand:

```
TEMP1[7-0] ← BIT_REFLECT8(SRC[7-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[39-0] ← TEMP1[7-0] << 32
TEMP4[39-0] ← TEMP2[31-0] << 8
TEMP5[39-0] ← TEMP3[39-0] XOR TEMP4[39-0]
TEMP6[31-0] ← TEMP5[39-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])
DEST[63-32] ← 00000000H
```

CRC32 instruction for 8-bit source operand and 32-bit destination operand:

```
TEMP1[7-0] ← BIT_REFLECT8(SRC[7-0])
TEMP2[31-0] ← BIT_REFLECT32 (DEST[31-0])
TEMP3[39-0] ← TEMP1[7-0] << 32
TEMP4[39-0] ← TEMP2[31-0] << 8
```

TEMP5[39-0] ← TEMP3[39-0] XOR TEMP4[39-0]
TEMP6[31-0] ← TEMP5[39-0] MOD2 11EDC6F41H
DEST[31-0] ← BIT_REFLECT (TEMP6[31-0])

Notes:

BIT_REFLECT64: DST[63-0] = SRC[0-63]
BIT_REFLECT32: DST[31-0] = SRC[0-31]
BIT_REFLECT16: DST[15-0] = SRC[0-15]
BIT_REFLECT8: DST[7-0] = SRC[0-7]

MOD2: Remainder from Polynomial division modulus 2

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

unsigned int _mm_crc32_u8(unsigned int crc, unsigned char data)
unsigned int _mm_crc32_u16(unsigned int crc, unsigned short data)
unsigned int _mm_crc32_u32(unsigned int crc, unsigned int data)
unsinged __int64 _mm_crc32_u64(unsinged __int64 crc, unsigned __int64 data)

SIMD Floating Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#UD	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

DPPD — Dot Product of Packed Double Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 41 /r ib	DPPD *xmm1, xmm2/m128, imm8*	Valid	Valid	Selectively multiply packed DP floating-point values from *xmm1* with packed DP floating-point values from *xmm2*, add and selectively store the packed DP floating-point values to *xmm1*.

Description

Conditionally multiplies the packed double precision floating-point values in the destination operand (first operand) with the packed double-precision floating-point values in the source(second operand) depending on a mask extracted from bits 4-5 of the immediate operand. Each of the two resulting double-precision values is summed and this sum is conditionally broadcast to each of 2 positions in the destination operand if the corresponding bit of the mask selected from bits 0-1 of the immediate operand is "1". If the corresponding low bit 0-1 of the mask is zero, the destination is set to zero.

DPPS follows the NaN forwarding rules stated in the Software Developer's Manual, vol. 1, table 4.7. These rules do not cover horizontal prioritization of NaNs. Horizontal propagation of NaNs to the destination and the positioning of those NaNs in the destination is implementation dependent. NaNs on the input sources or computationally generated NaNs will have at least one NaN propagated to the destination.

Operation

DPPD

```
IF (imm8[4] == 1) THEN Temp1[63:0] ← DEST[63:0] * SRC[63:0];
    ELSE Temp1[63:0] ← +0.0;
IF (imm8[5] == 1) THEN Temp1[127:64] ← DEST[127:64] * SRC[127:64];
    ELSE Temp1[127:64] ← +0.0;

Temp2[63:0] ← Temp1[63:0] + Temp1[127:64];

IF (imm8[0] == 1) THEN DEST[63:0] ← Temp2[63:0];
    ELSE DEST[63:0] ← +0.0;
IF (imm8[1] == 1) THEN DEST[127:64] ← Temp2[63:0];
    ELSE DEST[127:64] ← +0.0;
```

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

DPPD __m128d _mm_dp_pd (__m128d a, __m128d b, const int mask);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Exceptions are determined separately for each add and multiply operation. Unmasked exceptions will leave the destination untouched.

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If an unmasked SIMD floating-point exception and OSXM-MEXCPT in CR4 is 0.
	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#XM	If an unmasked SIMD floating-point exception and CR4.OSXM-MEXCPT[bit 10] = 1.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If an unmasked SIMD floating-point exception and OSXM-MEXCPT in CR4 is 0.
	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.

	If LOCK prefix is used.
#XM	If an unmasked SIMD floating-point exception and CR4.OSXM-MEXCPT[bit 10] = 1.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If an unmasked SIMD floating-point exception and OSXM-MEXCPT in CR4 is 0.
	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#XM	If an unmasked SIMD floating-point exception and CR4.OSXM-MEXCPT[bit 10] = 1.

DPPS — Dot Product of Packed Single Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 40 /r ib	DPPS *xmm1, xmm2/m128, imm8*	Valid	Valid	Selectively multiply packed SP floating-point values from *xmm1* with packed SP floating-point values from *xmm2*, add and selectively store the packed SP floating-point values or zero values to *xmm1*.

Description

Conditionally multiplies the packed single precision floating-point values in the destination operand (first operand) with the packed single-precision floats in the source (second operand) depending on a mask extracted from the high 4 bits of the immediate operand (third operand). Each of the four resulting single-precision values is summed and this sum is conditionally broadcast to each of 4 positions in the destination operand if the corresponding bit of the mask selected from the low 4 bits of the immediate operand is "1". If the corresponding low bit 0-3 of the mask is zero, the destination is set to zero.

DPPS follows the NaN forwarding rules stated in the Software Developer's Manual, vol. 1, table 4.7. These rules do not cover horizontal prioritization of NaNs. Horizontal propagation of NaNs to the destination and the positioning of those NaNs in the destination is implementation dependent. NaNs on the input sources or computationally generated NaNs will have at least one NaN propagated to the destination.

Operation

DPPS

```
IF (imm8[4] == 1) THEN Temp1[31:0] ← DEST[31:0] * SRC[31:0];
    ELSE Temp1[31:0] ← +0.0;
IF (imm8[5] == 1) THEN Temp1[63:32] ← DEST[63:32] * SRC[63:32];
    ELSE Temp1[63:32] ← +0.0;
IF (imm8[6] == 1) THEN Temp1[95:64] ← DEST[95:64] * SRC[95:64];
    ELSE Temp1[95:64] ← +0.0;
IF (imm8[7] == 1) THEN Temp1[127:96] ← DEST[127:96] * SRC[127:96];
    ELSE Temp1[127:96] ← +0.0;

Temp2[31:0] ← Temp1[31:0] + Temp1[63:32];
Temp3[31:0] ← Temp1[95:64] + Temp1[127:96];
Temp4[31:0] ← Temp2[31:0] + Temp3[31:0];
```

```
IF (imm8[0] == 1) THEN DEST[31:0] ← Temp4[31:0];
    ELSE DEST[31:0] ← +0.0;
IF (imm8[1] == 1) THEN DEST[63:32] ← Temp4[31:0];
    ELSE DEST[63:32] ← +0.0;
IF (imm8[2] == 1) THEN DEST[95:64] ← Temp4[31:0];
    ELSE DEST[95:64] ← +0.0;
IF (imm8[3] == 1) THEN DEST[127:96] ← Temp4[31:0];
    ELSE DEST[127:96] ← +0.0;
```

Intel C/C++ Compiler Intrinsic Equivalent

DPPS __m128 _mm_dp_ps (__m128 a, __m128 b, const int mask);

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal

Exceptions are determined separately for each add and multiply operation. Unmasked exceptions will leave the destination untouched.

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If an unmasked SIMD floating-point exception and OSXM-MEXCPT in CR4 is 0.
	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#XM	If an unmasked SIMD floating-point exception and CR4.OSXM-MEXCPT[bit 10] = 1.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.

#NM	If CR0.TS[bit 3] = 1.
#UD	If an unmasked SIMD floating-point exception and OSXM-MEXCPT in CR4 is 0.
	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#XM	If an unmasked SIMD floating-point exception and CR4.OSXM-MEXCPT[bit 10] = 1.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If an unmasked SIMD floating-point exception and OSXM-MEXCPT in CR4 is 0.
	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#XM	If an unmasked SIMD floating-point exception and CR4.OSXM-MEXCPT[bit 10] = 1.

EXTRACTPS — Extract Packed Single Precision Floating-Point Value

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 17 /r ib	EXTRACTPS *r/m32, xmm2, imm8*	Valid	Valid	Extract a single-precision floating-point value from *xmm2* at the source offset specified by *imm8* and store the result to *r/m32*.
66 REX.W 0F 3A 17 /r ib	EXTRACTPS *r64/m32, xmm2, imm8*	Valid	N.E.	Extract a single-precision floating-point value from *xmm2* at the source offset specified by *imm8* and store the result to *r64/m32*. Zero extend the result.

Description

Extract the single-precision floating-point value from the source xmm register (second argument) at a 32 bit offset determined from imm8[1-0]. The extracted single precision floating-point value is stored into the low 32-bits of the destination register or to the 32-bit memory location. When a REX.W prefix is used in 64-bit mode to a general purpose register (GPR), the packed single quantity is zero extended to 64 bits.

Operation

EXTRACTPS

```
IF (64-Bit Mode and REX.W used and the destination is a GPR )
THEN
    SRC_OFFSET ← imm8[1:0];
    r/m64[31:0] ← (SRC >> (32 * SRC_OFFSET)) AND 0FFFFFFFFh;
    r/m64[63:32] ← ZERO_FILL;
ELSE
    SRC_OFFSET ← imm8[1:0];
    r/m32[31:0] ← (SRC >> (32 * SRC_OFFSET)) AND 0FFFFFFFFh;
```

Intel C/C++ Compiler Intrinsic Equivalent

EXTRACTPS int _mm_extract_ps(__m128 src, const int ndx);

SIMD Floating-Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

INSERTPS — Insert Packed Single Precision Floating-Point Value

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 21 /r ib	INSERTPS *xmm1, xmm2/m32, imm8*	Valid	Valid	Insert a single precision floating-point value selected by *imm8* from *xmm2/m32* into xmm1 at the specified destination element specified by *imm8* and zero out destination elements in *xmm1* as indicated in *imm8*.

Description

Select a single precision floating-point element from source register (second operand, register form) as indicated by Count_S bits of the immediate operand (third operand) or load a floating-point element from memory indicated by the source (second operand, memory form) and insert it into the destination (first operand) at the location indicated by the Count_D bits of the immediate operand. Zero out destination elements based on the ZMask bits of the immediate operand.

Operation

INSERTPS

```
IF (SRC == REG) THEN COUNT_S ← imm8[7:6];
    ELSE COUNT_S ← 0;
COUNT_D ← imm8[5:4];
ZMASK ← imm8[3:0];

CASE (COUNT_S) OF
    0:  TMP ← SRC[31:0];
    1:  TMP ← SRC[63:32];
    2:  TMP ← SRC[95:64];
    3:  TMP ← SRC[127:96];

CASE (COUNT_D) OF
    0:  TMP2[31:0] ← TMP;
        TMP2[127:32] ← DEST[127:32];
    1:  TMP2[63:32] ← TMP;
        TMP2[31:0] ← DEST[31:0];
        TMP2[127:64] ← DEST[127:64];
    2:  TMP2[95:64] ← TMP;
        TMP2[63:0] ← DEST[63:0];
        TMP2[127:96] ← DEST[127:96];
```

```
    3:  TMP2[127:96] ← TMP;
        TMP2[95:0] ← DEST[95:0];

IF (ZMASK[0] == 1) THEN DEST[31:0] ← 00000000H;
    ELSE DEST[31:0] ← TMP2[31:0];
IF (ZMASK[1] == 1) THEN DEST[63:32] ← 00000000H;
    ELSE DEST[63:32] ← TMP2[63:32];
IF (ZMASK[2] == 1) THEN DEST[95:64] ← 00000000H;
    ELSE DEST[95:64] ← TMP2[95:64];
IF (ZMASK[3] == 1) THEN DEST[127:96] ← 00000000H;
    ELSE DEST[127:96] ← TMP2[127:96];
```

Intel C/C++ Compiler Intrinsic Equivalent

INSERTPS __m128 _mm_insert_ps(__m128 dst, __m128 src, const int ndx);

SIMD Floating-Point Exceptions

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

MOVNTDQA — Load Double Quadword Non-Temporal Aligned Hint

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 2A /r	MOVNTDQA *xmm1, m128*	Valid	Valid	Move double quadword from *m128* to *xmm* using non-temporal hint if WC memory type.

Description

MOVNTDQA loads a double quadword from the source operand (second operand) to the destination operand (first operand) using a non-temporal hint if the memory source is WC (write combining) memory type. For WC memory type, the non-temporal hint may be implemented by loading a temporary internal buffer with the equivalent of an aligned cache line without filling this data to the cache. Any memory-type aliased lines in the cache will be snooped and flushed. Subsequent MOVNTDQA reads to unread portions of the WC cache line will receive data from the temporary internal buffer if data is available. The temporary internal buffer may be flushed by the processor at any time for any reason, for example:

- A load operation other than a MOVNTDQA which references memory already resident in a temporary internal buffer.
- A non-WC reference to memory already resident in a temporary internal buffer.
- Interleaving of reads and writes to a single temporary internal buffer.
- Repeated MOVNTDQA loads of a particular 16-byte item in a streaming line.
- Certain micro-architectural conditions including resource shortages, detection of a mis-speculation condition, and various fault conditions

The non-temporal hint is implemented by using a write combining (WC) memory type protocol when reading the data from memory. Using this protocol, the processor does not read the data into the cache hierarchy, nor does it fetch the corresponding cache line from memory into the cache hierarchy. The memory type of the region being read can override the non-temporal hint, if the memory address specified for the non-temporal read is not a WC memory region. Information on non-temporal reads and writes can be found in Chapter 10, “Memory Cache Control” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Because the WC protocol uses a weakly-ordered memory consistency model, a fencing operation implemented with a MFENCE instruction should be used in conjunction with MOVNTDQA instructions if multiple processors might use different memory types for the referenced memory locations or in order to synchronize reads of a processor with writes by other agents in the system.

A processor’s implementation of the streaming load hint does not override the effective memory type, but the implementation of the hint is processor dependent. For example, a processor implementation may choose to ignore the hint and process the

instruction as a normal MOVDQA for any memory type. Alternatively, another implementation may optimize cache reads generated by MOVNTDQA on WB memory type to reduce cache evictions.

Operation

MOVNTDQA
DST ← SRC;

Intel C/C++ Compiler Intrinsic Equivalent

MOVNTDQA __m128i _mm_stream_load_si128 (__m128i *p);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

MPSADBW — Compute Multiple Packed Sums of Absolute Difference

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 42 /r ib	MPSADBW *xmm1, xmm2/m128, imm8*	Valid	Valid	Sums absolute 8-bit integer difference of adjacent groups of 4 byte integers in *xmm1* and *xmm2/m128* and writes the results in *xmm1*. Starting offsets within *xmm1* and *xmm2/m128* are determined by *imm8*.

Description

MPSADBW sums the absolute difference of 4 unsigned bytes, selected by bits [0:1] of the immediate byte (third operand), from the source (second operand) with sequential groups of 4 unsigned bytes in the destination operand. The first group of eight sequential groups of bytes from the destination operand (first operand) start at an offset determined by bit 2 of the immediate. The operation is repeated 8 times, each time using the same source input but selecting the next group of 4 bytes starting at the next higher byte in the destination. Each 16-bit sum is written to dest.

Operation

MPSADBW

SRC_OFFSET ← imm8[1:0]*32
DEST_OFFSET ← imm8[2]*32
DEST_BYTE0 ← DEST[DEST_OFFSET+7:DEST_OFFSET]
DEST_BYTE1 ← DEST[DEST_OFFSET+15:DEST_OFFSET+8]
DEST_BYTE2 ← DEST[DEST_OFFSET+23:DEST_OFFSET+16]
DEST_BYTE3 ← DEST[DEST_OFFSET+31:DEST_OFFSET+24]
DEST_BYTE4 ← DEST[DEST_OFFSET+39:DEST_OFFSET+32]
DEST_BYTE5 ← DEST[DEST_OFFSET+47:DEST_OFFSET+40]
DEST_BYTE6 ← DEST[DEST_OFFSET+55:DEST_OFFSET+48]
DEST_BYTE7 ← DEST[DEST_OFFSET+63:DEST_OFFSET+56]
DEST_BYTE8 ← DEST[DEST_OFFSET+71:DEST_OFFSET+64]
DEST_BYTE9 ← DEST[DEST_OFFSET+79:DEST_OFFSET+72]
DEST_BYTE10 ← DEST[DEST_OFFSET+87:DEST_OFFSET+80]

SRC_BYTE0 ← SRC[SRC_OFFSET+7:SRC_OFFSET]
SRC_BYTE1 ← SRC[SRC_OFFSET+15:SRC_OFFSET+8]
SRC_BYTE2 ← SRC[SRC_OFFSET+23:SRC_OFFSET+16]
SRC_BYTE3 ← SRC[SRC_OFFSET+31:SRC_OFFSET+24]

TEMP0 ← ABS(DEST_BYTE0 - SRC_BYTE0)

TEMP1 ← ABS(DEST_BYTE1 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE2 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE3 - SRC_BYTE3)
DEST[15:0] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE1 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE2 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE3 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE4 - SRC_BYTE3)
DEST[31:16] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE2 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE3 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE4 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE5 - SRC_BYTE3)
DEST[47:32] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE3 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE4 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE5 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE6 - SRC_BYTE3)
DEST[63:48] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE4 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE5 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE6 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE7 - SRC_BYTE3)
DEST[79:64] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE5 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE6 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE7 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE8 - SRC_BYTE3)
DEST[95:80] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE6 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE7 - SRC_BYTE1)
TEMP2 ← ABS(DEST_BYTE8 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE9 - SRC_BYTE3)
DEST[111:96] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

TEMP0 ← ABS(DEST_BYTE7 - SRC_BYTE0)
TEMP1 ← ABS(DEST_BYTE8 - SRC_BYTE1)

TEMP2 ← ABS(DEST_BYTE9 - SRC_BYTE2)
TEMP3 ← ABS(DEST_BYTE10 - SRC_BYTE3)
DEST[127:112] ← TEMP0 + TEMP1 + TEMP2 + TEMP3

Intel C/C++ Compiler Intrinsic Equivalent

MPSADBW __m128i _mm_mpsadbw_epu8 (__m128i s1, __m128i s2, const int mask);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PACKUSDW — Pack with Unsigned Saturation

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 2B /r	PACKUSDW *xmm1, xmm2/m128*	Valid	Valid	Convert 4 packed signed doubleword integers from *xmm1* and 4 packed signed doubleword integers from *xmm2/m128* into 8 packed unsigned word integers in *xmm1* using unsigned saturation.

Description

Converts packed signed doubleword integers into packed unsigned word integers using unsigned saturation to handle overflow conditions. If the signed doubleword value is beyond the range of an unsigned word (that is, greater than FFFFH or less than 0000H), the saturated unsigned word integer value of FFFFH or 0000H, respectively, is stored in the destination.

Operation

PACKUSDW

```
TMP[15:0] ← (DEST[31:0] < 0) ? 0 : DEST[15:0];
DEST[15:0] ← (DEST[31:0] > FFFFH) ? FFFFH : TMP[15:0] ;
TMP[31:16] ← (DEST[63:32] < 0) ? 0 : DEST[47:32];
DEST[31:16] ← (DEST[63:32] > FFFFH) ? FFFFH : TMP[31:16] ;
TMP[47:32] ← (DEST[95:64] < 0) ? 0 : DEST[79:64];
DEST[47:32] ← (DEST[95:64] > FFFFH) ? FFFFH : TMP[47:32] ;
TMP[63:48] ← (DEST[127:96] < 0) ? 0 : DEST[111:96];
DEST[63:48] ← (DEST[127:96] > FFFFH) ? FFFFH : TMP[63:48] ;
TMP[63:48] ← (DEST[127:96] < 0) ? 0 : DEST[111:96];
DEST[63:48] ← (DEST[127:96] > FFFFH) ? FFFFH : TMP[63:48] ;
TMP[79:64] ← (SRC[31:0] < 0) ? 0 : SRC[15:0];
DEST[63:48] ← (SRC[31:0] > FFFFH) ? FFFFH : TMP[79:64] ;
TMP[95:80] ← (SRC[63:32] < 0) ? 0 : SRC[47:32];
DEST[95:80] ← (SRC[63:32] > FFFFH) ? FFFFH : TMP[95:80] ;
TMP[111:96] ← (SRC[95:64] < 0) ? 0 : SRC[79:64];
DEST[111:96] ← (SRC[95:64] > FFFFH) ? FFFFH : TMP[111:96] ;
TMP[127:112] ← (SRC[127:96] < 0) ? 0 : SRC[111:96];
DEST[128:112] ← (SRC[127:96] > FFFFH) ? FFFFH : TMP[127:112] ;
```

Intel C/C++ Compiler Intrinsic Equivalent

PACKUSDW __m128i _mm_packus_epi32(__m128i m1, __m128i m2);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0):	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.SSE4_1(ECX bit 19) = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.SSE4_1(ECX bit 19) = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PBLENDVB — Variable Blend Packed Bytes

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 10 /r	PBLENDVB *xmm1, xmm2/m128, <XMM0>*	Valid	Valid	Select byte values from *xmm1* and *xmm2/m128* from mask specified in the high bit of each byte in *XMM0* and store the values into *xmm1*.

Description

Bytes from the source operand (second operand) are conditionally written to the destination operand (first operand) depending on mask bits defined in the implicit third register argument, XMM0. The most significant bit in the corresponding byte of XMM0 determines whether the destination byte is copied from the source byte. The presence of a "1" in the mask bit indicates that the byte is copied, else it is not copied and destination byte remains unchanged. The register assignment of the implicit third operand is defined to be the architectural register XMM0.

Operation

PBLENDVB with implicit XMM0 register operand

```
MASK ← XMM0;
IF (MASK[7] == 1) THEN DEST[7:0] ← SRC[7:0];
    ELSE DEST[7:0] ← DEST[7:0];
IF (MASK[15] == 1) THEN DEST[15:8] ← SRC[15:8];
    ELSE DEST[15:8] ← DEST[15:8];
IF (MASK[23] == 1) THEN DEST[23:16] ← SRC[23:16]
    ELSE DEST[23:16] ← DEST[23:16];
IF (MASK[31] == 1) THEN DEST[31:24] ← SRC[31:24]
    ELSE DEST[31:24] ← DEST[31:24];
IF (MASK[39] == 1) THEN DEST[39:32] ← SRC[39:32]
    ELSE DEST[39:32] ← DEST[39:32];
IF (MASK[47] == 1) THEN DEST[47:40] ← SRC[47:40]
    ELSE DEST[47:40] ← DEST[47:40];
IF (MASK[55] == 1) THEN DEST[55:48] ← SRC[55:48]
    ELSE DEST[55:48] ← DEST[55:48];
IF (MASK[63] == 1) THEN DEST[63:56] ← SRC[63:56]
    ELSE DEST[63:56] ← DEST[63:56];
IF (MASK[71] == 1) THEN DEST[71:64] ← SRC[71:64]
    ELSE DEST[71:64] ← DEST[71:64];
IF (MASK[79] == 1) THEN DEST[79:72] ← SRC[79:72]
```

```
    ELSE DEST[79:72] ← DEST[79:72];
IF (MASK[87] == 1) THEN DEST[87:80] ← SRC[87:80]
    ELSE DEST[87:80] ← DEST[87:80];
IF (MASK[95] == 1) THEN DEST[95:88] ← SRC[95:88]
    ELSE DEST[95:88] ← DEST[95:88];
IF (MASK[103] == 1) THEN DEST[103:96] ← SRC[103:96]
    ELSE DEST[103:96] ← DEST[103:96];
IF (MASK[111] == 1) THEN DEST[111:104] ← SRC[111:104]
    ELSE DEST[111:104] ← DEST[111:104];
IF (MASK[119] == 1) THEN DEST[119:112] ← SRC[119:112]
    ELSE DEST[119:112] ← DEST[119:112];
IF (MASK[127] == 1) THEN DEST[127:120] ← SRC[127:120]
    ELSE DEST[127:120] ← DEST[127:120])
```

Intel C/C++ Compiler Intrinsic Equivalent

PBLENDVB __m128i _mm_blendv_epi8 (__m128i v1, __m128i v2, __m128i mask);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If not aligned on 16-byte boundary, regardless of segment
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.

	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PBLENDW — Blend Packed Words

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 0E /r ib	PBLENDW *xmm1, xmm2/m128, imm8*	Valid	Valid	Select words from *xmm1* and *xmm2/m128* from mask specified in *imm8* and store the values into *xmm1*.

Description

Words from the source operand (second operand) are conditionally written to the destination operand (first operand) depending on bits in the immediate operand (third operand). The immediate bits (bits 7-0) form a mask that determines whether the corresponding word in the destination is copied from the source. If a bit in the mask, corresponding to a word, is "1", then the word is copied, else the word is unchanged.

Operation

PBLENDW

```
IF (imm8[0] == 1) THEN DEST[15:0] ← SRC[15:0];
    ELSE DEST[15:0] ← DEST[15:0];
IF (imm8[1] == 1) THEN DEST[31:16] ← SRC[31:16];
    ELSE DEST[31:16] ← DEST[31:16]);
IF (imm8[2] == 1) THEN DEST[47:32] ← SRC[47:32];
    ELSE DEST[47:32] ← DEST[47:32];
IF (imm8[3] == 1) THEN DEST[63:48] ← SRC[63:48];
    ELSE DEST[63:48] ← DEST[63:48];
IF (imm8[4] == 1) THEN DEST[79:64] ← SRC[79:64];
    ELSE DEST[79:64] ← DEST[79:64];
IF (imm8[5] == 1) THEN DEST[95:80] ← SRC[95:80];
    ELSE DEST[95:80] ← DEST[95:80];
IF (imm8[6] == 1) THEN DEST[111:96] ← SRC[111:96];
    ELSE DEST[111:96] ← DEST[111:96];
IF (imm8[7] == 1) THEN DEST[127:112] ← SRC[127:112];
    ELSE DEST[127:112] ← DEST[127:112];
```

Intel C/C++ Compiler Intrinsic Equivalent

PBLENDW __m128i _mm_blend_epi16 (__m128i v1, __m128i v2, const int mask);

Flags Affected

None

Protected Mode and Compatiblity Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PCMPEQQ — Compare Packed Qword Data for Equal

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 29 /r	PCMPEQQ *xmm1, xmm2/m128*	Valid	Valid	Compare packed qwords in *xmm2/m128* and *xmm1* for equality.

Description

Performs an SIMD compare for equality of the packed quadwords in the destination operand (first operand) and the source operand (second operand). If a pair of data elements is equal, the corresponding data element in the destination is set to all 1s; otherwise, it is set to 0s.

Operation

PCMPEQQ

```
IF (DEST[63:0] == SRC[63:0]) THEN DEST[63:0] ← FFFFFFFFFFFFFFFFH;
    ELSE DEST[63:0] ← 0;
IF (DEST[127:64] == SRC[127:64]) THEN DEST[127:64] ← FFFFFFFFFFFFFFFFH;
    ELSE DEST[127:64] ← 0;
```

Intel C/C++ Compiler Intrinsic Equivalent

PCMPEQQ __m128i _mm_cmpeq_epi64(__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PCMPESTRI — Packed Compare Explicit Length Strings, Return Index

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 3A 61 */r imm8*	PCMPESTRI *xmm1, xmm2/m128, imm8*	Valid	Valid	Perform a packed comparison of string data with explicit lengths, generating an index, and storing the result in ECX.

Description

The instruction compares and processes data from two string fragments based on the encoded value in the Imm8 Control Byte (see Section 5.3), and generates an index stored to ECX.

Each string fragment is represented by two values. The first value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). The second value is stored in EAX (for xmm1) or EDX (for xmm2/m128) and represents the number of bytes/words which are valid for the respective xmm/m128 data.

The length of each input is interpreted as being the absolute-value of the value in EAX (EDX). The absolute-value computation saturates to 16 (for bytes) and 8 (for words), based on the value of imm8[bit3] when the value in EAX (EDX) is greater than 16 (8) or less than -16 (-8).

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 5.3, “Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM”). The index of the first (or last, according to imm8[6]) set bit of IntRes2 (see Section 5.3.1.4) is returned in ECX. If no bits are set in IntRes2, ECX is set to 16 (8).

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag – Reset if IntRes2 is equal to zero, set otherwise
ZFlag – Set if absolute-value of EDX is < 16 (8), reset otherwise
SFlag – Set if absolute-value of EAX is < 16 (8), reset otherwise
OFlag – IntRes2[0]
AFlag – Reset
PFlag – Reset

Effective Operand Size

Operating mode/size	Operand 1	Operand 2	Length 1	Length 2	Result
16 bit	xmm	xmm/m128	EAX	EDX	ECX
32 bit	xmm	xmm/m128	EAX	EDX	ECX
64 bit	xmm	xmm/m128	EAX	EDX	ECX
64 bit + REX.W	xmm	xmm/m128	RAX	RDX	RCX

Intel C/C++ Compiler Intrinsic Equivalent For Returning Index

int _mm_cmpestri (__m128i a, int la, __m128i b, int lb, const int mode);

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

int _mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrz (__m128i a, int la, __m128i b, int lb, const int mode);

SIMD Floating-Point Exceptions

N/A.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#SS(0)	For an illegal address in the SS segment
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.

Real-Address Mode Exceptions

#GP(0)	Interrupt 13 If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If TS in CR0 is set.

#UD	If EM in CR0 is set. If OSFXSR in CR4 is 0. If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0. If LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set. If OSFXSR in CR4 is 0. If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0. If LOCK prefix is used.

PCMPESTRM — Packed Compare Explicit Length Strings, Return Mask

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 3A 60 /r imm8	PCMPESTRM *xmm1, xmm2/m128, imm8*	Valid	Valid	Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in *XMM0*

Description

The instruction compares data from two string fragments based on the encoded value in the imm8 contol byte (see Section 5.3), and generates a mask stored to XMM0.

Each string fragment is represented by two values. The first value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). The second value is stored in EAX (for xmm1) or EDX (for xmm2/m128) and represents the number of bytes/words which are valid for the respective xmm/m128 data.

The length of each input is interpreted as being the absolute-value of the value in EAX (EDX). The absolute-value computation saturates to 16 (for bytes) and 8 (for words), based on the value of imm8[bit3] when the value in EAX (EDX) is greater than 16 (8) or less than -16 (-8).

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 5.3, “Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM”). As defined by imm8[6], IntRes2 is then either stored to the least significant bits of XMM0 (zero extended to 128 bits) or expanded into a byte/word-mask and then stored to XMM0.

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag – Reset if IntRes2 is equal to zero, set otherwise
ZFlag – Set if absolute-value of EDX is < 16 (8), reset otherwise
SFlag – Set if absolute-value of EAX is < 16 (8), reset otherwise
OFlag –IntRes2[0]
AFlag – Reset
PFlag – Reset

Effective Operand Size

Operating mode/size	Operand1	Operand2	Length1	Length2	Result
16 bit	xmm	xmm/m128	EAX	EDX	XMM0
32 bit	xmm	xmm/m128	EAX	EDX	XMM0
64 bit	xmm	xmm/m128	EAX	EDX	XMM0
64 bit + REX.W	xmm	xmm/m128	RAX	RDX	XMM0

Intel C/C++ Compiler Intrinsic Equivalent For Returning Mask

__m128i _mm_cmpestrm (__m128i a, int la, __m128i b, int lb, const int mode);

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

int _mm_cmpestra (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrc (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestro (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrs (__m128i a, int la, __m128i b, int lb, const int mode);
int _mm_cmpestrz (__m128i a, int la, __m128i b, int lb, const int mode);

SIMD Floating-Point Exceptions

N/A.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#SS(0)	For an illegal address in the SS segment
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.

Real-Address Mode Exceptions

#GP(0)	Interrupt 13 If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.

	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

PCMPISTRI — Packed Compare Implicit Length Strings, Return Index

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 3A 63 */r imm8*	PCMPISTRI *xmm1, xmm2/m128, imm8*	Valid	Valid	Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.

Description

The instruction compares data from two strings based on the encoded value in the Imm8 Control Byte (see Section 5.3), and generates an index stored to ECX.

Each string is represented by a single value. The value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). Each input byte/word is augmented with a valid/invalid tag. A byte/word is considered valid only if it has a lower index than the least significant null byte/word. (The least significant null byte/word is also considered invalid.)

The comparison and aggregation operations are performed according to the encoded value of Imm8 bit fields (see Section 5.3, "Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM"). The index of the first (or last, according to imm8[6]) set bit of IntRes2 is returned in ECX. If no bits are set in IntRes2, ECX is set to 16 (8).

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag – Reset if IntRes2 is equal to zero, set otherwise
ZFlag – Set if any byte/word of xmm2/mem128 is null, reset otherwise
SFlag – Set if any byte/word of xmm1 is null, reset otherwise
OFlag –IntRes2[0]
AFlag – Reset
PFlag – Reset

Effective Operand Size

Operating mode/size	Operand1	Operand2	Result
16 bit	xmm	xmm/m128	ECX
32 bit	xmm	xmm/m128	ECX
64 bit	xmm	xmm/m128	ECX
64 bit + REX.W	xmm	xmm/m128	RCX

Intel C/C++ Compiler Intrinsic Equivalent For Returning Index

int _mm_cmpistri (__m128i a, __m128i b, const int mode);

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

int _mm_cmpistra (__m128i a, __m128i b, const int mode);
int _mm_cmpistrc (__m128i a, __m128i b, const int mode);
int _mm_cmpistro (__m128i a, __m128i b, const int mode);
int _mm_cmpistrs (__m128i a, __m128i b, const int mode);
int _mm_cmpistrz (__m128i a, __m128i b, const int mode);

SIMD Floating-Point Exceptions

N/A.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#SS(0)	For an illegal address in the SS segment.
#UD	If EM in CR0 is set. If OSFXSR in CR4 is 0. If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0. If LOCK prefix is used.

Real-Address Mode Exceptions

#GP(0)	Interrupt 13 If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

PCMPISTRM — Packed Compare Implicit Length Strings, Return Mask

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 3A 62 */r imm8*	PCMPISTRM *xmm1, xmm2/m128, imm8*	Valid	Valid	Perform a packed comparison of string data with implicit lengths, generating a mask, and storing the result in *XMM0.*

Description

The instruction compares data from two strings based on the encoded value in the imm8 byte (see Section 5.3) generating a mask stored to XMM0.

Each string is represented by a single value. The The value is an xmm (or possibly m128 for the second operand) which contains the data elements of the string (byte or word data). Each input byte/word is augmented with a valid/invalid tag. A byte/word is considered valid only if it has a lower index than the least significant null byte/word. (The least significant null byte/word is also considered invalid.)

The comparison and aggregation operation are performed according to the encoded value of Imm8 bit fields (see Section 5.3, "Imm8 Control Byte Operation for PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM"). As defined by imm8[6], IntRes2 is then either stored to the least significant bits of XMM0 (zero extended to 128 bits) or expanded into a byte/word-mask and then stored to XMM0.

Note that the Arithmetic Flags are written in a non-standard manner in order to supply the most relevant information:

CFlag – Reset if IntRes2 is equal to zero, set otherwise
ZFlag – Set if any byte/word of xmm2/mem128 is null, reset otherwise
SFlag – Set if any byte/word of xmm1 is null, reset otherwise
OFlag – IntRes2[0]
AFlag – Reset
PFlag – Reset

Effective Operand Size

Operating mode/size	Operand1	Operand2	Result
16 bit	xmm	xmm/m128	XMM0
32 bit	xmm	xmm/m128	XMM0
64 bit	xmm	xmm/m128	XMM0
64 bit + REX.W	xmm	xmm/m128	XMM0

Intel C/C++ Compiler Intrinsic Equivalent For Returning Mask

__m128i _mm_cmpistrm (__m128i a, __m128i b, const int mode);

Intel C/C++ Compiler Intrinsics For Reading EFlag Results

int _mm_cmpistra (__m128i a, __m128i b, const int mode);
int _mm_cmpistrc (__m128i a, __m128i b, const int mode);
int _mm_cmpistro (__m128i a, __m128i b, const int mode);
int _mm_cmpistrs (__m128i a, __m128i b, const int mode);
int _mm_cmpistrz (__m128i a, __m128i b, const int mode);

SIMD Floating-Point Exceptions

N/A.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#SS(0)	For an illegal address in the SS segment
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.

Real-Address Mode Exceptions

#GP(0)	Interrupt 13 If any part of the operand lies outside the effective address space from 0 to FFFFH.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] is 0.
	If LOCK prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

PCMPGTQ — Compare Packed Data for Greater Than

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
66 0F 38 37 */r*	PCMPGTQ *xmm1,xmm2/m128*	Valid	Valid	Compare packed qwords in *xmm2/m128* and *xmm1* for greater than.

Description

Performs an SIMD compare for the packed quadwords in the destination operand (first operand) and the source operand (second operand). If the data element in the first (destination) operand is greater than the corresponding element in the second (source) operand, the corresponding data element in the destination is set to all 1s; otherwise, it is set to 0s.

Operation

```
IF (DEST[63-0] > SRC[63-0])
    THEN DEST[63-0] ← FFFFFFFFFFFFFFFFH;
    ELSE DEST[63-0] ← 0;
FI
IF (DEST[127-64] > SRC[127-64])
    THEN DEST[127-64] ← FFFFFFFFFFFFFFFFH;
    ELSE DEST[127-64] ← 0;
FI
```

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

PCMPGTQ __m128i _mm_cmpgt_epi64(__m128i a, __m128i b)

Protected Mode and Compatibility Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
	If not aligned on 16-byte boundary, regardless of segment.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.

	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.
#NM	If TS bit in CR0 is set.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If not aligned on 16-byte boundary, regardless of segment.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.
#NM	If TS bit in CR0 is set.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If not aligned on 16-byte boundary, regardless of segment.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.
#NM	If TS bit in CR0 is set.
#PF (fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If not aligned on 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF (fault-code)	For a page fault.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.01H:ECX.SSE4_2 [Bit 20] = 0.
	If LOCK prefix is used.

#NM	If TS bit in CR0 is set.

PEXTRB — Extract Byte

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 14 /r ib	PEXTRB *r32/m8, xmm2, imm8*	Valid	Valid	Extract a byte integer value from *xmm2* at the source byte offset specified by *imm8* into *r32/m8*.
66 REX.W 0F 3A 14 /r ib	PEXTRB *r64/m8, xmm2, imm8*	Valid	N. E.	Extract a byte integer value from *xmm2* at the source byte offset specified by *imm8* into *r64/m8*.

Description

Extract a byte integer value from the source xmm register (second argument) at a byte offset determined from imm8[3:0]. The extracted integer value is stored into the low 8 bits of the destination. If the destination is a register, the upper bits of the register are zero extended.

Operation

PEXTRB (dest=m8)

```
SRC_Offset ← Imm8[3:0];
Mem8 ← (Src >> Src_Offset*8);
```

PEXTRB (dest=r32)

```
IF (64-Bit Mode and REX.W used and 64-bit destination GPR)
THEN
    SRC_Offset ← Imm8[3:0];
    r64 ← Zero_Extend64((Src >> Src_Offset*8) AND 0FFh);
ELSE
    SRC_Offset ← Imm8[3:0];
    r32 ← Zero_Extend32((Src >> Src_Offset*8) AND 0FFh);
```

Intel C/C++ Compiler Intrinsic Equivalent

PEXTRB int _mm_extract_epi8 (__m128i src, const int ndx);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.

#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PEXTRD/PEXTRQ — Extract Dword/Qword

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 16 /r ib	PEXTRD *r/m32, xmm2, imm8*	Valid	Valid	Extract a dword integer value from *xmm2* at the source dword offset specified by *imm8* into *r/m32*.
66 REX.W 0F 3A 16 /r ib	PEXTRQ *r/m64, xmm2, imm8*	Valid	N. E.	Extract a qword integer value from *xmm2* at the source dword offset specified by *imm8* into r/m64.

Description

PEXTRD extracts a dword integer value from the source xmm register (second argument) at a dword offset determined from imm8[1:0]. The extracted integer value is stored into the low 32 bits of the destination. If the destination is a register, the upper bits of the register are zero extended.

Operation

PEXTRTD/PEXTRQ

```
IF (64-Bit Mode and REX.W used and 64-bit destination operand)
THEN
    Src_Offset ← Imm8[0];
    r/m64= (Src >> Src_Offset * 64);
ELSE
    Src_Offset ← Imm8[1:0];
    r/m32 ← ((Src >> Src_Offset *32) AND 0FFFFFFFFh);
FI;
```

Intel C/C++ Compiler Intrinsic Equivalent

PEXTRD int _mm_extract_epi32 (__m128i src, const int ndx);
PEXTRQ __int64 _mm_extract_epi64 (__m128i src, const int ndx);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.

#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#NM	If CR0.TS[bit 3] = 1.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

PEXTRW — Extract Word

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 15 /r ib	PEXTRW *r32/m16, xmm2, imm8*	Valid	Valid	Extract a word integer value from *xmm2* at the source word offset specified by *imm8* into *r32/m16*.
66 REX.W 0F 3A 15 /r ib	PEXTRW *r64/m16, xmm2, imm8*	Valid	N.E.+	Extract a word integer value from *xmm2* at the source word offset specified by *imm8* into *r64/m16*.

Description

Extract a word integer value from the source xmm register (second argument) at a word offset determined from imm8[2:0]. The extracted integer value is stored into the low 16 bits of the destination. If the destination is a register, the upper bits of the register are zero extended.

Operation

PEXTRW (dest=m16)

```
SRC_Offset ← Imm8[2:0];
Mem16 ← (Src >> Src_Offset*16);
```

PEXTRW (dest=r32 or r64)

```
IF (64-Bit Mode and REX.W used and 64-bit destination GPR)
THEN
    SRC_Offset ← Imm8[2:0];
    r64 ← Zero_Extend64((Src >> Src_Offset*16) AND 0FFFFh);
ELSE
    SRC_Offset ← Imm8[2:0];
    r32 ← Zero_Extend32((Src >> Src_Offset*16) AND 0FFFFh);
```

Intel C/C++ Compiler Intrinsic Equivalent

PEXTRW int _mm_extract_epi16 (__m128i src, int ndx);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.

#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

PHMINPOSUW — Packed Horizontal Word Minimum

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 41 /r	PHMINPOSUW *xmm1, xmm2/m128*	Valid	Valid	Find the minimum unsigned word in *xmm2/m128* and place its value in the low word of *xmm1* and its index in the second-lowest word of *xmm1*.

Description

Determine the minimum unsigned word value in the source operand (second operand) and place the unsigned word in the low word (bits 0-15) of the destination operand (first operand). The word index of the minimum value is stored in bits 16-18 of the destination operand. The remaining upper bits of the destination are set to zero.

Operation

PHMINPOSUW

```
INDEX ← 0;
MIN ← SRC[15:0]
IF (SRC[31:16] < MIN) THEN INDEX ← 1;  MIN ← SRC[31:16];
IF (SRC[47:32] < MIN) THEN INDEX ← 2;  MIN ← SRC[47:32];
* Repeat operation for words 3 through 6
IF (SRC[127:112] < MIN) THEN INDEX ← 7;  MIN ← SRC[127:112];
DEST[15:0] ← MIN;
DEST[18:16] ← INDEX;
DEST[127:19] ← 00000000000000000000000000000H;
```

Intel C/C++ Compiler Intrinsic Equivalent

PHMINPOSUW __m128i _mm_minpos_epu16(__m128i packed_words);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.

	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.

If LOCK prefix is used.

PINSRB — Insert Byte

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 20 /r ib	PINSRB *xmm1, r32/m8, imm8*	Valid	Valid	Insert a byte integer value from *r32/m8* into *xmm1* at the destination element in *xmm1* specified by *imm8*.

Description

Copies a byte from the source operand (second operand) and inserts it into the destination operand (first operand) at the location specified with the immediate operand (third operand). The other words in the destination register are left unchanged. The byte select is specified by the 4 least-significant bits of the immediate.

Operation

PINSRB xmm1, m8, imm8

```
SEL ← imm8[3:0];
MASK ← (0FFH << (SEL * 8)); // Shift in zeros from right
DEST ← (DEST AND NOT MASK) OR (((SRC << (SEL *8)) AND MASK);
```

Intel C/C++ Compiler Intrinsic Equivalent

PINSRB __m128i _mm_insert_epi8 (__m128i s1, int s2, const int ndx);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PINSRD/PINSRQ — Insert Dword/Qword

Opcode	Instruction	Compat/ Leg Mode	64-bit Mode	Description
66 0F 3A 22 /r ib	PINSRD *xmm1, r/m32, imm8*	Valid	Valid	Insert a dword integer value from *r/m32* into the *xmm1* at the destination elements specified by *imm8.*
66 REX.W 0F 3A 22 /r ib	PINSRQ *xmm1, r/m64, imm8*	N. E.	Valid	Insert a qword integer value from *r/m32* into the *xmm1* at the destination elements specified by *imm8.*

Description

Copies a dword from the source operand (second operand) and inserts it into the destination operand (first operand) at the location specified by the immediate operand (third operand). The other dwords in the destination register are left unchanged. The dword select is specified by the 2 least-significant bits of the immediate.

Operation

PINSRD xmm1, m32, imm8

```
IF (64-Bit Mode and REX.W used )
THEN
    SEL ← imm8[0]
        MASK ← (0FFFFFFFFFFFFFFFFH << (SEL * 64));  // Shift in zeros from right
    DEST ← (DEST AND NOT MASK) OR (((SRC << (SEL *64)) AND MASK);
ELSE
    SEL ← imm8[1:0]
        MASK ← (0FFFFFFFFH << (SEL * 32));    // Shift in zeros from right
    DEST ← (DEST AND NOT MASK) OR (((SRC << (SEL *32)) AND MASK);
FI
```

Intel C/C++ Compiler Intrinsic Equivalent

PINSRD __m128i _mm_insert_epi32 (__m128i s2, int s, const int ndx);
PINSRQ __m128i _mm_insert_epi64(__m128i s2, __int64 s, const int ndx);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

PMAXSB — Maximum of Packed Signed Byte Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 3C /r	PMAXSB *xmm1, xmm2/m128*	Valid	Valid	Compare packed signed byte integers in *xmm1* and *xmm2/m128* and store packed maximum values in *xmm1*.

Description

Compares packed signed byte integers in the destination operand (first operand) and the source operand (second operand), and returns the maximum for each packed value in the destination operand.

Operation

PMAXSB

```
IF (DEST[7:0] > SRC[7:0]) THEN DEST[7:0] ← DEST[7:0];
    ELSE DEST[7:0] ← SRC[7:0];
IF (DEST[15:8] > SRC[15:8]) THEN DEST[15:8] ← DEST[15:8];
    ELSE DEST[15:8] ← SRC[15:8];
IF (DEST[23:16] > SRC[23:16]) THEN DEST[23:16] ← DEST[23:16];
    ELSE DEST[23:16] ← SRC[23:16];
IF (DEST[31:24] > SRC[31:24]) THEN DEST[31:24] ← DEST[31:24];
    ELSE DEST[31:24] ← SRC[31:24];
IF (DEST[39:32] > SRC[39:32]) THEN DEST[39:32] ← DEST[39:32];
    ELSE DEST[39:32] ← SRC[39:32];
IF (DEST[47:40] > SRC[47:40]) THEN DEST[47:40] ← DEST[47:40];
    ELSE DEST[47:40] ← SRC[47:40];
IF (DEST[55:48] > SRC[55:48]) THEN DEST[55:48] ← DEST[55:48];
    ELSE DEST[55:48] ← SRC[55:48];
IF (DEST[63:56] > SRC[63:56]) THEN DEST[63:56] ← DEST[63:56];
    ELSE DEST[63:56] ← SRC[63:56];
IF (DEST[71:64] > SRC[71:64]) THEN DEST[71:64] ← DEST[71:64];
    ELSE DEST[71:64] ← SRC[71:64];
IF (DEST[79:72] > SRC[79:72]) THEN DEST[79:72] ← DEST[79:72];
    ELSE DEST[79:72] ← SRC[79:72];
IF (DEST[87:80] > SRC[87:80]) THEN DEST[87:80] ← DEST[87:80];
    ELSE DEST[87:80] ← SRC[87:80];
IF (DEST[95:88] > SRC[95:88]) THEN DEST[95:88] ← DEST[95:88];
    ELSE DEST[95:88] ← SRC[95:88];
IF (DEST[103:96] > SRC[103:96]) THEN DEST[103:96] ← DEST[103:96];
    ELSE DEST[103:96] ← SRC[103:96];
```

```
IF (DEST[111:104] > SRC[111:104]) THEN DEST[111:104] ← DEST[111:104];
    ELSE DEST[111:104] ← SRC[111:104];
IF (DEST[119:112] > SRC[119:112]) THEN DEST[119:112] ← DEST[119:112];
    ELSE DEST[119:112] ← SRC[119:112];
IF (DEST[127:120] > SRC[127:120]) THEN DEST[127:120] ← DEST[127:120];
    ELSE DEST[127:120] ← SRC[127:120];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMAXSB __m128i _mm_max_epi8 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMAXSD — Maximum of Packed Signed Dword Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 3D /r	PMAXSD *xmm1*, *xmm2/m128*	Valid	Valid	Compare packed signed dword integers in *xmm1* and *xmm2/m128* and store packed maximum values in *xmm1*.

Description

Compares packed signed dword integers in the destination operand (first operand) and the source operand (second operand), and returns the maximum for each packed value in the destination operand.

Operation

PMAXSD

```
IF (DEST[31:0] > SRC[31:0]) THEN DEST[31:0] ← DEST[31:0];
    ELSE DEST[31:0] ← SRC[31:0];
IF (DEST[63:32] > SRC[63:32]) THEN DEST[63:32] ← DEST[63:32];
    ELSE DEST[63:32] ← SRC[63:32];
IF (DEST[95:64] > SRC[95:64]) THEN DEST[95:64] ← DEST[95:64];
    ELSE DEST[95:64] ← SRC[95:64];
IF (DEST[127:96] > SRC[127:96]) THEN DEST[127:96] ← DEST[127:96];
    ELSE DEST[127:96] ← SRC[127:96];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMAXSD __m128i _mm_max_epi32 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMAXUD — Maximum of Packed Unsigned Dword Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 3F /r	PMAXUD *xmm1*, *xmm2/m128*	Valid	Valid	Compare packed unsigned dword integers in *xmm1* and *xmm2/m128* and store packed maximum values in *xmm1*.

Description

Compares packed unsigned dword integers in the destination operand (first operand) and the source operand (second operand), and returns the maximum for each packed value in the destination operand.

Operation

PMAXUD

```
IF (DEST[31:0] > SRC[31:0]) THEN DEST[31:0] ← DEST[31:0];
    ELSE DEST[31:0] ← SRC[31:0];
IF (DEST[63:32] > SRC[63:32]) THEN DEST[63:32] ← DEST[63:32];
    ELSE DEST[63:32] ← SRC[63:32];
IF (DEST[95:64] > SRC[95:64]) THEN DEST[95:64] ← DEST[95:64];
    ELSE DEST[95:64] ← SRC[95:64];
IF (DEST[127:96] > SRC[127:96]) THEN DEST[127:96] ← DEST[127:96];
    ELSE DEST[127:96] ← SRC[127:96];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMAXUD __m128i _mm_max_epu32 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMAXUW — Maximum of Packed Word Integers

Opcode	Instruction	Compat/ Leg Mode	64-bit Mode	Description
66 0F 38 3E /r	PMAXUW *xmm1, xmm2/m128*	Valid	Valid	Compare packed unsigned word integers in *xmm1* and *xmm2/m128* and store packed maximum values in *xmm1*.

Description

Compares packed unsigned word integers in the destination operand (first operand) and the source operand (second operand), and returns the maximum for each packed value in the destination operand.

Operation

PMAXUW

```
IF (DEST[15:0] > SRC[15:0]) THEN DEST[15:0] ← DEST[15:0];
    ELSE DEST[15:0] ← SRC[15:0];
IF (DEST[31:16] > SRC[31:16]) THEN DEST[31:16] ← DEST[31:16];
    ELSE DEST[31:16] ← SRC[31:16];
IF (DEST[47:32] > SRC[47:32]) THEN DEST[47:32] ← DEST[47:32];
    ELSE DEST[47:32] ← SRC[47:32];
IF (DEST[63:48] > SRC[63:48]) THEN DEST[63:48] ← DEST[63:48];
    ELSE DEST[63:48] ← SRC[63:48];
IF (DEST[79:64] > SRC[79:64]) THEN DEST[79:64] ← DEST[79:64];
    ELSE DEST[79:64] ← SRC[79:64];
IF (DEST[95:80] > SRC[95:80]) THEN DEST[95:80] ← DEST[95:80];
    ELSE DEST[95:80] ← SRC[95:80];
IF (DEST[111:96] > SRC[111:96]) THEN DEST[111:96] ← DEST[111:96];
    ELSE DEST[111:96] ← SRC[111:96];
IF (DEST[127:112] > SRC[127:112]) THEN DEST[127:112] ← DEST[127:112];
    ELSE DEST[127:112] ← SRC[127:112];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMAXUW__m128i _mm_max_epu16 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMINSB — Minimum of Packed Signed Byte Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 38 /r	PMINSB *xmm1, xmm2/m128*	Valid	Valid	Compare packed signed byte integers in *xmm1* and *xmm2/m128* and store packed minimum values in *xmm1*.

Description

Compares packed signed byte integers in the destination operand (first operand) and the source operand (second operand), and returns the minimum for each packed value in the destination operand.

Operation

PMINSB

```
IF (DEST[7:0] < SRC[7:0]) THEN DEST[7:0] ← DEST[7:0];
    ELSE DEST[7:0] ← SRC[7:0];
IF (DEST[15:8] < SRC[15:8]) THEN DEST[15:8] ← DEST[15:8];
    ELSE DEST[15:8] ← SRC[15:8];
IF (DEST[23:16] < SRC[23:16]) THEN DEST[23:16] ← DEST[23:16];
    ELSE DEST[23:16] ← SRC[23:16];
IF (DEST[31:24] < SRC[31:24]) THEN DEST[31:24] ← DEST[31:24];
    ELSE DEST[31:24] ← SRC[31:24];
IF (DEST[39:32] < SRC[39:32]) THEN DEST[39:32] ← DEST[39:32];
    ELSE DEST[39:32] ← SRC[39:32];
IF (DEST[47:40] < SRC[47:40]) THEN DEST[47:40] ← DEST[47:40];
    ELSE DEST[47:40] ← SRC[47:40];
IF (DEST[55:48] < SRC[55:48]) THEN DEST[55:48] ← DEST[55:48];
    ELSE DEST[55:48] ← SRC[55:48];
IF (DEST[63:56] < SRC[63:56]) THEN DEST[63:56] ← DEST[63:56];
    ELSE DEST[63:56] ← SRC[63:56];
IF (DEST[71:64] < SRC[71:64]) THEN DEST[71:64] ← DEST[71:64];
    ELSE DEST[71:64] ← SRC[71:64];
IF (DEST[79:72] < SRC[79:72]) THEN DEST[79:72] ← DEST[79:72];
    ELSE DEST[79:72] ← SRC[79:72];
IF (DEST[87:80] < SRC[87:80]) THEN DEST[87:80] ← DEST[87:80];
    ELSE DEST[87:80] ← SRC[87:80];
IF (DEST[95:88] < SRC[95:88]) THEN DEST[95:88] ← DEST[95:88];
    ELSE DEST[95:88] ← SRC[95:88];
IF (DEST[103:96] < SRC[103:96]) THEN DEST[103:96] ← DEST[103:96];
```

```
    ELSE DEST[103:96] ← SRC[103:96];
IF (DEST[111:104] < SRC[111:104]) THEN DEST[111:104] ← DEST[111:104];
    ELSE DEST[111:104] ← SRC[111:104];
IF (DEST[119:112] < SRC[119:112]) THEN DEST[119:112] ← DEST[119:112];
    ELSE DEST[119:112] ← SRC[119:112];
IF (DEST[127:120] < SRC[127:120]) THEN DEST[127:120] ← DEST[127:120];
    ELSE DEST[127:120] ← SRC[127:120];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMINSB __m128i _mm_min_epi8 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMINSD — Minimum of Packed Dword Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 39 /r	PMINSD *xmm1, xmm2/m128*	Valid	Valid	Compare packed signed dword integers in *xmm1* and *xmm2/m128* and store packed mimum values in *xmm1*.

Description

Compares packed signed dword integers in the destination operand (first operand) and the source operand (second operand), and returns the minimum for each packed value in the destination operand.

Operation

PMINSD

```
IF (DEST[31:0] < SRC[31:0]) THEN DEST[31:0] ← DEST[31:0];
    ELSE DEST[31:0] ← SRC[31:0];
IF (DEST[63:32] < SRC[63:32]) THEN DEST[63:32] ← DEST[63:32];
    ELSE DEST[63:32] ← SRC[63:32];
IF (DEST[95:64] < SRC[95:64]) THEN DEST[95:64] ← DEST[95:64];
    ELSE DEST[95:64] ← SRC[95:64];
IF (DEST[127:96] < SRC[127:96]) THEN DEST[127:96] ← DEST[127:96];
    ELSE DEST[127:96] ← SRC[127:96];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMINSD __m128i _mm_min_epi32 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMINUD — Minimum of Packed Dword Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 3B /r	PMINUD *xmm1, xmm2/m128*	Valid	Valid	Compare packed unsigned dword integers in *xmm1* and *xmm2/m128* and store packed minimum values in *xmm1*.

Description

Compares packed unsigned dword integers in the destination operand (first operand) and the source operand (second operand), and returns the minimum for each packed value in the destination operand.

Operation

PMINUD

```
IF (DEST[31:0] < SRC[31:0]) THEN DEST[31:0] ← DEST[31:0];
    ELSE DEST[31:0] ← SRC[31:0];
IF (DEST[63:32] < SRC[63:32]) THEN DEST[63:32] ← DEST[63:32];
    ELSE DEST[63:32] ← SRC[63:32];
IF (DEST[95:64] < SRC[95:64]) THEN DEST[95:64] ← DEST[95:64];
    ELSE DEST[95:64] ← SRC[95:64];
IF (DEST[127:96] < SRC[127:96]) THEN DEST[127:96] ← DEST[127:96];
    ELSE DEST[127:96] ← SRC[127:96];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMINUD __m128i _mm_min_epu32 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMINUW — Minimum of Packed Word Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 3A /r	PMINUW *xmm1, xmm2/m128*	Valid	Valid	Compare packed unsigned word integers in *xmm1* and *xmm2/m128* and store packed minimum values in *xmm1*.

Description

Compares packed unsigned word integers in the destination operand (first operand) and the source operand (second operand), and returns the minimum for each packed value in the destination operand.

Operation

PMINUW

```
IF (DEST[15:0] < SRC[15:0]) THEN DEST[15:0] ← DEST[15:0];
    ELSE DEST[15:0] ← SRC[15:0];
IF (DEST[31:16] < SRC[31:16]) THEN DEST[31:16] ← DEST[31:16];
    ELSE DEST[31:16] ← SRC[31:16];
IF (DEST[47:32] < SRC[47:32]) THEN DEST[47:32] ← DEST[47:32];
    ELSE DEST[47:32] ← SRC[47:32];
IF (DEST[63:48] < SRC[63:48]) THEN DEST[63:48] ← DEST[63:48];
    ELSE DEST[63:48] ← SRC[63:48];
IF (DEST[79:64] < SRC[79:64]) THEN DEST[79:64] ← DEST[79:64];
    ELSE DEST[79:64] ← SRC[79:64];
IF (DEST[95:80] < SRC[95:80]) THEN DEST[95:80] ← DEST[95:80];
    ELSE DEST[95:80] ← SRC[95:80];
IF (DEST[111:96] < SRC[111:96]) THEN DEST[111:96] ← DEST[111:96];
    ELSE DEST[111:96] ← SRC[111:96];
IF (DEST[127:112] < SRC[127:112]) THEN DEST[127:112] ← DEST[127:112];
    ELSE DEST[127:112] ← SRC[127:112];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMINUW __m128i _mm_min_epu16 (__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMOVSX — Packed Move with Sign Extend

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0f 38 20 /r	PMOVSXBW *xmm1, xmm2/m64*	Valid	Valid	Sign extend 8 packed signed 8-bit integers in the low 8 bytes of *xmm2/m64* to 8 packed signed 16-bit integers in *xmm1*.
66 0f 38 21 /r	PMOVSXBD *xmm1, xmm2/m32*	Valid	Valid	Sign extend 4 packed signed 8-bit integers in the low 4 bytes of *xmm2/m32* to 4 packed signed 32-bit integers in *xmm1*.
66 0f 38 22 /r	PMOVSXBQ *xmm1, xmm2/m16*	Valid	Valid	Sign extend 2 packed signed 8-bit integers in the low 2 bytes of *xmm2/m16* to 2 packed signed 64-bit integers in *xmm1*.
66 0f 38 23 /r	PMOVSXWD *xmm1, xmm2/m64*	Valid	Valid	Sign extend 4 packed signed 16-bit integers in the low 8 bytes of *xmm2/m64* to 4 packed signed 32-bit integers in *xmm1*.
66 0f 38 24 /r	PMOVSXWQ *xmm1, xmm2/m32*	Valid	Valid	Sign extend 2 packed signed 16-bit integers in the low 4 bytes of *xmm2/m32* to 2 packed signed 64-bit integers in *xmm1*.
66 0f 38 25 /r	PMOVSXDQ *xmm1, xmm2/m64*	Valid	Valid	Sign extend 2 packed signed 32-bit integers in the low 8 bytes of *xmm2/m64* to 2 packed signed 64-bit integers in *xmm1*.

Description

Packed byte, word, or dword integers in the low bytes of the source operand (second operand) are sign extended to word, dword, or quadword integers and stored as packed data in the destination operand.

Operation

PMOVSXBW

```
DEST[15:0] ← SignExtend(SRC[7:0]);
DEST[31:16] ← SignExtend(SRC[15:8]);
DEST[47:32] ← SignExtend(SRC[23:16]);
DEST[63:48] ← SignExtend(SRC[31:24]);
DEST[79:64] ← SignExtend(SRC[39:32]);
```

DEST[95:80] ← SignExtend(SRC[47:40]);
DEST[111:96] ← SignExtend(SRC[55:48]);
DEST[127:112] ← SignExtend(SRC[63:56]);

PMOVSXBD
DEST[31:0] ← SignExtend(SRC[7:0]);
DEST[63:32] ← SignExtend(SRC[15:8]);
DEST[95:64] ← SignExtend(SRC[23:16]);
DEST[127:96] ← SignExtend(SRC[31:24]);

PMOVSXBQ
DEST[63:0] ← SignExtend(SRC[7:0]);
DEST[127:64] ← SignExtend(SRC[15:8]);

PMOVSXWD
DEST[31:0] ← SignExtend(SRC[15:0]);
DEST[63:32] ← SignExtend(SRC[31:16]);
DEST[95:64] ← SignExtend(SRC[47:32]);
DEST[127:96] ← SignExtend(SRC[63:48]);

PMOVSXWQ
DEST[63:0] ← SignExtend(SRC[15:0]);
DEST[127:64] ← SignExtend(SRC[31:16]);

PMOVSXDQ
DEST[63:0] ← SignExtend(SRC[31:0]);
DEST[127:64] ← SignExtend(SRC[63:32]);

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

PMOVSXBW	__m128i _mm_ cvtepi8_epi16 (__m128i a);
PMOVSXBD	__m128i _mm_ cvtepi8_epi32 (__m128i a);
PMOVSXBQ	__m128i _mm_ cvtepi8_epi64 (__m128i a);
PMOVSXWD	__m128i _mm_ cvtepi16_epi32 (__m128i a);
PMOVSXWQ	__m128i _mm_ cvtepi16_epi64 (__m128i a);
PMOVSXDQ	__m128i _mm_ cvtepi32_epi64 (__m128i a);

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.

#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

PMOVZX — Packed Move with Zero Extend

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0f 38 30 /r	PMOVZXBW *xmm1, xmm2/m64*	Valid	Valid	Zero extend 8 packed 8-bit integers in the low 8 bytes of *xmm2/m64* to 8 packed 16-bit integers in *xmm1*.
66 0f 38 31 /r	PMOVZXBD *xmm1, xmm2/m32*	Valid	Valid	Zero extend 4 packed 8-bit integers in the low 4 bytes of *xmm2/m32* to 4 packed 32-bit integers in *xmm1*.
66 0f 38 32 /r	PMOVZXBQ *xmm1, xmm2/m16*	Valid	Valid	Zero extend 2 packed 8-bit integers in the low 2 bytes of *xmm2/m16* to 2 packed 64-bit integers in *xmm1*.
66 0f 38 33 /r	PMOVZXWD *xmm1, xmm2/m64*	Valid	Valid	Zero extend 4 packed 16-bit integers in the low 8 bytes of *xmm2/m64* to 4 packed 32-bit integers in *xmm1*.
66 0f 38 34 /r	PMOVZXWQ *xmm1, xmm2/m32*	Valid	Valid	Zero extend 2 packed 16-bit integers in the low 4 bytes of *xmm2/m32* to 2 packed 64-bit integers in *xmm1*.
66 0f 38 35 /r	PMOVZXDQ *xmm1, xmm2/m64*	Valid	Valid	Zero extend 2 packed 32-bit integers in the low 8 bytes of *xmm2/m64* to 2 packed 64-bit integers in *xmm1*.

Description

Packed byte, word, or dword integers in the low bytes of the source operand (second operand) are zero extended to word, dword, or quadword integers and stored as packed data in the destination operand.

Operation

PMOVZXBW

DEST[15:0] ← ZeroExtend(SRC[7:0]);
DEST[31:16] ← ZeroExtend(SRC[15:8]);
DEST[47:32] ← ZeroExtend(SRC[23:16]);
DEST[63:48] ← ZeroExtend(SRC[31:24]);
DEST[79:64] ← ZeroExtend(SRC[39:32]);
DEST[95:80] ← ZeroExtend(SRC[47:40]);
DEST[111:96] ← ZeroExtend(SRC[55:48]);
DEST[127:112] ← ZeroExtend(SRC[63:56]);

PMOVZXBD

DEST[31:0] ← ZeroExtend(SRC[7:0]);

DEST[63:32] ← ZeroExtend(SRC[15:8]);
DEST[95:64] ← ZeroExtend(SRC[23:16]);
DEST[127:96] ← ZeroExtend(SRC[31:24]);

PMOVZXQB
DEST[63:0] ← ZeroExtend(SRC[7:0]);
DEST[127:64] ← ZeroExtend(SRC[15:8]);

PMOVZXWD
DEST[31:0] ← ZeroExtend(SRC[15:0]);
DEST[63:32] ← ZeroExtend(SRC[31:16]);
DEST[95:64] ← ZeroExtend(SRC[47:32]);
DEST[127:96] ← ZeroExtend(SRC[63:48]);

PMOVZXWQ
DEST[63:0] ← ZeroExtend(SRC[15:0]);
DEST[127:64] ← ZeroExtend(SRC[31:16]);

PMOVZXDQ
DEST[63:0] ← ZeroExtend(SRC[31:0]);
DEST[127:64] ← ZeroExtend(SRC[63:32]);

Flags Affected

None

Intel C/C++ Compiler Intrinsic Equivalent

PMOVZXBW __m128i _mm_ cvtepu8_epi16 (__m128i a);
PMOVZXBD __m128i _mm_ cvtepu8_epi32 (__m128i a);
PMOVZXBQ __m128i _mm_ cvtepu8_epi64 (__m128i a);
PMOVZXWD __m128i _mm_ cvtepu16_epi32 (__m128i a);
PMOVZXWQ __m128i _mm_ cvtepu16_epi64 (__m128i a);
PMOVZXDQ __m128i _mm_ cvtepu32_epi64 (__m128i a);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

PMULDQ — Multiply Packed Signed Dword Integers

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 28 /r	PMULDQ *xmm1*, *xmm2/m128*	Valid	Valid	Multiply the packed signed dword integers in *xmm1* and *xmm2/m128* and store the quadword product in *xmm1*.

Description

Performs a signed multiply of the first (low) and third packed signed dword integers in the destination operand (first operand) and the first and third packed signed dword integers in the source operand (second operand), and stores the 64 bit product in the destination operand. If the source is a memory operand then all 128 bits will be fetched from memory but the second and fourth dwords will not be used in the computation.

Operation

PMULDQ

```
DEST[63:0]   = DEST[31:0] * SRC[31:0];
DEST[127:64] = DEST[95:64] * SRC[95:64];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMULDQ __m128i _mm_mul_epi32(__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.

	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

PMULLD — Multiply Packed Signed Dword Integers and Store Low Result

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 40 /r	PMULLD *xmm1, xmm2/m128*	Valid	Valid	Multiply the packed dword signed integers in *xmm1* and *xmm2/m128* and store the low 32 bits of each product in *xmm1*.

Description

Performs a multiply of the packed signed dword integers in the destination operand (first operand) and the source operand (second operand), and stores the low 32 bits of each intermediate 64-bit product in the destination operand.

Operation

PMULLD

```
Temp0[63:0] ← DEST[31:0] * SRC[31:0];
Temp1[63:0] ← DEST[63:32] * SRC[63:32];
Temp2[63:0] ← DEST[95:64] * SRC[95:64];
Temp3[63:0] ← DEST[127:96] * SRC[127:96];
DEST[31:0] ← Temp0[31:0];
DEST[63:32] ← Temp1[31:0];
DEST[95:64] ← Temp2[31:0];
DEST[127:96] ← Temp3[31:0];
```

Intel C/C++ Compiler Intrinsic Equivalent

PMULLUD __m128i _mm_mullo_epi32(__m128i a, __m128i b);

Flags Affected

None

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments. If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.

#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

POPCNT — Return the Count of Number of Bits Set to 1

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
F3 0F B8 /r	POPCNT *r16, r/m16*	Valid	Valid	POPCNT on *r/m16*
F3 0F B8 /r	POPCNT *r32, r/m32*	Valid	Valid	POPCNT on *r/m32*
F3 REX.W 0F B8 /r	POPCNT *r64, r/m64*	Valid	N.E.	POPCNT on *r/m64*

Description

This instruction calculates of number of bits set to 1 in the second operand (source) and returns the count in the first operand (a destination register).

Operation

```
Count = 0;

//16-bit case
For (i=0; i < (16);i++) {
IF src16[i] == 1
    Then Count++
}
R16 ← Count;

//32-bit case
For (i=0; i < (32);i++) {
IF src32[i] == 1
    THEN Count++
}
R32 ← Count;

//64-bit case
For (i=0; i < (64);i++) {
IF src64[i] == 1
    THEN Count++
}
R64 ← Count;
```

Flags Affected

OF, SF, ZF, AF, CF, PF are all cleared. ZF is set if SRC == 0, otherwise ZF is cleared

Intel C/C++ Compiler Intrinsic Equivalent

POPCNT int _mm_popcnt_u32(unsigned int a);

POPCNT int64_t _mm_popcnt_u64(unsigned __int64 a);

Protected Mode and Compatibility Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segments.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

#GP(0)	If any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF (fault-code)	For a page fault.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0.
	If LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.

#PF (fault-code)	For a page fault.
#UD	If CPUID.01H:ECX.POPCNT [Bit 23] = 0. If LOCK prefix is used.

PTEST- Logical Compare

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 38 17 /r	PTEST *xmm1, xmm2/m128*	Valid	Valid	Set ZF if *xmm2/m128* AND *xmm1* result is all 0s. Set CF if *xmm2/m128* AND NOT *xmm1* result is all 0s.

Description

PTEST sets the ZF flag only if all bits in the result are 0 of the bitwise AND of the destination operand (first operand) and the source operand (second operand). PTEST sets the CF flag if all bits in the result are 0 of the bitwise AND of the source operand (second operand) and the logical NOT of the destination operand.

Operation

PTEST

```
IF (SRC[127:0] AND DEST[127:0] == 0) THEN ZF ← 1;
    ELSE ZF ← 0;
IF (SRC[127:0] AND NOT DEST[127:0] == 0) THEN CF ← 1;
    ELSE CF ← 0;
DEST[127:0] Unmodified;
AF = OF = PF = SF ← 0;
```

Intel C/C++ Compiler Intrinsic Equivalent

PTEST int _mm_testz_si128 (__m128i s1, __m128i s2);
int _mm_testc_si128 (__m128i s1, __m128i s2);
int _mm_testnzc_si128 (__m128i s1, __m128i s2);

Flags Affected

The 0F, AF, PF, SF flags are cleared and the ZF, CF flags are set according to the operation

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.

#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

ROUNDPD — Round Packed Double Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 09 /r ib	ROUNDPD *xmm1, xmm2/m128, imm8*	Valid	Valid	Round packed double precision floating-point values in *xmm2/m128* and place the result in *xmm1*. The rounding mode is determined by *imm8*.

Description

Round the 2 double precision floating-point values in the source operand (second operand) by the rounding mode specified in the immediate operand (third operand) and place the result in the destination operand (first operand). The rounding process rounds each input value to an integer value. The immediate operand specifies control fields for the rounding operation, three bit fields are defined and shown in Figure 5-2. Bit 3 of the immediate byte controls processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Table 5-9 lists the encoded values for rounding-mode field). The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN. If DAZ is set to ‘1 then denormals will be converted to zero before rounding.

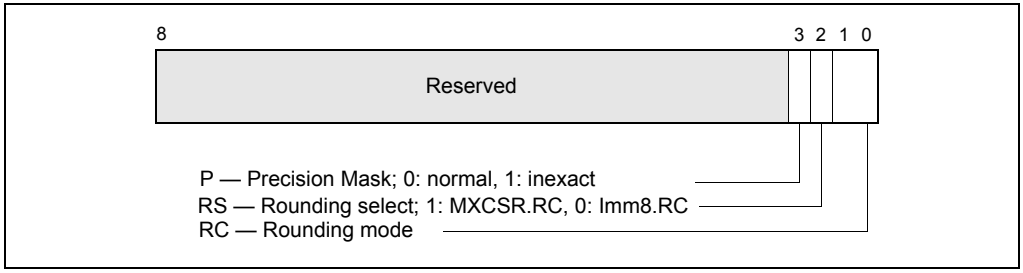

Figure 5-2. Bit Control Fields of Immediate Byte for ROUNDxx Instruction

Table 5-9. Rounding Modes and Encoding of Rounding Control (RC) Field

Rounding Mode	RC Field Setting	Description
Round to nearest (even)	00B	Rounded result is the closest to the infinitely precise result. If two values are equally close, the result is the even value (i.e., the integer value with the least-significant bit of zero).
Round down (toward −∞)	01B	Rounded result is closest to but no greater than the infinitely precise result.
Round up (toward +∞)	10B	Rounded result is closest to but no less than the infinitely precise result.
Round toward zero (Truncate)	11B	Rounded result is closest to but no greater in absolute value than the infinitely precise result.

Operation

ROUNDPD

```
IF (imm[2] == ‘1) THEN     // rounding mode is determined by MXCSR.RC
    DEST[63:0] ← ConvertDPFPToInteger_M(SRC[63:0]);
    DEST[127:64] ← ConvertDPFPToInteger_M(SRC[127:64]);
ELSE          // rounding mode is determined by IMM8.RC
    DEST[63:0] ← ConvertDPFPToInteger_I(SRC[63:0]);
    DEST[127:64] ← ConvertDPFPToInteger_I(SRC[127:64]);
FI  // If SRC == SNaN then RoundToIntegralValue will set DEST ← QNaN
    // The Precision exception is signaled only if imm[3] == ‘0
    // The Precision exception is not signaled if imm[3] == ‘1
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ROUNDPD  __m128 mm_round_pd(__m128d s1, int iRoundMode);
         __m128 mm_floor_pd(__m128d s1);
         __m128 mm_ceil_pd(__m128d s1);
```

SIMD Floating-Point Exceptions

Invalid (signaled only if SRC = SNaN)

Precision (signaled only if imm[3] == ‘0; if imm[3] == ‘1, then the Precision Mask in the MXSCSR is ignored and precision exception is not signaled.)

Note that Denormal is not signaled by ROUNDPD.

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.

	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.

If LOCK prefix is used.

ROUNDPS — Round Packed Single Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 08 /r ib	ROUNDPS *xmm1, xmm2/m128, imm8*	Valid	Valid	Round packed single precision floating-point values in *xmm2/m128* and place the result in *xmm1*. The rounding mode is determined by *imm8*.

Description

Round the 4 single precision floating-point values in the source operand (second operand) by the rounding mode specified in the immediate operand (third operand) and place the result in the destination operand (first operand). The rounding process rounds the input to an integral value and returns the result as a single precision floating-point value. The immediate operand specifies control fields for the rounding operation, three bit fields are defined and shown in Figure 5-2. Bit 3 of the immediate byte controls processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Table 5-9 lists the encoded values for rounding-mode field). The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN. If DAZ is set to ‘1 then denormals will be converted to zero before rounding.

Operation

ROUNDPS

```
IF (imm[2] == ‘1) THEN     // rounding mode is determined by MXCSR.RC
    DEST[31:0] ← ConvertSPFPToInteger_M(SRC[31:0]);
    DEST[63:32] ← ConvertSPFPToInteger_M(SRC[63:32]);
    DEST[95:64] ← ConvertSPFPToInteger_M(SRC[95:64]);
    DEST[127:96] ← ConvertSPFPToInteger_M(SRC[127:96]);
ELSE          // rounding mode is determined by IMM8.RC
    DEST[31:0] ← ConvertSPFPToInteger_I(SRC[31:0]);
    DEST[63:32] ← ConvertSPFPToInteger_I(SRC[63:32]);
    DEST[95:64] ← ConvertSPFPToInteger_I(SRC[95:64]);
    DEST[127:96] ← ConvertSPFPToInteger_I(SRC[127:96]);
FI
    // If SRC == SNaN then RoundToIntegralValue will set DEST ← QNaN
    // The Precision exception is signaled only if imm[3] == ‘0
    // The Precision exception is not signaled if imm[3] == ‘1
```

Intel C/C++ Compiler Intrinsic Equivalent

ROUNDPS __m128 mm_round_ps(__m128 s1, int iRoundMode);
__m128 mm_floor_ps(__m128 s1);
__m128 mm_ceil_ps(__m128 s1);

SIMD Floating-Point Exceptions

Invalid (signaled only if SRC = SNaN)

Precision (signaled only if imm[3] == ‘0; if imm[3] == ‘1, then the Precision Mask in the MXSCSR is ignored and precision exception is not signaled.)

Note that Denormal is not signaled by ROUNDPS.

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If a memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.

ROUNDSD — Round Scalar Double Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 0B /r ib	ROUNDSD *xmm1, xmm2/m64, imm8*	Valid	Valid	Round the low packed double precision floating-point value in *xmm2/m64* and place the result in *xmm1*. The rounding mode is determined by *imm8*.

Description

Round the DP FP value in the source operand (second operand) by the rounding mode specified in the immediate operand (third operand) and place the result in the destination operand (first operand). The rounding process rounds the lowest double precision floating-point input to an integral value and returns the result as a double precision floating-point value in the lowest position. The upper double precision floating-point value in the destination is retained. The immediate operand specifies control fields for the rounding operation, three bit fields are defined and shown in Figure 5-2. Bit 3 of the immediate byte controls processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Table 5-9 lists the encoded values for rounding-mode field). The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN. If DAZ is set to ‘1 then denormals will be converted to zero before rounding.

Operation

ROUNDSD

```
IF (imm[2] == ‘1) THEN     // rounding mode is determined by MXCSR.RC
    DEST[63:0] ← ConvertDPFPToInteger_M(SRC[63:0]);
ELSE          // rounding mode is determined by IMM8.RC
    DEST[63:0] ← ConvertDPFPToInteger_I(SRC[63:0]);
FI
    // If SRC == SNaN then RoundToIntegralValue will set DEST ← QNaN
    // The Precision exception is signaled only if imm[3] == ‘0
    // The Precision exception is not signaled if imm[3] == ‘1
    // DEST[127:63] remains unchanged ;
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ROUNDSD   __m128d mm_round_sd(__m128d dst, __m128d s1, int iRoundMode);
          __m128d mm_floor_sd(__m128d dst, __m128d s1);
          __m128d mm_ceil_sd(__m128d dst, __m128d s1);
```

SIMD Floating-Point Exceptions

Invalid (signaled only if SRC = SNaN)

Precision (signaled only if imm[3] == ‘0; if imm[3] == ‘1, then the Precision Mask in the MXSCSR is ignored and precision exception is not signaled.)

Note that Denormal is not signaled by ROUNDSD.

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

ROUNDSS — Round Scalar Single Precision Floating-Point Values

Opcode	Instruction	64-bit Mode	Compat/ Leg Mode	Description
66 0F 3A 0A /r ib	ROUNDSS *xmm1, xmm2/m32, imm8*	Valid	Valid	Round the low packed single precision floating-point value in *xmm2/m32* and place the result in *xmm1*. The rounding mode is determined by *imm8*.

Description

Round the single precision floating-point value in the source operand (second operand) by the rounding mode specified in the immediate operand (third operand) and place the result in the destination operand (first operand). The rounding process rounds the lowest single precision floating-point input to an integral value and returns the result as a single precision floating-point value in the lowest position. The upper three single precision floating-point values in the destination are retained. The immediate operand specifies control fields for the rounding operation, three bit fields are defined and shown in Figure 5-2. Bit 3 of the immediate byte controls processor behavior for a precision exception, bit 2 selects the source of rounding mode control. Bits 1:0 specify a non-sticky rounding-mode value (Table 5-9 lists the encoded values for rounding-mode field). The Precision Floating-Point Exception is signaled according to the immediate operand. If any source operand is an SNaN then it will be converted to a QNaN. If DAZ is set to ‘1 then denormals will be converted to zero before rounding.

Operation

ROUNDSS

```
IF (imm[2] == ‘1) THEN     // rounding mode is determined by MXCSR.RC
    DEST[31:0] ← ConvertSPFPToInteger_M(SRC[31:0]);
ELSE          // rounding mode is determined by IMM8.RC
    DEST[31:0] ← ConvertSPFPToInteger_I(SRC[31:0]);
FI
    // If SRC == SNaN then RoundToIntegralValue will set DEST ← QNaN
    // The Precision exception is signaled only if imm[3] == ‘0
    // The Precision exception is not signaled if imm[3] == ‘1
    // DEST[127:32] remains unchanged ;
```

Intel C/C++ Compiler Intrinsic Equivalent

ROUNDSS __m128 mm_round_ss(__m128 dst, __m128 s1, int iRoundMode);
__m128 mm_floor_ss(__m128 dst, __m128 s1);
__m128 mm_ceil_ss(__m128 dst, __m128 s1);

SIMD Floating-Point Exceptions

Invalid (signaled only if SRC = SNaN)

Precision (signaled only if imm[3] == ‘0; if imm[3] == ‘1, then the Precision Mask in the MXSCSR is ignored and precision exception is not signaled.)

Note that Denormal is not signaled by ROUNDSS.

Protected Mode and Compatibility Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segments.
#SS(0)	For an illegal address in the SS segment.
#PF(fault:code)	For a page fault.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Real Mode Exceptions

#GP(0)	if any part of the operand lies outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.TS[bit 3] = 1.
#UD	If CR0.EM[bit 2] = 1.
	If CR4.OSFXSR[bit 9] = 0.
	If CPUID.01H:ECX.SSE4_1[bit 19] = 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

#PF(fault-code)	For a page fault.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

Compatibility Mode Exceptions

Same exceptions as in Protected Mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	For a page fault.
#NM	If TS in CR0 is set.
#UD	If EM in CR0 is set.
	If OSFXSR in CR4 is 0.
	If CPUID feature flag ECX.SSE4_1 is 0.
	If LOCK prefix is used.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privelege level is 3.

APPENDIX A
INSTRUCTION SUMMARY AND ENCODINGS

1.1 SSE4.1 INSTRUCTION SUMMARY AND ENCODINGS

Table A-1. SSE4.1 Instruction Set Summary

Opcodes	Instruction	Description
66 0F 3A 0D	BLENDPD xmm1, xmm2/m128, imm8	Blend Packed Double Precision Floating-Point Values
66 0F 3A 0C	BLENDPS xmm1, xmm2/m128, imm8	Blend Packed Single Precision Floating-Point Values
66 0F 38 15	BLENDVPD xmm1, xmm2/m128, <XMM0>	Variable Blend Packed Double Precsion Floating-Point Values
66 0F 38 14	BLENDVPS xmm1, xmm2/m128, <XMM0>	Variable Blend Packed Single Precision Floating-Point Values
66 0F 3A 41	DPPD xmm1, xmm2/m128 imm8	Dot Product of Packed Double Precision Floating Point Values
66 0F 3A 40	DPPS xmm1, xmm2/m128. imm8	Dot Product of Packed Single Precision Floating Point Values
66 0F 3A 17	EXTRACTPS r/m32, xmm imm8	Extract Packed Single Precision Floating-Point Value
66 0F 3A 21	INSERTPS xmm1, xmm2/m32, imm8	Insert Packed Single Precision Floating-Point Value
66 0F 38 2A	MOVNTDQA xmm, m128	Load Double Quadword Non-Temporal Aligned Hint
66 0F 3A 42	MPSADBW xmm1, xmm2/m128, imm8	Compute Multiple Packed Sums of Absolute Difference
66 0F 38 2B	PACKUSDW xmm1, xmm2/m128	Pack with Unsigned Saturation
66 0F 38 10	PBLENDVB xmm1, xmm2/m128, <XMM0>	Variable Blend Packed Bytes
66 0F 3A 0E	PBLENDW xmm1, xmm2/m128, imm8	Blend Packed Words
66 0F 38 29	PCMPEQQ xmm1, xmm2/m128	Compare Packed Qword Data for Equal

Table A-1. SSE4.1 Instruction Set Summary

Opcodes	Instruction	Description
66 0F 3A 14	PEXTRB r32/m8, xmm, imm8	Extract Byte
66 0F 3A 16	PEXTRD r/m32, xmm, imm8	Extract Dword
66 REX.w 0F 3A 16	PEXTRQ r/m64, xmm, imm8	Extract Qword
66 0F 3A 15	PEXTRW r/m16, xmm, imm8	Extract Word
66 0F 38 41	PHMINPOSUW xmm1, xmm2/m128	Packed Horizontal Word Minimum
66 0F 3A 20	PINSRB xmm1, r32/m8, imm8	Insert Byte
66 0F 3A 22	PINSRD xmm1, r/m32, imm8	Insert Dword
66 REX.w 0F 3A 22	PINSRQ xmm1, r/m64, imm8	Insert Qword
66 0F 38 3C	PMAXSB xmm1, xmm2/m128	Maximum of Packed Signed Byte Integers
66 0F 38 3D	PMAXSD xmm1, xmm2/m128	Maximum of Packed Signed Dword Integers
66 0F 38 3F	PMAXUD xmm1, xmm2/m128	Maximum of Packed Unsigned Dword Integers
66 0F 38 3E	PMAXUW xmm1, xmm2/m128	Maximum of Packed Unsigned Word Integers
66 0F 38 38	PMINSB xmm1, xmm2/m128	Minimum of Packed Signed Byte Integers
66 0F 38 39	PMINSD xmm1, xmm2/m128	Minimum of Packed Signed Dword Integers
66 0F 38 3B	PMINUD xmm1, xmm2/m128	Minimum of Packed Unsigned Dword Integers
66 0F 38 3A	PMINUW xmm1, xmm2/m128	Minimum of Packed Unsigned Word Integers
66 0F 38 21	PMOVSXBD xmm1, xmm2/m32	Packed Move with Sign Extend - Byte to Dword
66 0F 38 22	PMOVSXBQ xmm1, xmm2/m16	Packed Move with Sign Extend - Byte to Qword
66 0F 38 20	PMOVSXBW xmm1, xmm2/m64	Packed Move with Sign Extend - Byte to Word
66 0F 38 23	PMOVSXWD xmm1, xmm2/m64	Packed Move with Sign Extend - Word to Dword
66 0F 38 24	PMOVSXWQ xmm1, xmm2/m32	Packed Move with Sign Extend - Word to Qword
66 0F 38 25	PMOVSXDQ xmm1, xmm2/m64	Packed Move with Sign Extend - Dword to Qword
66 0F 38 31	PMOVZXBD xmm1, xmm2/m32	Packed Move with Zero Extend - Byte to Dword

Table A-1. SSE4.1 Instruction Set Summary

Opcodes	Instruction	Description
66 0F 38 32	PMOVZXBQ xmm1, xmm2/m16	Packed Move with Zero Extend - Byte to Qword
66 0F 38 30	PMOVZXBW xmm1, xmm2/m64	Packed Move with Zero Extend - Byte to Word
66 0F 38 33	PMOVZXWD xmm1, xmm2/m64	Packed Move with Zero Extend - Word to Dword
66 0F 38 34	PMOVZXWQ xmm1, xmm2/m32	Packed Move with Zero Extend - Word to Qword
66 0F 38 35	PMOVZXDQ xmm1, xmm2/m64	Packed Move with Zero Extend - Dword to Qword
66 0F 38 28	PMULDQ xmm1, xmm2/m128	Multiply Packed Signed Dword Integers
66 0F 38 40	PMULLD xmm1, xmm2/m128	Multiply Packed Signed Dword Integers and Store Low Result
66 0F 38 17	PTEST xmm1, xmm2/m128	Logical Compare
66 0F 3A 09	ROUNDPD xmm1, xmm2/m128, imm8	Round Packed Double Precision Floating-Point Values
66 0F 3A 08	ROUNDPS xmm1, xmm2/m128, imm8	Round Packed Single Precision Floating-Point Values
66 0F 3A 0B	ROUNDSD xmm1, xmm2/m64, imm8	Round Scalar Double Precision Floating-Point Values
66 0F 3A 0A	ROUNDSS xmm1, xmm2/m32, imm8	Round Scalar Single Precision Floating-Point Values

Table A-2 provides SSE4.1 formats and encodings. Some SSE4.1 instructions require a mandatory prefix (66H, F2H, F3H) as part of the three-byte opcode. These prefixes are included in the tables.

In 64-bit mode, some instructions requires REX.W, the byte sequence of REX.W prefix in the opcode sequence is shown.

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
BLENDPD — Blend Packed Double-Precision Floats	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1010: 0000 1101:11 xmmreg1 xmmreg2

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0000 1101: mod xmmreg r/m
BLENDPS — Blend Packed Single-Precision Floats	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1010: 0000 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0000 1100: mod xmmreg r/m
BLENDVPD — Variable Blend Packed Double-Precision Floats	
xmmreg to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0101:11 xmmreg1 xmmreg2
mem to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0101: mod xmmreg r/m
BLENDVPS — Variable Blend Packed Single-Precision Floats	
xmmreg to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0100:11 xmmreg1 xmmreg2
mem to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0100: mod xmmreg r/m
DPPD — Packed Double-Precision Dot Products	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0001: mod xmmreg r/m: imm8
DPPS — Packed Single-Precision Dot Products	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0000: mod xmmreg r/m: imm8
EXTRACTPS — Extract From Packed Single-Precision Floats	
reg from xmmreg , imm8	0110 0110:0000 1111:0011 1010: 0001 0111:11 reg xmmreg: imm8

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
mem from xmmreg , imm8	0110 0110:0000 1111:0011 1010: 0001 0111: mod r/m xmmreg: imm8
INSERTPS — Insert Into Packed Single-Precision Floats	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0001: mod xmmreg r/m: imm8
MOVNTDQA — Load Double Quadword Non-temporal Aligned	
m128 to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1010:11 r/m xmmreg2
MPSADBW — Multiple Packed Sums of Absolute Difference	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0100 0010: mod xmmreg r/m: imm8
PACKUSDW — Pack with Unsigned Saturation	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1011: mod xmmreg r/m
PBLENDVB — Variable Blend Packed Bytes	
xmmreg to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0000:11 xmmreg1 xmmreg2
mem to xmmreg <xmm0>	0110 0110:0000 1111:0011 1000: 0001 0000: mod xmmreg r/m
PBLENDW — Blend Packed Words	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 1110:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1110: mod xmmreg r/m: imm8
PCMPEQQ — Compare Packed Qword Data of Equal	

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1001: mod xmmreg r/m
PEXTRB — Extract Byte	
reg from xmmreg , imm8	0110 0110:0000 1111:0011 1010: 0001 0100:11 reg xmmreg: imm8
xmmreg to mem, imm8	0110 0110:0000 1111:0011 1010: 0001 0100: mod r/m xmmreg: imm8
PEXTRD — Extract DWord	
reg from xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 0110:11 reg xmmreg: imm8
xmmreg to mem, imm8	0110 0110:0000 1111:0011 1010: 0001 0110: mod r/m xmmreg: imm8
PEXTRQ — Extract QWord	
r64 from xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0001 0110:11 reg xmmreg: imm8
m64 from xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0001 0110: mod r/m xmmreg: imm8
PEXTRW — Extract Word	
reg from xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 0101:11 reg xmmreg: imm8
mem from xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0001 0101: mod r/m xmmreg: imm8
PHMINPOSUW — Packed Horizontal Word Minimum	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0100 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0100 0001: mod xmmreg r/m
PINSRB — Extract Byte	
reg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0000:11 xmmreg reg: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0000: mod xmmreg r/m: imm8

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
PINSRD — Extract DWord	
reg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0010:11 xmmreg reg: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0010 0010: mod xmmreg r/m: imm8
PINSRQ — Extract QWord	
r64 to xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0010 0010:11 xmmreg reg: imm8
m64 to xmmreg, imm8	0110 0110:REX.W:0000 1111:0011 1010: 0010 0010: mod xmmreg r/m: imm8
PMAXSB — Maximum of Packed Signed Byte Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1100: mod xmmreg r/m
PMAXSD — Maximum of Packed Signed Dword Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1101: mod xmmreg r/m
PMAXUD — Maximum of Packed Unsigned Dword Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1111:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1111: mod xmmreg r/m
PMAXUW — Maximum of Packed Unsigned Word Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1110:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1110: mod xmmreg r/m
PMINSB — Minimum of Packed Signed Byte Integers	

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1000: mod xmmreg r/m
PMINSD — Minimum of Packed Signed Dword Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1001: mod xmmreg r/m
PMINUD — Minimum of Packed Unsigned Dword Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1011: mod xmmreg r/m
PMINUW — Minimum of Packed Unsigned Word Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 1010: mod xmmreg r/m
PMOVSXBD — Packed Move Sign Extend - Byte to Dword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0001: mod xmmreg r/m
PMOVSXBQ — Packed Move Sign Extend - Byte to Qword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0010: mod xmmreg r/m
PMOVSXBW — Packed Move Sign Extend - Byte to Word	

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0000: mod xmmreg r/m
PMOVSXWD — Packed Move Sign Extend - Word to Dword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0011: mod xmmreg r/m
PMOVSXWQ — Packed Move Sign Extend - Word to Qword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0100: mod xmmreg r/m
PMOVSXDQ — Packed Move Sign Extend - Dword to Qword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 0101: mod xmmreg r/m
PMOVZXBD — Packed Move Zero Extend - Byte to Dword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0001:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0001: mod xmmreg r/m
PMOVZXBQ — Packed Move Zero Extend - Byte to Qword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0010:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0010: mod xmmreg r/m
PMOVZXBW — Packed Move Zero Extend - Byte to Word	

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0000: mod xmmreg r/m
PMOVZXWD — Packed Move Zero Extend - Word to Dword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0011:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0011: mod xmmreg r/m
PMOVZXWQ — Packed Move Zero Extend - Word to Qword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0100:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0100: mod xmmreg r/m
PMOVZXDQ — Packed Move Zero Extend - Dword to Qword	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0101:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0101: mod xmmreg r/m
PMULDQ — Multiply Packed Signed Dword Integers	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0010 1000: mod xmmreg r/m
PMULLD — Multiply Packed Signed Dword Integers, Store low Result	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0100 0000:11 xmmreg1 xmmreg2
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0100 0000: mod xmmreg r/m
PTEST — Logical Compare	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0001 0111:11 xmmreg1 xmmreg2

Table A-2. Encodings of SSE4.1 instructions

Instruction and Format	Encoding
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0001 0111: mod xmmreg r/m
ROUNDPD — Round Packed Double-Precision Values	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1001: mod xmmreg r/m: imm8
ROUNDPS — Round Packed Single-Precision Values	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1000: mod xmmreg r/m: imm8
ROUNDSD — Round Scalar Double-Precision Value	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1011:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1011: mod xmmreg r/m: imm8
ROUNDSS — Round Scalar Single-Precision Value	
xmmreg to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg, imm8	0110 0110:0000 1111:0011 1010: 0000 1010: mod xmmreg r/m: imm8

1.2 SSE4.2 INSTRUCTION SUMMARY AND ENCODINGS

Table A-3. SSE4.2 Instruction Set Summary

Opcodes	**Instruction**	**Description**
F2 0F 38 F0 /r	CRC32 r32, r/m8	Accumulate CRC32 on r/m8

Table A-3. SSE4.2 Instruction Set Summary

Opcodes	Instruction	Description
F2 REX.W 0F 38 F0 /r	CRC32 r64, r/m8	Accumulate CRC32 on r/m8
F2 0F 38 F1 /r	CRC32 r32, r/m16	Accumulate CRC32 on r/m16
F2 0F 38 F1 /r	CRC32 r32, r/m32	Accumulate CRC32 on r/m32
F2 REX.W 0F 38 F1 /r	CRC32 r64, r/m64	Accumulate CRC32 on r/m64
66 0F 3A 61 /r imm8	PCMPESTRI xmm1, xmm2/m128, imm8	Perform a packed comparison of string data with explicit lengths, generating an index in ECX
66 0F 3A 60 /r imm8	PCMPESTRM xmm1, xmm2/m128, imm8	Perform a packed comparison of string data with explicit lengths, generating a mask in XMM0
66 0F 3A 63 /r imm8	PCMPISTRI xmm1, xmm2/m128, imm8	Perform a packed comparison of string data with implicit lengths, generating an index in ECX
66 0F 3A 62 /r imm8	PCMPISTRM xmm1, xmm2/m128, imm8	Perform a packed comparison of string data with implicit lengths, generating a mask in XMM0
66 0F 38 37 /r	PCMPGTQ xmm1, xmm2/m128	Compare packed qwords in xmm2/m128 and xmm1 for greater than
F3 0F B8 /r	POPCNT r16, r/m16	Calculate the number of bits set to 1 from r/m16bb
F3 0F B8 /r	POPCNT r32, r/m32	Calculate the number of bits set to 1 from r/m32
F3 REX.W 0F B8 /r	POPCNT r64, r/m64	Calculate the number of bits set to 1 from r/m64

Table A-4 provides SSE4.2 formats and encodings. Some SSE4.2 instructions require a mandatory prefix (66H, F2H, F3H) as part of the three-byte opcode. These prefixes are included in the tables. In 64-bit mode, some instructions requires REX.W, the byte sequence of REX.W prefix in the opcode sequence is shown.

Table A-4. Encodings of SSE4.2 instructions

Instruction and Format	Encoding
CRC32 — Accumulate CRC32	
reg2 to reg1	1111 0010:0000 1111:0011 1000: 1111 000w :11 reg1 reg2
mem to reg	1111 0010:0000 1111:0011 1000: 1111 000w : mod reg r/m

Table A-4. Encodings of SSE4.2 instructions

Instruction and Format	Encoding
bytereg2 to reg1	1111 0010:0100 WR0B:0000 1111:0011 1000: 1111 0000 :11 reg1 bytereg2
m8 to reg	1111 0010:0100 WR0B:0000 1111:0011 1000: 1111 0000 : mod reg r/m
qwreg2 to qwreg1	1111 0010:0100 1R0B:0000 1111:0011 1000: 1111 0000 :11 qwreg1 qwreg2
mem64 to qwreg	1111 0010:0100 1R0B:0000 1111:0011 1000: 1111 0000 : mod qwreg r/m
PCMPESTRI— Packed Compare Explicit-Length Strings To Index	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0001:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0001: mod xmmreg r/m
PCMPESTRM— Packed Compare Explicit-Length Strings To Mask	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0000:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0000: mod xmmreg r/m
PCMPISTRI— Packed Compare Implicit-Length String To Index	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0011:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0011: mod xmmreg r/m
PCMPISTRM— Packed Compare Implicit-Length Strings To Mask	
xmmreg2 to xmmreg1, imm8	0110 0110:0000 1111:0011 1010: 0110 0010:11 xmmreg1 xmmreg2: imm8
mem to xmmreg	0110 0110:0000 1111:0011 1010: 0110 0010: mod xmmreg r/m
PCMPGTQ— Packed Compare Greater Than	
xmmreg to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0111:11 xmmreg1 xmmreg2

Table A-4. Encodings of SSE4.2 instructions

Instruction and Format	Encoding
mem to xmmreg	0110 0110:0000 1111:0011 1000: 0011 0111: mod xmmreg r/m
POPCNT— Return Number of Bits Set to 1	
reg2 to reg1	1111 0011:0000 1111:1011 1000:11 reg1 reg2
mem to reg1	1111 0011:0000 1111:1011 1000:mod reg1 r/m
qwreg2 to qwreg1	1111 0011:0100 1R0B:0000 1111:1011 1000:11 reg1 reg2
mem64 to qwreg1	1111 0011:0100 1R0B:0000 1111:1011 1000:mod reg1 r/m

APPENDIX B
INSTRUCTION OPCODE MAP

SSE4.1 opcodes are indicated by blue table cells. SSE4.1 are indicated by yellow table cells.

Table B-1. Three-byte Opcode Map: 00H — 7FH (First Two Bytes are 0F 38H)

	0	1	2	3	4	5	6	7
0	pshufb Pq, Qq pshufb (66) Vdq, Wdq	phaddw Pq, Qq phaddw (66) Vdq, Wdq	phaddd Pq, Qq phaddd (66) Vdq, Wdq	phaddsw Pq, Qq phaddsw (66) Vdq, Wdq	pmaddubsw Pq, Qq pmaddubsw (66) Vdq, Wdq	phsubw Pq, Qq phsubw (66) Vdq, Wdq	phsubd Pq, Qq phsubd (66) Vdq, Wdq	phsubsw Pq, Qq phsubsw (66) Vdq, Wdq
1	**pblendvb (66) Vdq, Wdq**				**blendvps (66) Vdq, Wdq**	**blendvpd (66) Vdq, Wdq**		**ptest (66) Vdq, Wdq**
2	**pmovsxbw (66) Vdq, Wdq**	**pmovsxbd (66) Vdq, Wdq**	**pmovsxbq (66) Vdq, Wdq**	**pmovsxwd (66) Vdq, Wdq**	**pmovsxwq (66) Vdq, Wdq**	**pmovsxdq (66) Vdq, Wdq**		
3	**pmovzxbw (66) Vdq, Wdq**	**pmovzxbd (66) Vdq, Wdq**	**pmovzxbq (66) Vdq, Wdq**	**pmovzxwd (66) Vdq, Wdq**	**pmovzxwq (66) Vdq, Wdq**	**pmovzxdq (66) Vdq, Wdq**		**pcmpgtq (66) Vdq, Wdq**
4	**pmulld (66) Vdq, Wdq**	**phminposuw (66) Vdq, Wdq**						
5-E	*...No changes or additions...*							
F	**crc32 (F2) Gv, Eb**	**crc32 (F2) Gv, Ev**						

Table B-2. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H)

	8	9	A	B	C	D	E	F
0	psignb Pq, Qq psignb (66) Vdq, Wdq	psignw Pq, Qq psignw (66) Vdq, Wdq	psignd Pq, Qq psignd (66) Vdq, Wdq	pmulhrsw Pq, Qq pmulhrsw (66) Vdq, Wdq				
1					psabsb Pq, Qq pabsb (66) Vdq, Wdq	psabsw Pq, Qq pabsw (66) Vdq, Wdq	psabsd Pq, Qq pabsd (66) Vdq, Wdq	

Table B-2. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H) (Contd.)

	8	9	A	B	C	D	E	F
2	**pmuldq (66) Vdq, Wdq**	**pcmpeqq (66) Vdq, Wdq**	**movntdqa (66) Mdq,Vdq**	**packusdw (66) Vdq, Wdq**				
3	**pminsb (66) Vdq, Wdq**	**pminsd (66) Vdq, Wdq**	**pminuw (66) Vdq, Wdq**	**pminud (66) Vdq, Wdq**	**pmaxsb (66) Vdq, Wdq**	**pmaxsd (66) Vdq, Wdq**	**pmaxuw (66) Vdq, Wdq**	**pmaxud (66) Vdq, Wdq**
4-F	*...No changes or additions...*							

Table B-3. Three-byte Opcode Map: 00H — 7FH (First Two Bytes are 0F 3AH)

	0	1	2	3	4	5	6	7
0								
1					**pextrb (66) Rd/Mb, Vdq, Ib**	**pextrw (66) Rd/Mw, Vdq, Ib**	**pextrd/pextrq (66) Ed/q, Vdq, Ib**	**extractps (66) Ed, Vdq, Ib**
2	**pinsrb (66) Vdq, Eb, Ib**	**insertps (66) Vdq, Udq/Md, Ib**	**pinsrd/pinsrq (66) Vdq, Ed/q, Ib**					
3								
4	**dpps (66) Vdq, Wdq, Ib**	**dppd (66) Vdq, Wdq, Ib**	**mpsadbw (66) Vdq, Wdq, Ib**					
5								
6	**pcmpestrm (66) Vdq,Wdq,Ib**	**pcmpestri (66) Vdq,Wdq,Ib**	**pcmpistrm (66) Vdq,Wdq,Ib**	**pcmpistrm (66) Vdq,Wdq,Ib**				
7-F	*....No changes or additions...*							

NOTE: Instructions pinsrq and pextrq require a REX.w prefix. If the REX.w prefix is not present then these instructions will be treated as pinsrd and pextrd.

Table B-4. Three-byte Opcode Map: 80H — FFH (First Two Bytes are 0F 3AH)

	8	9	A	B	C	D	E	F
0	**roundps (66) Vdq, Wdq, Ib**	**roundpd (66) Vdq, Wdq, Ib**	**roundss (66) Vss, Wss, Ib**	**roundsd (66) Vsd, Wsd, Ib**	**blendps (66) Vdq, Wdq, Ib**	**blendpd (66) Vdq, Wdq, Ib**	**pblendw (66) Vdq, Wdq, Ib**	palign Pq, Qq, Ib palign (66) Vdq, Wdq, Ib
1-F	*...No changes or additions...*							

Table B-5. Two-byte Opcode Map: B8H (First Byte is 0FH)

	8
B	JMPE (reserved for emulator on IPF)
	POPCNT **(F3)** **Gv, Ev**