AMD Extensions to the 3DNow!™ and MMX™ Instruction Sets

Manual

AMD

Trademarks

Contents

Revision History vii

1 Extensions to the 3DNow!™ and MMX™ Instruction Sets 1

Introduction 1

2 Extensions to the 3DNow!™ Instruction Set 3

PF2IW 4
PFNACC 6
PFPNACC 7
PI2FW 8
PSWAPD 9

3 Extensions to the MMX™ Instruction Set 11

MASKMOVQ 12
MOVNTQ 13
PAVGB 14
PAVGW 17
PEXTRW 19
PINSRW 20
PMAXSW 21
PMAXUB 22
PMINSW 24
PMINUB 25
PMOVMSKB 27
PMULHUW 28
PREFETCHNTA - PREFETCHT0 - PREFETCHT1 - PREFETCHT2 30
PSADBW 31
PSHUFW 33
SFENCE 35

List of Tables

Table 1. 3DNow!™ Technology DSP Extensions 2
Table 2. MMX™ Instruction Set Extensions 2
Table 3. Numerical Range for the PF2IW Instruction 5
Table 4. Locality References for the Prefetch Instructions 30

Revision History

Date	Rev	Description
August 1999	B	Initial public release
February 2000	C	■ Clarification of PSWAPD operation on page 9. ■ Clarification of PINSRW description and operation on page 20. ■ Clarification of PSHUFW description and operation on page 33. ■ Clarification of SFENCE encoding on page 35.

1

Extensions to the 3DNow!™ and MMX™ Instruction Sets

Introduction

With the advent of the AMD Athlon™ processor, AMD has taken 3DNow!™ Technology to the next level of performance and functionality. The AMD Athlon processor adds 24 new instructions to the existing 3DNow! and MMX™ instruction sets. Along with the new instructions, the AMD Athlon processor implements additional microarchitecture enhancements that enable more efficient operation of all these instructions, and programming may be simplified because there are fewer coding restrictions.

3DNow! technology enabled fast frame rates on high-resolution 3D rendered scenes, amazing physical modeling of real-world environments, sharp and detailed 3D imaging, smooth video playback, and theater-quality audio. The new enhanced 3DNow! technology implemented in the AMD Athlon processor adds streaming and digital signal processing (DSP) technologies, which allow faster, more accurate speech recognition, DVD-quality audio and video, and streaming audio and video for a rich Internet experience.

The instructions described in this document are extensions to the instruction sets described in the *3DNow!™ Technology Manual,* order# 21928 and the *Multimedia Technology Manual,* order# 20726. The five new 3DNow! technology DSP extensions

are summarized in Table 1 and fully described in Chapter 2. The 19 new instructions augmenting existing MMX technology are summarized in Table 2 and fully described in Chapter 3.

Table 1. 3DNow!™ Technology DSP Extensions

Operation	Function	Opcode / imm8
PF2IW	Packed Floating-Point to Integer Word Conversion with Sign Extend	0Fh 0Fh / 1Ch
PFNACC	Packed Floating-Point Negative Accumulate	0Fh 0Fh / 8Ah
PFPNACC	Packed Floating-Point Mixed Positive-Negative Accumulate	0Fh 0Fh / 8Eh
PI2FW	Packed Integer Word to Floating-Point Conversion	0Fh 0Fh / 0Ch
PSWAPD	Packed Swap Doubleword	0Fh 0Fh / BBh

Table 2. MMX™ Instruction Set Extensions

Operation	Function	Opcode / imm8
MASKMOVQ	Streaming (Cache Bypass) Store Using Byte Mask	0Fh F7h
MOVNTQ	Streaming (Cache Bypass) Store	0Fh E7h
PAVGB	Packed Average of Unsigned Byte	0Fh E0h
PAVGW	Packed Average of Unsigned Word	0Fh E3h
PEXTRW	Extract Word into Integer Register	0Fh C5h
PINSRW	Insert Word from Integer Register	0Fh C4h
PMAXSW	Packed Maximum Signed Word	0Fh EEh
PMAXUB	Packed Maximum Unsigned Byte	0Fh DEh
PMINSW	Packed Minimum Signed Word	0Fh EAh
PMINUB	Packed Minimum Unsigned Byte	0Fh DAh
PMOVMSKB	Move Byte Mask to Integer Register	0Fh D7h
PMULHUW	Packed Multiply High Unsigned Word	0Fh E4h
PREFETCHNTA	Move Data Closer to the Processor Using the NTA Reference	0Fh 18h 0*
PREFETCHT0	Move Data Closer to the Processor Using the T0 Reference	0Fh 18h 1*
PREFETCHT1	Move Data Closer to the Processor Using the T1 Reference	0Fh 18h 2*
PREFETCHT2	Move Data Closer to the Processor Using the T2 Reference	0Fh 18h 3*
PSADBW	Packed Sum of Absolute Byte Differences	0Fh F6h
PSHUFW	Packed Shuffle Word	0Fh 70h
SFENCE	Store Fence	0Fh AEh / 7h

Note:
** The number after the opcode indicates the different prefetch modes in the ModR/M byte.*

2

Extensions to the 3DNow!™ Instruction Set

This chapter describes the five new DSP instructions added to the 3DNow! instruction set first defined in the *3DNow!™ Technology Manual*, order# 21928. The five instructions enhance the performance of communications applications, including soft modems, soft ADSL, MP3, and Dolby Digital and Surround sound processing.

Programmers should check bit 30 of the Extended Feature Flags in the EDX register after executing extended function 8000_0001h of the CPUID instruction. If bit 30 is set, the AMD processor supports these five instructions. For more information, refer to the *AMD Processor Recognition Application Note*, order# 20734.

Instruction definitions are in alphabetical order according to the instruction mnemonics.

PF2IW

mnemonic	*opcode / imm8*	*description*
PF2IW mmreg1, mmreg2 PF2IW mmreg, mem64	0Fh 0Fh / 1Ch	Packed floating-point to integer word conversion with sign extend

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

PF2IW converts a register containing single-precision floating-point operands to 16-bit signed integers using truncation. Arguments outside the range representable by signed 16-bit integers are saturated to the largest and smallest 16-bit integer, depending on their sign. All results are sign-extended to 32-bits. Table 3 on page 5 shows the numerical range of the PF2IW instruction.

"PF2IW mmreg1, mmreg2" performs the following operations:

```
IF (mmreg2[31:0] >= 2^15)
   THEN mmreg1[31:0] = 0x0000_7FFF
ELSE IF (mmreg2[31:0] <= -2^15)
   THEN mmreg1[31:0] = 0xFFFF_8000
ELSE mmreg1[31:0] = int(mmreg2[31:0])
IF (mmreg2[63:32] >= 2^15)
   THEN mmreg1[63:32] = 0x0000_7FFF
ELSE IF (mmreg2[63:32] <= -2^15)
   THEN mmreg1[63:32] = 0xFFFF_8000
ELSE mmreg1[63:32] = int(mmreg2[63:32])
```

"PF2IW mmreg, mem64" performs the following operations:

```
IF (mem64[31:0] >= 2^15)
   THEN mmreg[31:0] = 0x0000_7FFF
ELSE IF (mem64[31:0] <= -2^15)
   THEN mmreg[31:0] = 0xFFFF_8000
ELSE mmreg[31:0] = int(mem64[31:0])

IF (mem64[63:32] >= 2^15)
   THEN mmreg[63:32] = 0x0000_7FFF
ELSE IF (mem64[63:32] <= -2^15)
   THEN mmreg[63:32] = 0xFFFF_8000
ELSE mmreg[63:32] = int(mem64[63:32])
```

Table 3. Numerical Range for the PF2IW Instruction

Source 2	Source 1 and Destination
0	0
Normal, abs(Source 1) <1	0
Normal, –32768 < Source 1 <= –1	round to zero (Source 1)
Normal, 1 <= Source 1< 32768	round to zero (Source 1)
Normal, Source 1 >= 32768	0x0000_7FFF
Normal, Source 1 <= –32768	0xFFFF_8000
Unsupported	Undefined

Related Instructions See the PF2ID, PI2FW, and PI2FD instructions.

PFNACC

mnemonic	*opcode / imm8*	*description*
PFNACC mmreg1, mmreg2 PFNACC mmreg, mem64	0Fh 0Fh / 8Ah	Packed floating-point negative accumulate

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

PFNACC performs negative accumulation of the two doublewords of the destination operand and the source operand. PFNACC then stores the results in the low and high words of the destination operand, respectively. Both operands are single-precision, floating-point operands with 24-bit significands.

"PFNACC mmreg1, mmreg2" performs the following operations:

```
mmreg1[31:0]  = mmreg1[31:0]   - mmreg1[63:32]
mmreg1[63:32] = mmreg2[31:0]   - mmreg2[63:32]
```

"PFNACC mmreg, mem64" performs the following operations:

```
mmreg[31:0]   = mmreg[31:0] - mmreg[63:32]
mmreg[63:32]  = mem64[31:0] - mem64[63:32]
```

Related Instructions See the PFACC and PFPNACC instructions.

PFPNACC

mnemonic	*opcode / imm8*	*description*
PFPNACC mmreg1, mmreg2 PFPNACC mmreg, mem64	0Fh 0Fh / 8Eh	Packed floating-point mixed positive-negative accumulate

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

PFPNACC performs mixed negative and positive accumulation of the two doublewords of the destination operand and the source operand. PFPNACC then stores the results in the low and high words of the destination operand, respectively. Both operands are single-precision, floating-point operands with 24-bit significands.

"PFPNACC mmreg1, mmreg2" performs the following operations:

```
mmreg1[31:0]  = mmreg1[31:0] - mmreg1[63:32]
mmreg1[63:32] = mmreg2[31:0] + mmreg2[63:32]
```

"PFPNACC mmreg, mem64" performs the following operations:

```
mmreg[31:0]   = mmreg[31:0]  - mmreg[63:32]
mmreg[63:32]  = mem64[31:0]  + mem64[63:32]
```

Related Instructions See the PFACC and PFNACC instructions.

PI2FW

mnemonic	*opcode / imm8*	*description*
PI2FW mmreg1, mmreg2 PI2FW mmreg, mem64	0Fh 0Fh / 0Ch	Packed 16-bit integer to floating-point conversion

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

PI2FW converts a register containing signed, 16-bit integers to single-precision, floating-point operands.

"PI2FW mmreg1, mmreg2" performs the following operations:

```
mmreg1[31:0]  = float(mmreg2[15:0])
mmreg1[63:32] = float(mmreg2[47:32])
```

"PI2FW mmreg, mem64" performs the following operations:

```
mmreg[31:0]  = float(mem64[15:0])
mmreg[63:32] = float(mem64[47:32])
```

Related Instructions See the PI2FD, PF2IW, and PF2ID instructions.

PSWAPD

mnemonic	*opcode / imm8*	*description*
PSWAPD mmreg1, mmreg2 PSWAPD mmreg, mem64	0Fh 0Fh / BBh	Packed swap doubleword

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PSWAPD instruction swaps or reverses the upper and lower doublewords of the source operand.

"PSWAPD mmreg1, mmreg2" performs the following operations:

```
temp = mmreg2
mmreg1[63:32]  = temp[31:0]
mmreg1[31:0]   = temp[63:32]
```

"PSWAPD mmreg, mem64" performs the following operations:

```
mmreg[63:32]   = mem64[31:0]
mmreg[31:0]    = mem64[63:32]
```

3

Extensions to the MMX™ Instruction Set

This chapter describes 19 new instructions added to the MMX instruction set defined in the *AMD-K6® MMX™ Enhanced Processor Multimedia Technology Manual*, order# 20726. Twelve of the instructions improve multimedia-enhanced integer math calculations used in such applications as speech recognition and high-quality video processing. Seven instructions are dedicated to efficiently moving multimedia data into and out of the processor.

Programmers should check bit 22 of the Extended Feature Flags in the EDX register after executing extended function 8000_0001h of the CPUID instruction. If bit 22 is set, the AMD processor supports these 19 instructions. See the *AMD Processor Recognition Application Note*, order# 20734 for more information.

Instruction definitions are in alphabetical order according to the instruction mnemonics.

MASKMOVQ

mnemonic	*opcode*	*description*
MASKMOVQ mmreg1, mmreg2 (edi)	0Fh F7h	Streaming (cache bypass) store using byte mask

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The MASKMOVQ instruction conditionally stores individual bytes of an MMX register to a memory location specified by the edi register, while using the byte mask in a second MMX register. The MASKMOVQ instruction acts as a streaming store to minimize cache pollution. It is used to store data without first reading in old data (no write allocate).

"MASKMOVQ mmreg1, mmreg2 (edi)" performs the following operations:

```
memory[edi][63:56] = mmreg2[63] ? mmreg1[63:56] : memory[edi][63:56]
memory[edi][55:48] = mmreg2[55] ? mmreg1[55:48] : memory[edi][55:48]
memory[edi][47:40] = mmreg2[47] ? mmreg1[47:40] : memory[edi][47:40]
memory[edi][39:32] = mmreg2[39] ? mmreg1[39:32] : memory[edi][39:32]
memory[edi][31:24] = mmreg2[31] ? mmreg1[31:24] : memory[edi][31:24]
memory[edi][23:16] = mmreg2[23] ? mmreg1[23:16] : memory[edi][23:16]
memory[edi][15:8]  = mmreg2[15] ? mmreg1[15:8]  : memory[edi][15:8]
memory[edi][7:0]   = mmreg2[7]  ? mmreg1[7:0]   : memory[edi][7:0]
```

MOVNTQ

mnemonic	*opcode*	*description*
MOVNTQ mem64, mmreg	0Fh E7h	Streaming (cache bypass) store

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The MOVNTQ instruction stores individual bytes of an MMX register to memory. The MOVNTQ instruction acts as a streaming store to minimize cache pollution. It is used to store data without first reading in old data (no write allocate).

"MOVNTQ mem64, mmreg" performs the following operations:

```
mem64[63:0] = mmreg
```

PAVGB

mnemonic	*opcode*	*description*
PAVGB mmreg1, mmreg2 PAVGB mmreg, mem64	0Fh E0h	Packed average of unsigned byte

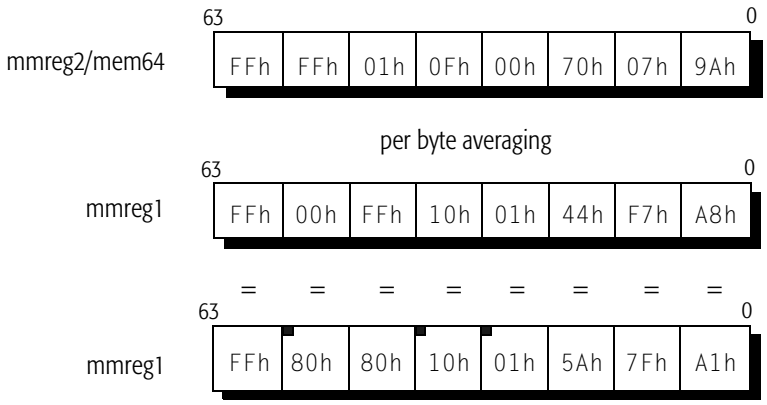
Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PAVGB instruction produces the rounded up averages of the eight unsigned 8-bit integer values in the source operand (an MMX register or a 64-bit memory location) and the eight corresponding unsigned 8-bit integer values in the destination operand (an MMX register). It does so by adding the source and destination byte values to get a 9-bit unsigned intermediate value. The intermediate value is then incremented by one and finally shifted to the right by one bit position. The eight unsigned 8-bit results are stored in the MMX register specified as the destination operand.

The PAVGB instruction is identical to the 3DNow! PAVGUSB instruction and can be used for pixel averaging in MPEG-2 motion compensation and video scaling operations.

Functional Illustration of the PAVGB Instruction



■ Indicates a value that was rounded-up

The following list explains the functional illustration of the PAVGB instruction:

- The rounded byte average of FFh and FFh is FFh.
- The rounded byte average of FFh and 00h is 80h.
- The rounded byte average of 01h and FFh is also 80h.
- The rounded byte average of 0Fh and 10h is 10h.
- The rounded byte average of 00h and 01h is 01h.
- The rounded byte average of 70h and 44h is 5Ah.
- The rounded byte average of 07h and F7h is 7Fh.
- The rounded byte average of 9Ah and A8h is A1h.

"PAVGB mmreg1, mmreg2" performs the following operations:

```
mmreg1[7:0]   = (mmreg1[7:0]   + mmreg2[7:0]    + 1) >> 1
mmreg1[15:8]  = (mmreg1[15:8]  + mmreg2[15:8]   + 1) >> 1
mmreg1[23:16] = (mmreg1[23:16] + mmreg2[23:16]  + 1) >> 1
mmreg1[31:24] = (mmreg1[31:24] + mmreg2[31:24]  + 1) >> 1
mmreg1[39:32] = (mmreg1[39:32] + mmreg2[39:32]  + 1) >> 1
mmreg1[47:40] = (mmreg1[47:40] + mmreg2[47:40]  + 1) >> 1
mmreg1[55:48] = (mmreg1[55:48] + mmreg2[55:48]  + 1) >> 1
mmreg1[63:56] = (mmreg1[63:56] + mmreg2[63:56]  + 1) >> 1
```

"PAVGB mmreg, mem64" performs the following operations:

```
mmreg[7:0]   = (mmreg[7:0]   + mem64[7:0]   + 1) >> 1
mmreg[15:8]  = (mmreg[15:8]  + mem64[15:8]  + 1) >> 1
mmreg[23:16] = (mmreg[23:16] + mem64[23:16] + 1) >> 1
mmreg[31:24] = (mmreg[31:24] + mem64[31:24] + 1) >> 1
mmreg[39:32] = (mmreg[39:32] + mem64[39:32] + 1) >> 1
mmreg[47:40] = (mmreg[47:40] + mem64[47:40] + 1) >> 1
mmreg[55:48] = (mmreg[55:48] + mem64[55:48] + 1) >> 1
mmreg[63:56] = (mmreg[63:56] + mem64[63:56] + 1) >> 1
```

Related Instructions See the PAVGW instruction.

PAVGW

mnemonic	*opcode*	*description*
PAVGW mmreg1, mmreg2 PAVGW mmreg, mem64	0Fh E3h	Packed average of unsigned word

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PAVGW instruction is the same as the PAVGB instruction, except it operates on packed unsigned words instead. PAVGW produces the rounded up averages of the four unsigned 16-bit integer values in the source operand (an MMX register or a 64-bit memory location) and the four corresponding unsigned 16-bit integer values in the destination operand (an MMX register). It does so by adding the source and destination byte values to get a 17-bit unsigned intermediate value. The intermediate value is then incremented by one and finally shifted to the right by one bit position. The four unsigned 16-bit results are stored in the MMX register specified as the destination operand.

"PAVGW mmreg1, mmreg2" performs the following operations:

```
mmreg1[15:0]  = (mmreg1[15:0]  + mmreg2[15:0]  + 1) >> 1
mmreg1[31:16] = (mmreg1[31:16] + mmreg2[31:16] + 1) >> 1
mmreg1[47:32] = (mmreg1[47:32] + mmreg2[47:32] + 1) >> 1
mmreg1[63:48] = (mmreg1[63:48] + mmreg2[63:48] + 1) >> 1
```

"PAVGW mmreg, mem64" performs the following operations:

```
mmreg[15:0]  = (mmreg[15:0]  + mem64[15:0]  + 1) >> 1
mmreg[31:16] = (mmreg[31:16] + mem64[31:16] + 1) >> 1
mmreg[47:32] = (mmreg[47:32] + mem64[47:32] + 1) >> 1
mmreg[63:48] = (mmreg[63:48] + mem64[63:48] + 1) >> 1
```

Related Instructions See the PAVGB instruction.

PEXTRW

mnemonic	*opcode*	*description*
PEXTRW reg32, mmreg, imm8	0Fh C5h	Extract word into integer register

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PEXTRW instruction extracts one of the four words pointed to by imm8 from an MMX register and stores that into the least significant word of a 32-bit integer register.

"PEXTRW reg32, mmreg, imm8" performs the following operations:

```
index = imm8[1:0] * 16
reg32[31:16] = 0
reg32[15:0]  = mmreg[index+15:index]
```

Related Instructions See the PINSRW instruction.

PINSRW

mnemonic	*opcode*	*description*
PINSRW mmreg, reg32, imm8 PINSRW mmreg, mem16, imm8	0Fh C4h	Insert word from integer register

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PINSRW instruction inserts the least significant word of the source operand (an integer register or a 16-bit memory location) into one of the four words of the destination operand (an MMX register).

"PINSRW mmreg, reg32, imm8" performs the following operations:

```
index = imm8[1:0] * 16
temp1 = 0
temp1[index+15:index] = reg32[15:0]
temp2 = mmreg
temp2[index+15:index] = 0
mmreg = temp1 | temp2
```

"PINSRW mmreg, mem16, imm8" performs the following operations:

```
index = imm8[1:0] * 16
temp1 = 0
temp1[index+15:index] = mem16[15:0]
temp2 = mmreg
temp2[index+15:index] = 0
mmreg = temp1 | temp2
```

Related Instructions See the PEXTRW instruction.

PMAXSW

mnemonic	*opcode*	*description*
PMAXSW mmreg1, mmreg2 PMAXSW mmreg, mem64	0Fh EEh	Packed maximum signed word

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PMAXSW instruction operates on signed 16-bit data and selects the maximum signed value between source 1 and source 2 for each of the four word positions.

"PMAXSW mmreg1, mmreg2" performs the following signed operations:

```
mmreg1[15:0]  = (mmreg1[15:0]  > mmreg2[15:0])  ? mmreg1[15:0]  : mmreg2[15:0]
mmreg1[31:16] = (mmreg1[31:16] > mmreg2[31:16]) ? mmreg1[31:16] : mmreg2[31:16]
mmreg1[47:32] = (mmreg1[47:32] > mmreg2[47:32]) ? mmreg1[47:32] : mmreg2[47:32]
mmreg1[63:48] = (mmreg1[63:48] > mmreg2[63:48]) ? mmreg1[63:48] : mmreg2[63:48]
```

"PMAXSW mmreg, mem64" performs the following signed operations:

```
mmreg[15:0]  = (mmreg[15:0]  > mem64[15:0])  ? mmreg[15:0]  : mem64[15:0]
mmreg[31:16] = (mmreg[31:16] > mem64[31:16]) ? mmreg[31:16] : mem64[31:16]
mmreg[47:32] = (mmreg[47:32] > mem64[47:32]) ? mmreg[47:32] : mem64[47:32]
mmreg[63:48] = (mmreg[63:48] > mem64[63:48]) ? mmreg[63:48] : mem64[63:48]
```

Related Instructions See the PMINSW instruction.

PMAXUB

mnemonic	*opcode*	*description*
PMAXUB mmreg1, mmreg2 PMAXUB mmreg, mem64	0Fh DEh	Packed maximum unsigned byte

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PMAXUB instruction operates on unsigned 8-bit data and selects the maximum unsigned value between source 1 and source 2 for each of the eight byte positions.

"PMAXUB mmreg1, mmreg2" performs the following unsigned operations:

```
mmreg1[7:0]   = (mmreg1[7:0]   > mmreg2[7:0])   ? mmreg1[7:0]   : mmreg2[7:0]
mmreg1[15:8]  = (mmreg1[15:8]  > mmreg2[15:8])  ? mmreg1[15:8]  : mmreg2[15:8]
mmreg1[23:16] = (mmreg1[23:16] > mmreg2[23:16]) ? mmreg1[23:16] : mmreg2[23:16]
mmreg1[31:24] = (mmreg1[31:24] > mmreg2[31:24]) ? mmreg1[31:24] : mmreg2[31:24]
mmreg1[39:32] = (mmreg1[39:32] > mmreg2[39:32]) ? mmreg1[39:32] : mmreg2[39:32]
mmreg1[47:40] = (mmreg1[47:40] > mmreg2[47:40]) ? mmreg1[47:40] : mmreg2[47:40]
mmreg1[55:48] = (mmreg1[55:48] > mmreg2[55:48]) ? mmreg1[55:48] : mmreg2[55:48]
mmreg1[63:56] = (mmreg1[63:56] > mmreg2[63:56]) ? mmreg1[63:56] : mmreg2[63:56]
```

"PMAXUB mmreg, mem64" performs the following unsigned operations:

```
mmreg[7:0]   = (mmreg[7:0]   > mem64[7:0])   ? mmreg[7:0]   : mem64[7:0]
mmreg[15:8]  = (mmreg[15:8]  > mem64[15:8])  ? mmreg[15:8]  : mem64[15:8]
mmreg[23:16] = (mmreg[23:16] > mem64[23:16]) ? mmreg[23:16] : mem64[23:16]
mmreg[31:24] = (mmreg[31:24] > mem64[31:24]) ? mmreg[31:24] : mem64[31:24]
mmreg[39:32] = (mmreg[39:32] > mem64[39:32]) ? mmreg[39:32] : mem64[39:32]
mmreg[47:40] = (mmreg[47:40] > mem64[47:40]) ? mmreg[47:40] : mem64[47:40]
mmreg[55:48] = (mmreg[55:48] > mem64[55:48]) ? mmreg[55:48] : mem64[55:48]
mmreg[63:56] = (mmreg[63:56] > mem64[63:56]) ? mmreg[63:56] : mem64[63:56]
```

Related Instructions See the PMINUB instruction.

PMINSW

mnemonic	*opcode*	*description*
PMINSW mmreg1, mmreg2 PMINSW mmreg, mem64	0Fh EAh	Packed minimum signed word

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PMINSW instruction operates on signed 16-bit data and selects the minimum arithmetic value between source 1 and source 2 for each word.

"PMINSW mmreg1, mmreg2" performs the following signed operations:

```
mmreg1[15:0]  = (mmreg1[15:0]  <= mmreg2[15:0])  ? mmreg1[15:0]  : mmreg2[15:0]
mmreg1[31:16] = (mmreg1[31:16] <= mmreg2[31:16]) ? mmreg1[31:16] : mmreg2[31:16]
mmreg1[47:32] = (mmreg1[47:32] <= mmreg2[47:32]) ? mmreg1[47:32] : mmreg2[47:32]
mmreg1[63:48] = (mmreg1[63:48] <= mmreg2[63:48]) ? mmreg1[63:48] : mmreg2[63:48]
```

"PMINSW mmreg, mem64" performs the following signed operations:

```
mmreg[15:0]  = (mmreg[15:0]  <= mem64[15:0])  ? mmreg[15:0]  : mem64[15:0]
mmreg[31:16] = (mmreg[31:16] <= mem64[31:16]) ? mmreg[31:16] : mem64[31:16]
mmreg[47:32] = (mmreg[47:32] <= mem64[47:32]) ? mmreg[47:32] : mem64[47:32]
mmreg[63:48] = (mmreg[63:48] <= mem64[63:48]) ? mmreg[63:48] : mem64[63:48]
```

Related Instructions See the PMAXSW instruction.

PMINUB

mnemonic	*opcode*	*description*
PMINUB mmreg1, mmreg2 PMINUB mmreg, mem64	0Fh DAh	Packed minimum unsigned byte

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PMINUB instruction operates on unsigned 8-bit data and selects the minimum value between source 1 and source 2 for each byte position.

"PMINUB mmreg1, mmreg2" performs the following unsigned operations:

```
mmreg1[7:0]   = (mmreg1[7:0]   <= mmreg2[7:0])   ? mmreg1[7:0]   : mmreg2[7:0]
mmreg1[15:8]  = (mmreg1[15:8]  <= mmreg2[15:8])  ? mmreg1[15:8]  : mmreg2[15:8]
mmreg1[23:16] = (mmreg1[23:16] <= mmreg2[23:16]) ? mmreg1[23:16] : mmreg2[23:16]
mmreg1[31:24] = (mmreg1[31:24] <= mmreg2[31:24]) ? mmreg1[31:24] : mmreg2[31:24]
mmreg1[39:32] = (mmreg1[39:32] <= mmreg2[39:32]) ? mmreg1[39:32] : mmreg2[39:32]
mmreg1[47:40] = (mmreg1[47:40] <= mmreg2[47:40]) ? mmreg1[47:40] : mmreg2[47:40]
mmreg1[55:48] = (mmreg1[55:48] <= mmreg2[55:48]) ? mmreg1[55:48] : mmreg2[55:48]
mmreg1[63:56] = (mmreg1[63:56] <= mmreg2[63:56]) ? mmreg1[63:56] : mmreg2[63:56]
```

"PMINUB mmreg1, mem64" performs the following unsigned operations:

```
mmreg[7:0]   = (mmreg[7:0]   <= mem64[7:0])   ? mmreg[7:0]   : mem64[7:0]
mmreg[15:8]  = (mmreg[15:8]  <= mem64[15:8])  ? mmreg[15:8]  : mem64[15:8]
mmreg[23:16] = (mmreg[23:16] <= mem64[23:16]) ? mmreg[23:16] : mem64[23:16]
mmreg[31:24] = (mmreg[31:24] <= mem64[31:24]) ? mmreg[31:24] : mem64[31:24]
mmreg[39:32] = (mmreg[39:32] <= mem64[39:32]) ? mmreg[39:32] : mem64[39:32]
mmreg[47:40] = (mmreg[47:40] <= mem64[47:40]) ? mmreg[47:40] : mem64[47:40]
mmreg[55:48] = (mmreg[55:48] <= mem64[55:48]) ? mmreg[55:48] : mem64[55:48]
mmreg[63:56] = (mmreg[63:56] <= mem64[63:56]) ? mmreg[63:56] : mem64[63:56]
```

Related Instructions See the PMAXUB instruction.

PMOVMSKB

mnemonic	*opcode*	*description*
PMOVMSKB reg32, mmreg	0Fh D7h	Move byte mask to integer register

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PMOVMSKB instruction selects the most significant bit from each byte position of an MMX register and collapses all eight bits into the least significant byte of an integer register.

"PMOVMSKB reg32, mmreg" performs the following operations:

```
reg32[31:8] = 0
reg32[7] = mmreg[63]
reg32[6] = mmreg[55]
reg32[5] = mmreg[47]
reg32[4] = mmreg[39]
reg32[3] = mmreg[31]
reg32[2] = mmreg[23]
reg32[1] = mmreg[15]
reg32[0] = mmreg[7]
```

PMULHUW

mnemonic	*opcode*	*description*
PMULHUW mmreg1, mmreg2 PMULHUW mmreg, mem64	0Fh E4h	Packed multiply high unsigned word

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PMULHUW instruction multiplies the four unsigned words in the source operand with the four unsigned words in the destination operand. The upper 16 bits of the 32-bit intermediate result is placed into the destination operand.

"PMULHUW mmreg1, mmreg2" performs the following operations:

```
temp1 = (mmreg1[15:0]  * mmreg2[15:0])
temp2 = (mmreg1[31:16] * mmreg2[31:16])
temp3 = (mmreg1[47:32] * mmreg2[47:32])
temp4 = (mmreg1[63:48] * mmreg2[63:48])

mmreg1[15:0]  = temp1[31:16]
mmreg1[31:16] = temp2[31:16]
mmreg1[47:32] = temp3[31:16]
mmreg1[63:48] = temp4[31:16]
```

"PMULHUW mmreg, mem64" performs the following operations:

```
temp1 = (mmreg1[15:0]  * mem642[15:0])
temp2 = (mmreg1[31:16] * mem64[31:16])
temp3 = (mmreg1[47:32] * mem64[47:32])
temp4 = (mmreg1[63:48] * mem64[63:48])

mmreg1[15:0]  = temp1[31:16]
mmreg1[31:16] = temp2[31:16]
mmreg1[47:32] = temp3[31:16]
mmreg1[63:48] = temp4[31:16]
```

PREFETCHNTA - PREFETCHT0 - PREFETCHT1 - PREFETCHT2

mnemonic	*opcode / ModR/M*	*description*
PREFETCHNTA mem8	0Fh 18h / 0	Move data closer to the processor using the NTA reference.
PREFETCHT0 mem8	0Fh 18h / 1	Move data closer to the processor using the T0 reference.
PREFETCHT1 mem8	0Fh 18h / 2	Move data closer to the processor using the T1 reference.
PREFETCHT2 mem8	0Fh 18h / 3	Move data closer to the processor using the T2 reference.

Privilege: none
Registers Affected: none
Flags Affected: none
Exceptions Generated: none

The prefetch instruction brings a cache line into the processor cache level(s) specified by a locality reference. The address of the prefetched cache line is specified by the mem8 value. The prefetch instruction loads a cache line even if the mem8 address is not aligned with the start of the line. If the cache line is already contained in a cache level that is lower than the locality reference or a memory fault is detected, then no bus cycle is initiated and the instruction is treated as a NOP.

The operation of the prefetch instructions is processor implementation dependent. The instructions can be ignored or changed by a processor implementation, though they will not change program behavior. The cache line size is also implementation dependent having a minimum size of 32 bytes.

Bits 5:3 of the ModR/M byte indicate the cache locality references.

Table 4. Locality References for the Prefetch Instructions

Locality Reference	Description
NTA	Move specified data into processor with minimal L1/L2 cache pollution.
T0	Move specified data into all cache levels.
T1	Move specified data into all cache levels except 0th level cache.
T2	Move specified data into all cache levels except 0th and 1st level caches.

Note:
A 0th level cache is implementation dependent.

PSADBW

mnemonic	*opcode*	*description*
PSADBW mmreg1, mmreg2 PSADBW mmreg, mem64	0Fh F6h	Packed sum of absolute byte differences

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PSADBW instruction is the sum of the absolute value of the differences between each byte position of source 1 and source 2.

"PSADBW mmreg1, mmreg2" performs the following operations:

```
mmreg1[63:16] =  0
mmreg1[15:0]  =  abs(mmreg1[7:0]   -  mmreg2[7:0])   +
                 abs(mmreg1[15:8]  -  mmreg2[15:8])  +
                 abs(mmreg1[23:16] -  mmreg2[23:16]) +
                 abs(mmreg1[31:24] -  mmreg2[31:24]) +
                 abs(mmreg1[39:32] -  mmreg2[39:32]) +
                 abs(mmreg1[47:40] -  mmreg2[47:40]) +
                 abs(mmreg1[55:48] -  mmreg2[55:48]) +
                 abs(mmreg1[63:56] -  mmreg2[63:56])
```

"PSADBW mmreg, mem64" performs the following operations:

```
mmreg[63:16] =  0
mmreg[15:0]  =  abs(mmreg[7:0]   - mem64[7:0])   +
                abs(mmreg[15:8]  - mem64[15:8])  +
                abs(mmreg[23:16] - mem64[23:16]) +
                abs(mmreg[31:24] - mem64[31:24]) +
                abs(mmreg[39:32] - mem64[39:32]) +
                abs(mmreg[47:40] - mem64[47:40]) +
                abs(mmreg[55:48] - mem64[55:48]) +
                abs(mmreg[63:56] - mem64[63:56])
```

Related Instructions See the PAVGUSB instruction.

PSHUFW

mnemonic	*opcode*	*description*
PSHUFW mmreg1, mmreg2, imm8 PSHUFW mmreg, mem64, imm8	0Fh 70h	Packed shuffle word

Privilege: none
Registers Affected: MMX
Flags Affected: none
Exceptions Generated:

Exception	Real	Virtual 8086	Protected	Description
Invalid opcode (6)	X	X	X	The emulate instruction bit (EM) of the control register (CR0) is set to 1.
Device not available (7)	X	X	X	The task switch bit (TS) of the control register (CR0) is set to 1.
Stack exception (12)			X	During instruction execution, the stack segment limit was exceeded.
General protection (13)			X	During instruction execution, the effective address of one of the segment registers used for the operand points to an illegal memory location.
Segment overrun (13)	X	X		One of the instruction data operands falls outside the address range 00000h to 0FFFFh.
Page fault (14)		X	X	A page fault resulted from the execution of the instruction.
Floating-point exception pending (16)	X	X	X	An exception is pending due to the floating-point execution unit.
Alignment check (17)		X	X	An unaligned memory reference resulted from the instruction execution, and the alignment mask bit (AM) of the control register (CR0) is set to 1. (In Protected Mode, CPL = 3.)

The PSHUFW instruction selects from the four words of the source operand (an MMX register or a 64-bit memory location) in one of 256 possible ways as defined by an immediate byte.

"PSHUFW mmreg1, mmreg2, imm8" performs the following operations:

```
index3 = imm8[7:6] * 16
index2 = imm8[5:4] * 16
index1 = imm8[3:2] * 16
index0 = imm8[1:0] * 16
temp = mmreg2
mmreg1[63:48] = temp[index3+15:index3]
mmreg1[47:32] = temp[index2+15:index2]
mmreg1[31:16] = temp[index1+15:index1]
mmreg1[15:0]  = temp[index0+15:index0]
```

"PSHUFW mmreg, mem64, imm8" performs the following operations:

```
index3 = imm8[7:6] * 16
index2 = imm8[5:4] * 16
index1 = imm8[3:2] * 16
index0 = imm8[1:0] * 16
mmreg[63:48] = mem64[index3+15:index3]
mmreg[47:32] = mem64[index2+15:index2]
mmreg[31:16] = mem64[index1+15:index1]
mmreg[15:0]  = mem64[index0+15:index0]
```

SFENCE

mnemonic	*opcode / imm8*	*description*
SFENCE	0Fh AEh / 7h	Store fence

Privilege:	none
Registers Affected:	none
Flags Affected:	none
Exceptions Generated:	none

In a weakly ordered system, hardware is allowed to reorder reads and writes between the processor and memory. For example, writeback stores can complete ahead of write-combining stores. SFENCE provides a mechanism to force a strong ordering between routines that produce weakly ordered results (such as WC memory types).

The SFENCE instruction makes all previous writes globally visible to any preceding store. For example, an SFENCE instruction will force a newer write-back store to wait until all older streaming stores or write-combining stores are completed.

Note: *Software should encode the SFENCE instruction with a ModR/M byte of 0xF8. All other possible ModR/M encodings are reserved for future use.*