Prescott New Instructions Software Developer's Guide

252490-003

June 2003

Revision History

.002	Table 2-1: Revised function 4H and 80000006H. Section 2.1.2: Corrected extended family encoding display algorithm. Table 2.5: Revised for consistency. Figure 2.6: Added clarification. Section 4.3.1: Corrected LDDQU type usage.
.003	Included intrinsics. Included opcodes. Corrected comment in Example 4.1.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® processors may contain design defects or errors known as errata. Current characterized errata are available on request.

Intel, Pentium, Intel Xeon, Intel Pentium III Xeon, Intel NetBurst, MMX, and Celeron, are trademarks or registered trademarks of Intel Corporation and its subsidiaries in the United States and other countries.

Prescott is a code name that is used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and a Hyper-Threading Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See http://www.intel.com/info/hyperthreading/ for more information including details on which processors support HT Technology.

*Other names and brands may be claimed as the property of others.

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

COPYRIGHT © 2003 INTEL CORPORATION

CONTENTS

PAGE

PAGE

CHAPTER 1
NEXT GENERATION
INTEL® PROCESSOR OVERVIEW

1.1. KEY FEATURES

Prescott is the code name for a new generation of IA32 processors. The technology incorporates an enhanced Intel® NetBurst™ microarchitecture. Other key features of the Prescott include:

- Support for Hyper-Threading (HT) Technology[1]
- Prescott New Instructions support
- Deeper pipelining to enable higher frequency
- A High-speed System Bus

Prescott improves on the Pentium® 4 processor's hyper-pipelined technology to achieve even higher clock rates than previous generations of Pentium 4 processors. At the same time, the new processor has larger first-level and second-level caches, more store buffers, write-combining buffers.

Support for Prescott New Instructions does not require new OS support for saving and restoring the new state during a context switch, beyond that provided for Streaming SIMD Extensions. Prescott New Instructions are fully compatible with all software written for Intel® architecture microprocessors.

1.2. HYPER-THREADING TECHNOLOGY

Hyper-Threading Technology (HT Technology) makes a single physical processor appear as multiple logical processors by running two threads simultaneously. This is accomplished by duplicating the architecture state for each logical processor in the physical processor and sharing the physical execution resources within a physical processor package between the logical processors. Each logical processor maintains a complete architecture state (see Figure 1-1).

From a software or architecture perspective, this means operating systems and user programs can schedule processes or threads to logical processors as they would on conventional physical processors. From a microarchitectural perspective, this means that instructions from both logical processors will persist and execute simultaneously on shared execution resources.

---

1. Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and a Hyper-Threading Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See http://www.intel.com/info/hyperthreading/ for more information including details on which processors support HT Technology.

HT Technology is available across the server, workstation and desktop segments in the IA-32 processor family. Software detects support for HT Technology in IA-32 processors by using the CPUID instruction. All HT Technology configurations require a chipset and BIOS that utilize the technology, and an operating system that includes optimizations for HT Technology. See *www.intel.com/info/hyperthreading* for more information.

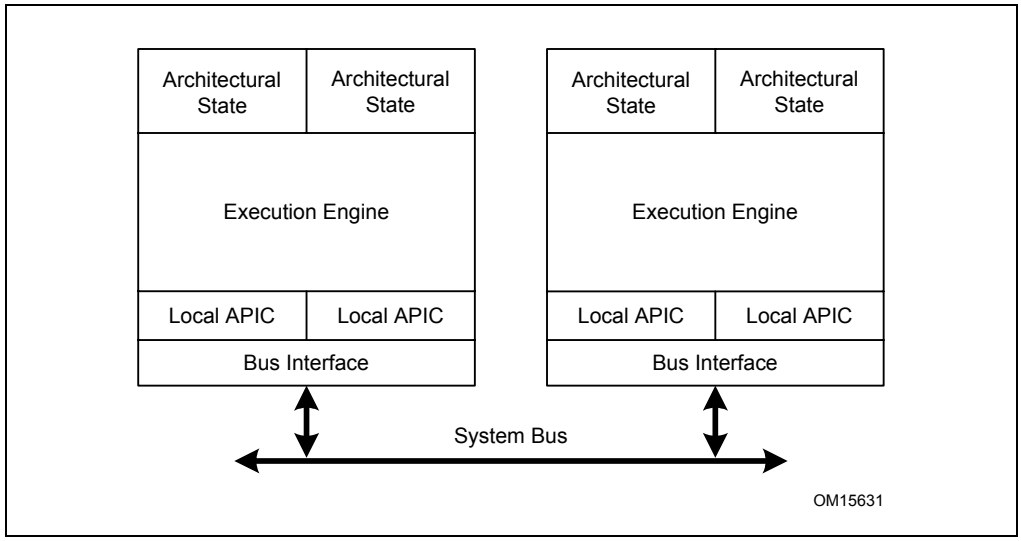


Figure 1-1. Two Logical Processors in One Physical Package

A system with processors that are HT Technology capable appear to the operating system and application software as having twice the number of processors as the number of physical processors. Operating systems manage logical processors as they do physical processors, scheduling run-able tasks or threads to logical processors.

Processors with HT Technology deliver higher performance than a comparable physical processor without HT technology, because two simultaneously running threads utilize more execution resources in the physical processor. However, HT Technology does not deliver the same performance as a multiprocessor system with two physical processors.

1.3. ENHANCED CPUID CAPABILITIES

The CPUID instruction has been enhanced to support the following new features:

- Prescott New Instructions
- Debug Trace Store Qualification
- Enhanced Intel SpeedStep® technology (uses model-specific registers on the processor)

- MONITOR-MWAIT support

The behavior of the CPUID instruction has not changed (although more values are returned). The instruction provides a wealth of information that are organized into pages or leaves; leaves are queried by loading different values in EAX and then executing the instruction.

For detailed information, see Chapter 2, “CPUID Extensions”.

1.4. PRESCOTT NEW INSTRUCTIONS

Prescott New Instruction technology for the IA-32 Intel architecture is a set of 13 new instructions that accelerate performance of Streaming SIMD Extensions technology, Streaming SIMD Extensions 2 technology, and x87-FP math capabilities. The new technology is compatible with existing software written for Intel architecture microprocessors and existing software should continue to run correctly, without modification, on microprocessors that incorporate Prescott New Instructions.

The thirteen new instructions are summarized in the following sections. For detailed information on each instruction, see Chapter 3, “Instruction Set Reference”.

1.4.1. One Instruction That Improves x87-FP Integer Conversion

FISTTP (Store Integer and Pop from x87-FP with Truncation) behaves like the FISTP instruction but uses truncation, irrespective of the rounding mode specified in the floating-point control word (FCW). The instruction converts the top of stack (ST0) to integer with rounding to truncate and pop the stack.

FISTTP is available in three precisions: short integer (word or 16-bit), integer (double word or 32-bit), and long integer (64-bit). With FISTTP, applications no longer need to change the FCW when truncation is desired. This instruction is the only x87-FP instruction in the Prescott New Instruction technology.

1.4.2. Three Instructions Enhance LOAD/MOVE/DUPLICATE Performance

MOVSHDUP loads/moves 128-bits, duplicating the second and fourth 32-bit data elements.

- MOVSHDUP OperandA OperandB
    - OperandA (128 bits, four data elements): 3_a, 2_a, 1_a, 0_a
    - OperandB (128 bits, four data elements): 3_b, 2_b, 1_b, 0_b
    - Result (stored in OperandA): 3_b, 3_b, 1_b, 1_b

MOVSLDUP loads/moves 128-bits, duplicating the first and third 32-bit data elements.

- MOVSLDUP OperandA OperandB

- OperandA (128 bits, four data elements): 3_a, 2_a, 1_a, 0_a
- OperandB (128 bits, four data elements): 3_b, 2_b, 1_b, 0_b
- Result (stored in OperandA): 2_b, 2_b, 0_b, 0_b

MOVDDUP loads/moves 64-bits (bits[63-0] if the source is a register) and returns the same 64 bits in both the lower and upper halves of the 128-bit result register. This duplicates the 64 bits from the source.

- MOVDDUP OperandA OperandB
  - OperandA (128 bits, two data elements): 1_a, 0_a
  - OperandB (64 bits, one data element): 0_b
  - Result (stored in OperandA): 0_b, 0_b

1.4.3. One Instruction Provides Specialized 128-bit Unaligned Data Load

LDDQU is a special 128-bit unaligned load designed to avoid cache line splits. If the address of the load is aligned on a 16-byte boundary, LDQQU loads the 16 bytes requested. If the address of the load is not aligned on a 16-byte boundary, LDDQU loads a 32-byte block starting at the 16-byte aligned address immediately below the load request. It then extracts the requested 16 bytes.

The instruction provides significant performance improvement on 128-bit unaligned memory accesses at the cost of some usage model restrictions.

1.4.4. Two Instructions Provide Packed Addition/Subtraction

ADDSUBPS has two 128-bit operands. The instruction performs single-precision addition on the second and fourth pairs of 32-bit data elements within the operands; and single-precision subtraction on the first and third pairs. This instruction is effective at evaluating complex products on packed single-precision data.

- ADDSUBPS OperandA OperandB
  - OperandA (128 bits, four data elements): 3_a, 2_a, 1_a, 0_a
  - OperandB (128 bits, four data elements): 3_b, 2_b, 1_b, 0_b
  - Result (stored in OperandA): 3_a+3_b, 2_a-2_b, 1_a+1_b, 0_a-0_b

ADDSUBPD has two 128-bit operands. The instruction performs double-precision addition on the second pair of quadwords, and double-precision subtraction on the first pair. This instruction is useful when evaluating complex products on packed double-precision data.

- ADDSUBPD OperandA OperandB
  - OperandA (128 bits, two data elements): 1_a, 0_a

- OperandB (128 bits, two data elements): 1_b, 0_b
- Result (stored in OperandA): 1_a+1_b, 0_a-0_b

1.4.5. Four Instructions Provide Horizontal Addition/Subtraction

Most SIMD instructions operate vertically. This means that the result in position i of the result is a function of the elements in position i of both operands. Horizontal addition/subtraction operates horizontally. This means that contiguous data elements from the same operand are used to produce a result data element.

HADDPS performs a single-precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand; the second element by adding the third and fourth elements of the first operand; the third by adding the first and second elements of the second operand; and the fourth by adding the third and fourth elements of the second operand.

- HADDPS OperandA OperandB
  - OperandA (128 bits, four data elements): 3_a, 2_a, 1_a, 0_a
  - OperandB (128 bits, four data elements): 3_b, 2_b, 1_b, 0_b
  - Result (Stored in OperandA): 3_b+2_b, 1_b+0_b, 3_a+2_a, 1_a+0_a

HSUBPS performs a single-precision subtraction on contiguous data elements. The first data element of the result is obtained by subtracting the second element of the first operand from the first element of the first operand; the second element by subtracting the fourth element of the first operand from the third element of the first operand; the third by subtracting the second element of the second operand from the first element of the second operand; and the fourth by subtracting the fourth element of the second operand from the third element of the second operand.

- HSUBPS OperandA OperandB
  - OperandA (128 bits, four data elements): 3_a, 2_a, 1_a, 0_a
  - OperandB (128 bits, four data elements): 3_b, 2_b, 1_b, 0_b
  - Result (Stored in OperandA): 2_b-3_b, 0_b-1_b, 2_a-3_a, 0_a-1_a

HADDPD performs a double-precision addition on contiguous data elements. The first data element of the result is obtained by adding the first and second elements of the first operand; the second element by adding the first and second elements of the second operand.

- HADDPD OperandA OperandB
  - OperandA (128 bits, two data elements): 1_a, 0_a
  - OperandB (128 bits, two data elements): 1_b, 0_b
  - Result (Stored in OperandA): 1_b+0_b, 1_a+0_a

HSUBPD performs a double-precision subtraction on contiguous data elements. The first data element of the result is obtained by subtracting the second element of the first operand from the

first element of the first operand; the second element by subtracting the second element of the second operand from the first element of the second operand.

- HSUBPD OperandA OperandB
  - OperandA (128 bits, two data elements): 1_a, 0_a
  - OperandB (128 bits, two data elements): 1_b, 0_b
  - Result (Stored in OperandA): 0_b-1_b, 0_a-1_a

1.4.6. Two Instructions Improve Synchronization Between Agents

MONITOR sets up an address range used to monitor write-back stores.

MWAIT enables a logical processor to enter into an optimized state while waiting for a write-back store to the address range set up by the MONITOR instruction.

Support for MONITOR/MWAIT is indicated by the CPUID MONITOR/MWAIT Software need not check for support of SSE in order to use the MONITOR/MWAIT.

CHAPTER 2
CPUID EXTENSIONS

2.1. VALUES RETURNED USING CPUID

CPUID instruction and feature-identification bits have been added for software to identify the features offered by Prescott New Instructions. Table 2-1 shows the value in EAX before a call to CPUID and the value returned. For impacted areas, see the bold type.

See also: Section 4.2.2., *Detecting Prescott New Instructions Technology Using CPUID*.

Table 2-1. CPUID Return Values

EAX Value	Leaf Information Returned	
0H	EAX	Maximum Input Value for Basic CPUID Information (see Table 2-2).
	EBX	"Genu"
	ECX	"ntel"
	EDX	"inel"
1H	EAX	**Version Information (Type, Family, Model, and Stepping ID; see Figure 2-3)**.
	EBX	Bits 7-0: Brand Index Bits 15-8: CLFLUSH line size. (Value ∗ 8 = cache line size in bytes) Bits 23-16: Number of logical processors per physical processor. (2 for the Prescott if Hyper-Threading Technology enabled) Bits 31-24: Processor Initial Local APIC ID
	ECX	**Extended Feature information (see Figure 2-1)**
	EDX	**Extended Feature Information (see Figure 2-2). PSN is not supported in Prescott)**
2H	EAX	Cache and TLB Information (see Table 2-5)
	EBX	Cache and TLB Information
	ECX	Cache and TLB Information
	EDX	Cache and TLB Information
3H	EAX	Reserved
	EBX	Reserved
	ECX	Reserved
	EDX	Reserved

EAX Value	Leaf Information Returned	
4H		**Deterministic Cache Parameters Leaf**
	EAX	**Bits 4-0: Cache Type****
		Bits 7-5: Cache Level (starts at 1)
		Bits 8: Self Initializing cache level (does not need SW initialization)
		Bits 9: Fully Associative cache
		Bits 13-10: Reserved
		Bits 25-14: Number of threads sharing this cache*
		Bits 31-26: Number of processor cores on this die (Multicore)*.
	EBX	**Bits 11-00: L = System Coherency Line Size***
		Bits 21-12: P = Physical Line partitions*
		Bits 31-22: W = Ways of associativity*.
	ECX	**Bits 31-00: S = Number of Sets***
	EDX	Reserved = 0
		*** Add one to the value in the register file to get the number. For example, the number of processor cores is EAX[31:26]+1.**
		**** Cache Types fields**
		0 = Null - No more caches
		1 = Data Cache
		2 = Instruction Cache
		3 = Unified Cache
		4-31 = Reserved
		NOTE: CPUID leaves > 3 < 80000000 are only visible when IA32_CR_MISC_ENABLES.BOOT_NT4 (bit 22) is clear (Default)
5H		**MONITOR/MWAIT Leaf**
	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	Reserved = 0
	EDX	Reserved = 0
		Extended Function CPUID Information
80000000H	EAX	Maximum Input Value for Extended Function CPUID Information (see Table 2-2).
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
80000001H	EAX	Extended Processor Signature and Extended Feature Bits. (Currently Reserved.)
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
80000002H	EAX	**Processor Brand String.**
	EBX	**Processor Brand String Continued.**
	ECX	**Processor Brand String Continued.**
	EDX	**Processor Brand String Continued.**

EAX Value	Leaf Information Returned
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
80000006H	EAX Reserved = 0 EBX Reserved = 0 ECX **Bits 7-0: Cache Line size** **Bits 15-12: L2 Associativity** **Bits 31-16: Cache size in 1K units** **In initial implementation ECX = 0x04008040** EDX Reserved = 0
80000007H	EAX Reserved = 0 EBX Reserved = 0 ECX Reserved = 0 EDX Reserved = 0

Table 2-2. Highest CPUID Source Operand for IA-32 Processors

IA-32 Processors	Highest Value in EAX	
	Basic Information	Extended Function Information
Prescott A Step	5H	80000008H

2.1.1. Extended Feature Information Returned in ECX and EDX

This section describes the feature information returned in ECX and EDX (by calling CPUID with EAX = 01H; see Table 2-1).

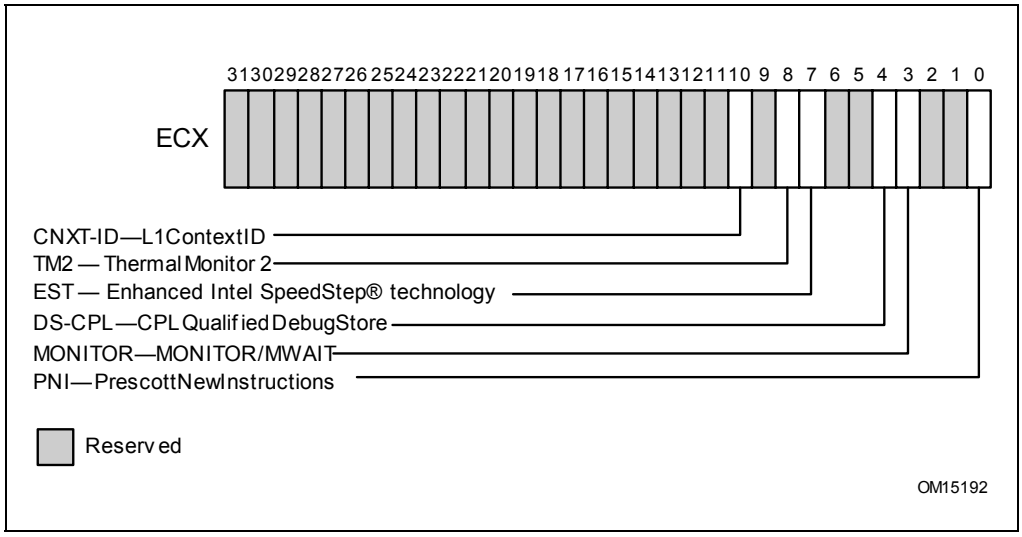


Figure 2-1. Prescott Feature Information Returned in the ECX Register

Table 2-3. More on Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	PNI	Prescott New Instructions. A value of 1 indicates the processor supports Prescott New Instructions.
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.
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.
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.

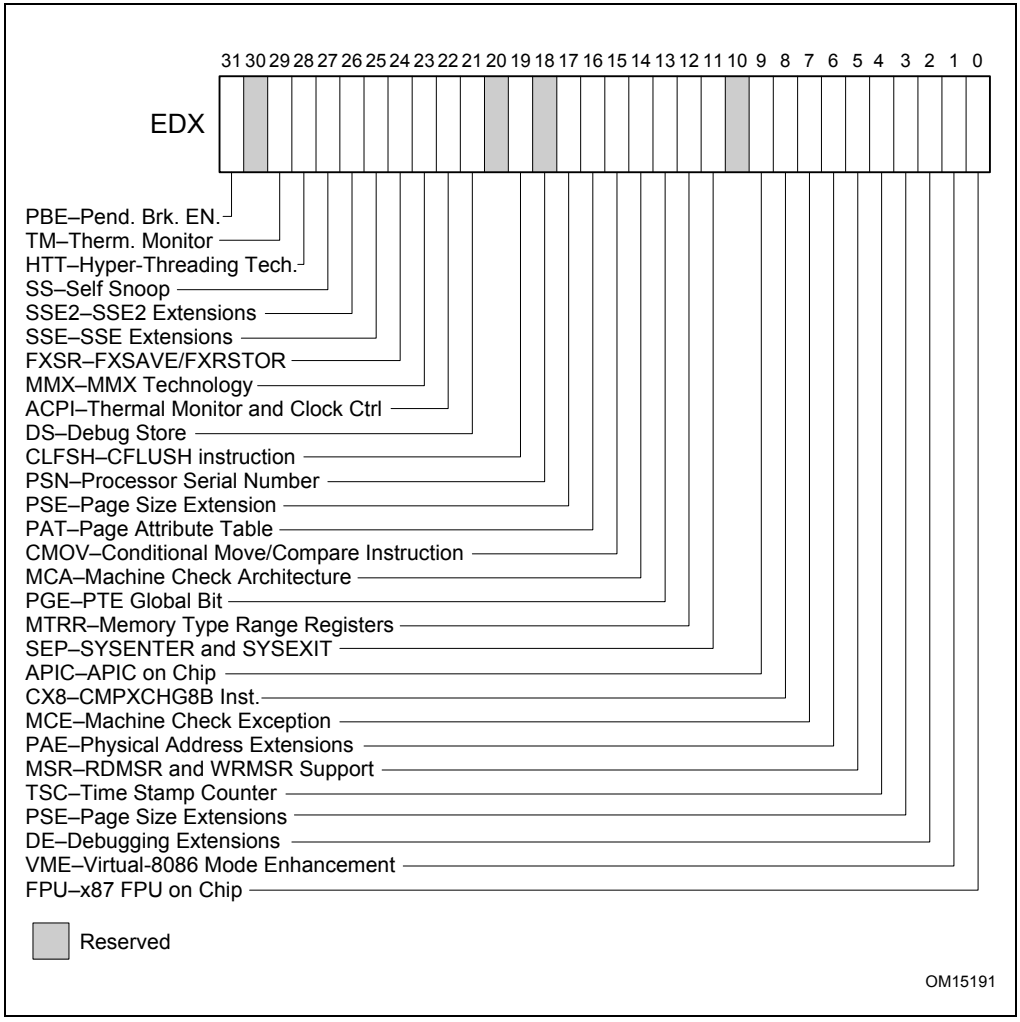


Figure 2-2. Prescott Feature Information in the EDX Register

2.1.2. Version Information Returned in EAX

This section describes version information returned in EAX (by calling CPUID with EAX = 01H; see Table 2-1). Note that Prescott does not support the Processor Serial Number.

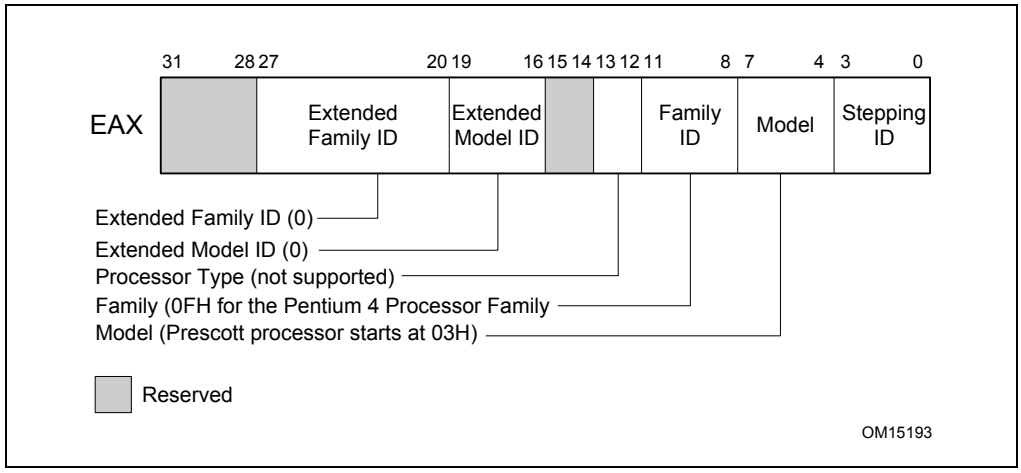


Figure 2-3. Version Information Returned by CPUID in EAX

The Extended Family ID and the Extended Model ID need to be examined only if the family is OFH. Often software will display processor information as a combination of family, model and stepping. Integrate the Family ID and Extended Family ID fields into the displayed family follows:

```
Displayed family = ((Extended Family ID(4-bits) << 4)) (8-bits)
+ Family ID (4-bits zero extended to 8-bits)
```

Compute the displayed model from the Model ID and the Extended Model ID as follows.

```
Displayed Model = ((Extended Model ID (4-bits) << 4))(8-bits)
+ Model (4-bits zero extended to 8-bits)
```

2.1.3. Brand and Maximum Frequency Identification

To facilitate the correct identification of Intel processors, the Pentium® 4 processor introduced the brand index and the brand string. Software uses the brand index to locate a brand identification string 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.

Table 2-4 shows the relationship between the brand indices and corresponding brand strings. All reserved entries in the brand identification table should be associated with a brand string that indicates that the index is reserved for future Intel processors. Software should be able to handle reserved brand indices suitably.

Table 2-4. Mapping Brand Index and Brand Identification String

Brand Index	Brand String
00H	Not supported
01H	Intel® Celeron® processor
02H	Intel® Pentium® III processor
03H	Intel® Pentium® III Xeon™ processor If processor signature = 000006B1H; then Intel® Celeron® processor
04H	Intel® Pentium® III processor
06H	Mobile Intel® Pentium® III processor -M
07H	Mobile Intel® Celeron® processor
08H	Intel® Pentium® 4 processor If processor signature ≥ 00000F13H; then Intel® Genuine processor
09H	Intel® Pentium® 4 processor
0AH	Intel® Celeron® processor
0BH	Intel® Xeon™ processor If processor signature < 00000F13H; then Intel® Xeon™ processor MP
0CH	Intel® Xeon™ processor MP
0EH	Mobile Intel® Pentium® 4 processor -M If processor signature < 00000F13H; then Intel® Xeon™ processor
0FH	Mobile Intel® Celeron® processor
Other Values	RESERVED

Using the brand string feature, future IA-32 architecture-based processors will return an ASCII brand identification string and a maximum operating frequency using CPUID calls. Note that the frequency returned is the maximum operating frequency, not the current frequency of the processor.

2.1.4. Determining Support for the Processor Brand String

Figure 2-5 describes the algorithm used for detection of the brand string feature. Processor brand identification software should execute this algorithm on all IA-32 architecture compatible processors.

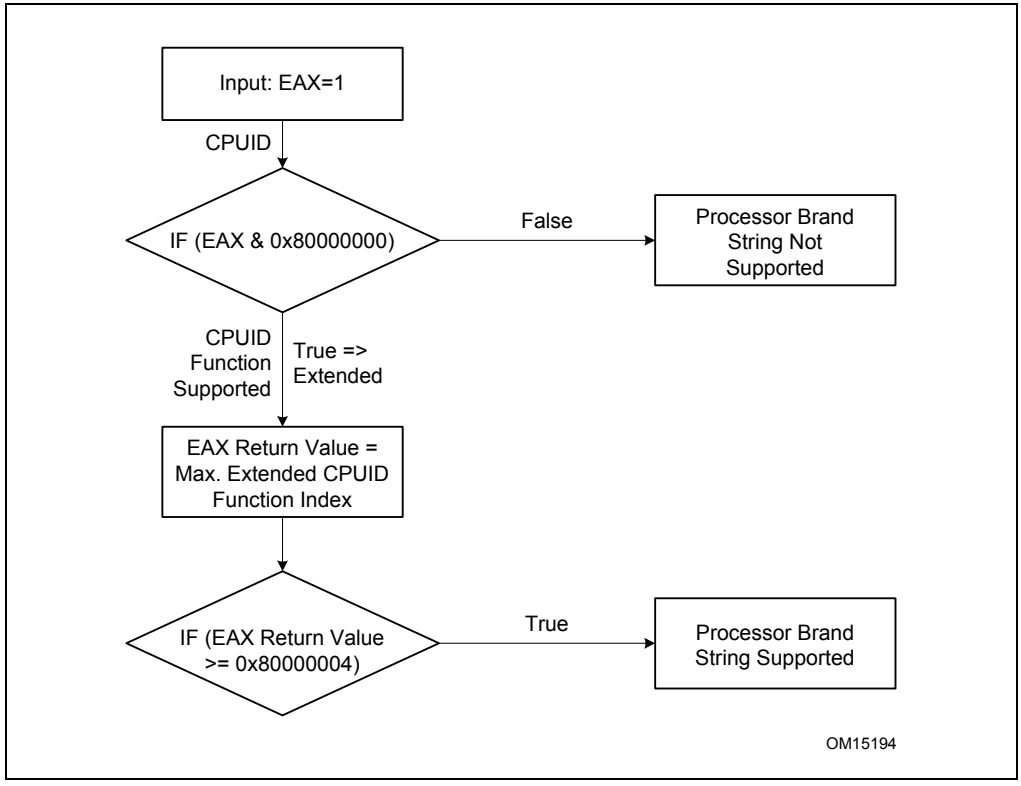


Figure 2-4. Determination of Support for the Processor Brand String

2.1.5. Algorithm for Extracting Maximum Processor Frequency

Figure 2-5 provides an algorithm (for IA-32 architecture based processors) which software can use to extract the maximum processor operating frequency from the processor brand string.

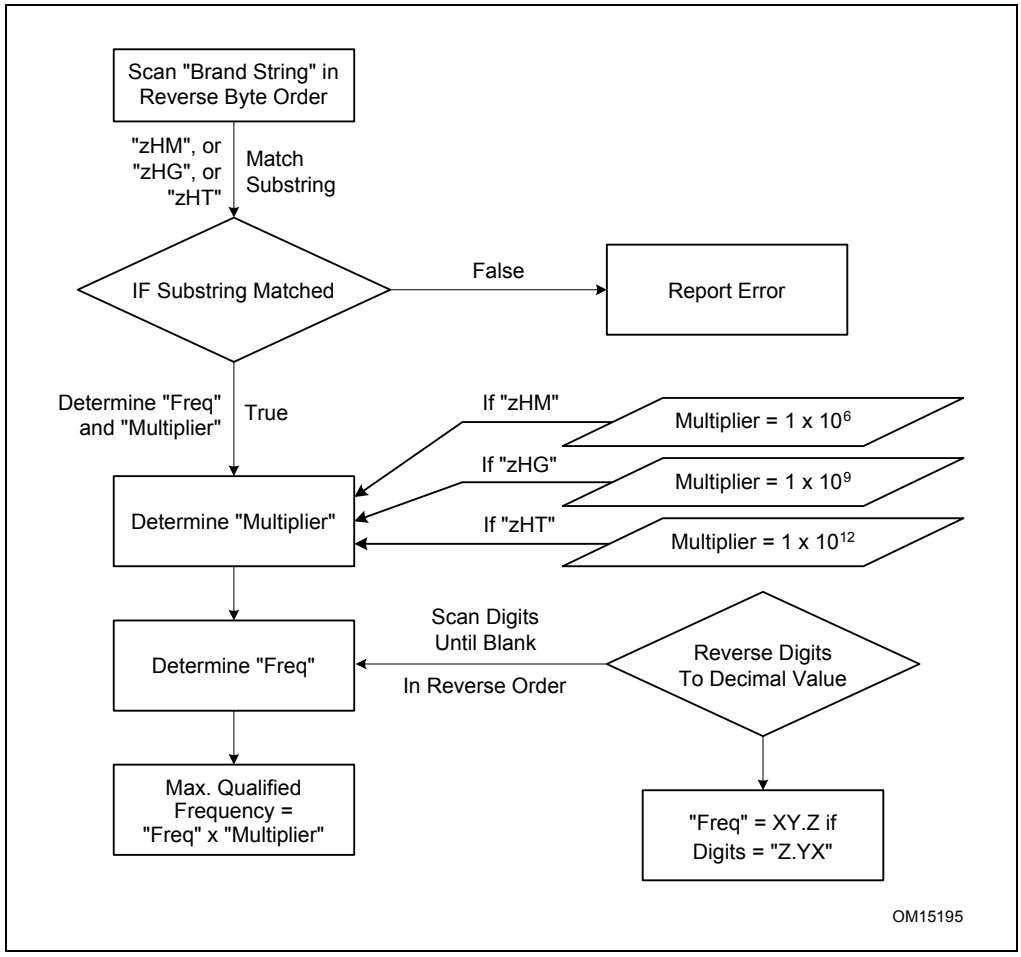


Figure 2-5. Algorithm for Extracting Maximum Processor Frequency

2.1.6. Determining Explicit Cache Descriptors

Use of the cache descriptor based mechanism to determine cache hierarchy information requires the operating system to have knowledge of specific descriptor values and the corresponding cache information. The need for a-priori knowledge limits the ability of deployed operating systems to correctly identify cache sizes and structures on processors not available at the time of their development. To address this limitation, Prescott provides a mechanism for enumerating/calculating details of the cache hierarchy at runtime by using CPUID. The algorithm for extracting cache parameters is as shown in Figure 2-6.

Information about Translation Lookaside Buffers (TLBs) is not provided using this mechanism. See the *IA-32 Intel® Architecture Software Developer's Guide* for more information about TLBs.

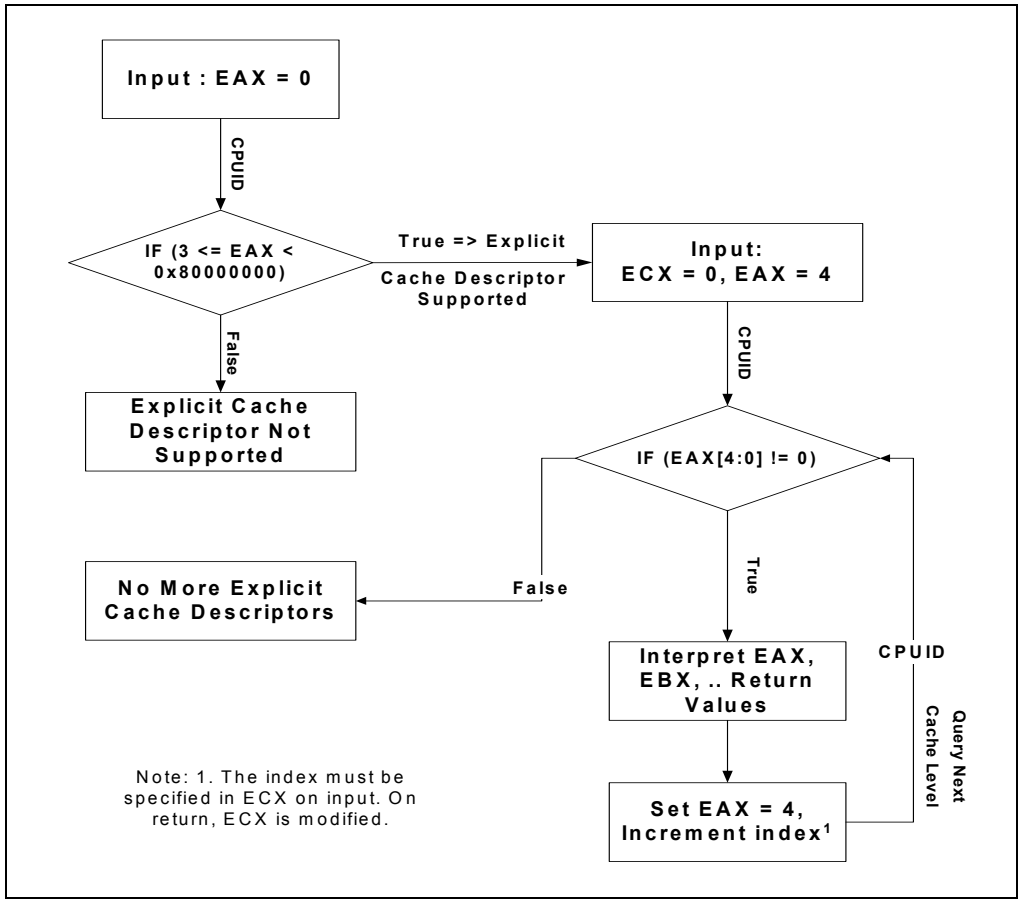


Figure 2-6. Determination of Support for Explicit Cache Descriptors

Prescott cache and TLB descriptor values are provided in Table 2-5 (returned in EAX/EBX/ECX/EDX by CPUID when EAX = 02H; see also Table 2-1). The cache level closest to the processor is referred to as the first-level cache, the next level as the second-level cache, and so forth.

Table 2-5. Prescott Cache and TLB Descriptors

Cache or TLB Descriptor Description	Descriptor value (Hex)
512K Third-Level Cache, 4-way, 64byte line size, 128 byte sector size	22h
1MB Third-Level Cache, 8-way, 64byte line size, 128 byte sector size	23h
2MB Third-Level Cache, 8-way, 64byte line size, 128 byte sector size	25h
4MB Third-Level Cache, 8-way, 64byte line size, 128 byte sector size	29h
No Third-Level Cache	40h
64 entries for 4KB pages & 2MB/4MB pages (ITLB)	50h
128 entries for 4KB pages & 2MB/4MB pages (ITLB)	51h
256 entries for 4KB pages & 2MB/4MB pages (ITLB)	52h
64 entries for 4KB pages & 4MB pages (DTLB)	5Bh
128 entries for 4KB pages & 4MB pages (DTLB)	5Ch
256 entries for 4KB pages & 4MB pages (DTLB)	5Dh
8KB First-Level Data Cache, 4-way set associative, 64 byte line size	66h
16KB First-Level Data Cache, 4-way set associative, 64 byte line size	67h
32KB First-Level Data Cache, 4-way set associative, 64 byte line size	68h
12K uops, Trace Cache, 8-way set associative	70h
16K uops, Trace Cache, 8-way set associative	71h
32K uops, Trace Cache, 8-way set associative	72h
128KB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size	79h
256KB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size	7Ah
512KB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size	7Bh
1MB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size	7Ch

CHAPTER 3
INSTRUCTION SET REFERENCE

3.1. INTERPRETING THE INSTRUCTION REFERENCE PAGES

Prescott New Instructions technology 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).

Besides opcodes, two kinds of notations describe information found 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 (i.e., 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, the following abbreviations are used:

r32	Intel architecture 32-bit integer register
xmm/m128	Indicates a 128-bit FP Streaming SIMD Extensions/Streaming SIMD Extensions 2 register or a 128-bit memory location.
xmm/m64	Indicates a 128-bit FP Streaming SIMD Extensions/Streaming SIMD Extensions 2 register or a 64-bit memory location.
xmm/m32	Indicates a 128-bit FP Streaming SIMD Extensions/Streaming SIMD Extensions 2 register or a 32-bit memory location.
mm/m64	Indicates a 64-bit integer register using MMX™ media enhancement technolgy or a 64-bit memory location.
xmm/m128	Indicates a 128-bit integer register using MMX media enhancement technolgy or a 128-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.

When there is ambiguity, xmm1 indicates the first source operand and xmm2 the second source operand. For more information on notation, refer to the notation section in the *IA-32 Intel® Architecture Software Developer's Manual*, Volume 3.

3.2. PRESCOTT NEW INSTRUCTIONS

This chapter describes the thirteen instructions which form Prescott New Instructions technology. Detailed information follows. Appendix A summarizes the new instructions.

ADDSUBPD: Packed Double-FP Add/Subtract

Opcode	Instruction	Description
66,0F,D0,/r	ADDSUBPD xmm1, xmm2/m128	Add/Subtract packed DP FP numbers from XMM2/Mem to XMM1.

Description

Adds the double-precision floating-point values in the high quadword of the source and destination operands and stores the result in the high quadword of the destination operand.

Subtracts the double-precision floating-point value in the low quadword of the source operand from the low quadword of the destination operand and stores the result in the low quadword of the destination operand.

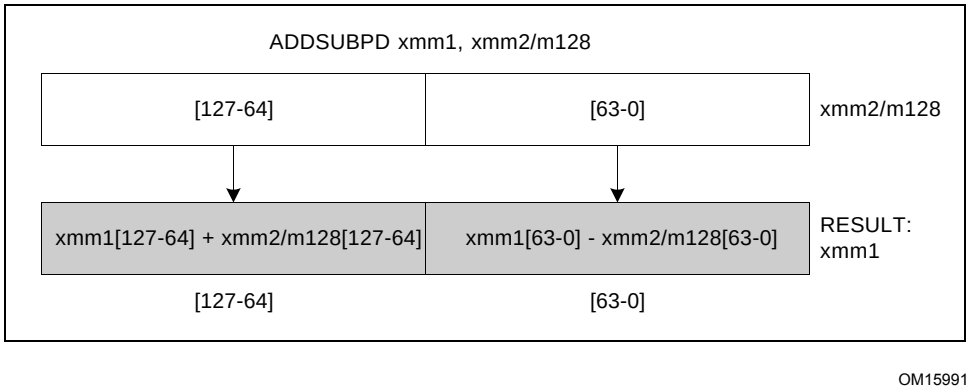


Figure 3-1. ADDSUBPD: Packed Double-FP Add/Subtract

Operation

```
xmm1[63-0]   = xmm1[63-0]   - xmm2/m128[63-0];
xmm1[127-64] = xmm1[127-64] + xmm2/m128[127-64];
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ADDSUBPD __m128d _mm_addsub_pd(__m128d a, __m128d b)
```

ADDSUBPD: Packed Double-FP Add/Subtract (Continued)

Exceptions

When the source operand is a memory operand, it must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

Numeric Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected 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 TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0);
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

ADDSUBPD: Packed Double-FP Add/Subtract (Continued)

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

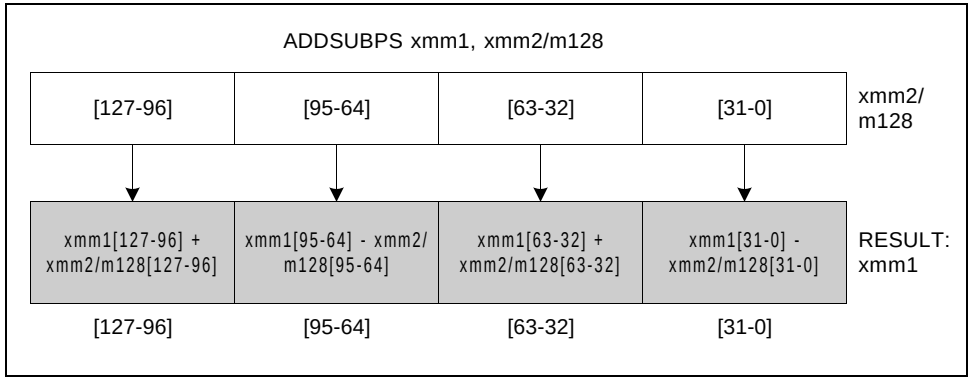
ADDSUBPS: Packed Single-FP Add/Subtract

Opcode	Instruction	Description
F2,0F,D0,/r	ADDSUBPS xmm1, xmm2/m128	Add/Subtract packed SP FP numbers from XMM2/Mem to XMM1.

Description

Adds odd-numbered single-precision floating-point values of the source operand with the corresponding single-precision floating-point values from the destination operand; stores the result in the odd-numbered values of the destination operand.

Subtracts the even-numbered single-precision floating-point values in the source operand from the corresponding single-precision floating values in the destination operand; stores the result into the even-numbered values of the destination operand.



OM15992

Figure 3-2. ADDSUBPS: Packed Single-FP Add/Subtract

Operation

```
xmm1[31-0]   = xmm1[31-0]   - xmm2/m128[31-0];
xmm1[63-32]  = xmm1[63-32]  + xmm2/m128[63-32];
xmm1[95-64]  = xmm1[95-64]  - xmm2/m128[95-64];
xmm1[127-96] = xmm1[127-96] + xmm2/m128[127-96];
```

Intel C/C++ Compiler Intrinsic Equivalent

```
ADDSUBPS __m128 _mm_addsub_ps(__m128 a, __m128 b)
```

ADDSUBPS: Packed Single-FP Add/Subtract (Continued)

Exceptions

When the source operand is a memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

Numeric Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected 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 TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

ADDSUBPS: Packed Single-FP Add/Subtract (Continued)

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

FISTTP: Store Integer with Truncation

Opcode	Instruction	Description
DF /1	FISTTP m16int	Store ST as a signed integer (truncate) in m16int and pop ST.
DB /1	FISTTP m32int	Store ST as a signed integer (truncate) in m32int and pop ST.
DD /1	FISTTP m64int	Store ST as a signed integer (truncate) in m64int and pop ST.

Description

FISTTP converts the value in ST into a signed integer using truncation (chop) as rounding mode, transfers the result to the destination, and pop ST. FISTTP accepts word, short integer, and long integer destinations.

The following table shows the results obtained when storing various classes of numbers in integer format.

ST(0)	DEST
−∞ or Value Too Large for DEST Format	*
F ≤ −1	−I
1 < F < +	0
F +	+I
+ or Value Too Large for DEST Format	*
NaN	*

Notes:

F Means finite floating-point value.

I Means integer.

* Indicates floating-point invalid-operation (#IA) exception.

Operation

```
DEST ← ST;
pop ST;
```

Flags Affected

C1 is cleared; C0, C2, C3 undefined.

Numeric Exceptions

Invalid, Stack Invalid (stack underflow), Precision.

FISTTP: Store Integer with Truncation (Continued)

Protected Mode Exceptions

#GP(0)	If the destination is in a nonwritable segment.
	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.
#AC	For unaligned memory reference if the current privilege level is 3.
#NM	If CR0.EM = 1.
	If TS bit in CR0 is set.
#UD	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.EM = 1.
	If TS bit in CR0 is set.
#UD	If CPUID.PNI(ECX bit 0) = 0.

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If CR0.EM = 1.
	If TS bit in CR0 is set.
#UD	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.
#AC	For unaligned memory reference if the current privilege is 3.

HADDPD: Packed Double-FP Horizontal Add

Opcode	Instruction	Description
66,0F,7C,/r	HADDPD xmm1, xmm2/m128	Add horizontally packed DP FP numbers from XMM2/Mem to XMM1.

Description

Adds the double-precision floating-point values in the high and low quadwords of the destination operand and stores the result in the low quadword of the destination operand.

Adds the double-precision floating-point values in the high and low quadwords of the source operand and stores the result in the high quadword of the destination operand.

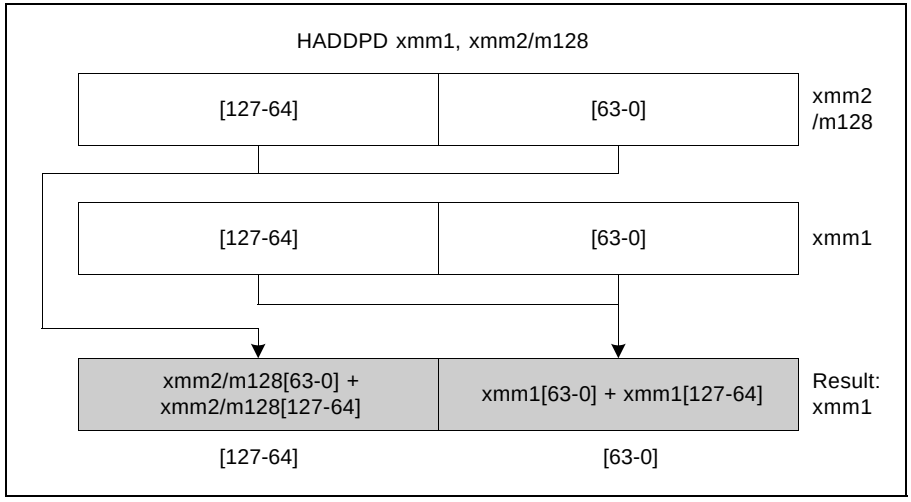


Figure 3-3. HADDPD: Packed Double-FP Horizontal Add

Operation

```
xmm1[63-0]   = xmm1[63-0]      + xmm1[127-64];
xmm1[127-64] = xmm2/m128[63-0] + xmm2/m128[127-64];
```

Intel C/C++ Compiler Intrinsic Equivalent

```
HADDPD __m128d _mm_hadd_pd(__m128d a, __m128d b)
```

HADDPD: Packed Double-FP Horizontal Add (Continued)

Exceptions

When the source operand is a memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

Numeric Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected 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 TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0);
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

HADDPD: Packed Double-FP Horizontal Add (Continued)

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

HADDPS: Packed Single-FP Horizontal Add

Opcode	Instruction	Description
F2,0F,7C,/r	HADDPS xmm1, xmm2/m128	Add horizontally packed SP FP numbers from XMM2/Mem to XMM1.

Description

Adds the single-precision floating-point values in the first and second dwords of the destination operand and stores the result in the first dword of the destination operand.

Adds single-precision floating-point values in the third and fourth dword of the destination operand and stores the result in the second dword of the destination operand.

Adds single-precision floating-point values in the first and second dword of the source operand and stores the result in the third dword of the destination operand.

Adds single-precision floating-point values in the third and fourth dword of the source operand and stores the result in the fourth dword of the destination operand.

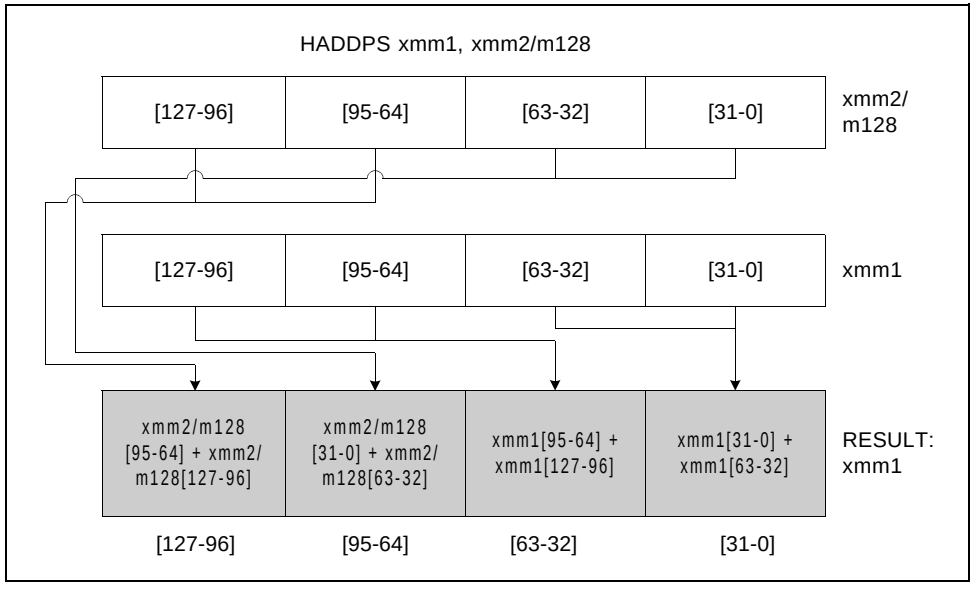


Figure 3-4. HADDPS: Packed Single-FP Horizontal Add

HADDPS: Packed Single-FP Horizontal Add (Continued)

Operation

```
xmm1[31-0] = xmm1[31-0] + xmm1[63-32];
xmm1[63-32] = xmm1[95-64] + xmm1[127-96];
xmm1[95-64] = xmm2/m128[31-0] + xmm2/m128[63-32];
xmm1[127-96] = xmm2/m128[95-64] + xmm2/m128[127-96];
```

Intel C/C++ Compiler Intrinsic Equivalent

```
HADDPS __m128 _mm_hadd_ps(__m128 a, __m128 b)
```

Exceptions

When the source operand is a memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

Numeric Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected 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 TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

HADDPS: Packed Single-FP Horizontal Add (Continued)

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

HSUBPD: Packed Double-FP Horizontal Subtract

Opcode	Instruction	Description
66,0F,7D,/r	HSUBPD xmm1, xmm2/m128	Subtract horizontally packed DP FP numbers in XMM2/Mem from XMM1.

Description

The HSUBPD instruction subtracts horizontally the packed DP FP numbers of both operands.

Subtracts the double-precision floating-point value in the high quadword of the destination operand from the low quadword of the destination operand and stores the result in the low quadword of the destination operand.

Subtracts the double-precision floating-point value in the high quadword of the source operand from the low quadword of the source operand and stores the result in the high quadword of the destination operand.

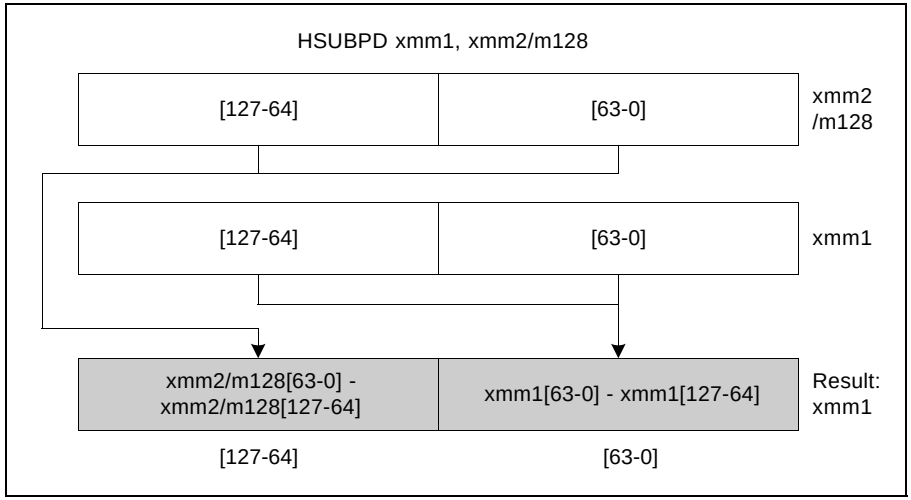


Figure 3-5. HSUBPD: Packed Double-FP Horizontal Subtract

Operation

```
xmm1[63-0]   = xmm1[63-0] - xmm1[127-64];
xmm1[127-64] = xmm2/m128[63-0] - xmm2/m128[127-64];
```

HSUBPD: Packed Double-FP Horizontal Subtract (Continued)

Intel C/C++ Compiler Intrinsic Equivalent

HSUBPD __m128d _mm_hsub_pd(__m128d a, __m128d b)

Exceptions

When the source operand is a memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

Numeric Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected 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 TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).

HSUBPD: Packed Double-FP Horizontal Subtract (Continued)

#UD	If CR0.EM = 1. For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0). If CR4.OSFXSR(bit 9) = 0. If CPUID.PNI(ECX bit 0) = 0.

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1. For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0). If CR4.OSFXSR(bit 9) = 0. If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

HSUBPS: Packed Single-FP Horizontal Subtract

Opcode	Instruction	Description
F2,0F,7D,/r	HSUBPS xmm1, xmm2/m128	Subtract horizontally packed SP FP numbers in XMM2/Mem from XMM1.

Description

Subtracts the single-precision floating-point value in the second dword of the destination operand from the first dword of the destination operand and stores the result in the first dword of the destination operand.

Subtracts the single-precision floating-point value in the fourth dword of the destination operand from the third dword of the destination operand and stores the result in the second dword of the destination operand.

Subtracts the single-precision floating-point value in the second dword of the source operand from the first dword of the source operand and stores the result in the third dword of the destination operand.

Subtracts the single-precision floating-point value in the fourth dword of the source operand from the third dword of the source operand and stores the result in the fourth dword of the destination operand.

HSUBPS: Packed Single-FP Horizontal Subtract (Continued)

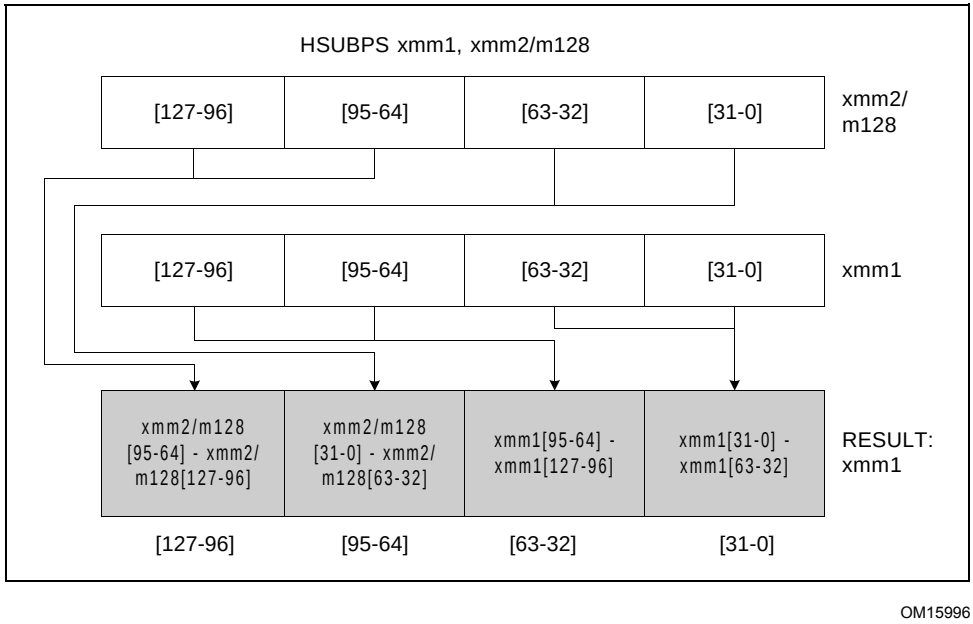


Figure 3-6. HSUBPS: Packed Single-FP Horizontal Subtract

Operation

```
xmm1[31-0] = xmm1[31-0] - xmm1[63-32];
xmm1[63-32] = xmm1[95-64] - xmm1[127-96];
xmm1[95-64] = xmm2/m128[31-0] - xmm2/m128[63-32];
xmm1[127-96] = xmm2/m128[95-64] - xmm2/m128[127-96];
```

Intel C/C++ Compiler Intrinsic Equivalent

```
HSUBPS __m128 _mm_hsub_ps(__m128 a, __m128 b)
```

Exceptions

When the source operand is a memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (#GP) will be generated.

HSUBPS: Packed Single-FP Horizontal Subtract (Continued)

Numeric Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected 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 TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

HSUBPS: Packed Single-FP Horizontal Subtract (Continued)

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 1).
#UD	If CR0.EM = 1.
	For an unmasked Streaming SIMD Extensions numeric exception (CR4.OSXMMEXCPT = 0).
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

LDDQU: Load Unaligned Integer 128 bits

Opcode	Instruction	Description
F2,0F,F0,/r	LDDQU xmm, m128	Load 128 bits from Mem to XMM register.

Description

The instruction is functionally equivalent to MOVDQU xmm, m128. It moves a double quadword from the source memory operand (second operand) to the destination register (first operand). The source operand may be unaligned on a 16-byte boundary without causing a general-protection exception (#GP) to be generated.

This instruction may improve performance relative to MOVDQU if the source operand crosses a cacheline boundary. This instruction may reduce performance if the load requires store to load forwarding. To move a double quadword to or from memory locations that are known to be aligned on 16-byte boundaries, use the MOVDQA instruction.

Implementation Note

Specific processors may implement LDDQU exactly like MOVDQU if the implementation of LDDQU as described above (with two aligned 16-byte memory accesses) would lead to lower performance than MOVDQU.

Operation

```
xmm[127-0] = m128;
```

Intel C/C++ Compiler Intrinsic Equivalent

```
HADDPS __m128i _mm_lddqu_si128(__m128i const *p)
```

Numeric Exceptions

None

Protected 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; #UD if CR0.EM = 1.
#NM	If TS bit in CR0 is set; #UD if CR4.OSFXSR(bit 9) = 0.
#UD	If CPUID.PNI(ECX bit 0) = 0.

LDDQU: Load Unaligned Integer 128 bits (Continued)

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

Comments

This instruction may emulate an unaligned memory access with two 128-bit aligned loads (for example, loading 32 bytes) if such an implementation gains a performance advantage relative to loading 16 bytes with MOVDQU.

If the source is aligned to a 16-byte boundary, based on the implementation, the 16 bytes may be loaded more than once. For that reason, the usage of LDDQU should be avoided when using uncached or write-combining (WC) memory regions. For uncached or WC memory regions, keep using MOVDQU.

This instruction is a replacement for MOVDQU (load) in situations where cache line splits significantly affect performance. It should not be used in situations where store-load forwarding is performance critical. If performance of store-load forwarding is critical to the application, use MOVDQA store-load pairs when data is 128-bit aligned or MOVDQU store-load pairs when data is 128-bit unaligned.

LDDQU behavior for offsets that are within 15 bytes of an unaligned (specifically 16-byte unaligned) base or limit is implementation specific; this situation may signal a #GP Exception. The bytes accessed may be different based on the segment definition (base, limit value). LDDQU behavior in 16b(32b) addressing mode, when the data access crosses a 16b(32b) boundary, is also implementation specific and may or may not signal a #GP Exception. The bytes accessed may be different based on the segment definition (base, limit value).

MONITOR: Setup Monitor Address

Opcode	Instruction	Description
0F,01,C8	MONITOR EAX, ECX, EDX	Sets up a linear address range to be monitored by hardware and activates the monitor. The address range should be of a write-back memory caching type.

Description

The MONITOR instruction relies on a state in the processor called the monitor event pending flag. The monitor event pending flag is either set or clear; its value is not architecturally visible except through the behavior of the MWAIT instruction. The monitor event pending flag is set by multiple events including a write to the address range being monitored and reset by the MONITOR instruction.

The MONITOR instruction sets up the address monitoring hardware using the address specified in EAX and resets the monitor event pending flag. The address range that the hardware checks for stores can be determined by calling CPUID. Additional details for determining setup of this address range to prevent false wake ups are provided in The monitoring hardware detects stores to an address within the address range and sets the monitor event pending flag when the write is detected. The state of the monitor event pending flag is used by the MWAIT instruction. Other events will also set the monitor event pending flag.

The address range must be in memory of write-back caching type. Only write-back memory type stores to the monitored address range set the monitor event pending flag.

The content of EAX is a logical address. By default, the DS segment is used to create a linear address that is then monitored. Segment overrides can be used with the MONITOR instruction. ECX and EDX are used to communicate other information to the MONITOR instruction. ECX specifies optional extensions for the MONITOR instruction. EDX may contain hints and will not change the architectural behavior of the instruction. For Prescott, no extensions or hints are defined and the value for ECX and EDX should be zero. Non-zero values for ECX and EDX are reserved and the processor will raise a general protection fault exception on the execution of the MONITOR instruction with reserved values in ECX. The processor ignores setting of reserved bits in EDX.

The MONITOR instruction is ordered as a load operation with respect to other memory transactions. The instruction can be used at all privilege levels and is subject to all permission checking and faults associated with a byte load. Like a load, the MONITOR instruction sets the A-bit but not the D-bit in page tables.

The MONITOR CPUID feature flag (bit 3 of ECX when CPUID is executed with EAX=1) indicates that a processor supports this instruction. The operating system or system BIOS may disable this instruction through the IA32_MISC_ENABLES MSR; disabling the instruction clears the CPUID feature flag and causes execution of the MONITOR instruction to generate an illegal opcode exception.

See also: Section 4.3.6, *Guideline for MONITOR/MWAIT.*

MONITOR: Setup Monitor Address (Continued)

Operation

MONITOR sets up an address range for the monitor hardware using the content of EAX as a logical address and resets the monitor event pending flag. The memory address range should be within memory of the write-back caching type. A store to the specified address range will set the monitor event pending flag.

The content of ECX and EDX are used to communicate other information to the MONITOR instruction.

Intel C/C++ Compiler Intrinsic Equivalent

```
MONITOR void _mm_monitor(void const *p, unsigned extensions, unsigned
hints)
```

Exceptions

None

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
#GP(0)	For ECX has a value other than 0.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault (TBD).
#UD	If CPUID feature flag MONITOR is 0.
	If LOCK, REP, REPNE/NZ and Operand Size override prefixes are used.

Real Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#GP(0)	For ECX has a value other than 0.
#UD	If CPUID feature flag MONITOR is 0.
	If LOCK, REP, REPNE/NZ and Operand Size override prefixes are used.

MONITOR: Setup Monitor Address (Continued)

Virtual 8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#GP(0)	For ECX has a value other than 0; #UD if CPUID feature flag MONITOR is 0.
#UD	If LOCK, REP, REPNE/NZ and Operand Size override prefixes are used.
#PF(fault-code)	For a page fault.

MOVDDUP: Move One Double-FP and Duplicate

Opcode	Instruction	Description
F2,0F,12,/r	MOVDDUP xmm1, xmm2/m64	Move 64 bits representing the lower DP data element from XMM2/Mem to XMM1 register and duplicate.

Description

The linear address corresponds to the address of the least-significant byte of the referenced memory data. When a memory address is indicated, the 8 bytes of data at memory location m64 are loaded. When the register-register form of this operation is used, the lower half of the 128-bit source register is duplicated and copied into the 128-bit destination register.

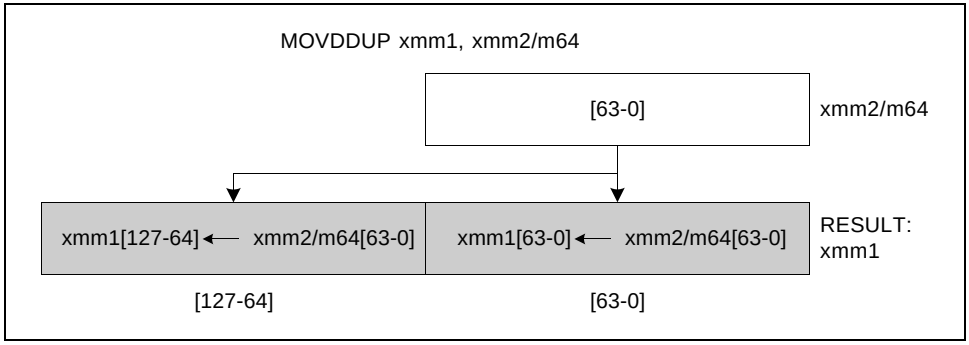


Figure 3-7. MOVDDUP: Move One Double-FP and Duplicate

Operation

```
if (source == m64) {
     // load instruction
     xmm1[63-0] = m64;
     xmm1[127-64] = m64;
}
else {
     // move instruction
     xmm1[63-0]   = xmm2[63-0];
     xmm1[127-64] = xmm2[63-0];
}
```

MOVDDUP: Move One Double-FP and Duplicate (Continued)

Intel C/C++ Compiler Intrinsic Equivalent

```
MOVDDUP    __m128d _mm_movedup_pd(__m128d a)
           __m128d _mm_loaddup_pd(double const * dp)
```

Exceptions

None

Numeric Exceptions

None

Protected 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 TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	I CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

MOVDDUP: Move One Double-FP and Duplicate (Continued)

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

MOVSHDUP: Move Packed Single-FP High and Duplicate

Opcode	Instruction	Description
F3,0F,16,/r	MOVSHDUP xmm1, xmm2/m128	Move 128 bits representing packed SP data elements from XMM2/Mem to XMM1 register and duplicate high.

Description

The linear address corresponds to the address of the least-significant byte of the referenced memory data. When a memory address is indicated, the 16 bytes of data at memory location m128 are loaded and the single-precision elements in positions 1 and 3 are duplicated. When the register-register form of this operation is used, the same operation is performed but with data coming from the 128-bit source register.

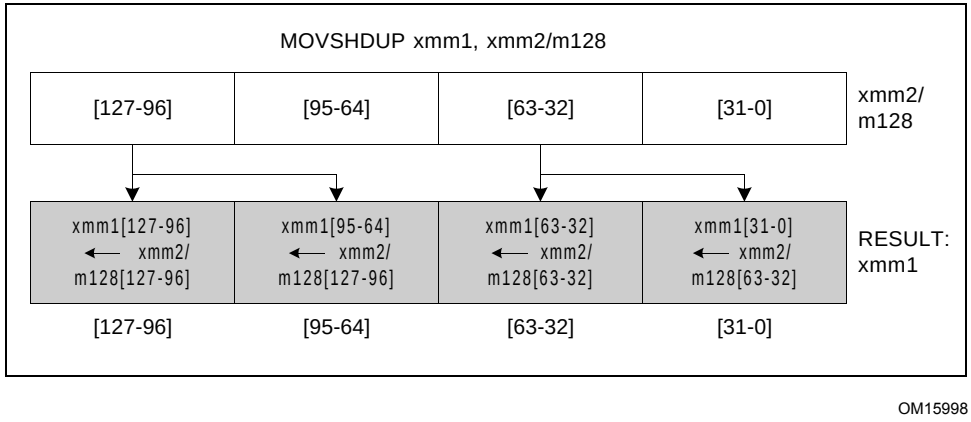


Figure 3-8. MOVSHDUP: Move Packed Single-FP High and Duplicate

MOVSHDUP: Move Packed Single-FP High and Duplicate (Continued)

Operation

```
if (source == m128) {
     // load instruction
     xmm1[31-0] = m128[63-32];
     xmm1[63-32] = m128[63-32]
     xmm1[95-64]  = m128[127-96];
     xmm1[127-96] = m128[127-96];
}
else {
     // move instruction
     xmm1[31-0] = xmm2[63-32];
     xmm1[63-32] = xmm2[63-32];
     xmm1[95-64] = xmm2[127-96];
     xmm1[127-96] = xmm2[127-96];
}
```

Intel C/C++ Compiler Intrinsic Equivalent

```
MOVSHDUP __m128 _mm_movehdup_ps(__m128 a)
```

Exceptions

General protection exception if not aligned on 16-byte boundary, regardless of segment.

Numeric Exceptions

None

Protected 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 TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

MOVSHDUP: Move Packed Single-FP High and Duplicate (Continued)

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

MOVSLDUP: Move Packed Single-FP Low and Duplicate

Opcode	Instruction	Description
F3,0F,12,/r	MOVSLDUP xmm1, xmm2/m128	Move 128 bits representing packed SP data elements from XMM2/Mem to XMM1 register and duplicate low.

Description

The linear address corresponds to the address of the least-significant byte of the referenced memory data. When a memory address is indicated, the 16 bytes of data at memory location m128 are loaded and the single-precision elements in positions 0 and 2 are duplicated. When the register-register form of this operation is used, the same operation is performed but with data coming from the 128-bit source register.

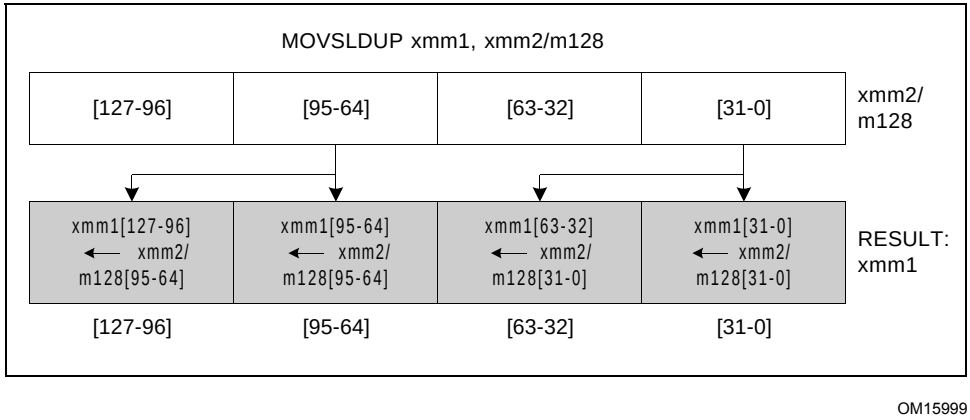


Figure 3-9. MOVSLDUP: Move Packed Single-FP Low and Duplicate

Operation

```
if (source == m128) {
     // load instruction
     xmm1[31-0] = m128[31-0];
     xmm1[63-32] = m128[31-0]
     xmm1[95-64] = m128[95-64];
     xmm1[127-96] = m128[95-64];
}
else {
     // move instruction
```

MOVSLDUP: Move Packed Single-FP Low and Duplicate (Continued)

```
      xmm1[31-0]   = xmm2[31-0];
      xmm1[63-32]  = xmm2[31-0];
      xmm1[95-64]  = xmm2[95-64];
      xmm1[127-96] = xmm2[95-64];
}
```

Intel C/C++Compiler Intrinsic Equivalent

```
MOVSLDUP __m128 _mm_moveldup_ps(__m128 a)
```

Exceptions

General protection exception if not aligned on 16-byte boundary, regardless of segment.

Numeric Exceptions

None

Protected 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 TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

Real Address Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.

MOVSLDUP: Move Packed Single-FP Low and Duplicate (Continued)

Virtual 8086 Mode Exceptions

Interrupt 13	If any part of the operand would lie outside of the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.PNI(ECX bit 0) = 0.
#PF(fault-code)	For a page fault.

MWAIT: Monitor Wait

Opcode	Instruction	Description
0F,01,C9	MWAIT EAX, ECX	A hint that allows the processor to stop instruction execution and enter an implementation-dependent optimized state until occurrence of a class of events.

Description

The MWAIT instruction is designed to operate with the MONITOR instruction to allow a processor to signal an address on which to wait (MONITOR) and an instruction that causes the wait operation to commence (MWAIT). The MWAIT instruction is also a hint to the processor that it can choose to enter an implementation-dependent state while waiting for an event or for the store to the address range set up by the preceding MONITOR instruction in program flow.

A given implementation may choose to ignore the hint and continue executing the next instruction. Future processor implementations may implement several optimized wait states and will select among those states based on the hint argument.

In future processor designs, EAX and ECX will be used to communicate other information to the MWAIT instruction, such as the kind of optimized state the processor should enter. ECX specifies optional extensions for the MWAIT instruction. EAX may contain hints such as the preferred optimized state the processor should enter. For Prescott, all non-zero values for EAX and ECX are reserved. The processor will raise a general protection fault on the execution of the MWAIT instruction with reserved values in ECX. The processor ignores setting of reserved bits in EDX.

A store to the address range set by the MONITOR instruction, an interrupt, NMI, SMI, a debug exception, a machine check exception, the BINIT# signal, the INIT# signal, or the RESET# signal will exit the optimized state. Note that an interrupt will cause the processor to exit the optimized state only if the state was entered with interrupts enabled. If a store to the address range caused the processor to exit then execution will resume at the instruction following the MWAIT instruction. If an interrupt (including NMI) caused the processor to exit the optimized state, the processor will exit the optimized state and handle the interrupt. If an SMI caused the processor to exit the optimized state, execution will resume at the instruction following the MWAIT after handling of the SMI. Unlike the HALT instruction, the MWAIT instruction does not support a restart at the MWAIT instruction. There may also be other implementation-dependent events or time-outs that may take the processor out of the optimized state and resume execution at the instruction following the MWAIT.

If the preceding MONITOR instruction did not successfully set an address range or if the MONITOR instruction has not been executed prior to executing MWAIT, then the processor will not enter the optimized state. Execution will resume at the instruction following the MWAIT.

The MWAIT instruction can be executed at any privilege level. The MONITOR CPUID feature flag (bit 3 of ECX when CPUID is executed with EAX=1) indicates that a processor supports this instruction. The operating system or system BIOS may disable this instruction through the IA32_MISC_ENABLES MSR; disabling the instruction clears the CPUID feature flag and causes execution of the MWAIT instruction to generate an illegal opcode exception

MWAIT: Monitor Wait (Continued)

Operation

```
// MWAIT takes the argument inEAX as a hint extension and is
// architected to take the argument in ECX as an instruction extension
// MWAIT EAX, ECX
{
WHILE (! ("monitor_event_pending_flag" OR "monitor_not_active")) {
     implementation_dependent_optimized_state(EAX, ECX);
}
Clear monitor_event_pending_flag;
}
```

Intel C/C++ Compiler Intrinsic Equivalent

```
MWAIT void _mm_mwait(unsigned extensions, unsigned hints)
```

Example

The MONITOR and MWAIT instructions must be coded in the same loop because execution of the MWAIT instruction will clear the monitor address range. It is not possible to execute MONITOR once and then execute MWAIT in a loop. Setting up MONITOR without executing MWAIT has no adverse effects.

Assume that there is a globally shared volatile integer trigger, that is written to zero (0x0) before the processor enters MWAIT, and written to one (0x1) by the other thread/processor that wants to wake this one up.

```
Trigger = 0;
If ( !trigger) {
     EAX = &trigger
     ECX = 0
     EDX = 0
     MONITOR EAX, ECX, EDX
     If ( !trigger ) {
          EAX = 0
          ECX = 0
          MWAIT EAX, ECX
     }
}
```

The above code sequence makes sure that a triggering store does not happen between the first check of the trigger and the execution of the monitor instruction. Without the second check that triggering store would go un-noticed. Typical usage of MONITOR and MWAIT would have the above code sequence within a loop.

MWAIT: Monitor Wait (Continued)

Exceptions

None

Numeric Exceptions

None

Protected Mode Exceptions

#GP(0)	For ECX has a value other than 0.
#UD	If CPUID feature flag MONITOR is 0.
	If LOCK prefix is used.

Real Address Mode Exceptions

#GP(0)	For ECX has a value other than 0.
#UD	If CPUID feature flag MONITOR is 0.
	If LOCK prefix is used.

Virtual 8086 Mode Exceptions

#GP(0)	For ECX has a value other than 0.
#UD	If CPUID feature flag MONITOR is 0.
	If LOCK prefix is used.

CHAPTER 4
SYSTEM AND APPLICATION PROGRAMMING GUIDELINES

4.1. SYSTEM PROGRAMMING MODEL AND REQUIREMENTS

The Prescott New Instructions state requires no new OS support for saving and restoring the new state during a context switch, beyond that provided for Streaming SIMD Extensions (SSE). The operating system or executive must provide support for initializing the processor to use Prescott New Instruction extensions, for handling the FXSAVE and FXRSTOR state saving instructions, and for handling SIMD floating-point exceptions. The following sections provide guidelines for providing this support.

4.1.1. Enabling Prescott New Instructions Support in a System Executive

Eleven of the thirteen new instructions are extensions to Streaming SIMD Extensions and Streaming SIMD Extensions 2 technologies. Providing operating system or executive support for Prescott New Instructions technology is similar to the steps described in *General Guidelines for Adding Support to an Operating System for the SSE and SSE2 Extensions*, Chapter 12, Vol. 3: *IA-32 Intel Software Developer's Manual*. The steps are:

1. Check that the processor supports SSE, SSE2 and Prescott New Instruction extensions.
2. Check that the processor supports FXSAVE and FXRESTOR.
3. Provide a procedure that initializes the SSE and SSE2 state.
4. Provide support for FXSAVE and FXRSTOR.
5. Provide support (if necessary) in non-numeric exception handlers for exceptions generated by SSE, SSE2 and Prescott New Instructions.
6. Provide a handler for the SIMD floating-point exception (#XF).

See also: Section 4.2.2., *Detecting Prescott New Instructions Technology Using CPUID*.

4.1.2. FXSAVE/FXRSTOR Replaces Use of FSAVE/FRSTOR

The FSAVE instruction does not save the new state associated with Streaming SIMD Extensions/Streaming SIMD Extensions 2.

FSAVE/FRSTOR should be replaced with FXSAVE/FXRSTOR; the new instructions saves and restore 128-bit registers. EXAMPLE: Exception handlers that use 64-bit integer MMX tech-

nology or x87-FP operations are a case where FSAVE/FRSTOR should be replaced by FXSAVE/FXRSTOR.

4.1.3. Initialization

The steps required for a system executive to initialize support for Prescott New Instructions are the same as the initialization steps required to support SSE and SSE2 extensions. See *Initialization of the SSE and SSE2 Extensions* in Chapter 12, Vol. 3: *IA-32 Intel Architecture Software Developer's Manual*.

4.1.4. Exception Handler

Prescott New Instructions do not introduce new exception types.

4.1.4.1. DEVICE NOT AVAILABLE (DNA) EXCEPTIONS

Prescott New Instructions will cause a DNA Exception (#NM) if the processor attempts to execute a Prescott New Instruction while CR0.TS is set. If CPUID.PNI is clear, execution of any Prescott New Instructions will cause an invalid opcode fault regardless of the state of CR0.TS.

4.1.4.2. NUMERIC ERROR FLAG AND IGNNE#

Most of the Prescott New Instructions ignore CR0.NE (treats it as if it were always set) and the IGNNE# pin. They uses the vector 19 software exception for error reporting. The exception is FISTTP. This instruction behaves like other x87-FP instructions.

4.1.4.3. TECHNOLOGY EMULATION

The CR0.EM bit used to emulate floating-point instructions cannot be used in the same way for MMX technology emulation. If a Prescott New Instructions execute when the CR0.EM bit is set, an Invalid Opcode exception (Int 6) is generated instead of a Device Not Available exception (Int 7).

4.1.5. Detecting Support for MONITOR/MWAIT

To use the MONITOR/MWAIT instruction, system software must detect support for these instructions using the CPUID instruction. The extended feature flag bit 3 [CPUID Function 01, ECX:3] indicates support for the MONITOR/MWAIT instructions. Software must also query CPUID's MONITOR/MWAIT leaf to obtain the monitor line size information.

4.2. APPLICATION PROGRAMMING MODEL

The application programming environment for using Prescott New Instructions is unchanged from that provided for Streaming SIMD Extensions and Streaming SIMD Extensions 2.

4.2.1. Detecting Support for MONITOR/MWAIT Instructions

Support for MONITOR/MWAIT can be detected by the Monitor bit in the CPUID extended feature flags. In Prescott implementation, the MONITOR/MWAIT instructions are targeted for use by system software to support efficient thread synchronization. While application software may attempt to use MONITOR/MWAIT, both instructions may be explicitly disabled either by the OS or the BIOS. Disabling the instructions will clear the CPUID feature flag; this also causes MWAIT execution to generate an illegal opcode exception.

4.2.2. Detecting Prescott New Instructions Technology Using CPUID

If an application attempts to use Prescott New Instructions and the processor is not capable of using the new instructions, an Interrupt 6 is generated. To use Prescott New Instructions, the following conditions must exist:

- CR0.EM = 0 (emulation disabled)
- CR4.OSFXSR = 1 (OS supports saving Streaming SIMD Extensions/Streaming SIMD Extensions 2 state during context switches)
- CPUID.PNI = 1 (processor supports Prescott New Instructions technology)

An application can verify that Prescott New Instructions are supported by using this code sequence:

```
boolean prescott_new_instructions_work = TRUE;
try {
        IssuePrescottNewInstructions();
        // Use ADDSUBPD
} except (UNWIND) {
        // if we get here, Prescott New Instructions doesn't work
        prescott_new_instructions_work = FALSE;
```

4.3. GUIDELINES FOR PRESCOTT NEW INSTRUCTIONS

4.3.1. Guideline for Data Movement Instructions

The MOVSHDUP and MOVSLDUP instructions require the source memory operand to be aligned to 16-byte boundary. MOVDDUP and LDDQU do not require the source memory operand to be 16-byte aligned.

The results of MOVSHDUP, MOVSLDUP, and MOVDDUP instructions are typed. The first two instructions should only be used with SSE single-precision floating point computations. The result of MOVDDUP instruction should only be used with SSE2 double-precision floating point computations. The result of LDDQU instruction is also typed, it should be used with SIMD packed integer instructions.

4.3.2. Guideline for Packed ADDSUBxx Instructions

Double-precision and single-precision packed ADDSUBxx instructions are designed to support complex arithmetic computations. These instructions can be used with arrays of complex data types declared to be a structure of a real and imaginary numbers. Example 4-1 shows two code samples: (a) multiplies two pairs of single-precision, complex values, (b) calculates the division of two pairs of single-precision, complex values.

Double-precision complex multiplication and division can be calculated one pair at a time in a similar fashion. When evaluating more sophisticated expressions involving complex values, such as fractions with complex multiplications, evaluate the expression to favor multiplications and reduce the number of divisions.

Example 4-1. Sample Code for Complex Multiplication and Complex Divisions

```
(A) Product of two pair of complex data (a_k + i b_k ) * (c_k + i d_k )
  movsldup xmm0, Src1       ; load real parts into the
                            ; destination, a1, a1, a0, a0
  movaps   xmm1, src2       ; load the 2nd pair of complex
                            ; values, i.e. d1, c1, d0, c0
  mulps    xmm0, xmm1       ; temporary results, a1d1, a1c1,
                            ; a0d0, a0c0
  shufps   xmm1, xmm1, b1   ; reorder the real and imaginary
                            ; parts, c1, d1, c0, d0
  movshdup xmm2, Src1       ; load the imaginary parts into the
                            ; destination, b1, b1, b0, b0
  mulps    xmm2, xmm1       ; temporary results, b1c1, b1d1,
```

```
                              ; b0c0, b0d0
  addsubps xmm0, xmm2         ; b1c1+a1d1, a1c1 -b1d1, b0c0+a0d0,
                              ; a0c0-b0d0
(B)Division of two pair of complex data (ak + i bk ) / (ck + i dk )
  movshdup xmm0, Src1         ; load imaginary parts into the
                              : destination, b1, b1, b0, b0
  movaps   xmm1, src2         ; load the 2nd pair of complex
                              ; values, i.e. d1, c1, d0, c0
  mulps    xmm0, xmm1         ; temporary results, b1d1, b1c1,
                              ; b0d0, b0c0
  shufps   xmm1, xmm1, b1     ; reorder the real and imaginary parts,
                              ; c1, d1, c0, d0
  movsldup xmm2, Src1         ; load the real parts into the
                              ; destination, a1, a1, a0, a0
  mulps    xmm2, xmm1         ; temporary results, a1c1, a1d1, a0c0, a0d0
  addsubps xmm0, xmm2         ; a1c1+b1d1, b1c1-a1d1, a0c0+b0d0, b0c0-a0d0
  mulps    xmm1, xmm1         ; c1c1, d1d1, c0c0, d0d0
  movps    xmm2, xmm1         ; c1c1, d1d1, c0c0, d0d0
  shufps   xmm2, xmm2, b1     ; d1d1, c1c1, d0d0, c0c0
  addps    xmm2, xmm1         ; c1c1+d1d1, c1c1+d1d1, c0c0+d0d0, c0c0+d0d0
  divps    xmm0, xmm2
  shufps   xmm0, xmm0, b1     ; (b1c1-a1d1)/(c1c1+d1d1), (a1c1+b1d1)/
                              ; (c1c1+d1d1), (b0c0-a0d0)/( c0c0+d0d0),
                              ; (a0c0+b0d0)/( c0c0+d0d0)
```

4.3.3. Guideline for FISTTP

The FISTTP instruction provides a quick way to truncate a floating-point value on the x87 stack to a signed integer, pop the stack and store the result in a memory destination. The behavior of FISTTP is identical to FISTP, except FISTTP does not require modification to the floating-point control word (FCW) to change the rounding mode. FISTTP is available in three precisions depending on the size of the destination operand: short integer (word or 16-bit), integer (double word or 32-bit), and long integer (64-bit).

Using FISTTP improves the performance of x87 code. It saves the extra code needed to maintain the current value of the FCW, to change to a new value appropriate to the operand size, and to write the new value back.

Example 4-1 compares the code that a compiler might generate for a simple C statement that cast a floating-point value to integer. Example 4-2 shows the assembly code that a compiler supporting Prescott New Instructions might generate for the same C statement.

Example 4-1. Converting a Floating-point Value to Integer without FISTTP

```
// Compiler output without Precott New Instructions
// for ivalue = (int) fvalue;
    fld    DWORD PTR [ebp-20]         ;Load fvalue from memory
    fnstcw [ebp-12]                   ;save a copy of current FCW
    mov    DWORD PTR [ebp-8], eax     ;Save the content of eax
    movzx  eax, WORD PTR [ebp-12]     ;Load FCW value for change
    or     eax, 3072                  ;Modify to desired rounding mode
    mov    DWORD PTR [ebp-4], eax     ;Prepare new value to
                                      ;write to FCW
    mov    eax, DWORD PTR [ebp-8]     ;Restore eax to its original
    fldcw  [ebp-4]                    ;Write new value to FCW
    fistp  DWORD PTR [ebp-16]         ;Convert fvalue to integer
                                      ;and pop stack
    fldcw  [ebp-12]                   ;Restore FCW to its
                                      ;original state
```

Example 4-2. Using FISTTP to Convert a Floating-point Value to an Integer

```
// Converting floating-point value to integer with
// Prescott New Instructions:
// ivalue = (int) fvalue;
    fld    DWORD PTR [ebp-20]  ;Load fvalue from memory
    fistp  DWORD PTR [ebp-16]  ;Convert fvalue to integer
                               ;and pop stack
```

4.3.4. Guideline for Unaligned 128-bit Load

The Streaming SIMD Extensions (SSE) provides the MOVDQU instruction for loading memory from addresses that are not aligned on 16-byte boundaries. Code sequences that use MOVDQU frequently encounter situations where the source spans across a 64-byte boundary (or cache line boundary). Loading from a memory address that span across a cache line boundary causes a hardware stall and degrades software performance.

LDDQU is a special 128-bit unaligned load designed to avoid cache line splits. If the address of the load is aligned on a 16-byte boundary, LDQQU loads the 16 bytes requested. If the address of the load is not aligned on a 16-byte boundary, LDDQU loads a 32-byte block starting at the 16-byte aligned address immediately below the address of the load request. It then provides the requested 16 bytes. If the address is aligned on a 16-byte boundary, the effective number of memory requests is implementation dependent (one, or more). Because LDDQU usually

accesses more data than is needed (32 bytes when 16 are needed) and because the number of memory accesses is implementation dependent, great care must be taken when dealing with uncached or write-combining (WC) memory regions.

LDDQU is a typed instruction for integer data, it is best used with integer data. Because of implementation issues, restrict the usage of LDDQU to situations where no store-to-load forwarding is expected. Restrict the usage of LDDQU to situations where no store-to-load forwarding is expected. For situations where store-to-load forwarding is expected, use regular store/load pairs (either aligned or unaligned based on the alignment of the data accessed).

4.3.5. Guideline for Horizontal Add/Subtract

Most SIMD instructions operate vertically. Data element of the result in position k are a function of data elements in position k the instructions operands. Horizontal instructions operate differently. Contiguous data elements from the same operand are used to produce the result.

Packed horizontal add instructions can be useful to evaluate dot products, matrix multiplications, and facilitate some SIMD computation operating on vectors that are arranged in arrays of structures. Example 4-1 demonstrates computing the dot product of a four component vector, and can be adapted and extended to compute matrix multiplication of 4x4 matrix.

Example 4-1. Using Horizontal Add to Compute Dot Products

```
// An example that computes a four component dot product and
// broadcasts the result which is stored in xmm0.
          movaps xmm0, Vector1
          movaps xmm1, Vector2
          mulps xmm0, xmm1
          haddps xmm0, xmm0
          haddps xmm0, xmm0
// An example that computes two four component
// dot product from 4 vectors.
          movaps xmm0, Vector1
          movaps xmm1, Vector2
          movaps xmm2, Vector3
          movaps xmm3, Vector4
          mulps xmm0, xmm1
          mulps xmm2, xmm3
          haddps xmm0, xmm2
          haddps xmm0, xmm0
```

4.3.6. Guideline for MONITOR/MWAIT

MONITOR and MWAIT are provided to improve synchronization between multiple agents. In the Prescott implementation, MONITOR/MWAIT are targeted for use by system software to provide more efficient thread synchronization primitives. MONITOR defines an address range used to monitor write-back stores. MWAIT is used to indicate that the software thread is waiting for a write-back store to the address range defined by the MONITOR instruction.

4.3.6.1. MONITOR/MWAIT ADDRESS RANGE DETERMINATION

Software should know the exact length of the region that will be monitored for writes by the MONITOR/MWAIT. Allocating and using a region smaller in length than the triggering area for the processor could lead to false wake-ups (resulting from writes to data variables that are incorrectly located in the triggering area). Conversely, allocating a region greater in length than the triggering area could lead to the processor not waking appropriately. CPUID allows for the determination of the exact length of the triggering area. This length has no relationship to any cache line size in the system and software should not make any assumptions to that effect. Based on the size provided by CPUID, the OS/software should dynamically allocate structures with appropriate padding. If correct allocation causes issues, choose to not use MONTIOR/MWAIT.

While a single length is expected to suffice for single cluster based systems, setting up the data layout for systems with multiple clusters is expected to be more complicated. Depending on the mechanism implemented by the chipset in such a system, a single monitor-line size may not suffice.

Typically software will have a set of data variables that it monitors for writes. It will be necessary to locate these in the monitor triggering area. To eliminate false wake-ups due to writes to other variables, software will need to add padding around the monitored variables. This is referred to as the padded area.

See also: Chapter 2, *CPUID Extensions* and Chapter 4.2.2., *Detecting Prescott New Instructions Technology Using CPUID*.

4.3.6.2. WAKING-UP FROM MWAIT

Multiple events other than a write to the triggering address range can cause a processor that executed MWAIT to wake up. These include:

- External interrupts: NMI, SMI, INIT, BINIT, MCERR
- Faults, Aborts including Machine Check
- Architectural TLB invalidations, including writes to CR0, CR3, CR4 and certain MSR writes
- Voluntary transitions due to fast system call and far calls

Power management related events such as Thermal Monitor, Enhanced Intel SpeedStep® technology transitions or chipset driven STP-CLK# assertion will not cause the Monitor event pending bit to be cleared (Chapter 3, *Instruction Set Reference*: “MONITOR: Setup Monitor Address”). Debug traps and faults will not cause the Monitor event pending bit to be cleared.

Example 4-1 below shows the typical usage of MONITOR/MWAIT.

Example 4-1. Pseudo Code to Use MONITOR/MWAIT

```
// Trigger[MONITORDATARANGE] is the memory address range that will be
// used as the trigger data range Trigger[0] = 0;
       If ( trigger[0] != TRIGERRDATAVALUE) {
          EAX = &trigger[0]
          ECX = 0
          EDX = 0
          MONITOR EAX, ECX, EDX
          If (trigger[0] != TRIGERRDATAVALUE ) {
             EAX = 0
             ECX = 0
             MWAIT EAX, ECX
          }
       }
```

APPENDIX A

A.1. INSTRUCTION SUMMARY

Table A-1, lists the six types of floating-point exceptions that can be generated by Prescott New Instructions. Table A-2 lists individual instructions and associated exceptions. All of the exceptions shown except the denormal-operand exception (#D) and invalid-operation exception for stack underflow or stack overflow (#IS) are defined in IEEE Standard 754 for Binary Floating-Point Arithmetic.

Table A-1. x87 FPU and SIMD Floating-point Exceptions

Floating-point Exception	Description
#IS	Invalid-operation exception for stack underflow or stack overflow. (Can only be generated for x87 FPU instructions.)
#IA or #I	Invalid-operation exception for invalid arithmetic operands and unsupported formats.
#D	Denormal-operand exception.
#Z	Divide-by-zero exception.
#O	Numeric-overflow exception.
#U	Numeric-underflow exception.
#P	Inexact-result (precision) exception.

Table A-2. Prescott New Instruction Technology Instruction Set Summary

Opcode	Instruction	Description	#I	#D	#Z	#O	#U	#P
66,0F,D0,/r	ADDSUBPD xmm1, xmm2/m128	Add /Sub packed DP FP numbers from XMM2/Mem to XMM1.	Y	Y		Y	Y	Y
F2,0F,D0,/r	ADDSUBPS xmm1, xmm2/m128	Add /Sub packed SP FP numbers from XMM2/Mem to XMM1.	Y	Y		Y	Y	Y
DF /1	FISTTP m16int	Store ST in int16 (chop) and pop.	Y					Y
DB /1	FISTTP m32int	Store ST in int32 (chop) and pop.	Y					Y
DD /1	FISTTP m64int	Store ST in int64 (chop) and pop.	Y					Y

Table A-2. Prescott New Instruction Technology Instruction Set Summary

Opcode	Instruction	Description	#I	#D	#Z	#O	#U	#P
66,0F,7C,/r	HADDPD xmm1, xmm2/m128	Add horizontally packed DP FP numbers XMM2/Mem to XMM1.	Y	Y		Y	Y	Y
F2,0F,7C,/r	HADDPS xmm1, xmm2/m128	Add horizontally packed SP FP numbers XMM2/Mem to XMM1	Y	Y		Y	Y	Y
66,0F,7D,/r	HSUBPD xmm1, xmm2/m128	Sub horizontally packed DP FP numbers XMM2/Mem to XMM1	Y	Y		Y	Y	Y
F2,0F,7D,/r	HSUBPS xmm1, xmm2/m128	Sub horizontally packed SP FP numbers XMM2/Mem to XMM1	Y	Y		Y	Y	Y
F2,0F,F0,/r	LDDQU xmm, m128	Load unaligned integer 128-bit.						
0F,01,C8	MONITOR eax, ecx, edx	Set up a linear address range to be monitored by hardware.						
F2,0F,12,/r	MOVDDUP xmm1, xmm2/m64	Move 64 bits representing one DP data from XMM2/Mem to XMM1 and duplicate.						
F3,0F,16,/r	MOVSHDUP xmm1, xmm2/m128	Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicate high.						
F3,0F,12,/r	MOVSLDUP xmm1, xmm2/m128	Move 128 bits representing 4 SP data from XMM2/Mem to XMM1 and duplicate low.						
0F,01,C9	MWAIT eax, ecx	Wait until write-back store performed within the range specified by the instruction MONITOR.						

INDEX