5

# Intel Architecture MMX™ Instruction Set

# CHAPTER 5
# INTEL ARCHITECTURE MMX™ INSTRUCTION SET

This chapter presents the Intel Architecture MMX™ instructions in alphabetical order, with a full description of each instruction.

The IA MMX technology defines fifty-seven new instructions. The instructions are grouped into the following functional categories:

- Arithmetic Instructions
- Comparison Instructions
- Conversion Instructions
- Logical Instructions
- Shift Instructions
- Data Transfer Instructions
- Empty MMX State (EMMS) Instruction

Appendix A summarizes the MMX instructions grouped by categories of related functions. Appendix B provides instruction formats and encodings, and Appendix C provides an alphabetical list of instruction mnemonics, their source data types, encodings in hexadecimal, and format. Appendix D provides an Opcode Map of the MMX instructions.

Many of the instructions have multiple variations depending on the data types they support. Each variation has a different suffix. For example the PADD instruction has three variations: PADDB, PADDW, and PADDD, where the letters B, W, and D represent byte, word, and doubleword.

## 5.1. INSTRUCTION SYNTAX

Instructions vary by:

- Data type: packed bytes, packed words, packed doublewords or quadwords
- Signed - Unsigned numbers
- Wraparound - Saturate arithmetic

A typical MMX instruction has this syntax:

- Prefix: **P** for Packed
- Instruction operation: for example - ADD, CMP, or XOR
- Suffix:
  - **US** for Unsigned Saturation
  - **S** for Signed saturation
  - **B, W, D, Q** for the data type: packed byte, packed word, packed doubleword, or quadword.

Instructions that have different input and output data elements have two data-type suffixes. For example, the conversion instruction converts from one data type to another. It has two suffixes: one for the original data type and the second for the converted data type.

This is an example of an instruction mnemonic syntax :

**PADDUSW (Packed Add Unsigned with Saturation for Word)**

| | | |
|---|---|---|
| **P** | = | Packed |
| **ADD** | = | the instruction operation |
| **US** | = | Unsigned Saturation |
| **W** | = | Word |

## 5.2. INSTRUCTION FORMAT

The IA MMX instructions use the existing IA instruction format. All instructions, except the EMMS instruction, use the ModR/M format. All are preceded by the 0F prefix byte. For more details about the ModR/M format refer to *Pentium® Processor Family Developer's Manual Volume 3*, Section 25.2.1.

For data-transfer instructions, the destination and source operands can reside in memory, integer registers, or MMX registers. For all other IA MMX instructions, the destination operands reside in MMX registers, and the source operands reside in memory, MMX registers, or immediate operands.

All existing address modes are supported using the SIB (Scale Index Base) format.

## 5.3. NOTATIONAL CONVENTIONS

The following conventions apply to all MMX instructions (except the EMMS instruction):

- The instructions reference and operate on two operands: the source and destination operands. The right operand is the source and the left operand is the destination. The destination operand may also supply one of the inputs for the operation. The instruction overwrites the destination operand with the result.
- When one of the operands is a memory location, the linear address corresponds to the address of the least significant byte of the referenced memory data.
- The MMX instructions do not affect the condition flags.

## 5.4. HOW TO READ THE INSTRUCTION SET PAGES

The following is an example of the format used for each MMX instruction description in this chapter:

# PSLL—Packed Shift Left Logical

| Opcode | Instruction | Description |
|---|---|---|
| 0F F1 /r | PSLLW mm, mm/m64 | Shift all words in MMX register to left by an amount specified in MMX register/memory, while shifting in zeros. |

The above table gives the instruction mnemonic and a brief description of the mnemonic. The columns content are explained below.

### Opcode Column

The "Opcode" column provides the complete opcode produced for each form of the instruction.

The codes are defined as hexadecimal bytes, in the same order in which they appear in memory. Definitions of entries other than hexadecimal bytes are as follows:

- **/digit:** (digit is between 0 and 7) indicates that the ModR/M byte of the instruction uses only the r/m (register or memory) operand. The **reg** field contains the digit that provides a technology to the instruction's opcode.
- **/r**: indicates that the ModR/M byte of the instruction contains both a register operand and an r/m operand.
- **ib**: a 1-byte, immediate operand to the instruction that follows the opcode, ModR/M bytes, and scale-indexing bytes. The opcode determines if the operand is a signed value.

### Instruction Column

The "Instruction" column contains the instruction syntax. The following is a list of the symbols used to represent operands in the instruction statements:

- **imm8**: an immediate byte value, imm8 is a signed number between -128 and +127 inclusive.
- **r/m32**: a doubleword register or memory operand used for instructions whose operand-size attribute is 32 bits.

- **mm/m32**: indicates the lowest 32 bits of an MMX register or a 32-bit memory location.
- **mm/m64**: indicates a 64-bit MMX register or a 64-bit memory location.

### Description Column

The "Description" column briefly explains the instruction activity.

## Operation

The "Operation" section contains an algorithmic description of the operation performed by the instruction.

The register name or memory location implies the contents of the register or memory.

The bit values are written from high-order to low-order and indicate the address within the register or memory. The bit addresses are specified along with the register name or memory location in brackets. For example mm(7..0) represents the low-order 8 bits in an MMX register.

The algorithms are composed of the following elements:

- Comments are enclosed with the symbol pairs “(*” and “*)”.
- Compound statements are enclosed between the keywords of the “if” statement (IF, THEN, ELSE).
- A ← B; indicates that the value of B is assigned to A.
- The symbols =, <>, >, <, ≥, and ≤ are relational operators used to compare two values, meaning equal, not equal, greater or equal, less or equal, respectively. A relational expression such as A=B is TRUE if the value for A is equal to B; otherwise it is FALSE.

The following functions are used in the algorithmic descriptions:

- **ZeroExtend (value)** returns a value zero-extended to the operand-size attribute of the instruction. For example, if OperandSize = 32, ZeroExtend of a byte value of -10 converts the byte from F6H to doubleword with hexadecimal value 000000F6H. If the value passed to ZeroExtend and the operand-size attribute are the same size, ZeroExtend returns the value unaltered.
- **SignExtend (value)** returns a value sign-extended to the operand-size attribute of the instruction. For example, if OperandSize = 32, SignExtend of a byte containing the value -10 converts the byte from F6H to doubleword with hexadecimal value FFFFFFF6H. If the value passed to SignExtend and the operand-size attribute are the same size, SignExtend returns the value unaltered.

- **SaturateSignedWordToSignedByte** converts a signed 16-bit value to a signed 8-bit value. If the signed 16-bit value is less than -128, it is represented by the saturated value -128 (0x80). If it is greater than 127, it is represented by the saturated value 127 (0x7F).
- **SaturateSignedDwordToSignedWord** converts a signed 32-bit value to a signed 16-bit value. If the signed 32-bit value is less than -32768, it is represented by the saturated value -32768 (0x8000). If it is greater than 32767, it is represented by the saturated value 32767 (0x7FFF).
- **SaturateSignedWordToUnsignedByte** converts a signed 16-bit value to an unsigned 8-bit value. If the signed 16-bit value is less than zero it is represented by the saturated value zero (0x00).If it is greater than 255 it is represented by the saturated value 255 (0xFF).
- **SaturateToSignedByte** represents the result of an operation as a signed 8-bit value. If the result is less than -128, it is represented by the saturated value -128 (0x80). If it is greater than 127, it is represented by the saturated value 127 (0x7F).
- **SaturateToSignedWord** represents the result of an operation as a signed 16-bit value. If the result is less than -32768, it is represented by the saturated value -32768 (0x8000).If it is greater than 32767, it is represented by the saturated value 32767 (0x7FFF).
- **SaturateToUnsignedByte** represents the result of an operation as a signed 8-bit value. If the result is less than zero it is represented by the saturated value zero (0x00). If it is greater than 255, it is represented by the saturated value 255 (0xFF).
- **SaturateToUnsignedWord** represents the result of an operation as a signed 16-bit value. If the result is less than zero it is represented by the saturated value zero (0x00).I If it is greater than 65535, it is represented by the saturated value 65535 (0xFFFF).

### Description

The "Description" section describes the operation for all variations of the instruction.

### Example

The “Example” section contains a graphical representation of the instruction's functional behavior.

### Exceptions

The "Exceptions" section lists the exceptions in the three different modes: Protected mode, Real Address mode, and Virtual-8086 mode.

Refer to Section 4.2 of this document for more detail on these exceptions. See also the *Pentium® Processor Family Manual, Volume 3*, Section 9.4 and Chapter 14.

# EMMS—Empty MMX™ State

| Opcode | Instruction | Description |
|---|---|---|
| 0F 77 | EMMS | Set the FP tag word to empty. |

## Operation

TW ← 0xFFFF;

## Description

The EMMS instruction sets the values of the floating-point (FP) tag word to empty (all ones). EMMS marks the registers as available, so they can subsequently be used by floating-point instructions.

If a floating-point instruction loads into one of the registers before it has been reset by the EMMS instruction, a floating-point stack overflow can occur, which results in an FP exception or incorrect result.

All other MMX instructions validate the entire FP tag word (all zeros).

NOTE

This instruction must be used to clear the MMX state at the end of all MMX routines, and before calling other routines that may execute floating-point instructions.

Figure 5-1 shows the format of the FP Tag Word.

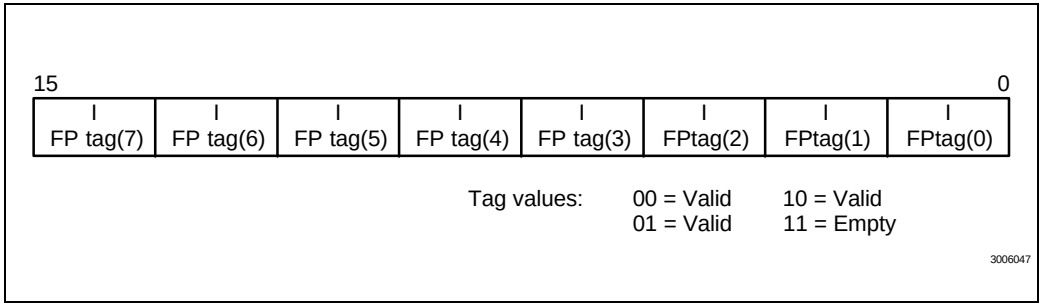



Figure 5-1. Floating Point Tag Word Format

## Flags Affected

None.

### Protected Mode Exceptions

#UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

#UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

#UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

# MOVD—Move 32 Bits

| Opcode | Instruction | Description |
|---|---|---|
| 0F 6E /r | MOVD mm, r/m32 | Move 32 bits from integer register/memory to MMX register. |
| 0F 7E /r | MOVD r/m32, mm | Move 32 bits from MMX register to integer register/memory. |

## Operation

```
IF destination = mm
THEN
   mm(63..0) ← ZeroExtend(r/m32);
ELSE
   r/m32 ← mm(31..0);
```

## Description

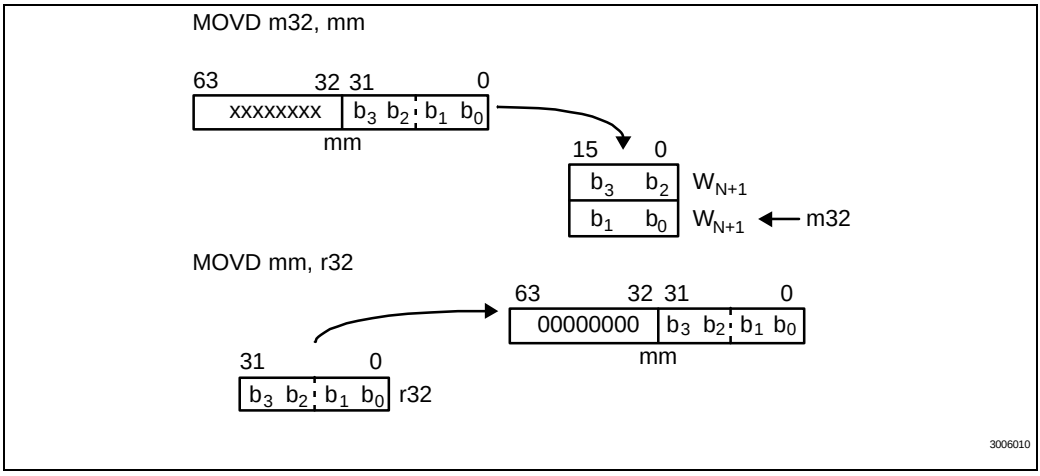
The MOVD instruction copies 32 bits from the source operand to the destination operand.

The destination and source operands can be either MMX registers, 32-bit memory operands, or 32-bit integer registers. The MOVD cannot transfer data from an MMX register to an MMX register, from memory to memory, or from an integer register to an integer register.

When the destination operand is an MMX register, the 32-bit source operand is written to the low-order 32 bits of the 64-bit destination register. The destination register is zero-extended to 64 bits.

When the source operand is an MMX register, the low-order 32 bits of the MMX register are written to the 32-bit integer register or 32-bit memory location.

Example




## Flags Affected

None.

## Protected Mode Exceptions

#GP(0) if the destination is in a nonwritable segment; #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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# MOVQ—Move 64 Bits

| Opcode | Instruction | Description |
|---|---|---|
| 0F 6F /r | MOVQ mm, mm/m64 | Move 64 bits from MMX register/memory to MMX register. |
| 0F 7F /r | MOVQ mm/m64, mm | Move 64 bits from MMX register to MMX register/memory. |

## Operation

```
IF destination = mm
THEN
   mm ← mm/m64;
ELSE
   mm/m64 ← mm;
```

## Description

The MOVQ instruction copies 64 bits from the source operand to the destination operand.

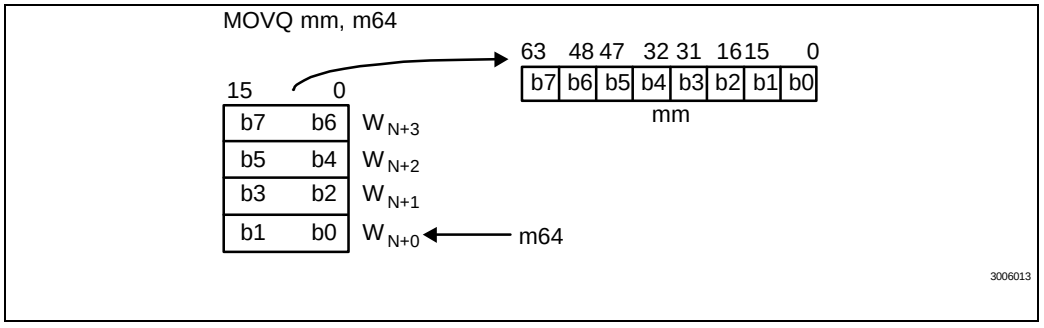
The destination and source operands can be either MMX registers or 64-bit memory operands. The MOVQ instruction cannot transfer data from memory to memory.

When the destination is an MMX register and the source is a 64-bit memory operand, the 64 bits of data at the memory location are copied into the MMX register.

When the destination is a 64-bit memory operand and the source is an MMX register, the 64 bits of data are copied from the MMX register into the memory location.

When the destination and source are both MMX registers, the contents of the MMX register (source) are copied into an MMX register (destination).

## Example



### Flags Affected

None.

### Protected Mode Exceptions

#GP(0) if the destination is in a nonwritable segment; #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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PACKSSWB /PACKSSDW —Pack with Signed Saturation

| Opcode | Instruction | Description |
|---|---|---|
| 0F 63 /r | PACKSSWB mm, mm/m64 | Pack and saturate signed words from MMX register and MMX register/memory into signed bytes in MMX register. |
| 0F 6B /r | PACKSSDW mm, mm/m64 | Pack and saturate signed dwords from MMX register and MMX register/memory into signed words in MMX register. |

## Operation

```
IF instruction is PACKSSWB
THEN {
   mm(7..0) ← SaturateSignedWordToSignedByte mm(15..0);
   mm(15..8) ← SaturateSignedWordToSignedByte mm(31..16);
   mm(23..16) ← SaturateSignedWordToSignedByte mm(47..32);
   mm(31..24) ← SaturateSignedWordToSignedByte mm(63..48);
   mm(39..32) ← SaturateSignedWordToSignedByte mm/m64(15..0);
   mm(47..40) ← SaturateSignedWordToSignedByte mm/m64(31..16);
   mm(55..48) ← SaturateSignedWordToSignedByte mm/m64(47..32);
   mm(63..56) ← SaturateSignedWordToSignedByte mm/m64(63..48);
   }
ELSE { (* instruction is PACKSSDW *)
   mm(15..0) ← SaturateSignedDwordToSignedWord mm(31..0);
   mm(31..16) ← SaturateSignedDwordToSignedWord mm(63..32);
   mm(47..32) ← SaturateSignedDwordToSignedWord mm/m64(31..0);
   mm(63..48) ← SaturateSignedDwordToSignedWord mm/m64(63..32);
   }
```

## Description

The PACKSS instruction packs and saturates the signed data elements from the source and the destination operands and writes the signed results to the destination operand.

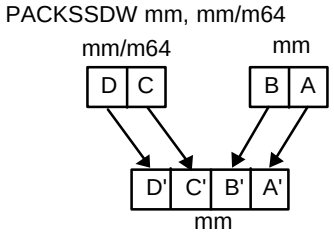
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PACKSSWB instruction packs four signed words from the source operand and four signed words from the destination operand into eight signed bytes in the destination register. If the signed value of a word is larger or smaller than the range of a signed byte, the value is saturated (in the case of an overflow - to 0x7F, and in the case of an underflow - to 0x80).

The PACKSSDW instruction packs two signed doublewords from the source operand and two signed doublewords from the destination operand into four signed words in the destination register. If the signed value of a doubleword is larger or smaller than the range of

a signed word, the value is saturated (in the case of an overflow - to 0x7FFF, and in the case of an underflow - to 0x8000).

### Example




### Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PACKUSWB —Pack with Unsigned Saturation

| Opcode | Instruction | Description |
|---|---|---|
| 0F 67 /r | PACKUSWB mm, mm/m64 | Pack and saturate signed words from MMX register and MMX register/memory into unsigned bytes in MMX register. |

## Operation

```
mm(7..0) ← SaturateSignedWordToUnsignedByte mm(15..0);
mm(15..8)← SaturateSignedWordToUnsignedByte mm(31..15);
mm(23..16) ← SaturateSignedWordToUnsignedByte mm(47..32);
mm(31..24) ← SaturateSignedWordToUnsignedByte mm(63..48);
mm(39..32) ← SaturateSignedWordToUnsignedByte mm/m64(15..0);
mm(47..40) ← SaturateSignedWordToUnsignedByte mm/m64(31..16);
mm(55..48) ← SaturateSignedWordToUnsignedByte mm/m64(47..32);
mm(63..56) ← SaturateSignedWordToUnsignedByte mm/m64(63..48);
```

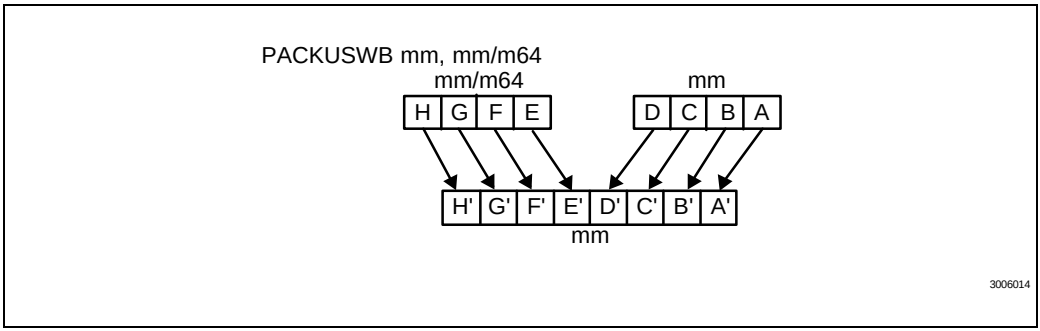
## Description:

The PACKUSWB packs and saturates four signed words of the source operand and four signed words of the destination operand into eight unsigned bytes. The result is written to the destination operand

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

If the signed value of the word is larger or smaller than the range of an unsigned byte, the value is saturated (in the case of an overflow - to 0xFF and in the case of an underflow - to 0x00).

Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PADDB/PADDW/PADDD —Packed Add

| Opcode | Instruction | Description |
|---|---|---|
| 0F FC /r | PADDB mm, mm/m64 | Add packed byte from MMX register/memory to packed byte in MMX register. |
| 0F FD /r | PADDW mm, mm/m64 | Add packed word from MMX register/memory to packed word in MMX register. |
| 0F FE /r | PADDD mm, mm/m64 | Add packed dword from MMX register/memory to packed dword in MMX register. |

## Operation

```
IF instruction is PADDB
THEN  {
   mm(7..0) ← mm(7..0) + mm/m64(7..0);
   mm(15..8) ← mm(15..8) + mm/m64(15..8);
   mm(23..16) ← mm(23..16)+ mm/m64(23..16);
   mm(31..24) ← mm(31..24) + mm/m64(31..24);
   mm(39..32) ← mm(39..32) + mm/m64(39..32);
   mm(47..40) ← mm(47..40)+ mm/m64(47..40);
   mm(55..48) ← mm(55..48) + mm/m64(55..48);
   mm(63..56) ← mm(63..56) + mm/m64(63..56);
   }
IF instruction is PADDW
THEN {
   mm(15..0) ← mm(15..0) + mm/m64(15..0);
   mm(31..16) ← mm(31..16) + mm/m64(31..16);
   mm(47..32) ← mm(47..32) + mm/m64(47..32);
   mm(63..48) ← mm(63..48) + mm/m64(63..48);
   }
ELSE { (* instruction is PADDD *)
   mm(31..0) ← mm(31..0) + mm/m64(31..0);
   mm(63..32) ← mm(63..32) + mm/m64(63..32);
   }
```

## Description

The PADD instructions add the data elements of the source operand to the data elements of the destination register. The result is written to the destination register. If the result exceeds the data-range limit for the data type, it wraps around.

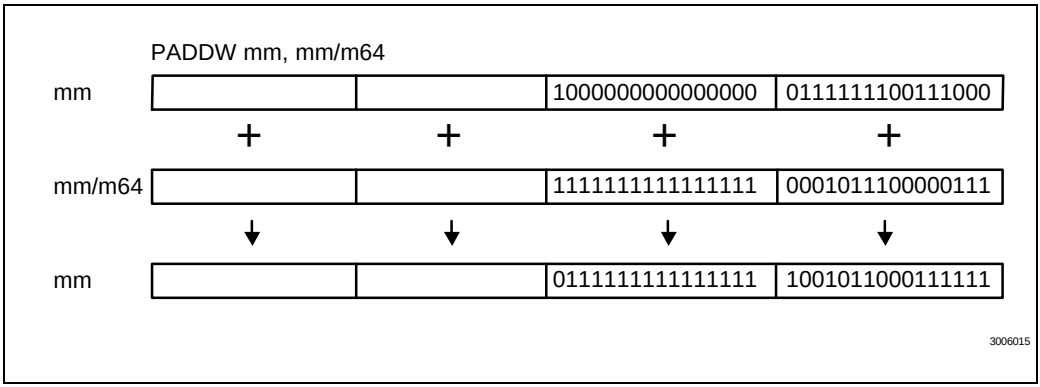
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PADDB instruction adds the bytes of the source operand to the bytes of the destination operand and writes the results to the MMX register. When the result is too large to be represented in a packed byte (overflow), the result wraps around and the lower 8 bits are written to the destination register.

The PADDW instruction adds the words of the source operand to the words of the destination operand and writes the results to the MMX register. When the result is too large to be represented in a packed word (overflow), the result wraps around and the lower 16 bits are written to the destination register.

The PADDD instruction adds the doublewords of the source operand to the doublewords of the destination operand and writes the results to the MMX register. When the result is too large to be represented in a packed doubleword (overflow), the result wraps around and the lower 32 bits are written to the destination register.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PADDSB/PADDSW —Packed Add with Saturation

| Opcode | Instruction | Description |
|---|---|---|
| 0F EC /r | PADDSB mm, mm/m64 | Add signed packed byte from MMX register/memory to signed packed byte in MMX register and saturate. |
| 0F ED /r | PADDSW mm, mm/m64 | Add signed packed word from MMX register/memory to signed packed word in MMX register and saturate. |

## Operation

```
IF instruction is PADDSB
THEN{
   mm(7..0) ← SaturateToSignedByte (mm(7..0) + mm/m64 (7..0)) ;
   mm(15..8) ← SaturateToSignedByte (mm(15..8) + mm/m64(15..8) );
   mm(23..16) ← SaturateToSignedByte (mm(23..16)+ mm/m64(23..16) );
   mm(31..24) ← SaturateToSignedByte (mm(31..24) + mm/m64(31..24) );
   mm(39..32) ← SaturateToSignedByte (mm(39..32) + mm/m64(39..32) );
   mm(47..40) ← SaturateToSignedByte (mm(47..40)+ mm/m64(47..40) );
   mm(55..48) ← SaturateToSignedByte (mm(55..48) + mm/m64(55..48) );
   mm(63..56) ← SaturateToSignedByte (mm(63..56) + mm/m64(63..56) );
   }
ELSE { (* instruction is PADDW *)
   mm(15..0) ← SaturateToSignedWord (mm(15..0) + mm/m64(15..0) );
   mm(31..16) ← SaturateToSignedWord (mm(31..16) + mm/m64(31..16) );
   mm(47..32) ← SaturateToSignedWord (mm(47..32) + mm/m64(47..32) );
   mm(63..48) ← SaturateToSignedWord (mm(63..48) + mm/m64(63..48) );
   }
```

## Description

The PADDS instructions add the packed signed data elements of the source operand to the packed signed data elements of the destination operand and saturate the result. The result is written to the destination operand.

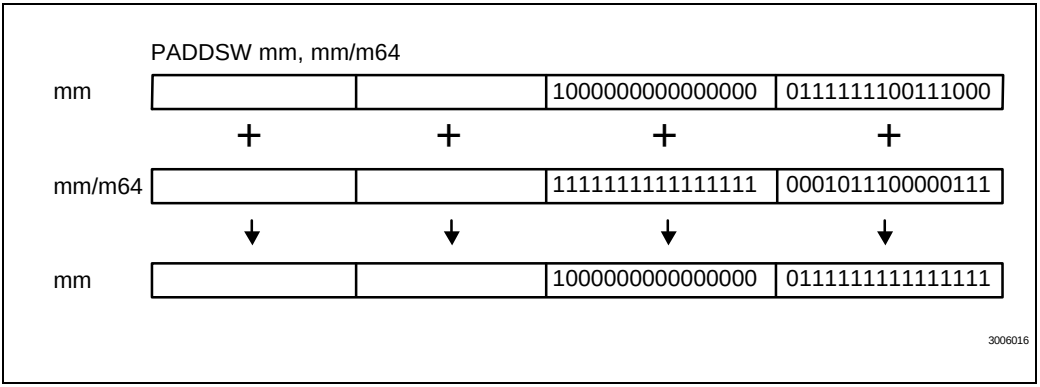
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PADDSB instruction adds the signed bytes of the source operand to the signed bytes of the destination operand and writes the results to the MMX register. If the result is larger or smaller than the range of a signed byte, the value is saturated (in the case of an overflow - to 0x7F, and in the case of an underflow - to 0x80).

The PADDSW instruction adds the signed words of the source operand to the signed words of the destination operand and writes the results to the MMX register. If the result is larger or

smaller than the range of a signed word, the value is saturated (in the case of an overflow - to 0x7FFF, and in the case of an underflow - to 0x8000) .

### Example




### Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PADDUSB/PADDUSW —Packed Add Unsigned with Saturation

| Opcode | Instruction | Description |
|---|---|---|
| 0F DC /r | PADDUSB mm, mm/m64 | Add unsigned packed byte from MMX register/memory to unsigned packed byte in MMX register and saturate. |
| 0F DD /r | PADDUSW mm, mm/m64 | Add unsigned packed word from MMX register/memory to unsigned packed word in MMX register and saturate. |

## Operation

```
IF instruction is PADDUSB
THEN{
   mm(7..0) ← SaturateToUnsignedByte (mm(7..0) + mm/m64 (7..0) );
   mm(15..8) ← SaturateToUnsignedByte (mm(15..8) + mm/m64(15..8) );
   mm(23..16) ← SaturateToUnsignedByte (mm(23..16)+ mm/m64(23..16) );
   mm(31..24) ← SaturateToUnsignedByte (mm(31..24) + mm/m64(31..24) );
   mm(39..32) ← SaturateToUnsignedByte (mm(39..32) + mm/m64(39..32) );
   mm(47..40) ← SaturateToUnsignedByte (mm(47..40)+ mm/m64(47..40) );
   mm(55..48) ← SaturateToUnsignedByte (mm(55..48) + mm/m64(55..48) );
   mm(63..56) ← SaturateToUnsignedByte (mm(63..56) + mm/m64(63..56) );
   }
ELSE { (* instruction is PADDUSW *)
   mm(15..0) ← SaturateToUnsignedWord (mm(15..0) + mm/m64(15..0) );
   mm(31..16) ← SaturateToUnsignedWord (mm(31..16) + mm/m64(31..16) );
   mm(47..32) ← SaturateToUnsignedWord (mm(47..32) + mm/m64(47..32) );
   mm(63..48) ← SaturateToUnsignedWord (mm(63..48) + mm/m64(63..48) );
   }
```

## Description

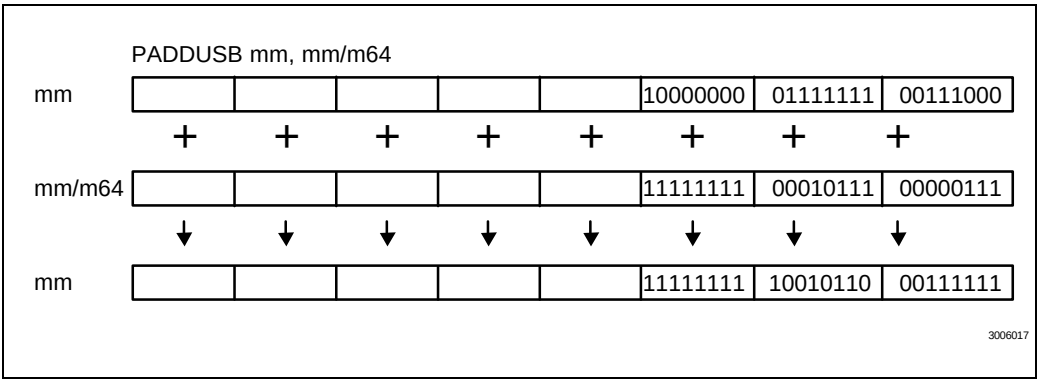
The PADDUS instructions add the packed unsigned data elements of the source operand to the packed unsigned data elements of the destination operand and saturate the results. The results are written to the destination operand.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PADDUSB instruction adds the unsigned bytes of the source operand to the unsigned bytes of the destination operand and writes the results to the MMX register. When the result is larger than the range of an unsigned byte (overflow), the value is saturated to 0xFF. When the result is smaller than the range of an unsigned byte (underflow), the value is saturated to 0x00.

The PADDUSW instruction adds the unsigned words of the source operand to the unsigned words of the destination operand and writes the results to the MMX register. When the result is larger than the range of an unsigned word (overflow), the value is saturated to 0xFFFF. When the result is smaller than the range of an unsigned word (underflow), the value is saturated to 0x0000.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PAND—Bitwise Logical And

| Opcode | Instruction | Description |
|---|---|---|
| 0F DB /r | PAND mm, mm/m64 | AND 64 bits from MMX register/memory to MMX register. |

## Operation

mm ←mm AND mm/m64;

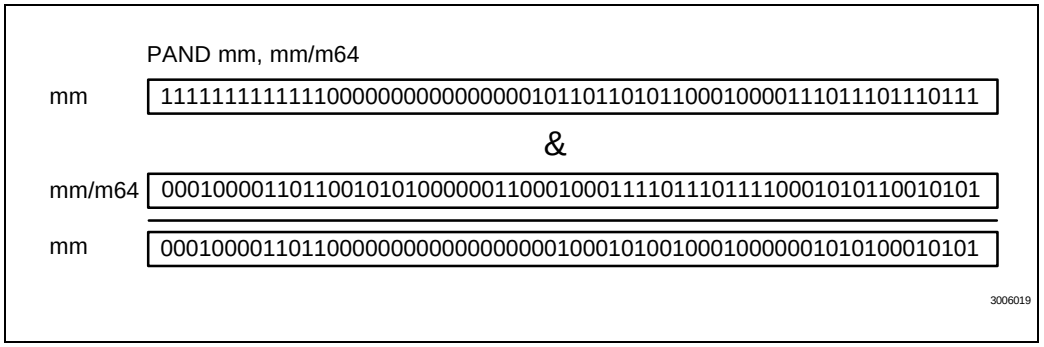
### Description

The PAND instruction performs a bitwise logical AND on 64 bits of the source and destination operands, and writes the result to the destination operand.

Each bit of the result of the PAND instruction is set to 1 if the corresponding bits of the operands are 1. Otherwise, it is set to 0.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

### Example




### Flags Affected

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;

#AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PANDN—Bitwise Logical And Not

| Opcode | Instruction | Description |
|---|---|---|
| 0F DF /r | PANDN mm, mm/m64 | Invert the 64 bits in MMX register, AND inverted MMX register with MMX register/memory. |

## Operation

mm ←(NOT mm) AND mm/m64;

## Description

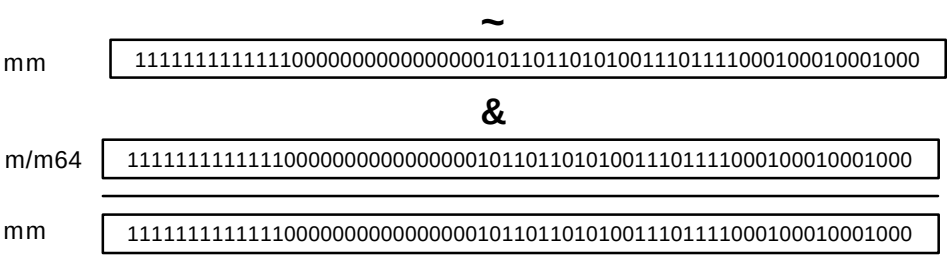
The PANDN instruction performs a bitwise logical NOT on the 64 bits of the destination operand. The NOT inverts each of the 64 bits of the destination register so that every 1 becomes a 0, and visa versa.

The instruction then performs a bitwise logical AND on the inverted 64 bits of the destination operand and on the source operand. Each bit of the result of the AND instruction is set to 1 if the corresponding bits are 1. Otherwise, it is set to 0. The result is written to the destination register.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

## Example

PANDN mm, mm/m64




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PCMPEQB/PCMPEQW/PCMPEQD —Packed Compare for Equal

| Opcode | Instruction | Description |
|---|---|---|
| 0F 74 /r | PCMPEQB mm, mm/m64 | Compare packed byte in MMX register/memory with packed byte in MMX register for equality. |
| 07, 75, /r | PCMPEQW mm, mm/m64 | Compare packed word in MMX register/memory with packed word in MMX register for equality. |
| 07, 76, /r | PCMPEQD mm, mm/m64 | Compare packed dword in MMX register/memory with packed dword in MMX register for equality. |

## Operation

```
IF instruction is PCMPEQB
THEN {
   IF mm(7..0) = mm/m64(7..0)
   THEN mm(7  0) ← 0xFF;
   ELSE mm(7..0) ← 0;
   IF mm(15..8) = mm/m64(15.. 8)
   THEN mm(15..8) ← 0xFF;
   ELSE mm(15..8) ← 0;
   ...
   IF mm(63..56) = mm/m64(63..56)
   THEN mm(63..56) ← 0xFF;
   ELSE mm(63..56) ← 0;
   }
ELSE IF instruction is PCMPEQW
THEN {
   IF mm(15..0) = mm/m64(15..0)
   THEN mm(15..0) ← 0xFFFF;
   ELSE mm(15..0) ← 0;
   IF mm(31..16) = mm/m64(31..16)
   THEN mm(31..16) ← 0xFFFF;
   ELSE mm(31..16) ← 0;
   ...
   IF mm(63..48) = mm/m64(63..48)
   THEN mm(63..48) ← 0xFFFF;
   ELSE mm(63..48) ← 0;
   }
```

```
ELSE {     (* instruction is PCMPEQD *)
   IF mm(31..0) = mm/m64(31..0)
   THEN mm(31..0) ← 0xFFFFFFFF;
   ELSE mm(31..0) ← 0;
   IF mm(63..32) = mm/m64(63..32)
   THEN mm(63..32) ← 0xFFFFFFFF;
   ELSE mm(63..32) ← 0;
   }
```

## Description

The PCMPEQ instructions compare the data elements in the destination operand to the corresponding data elements in the source operand. If the data elements are equal, the corresponding data element in the destination register is set to all ones. If they are not equal, the corresponding data element is set to all zeros.

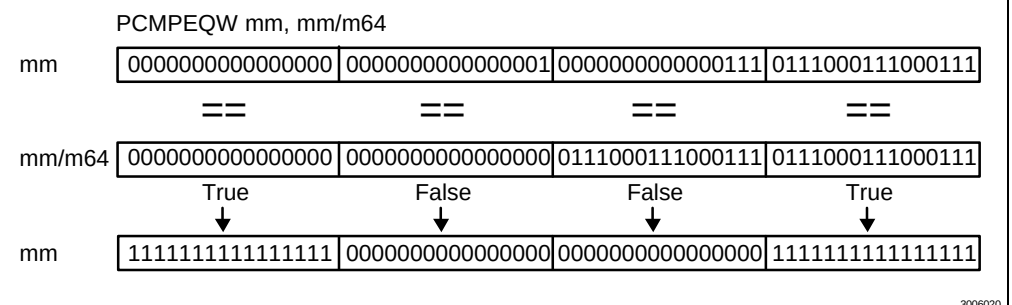
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PCMPEQB instruction compares the bytes in the destination operand to the bytes in the source operand. The bytes in the destination operand are set accordingly.

The PCMPEQW instruction compares the words in the destination operand to the words in the source operand. The words in the destination operand are set accordingly.

The PCMPEQD instruction compares the doublewords in the destination operand to the doublewords in the source operand. The doublewords in the destination operand are set accordingly.

**Example**




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PCMPGTB/PCMPGTW/PCMPGTD —Packed Compare for Greater Than

| Opcode | Instruction | Description |
|---|---|---|
| 0F 64 /r | PCMPGTB mm, mm/m64 | Compare packed byte in MMX register with packed byte in MMX register/memory for greater value. |
| 0F 65 /r | PCMPGTW mm, mm/m64 | Compare packed word in MMX register with packed word in MMX register/memory for greater value. |
| 0F 66 /r | PCMPGTD mm, mm/m64 | Compare packed dword in MMX register with packed dword in MMX register/memory for greater value. |

## Operation

```
IF instruction is PCMPGTB
THEN {
   IF mm(7..0) > mm/m64(7..0)
   THEN mm(7  0) ← 0xFF;
   ELSE mm(7..0) ← 0;
   IF mm(15..8) > mm/m64(15.. 8)
   THEN mm(15..8) ← 0xFF;
   ELSE mm(15..8) ← 0;
   ...
   IF mm(63..56) > mm/m64(63..56)
   THEN mm(63..56) ← 0xFF;
   ELSE mm(63..56) ← 0;
   }
 ELSE IF instruction is PCMPGTW
THEN {
   IF mm(15..0) > mm/m64(15..0)
   THEN mm(15..0) ← 0xFFFF;
   ELSE mm(15..0) ←0;
   IF mm(31..16) > mm/m64(31..16)
   THEN mm(31..16) ← 0xFFFF;
   ELSE mm(31..16) ← 0;
   ...
   IF mm(63..48) > mm/m64(63..48)
   THEN mm(63..48) ← 0xFFFF;
   ELSE mm(63..48) ← 0;
   }
ELSE { (* instruction is PCMPGTD *)
   IF mm(31..0) > mm/m64(31..0)
   THEN mm(31..0) ← 0xFFFFFFFF;
   ELSE mm(31..0) ← 0;
   IF mm(63..32) > mm/m64(63..32)
```

```
THEN mm(63..32) ← 0xFFFFFFFF;
ELSE mm(63..32) ← 0;
}
```

## Description

The PCMPGT instructions compare the signed data elements in the destination operand to the signed data elements in the source operand. If the signed data elements in the destination register are greater than those in the source operand, the corresponding data element in the destination operand is set to all ones. Otherwise, it is set to all zeros.

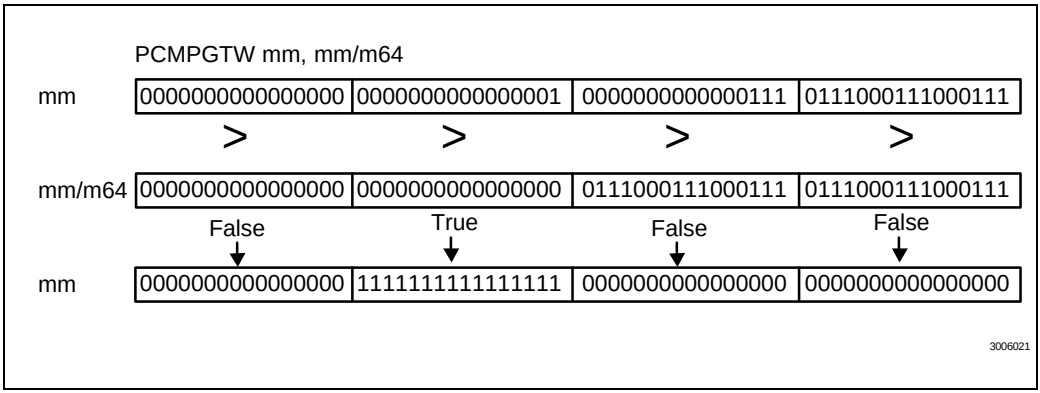
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PCMPGTB instruction compares the signed bytes in the destination operand to the corresponding signed bytes in the source operand. The bytes in the destination register are set accordingly.

The PCMPGTW instruction compares the signed words in the destination operand to the corresponding signed words in the source operand. The words in the destination register are set accordingly.

The PCMPGTD instruction compares the signed doublewords in the destination operand to the corresponding signed words in the source operand. The doublewords in the destination register are set accordingly.

Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PMADDWD —Packed Multiply and Add

| Opcode | Instruction | Description |
|---|---|---|
| 0F F5 /r | PMADDWD mm, mm/m64 | Multiply the packed word in MMX register by the packed word in MMX reg/memory. Add the 32-bit results pairwise and store in MMX register as dword |

## Operation

mm(31..0) ← mm(15..0) * mm/m64(15..0) + mm(31..16) * mm/m64(31..16);
mm(63..32) ← mm(47..32) * mm/m64(47..32) + mm(63..48) * mm/m64(63..48);

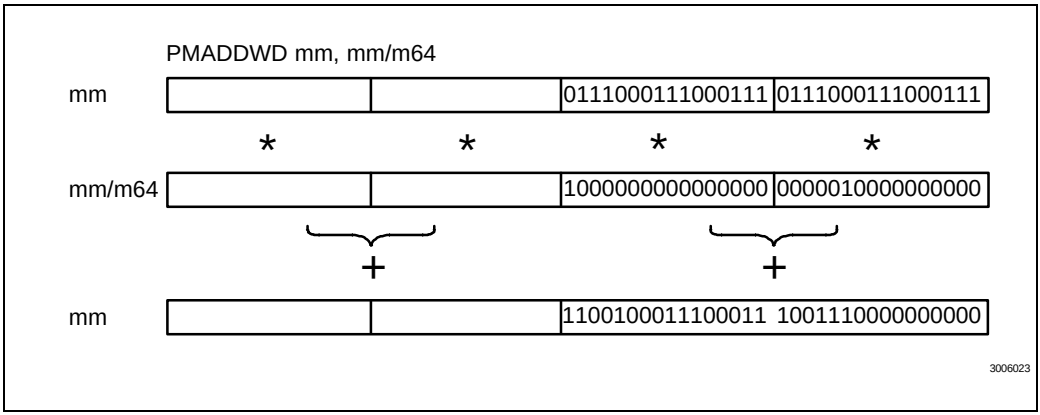
## Description

The PMADDWD instruction multiplies the four signed words of the destination operand by the four signed words of the source operand. The result is two 32-bit doublewords. The two high-order words are summed and stored in the upper doubleword of the destination operand. The two low-order words are summed and stored in the lower doubleword of the destination operand. This result is written to the destination operand.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PMADDWD instruction wraps around to 0x80000000 only when all four words of both the source and destination operands are 0x8000.

Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PMULHW—Packed Multiply High

| Opcode | Instruction | Description |
|---|---|---|
| 0F E5 /r | PMULHW mm, mm/m64 | Multiply the signed packed word in MMX register with the signed packed word in MMX reg/memory, then store the high-order 16 bits of the results in MMX register. |

## Operation

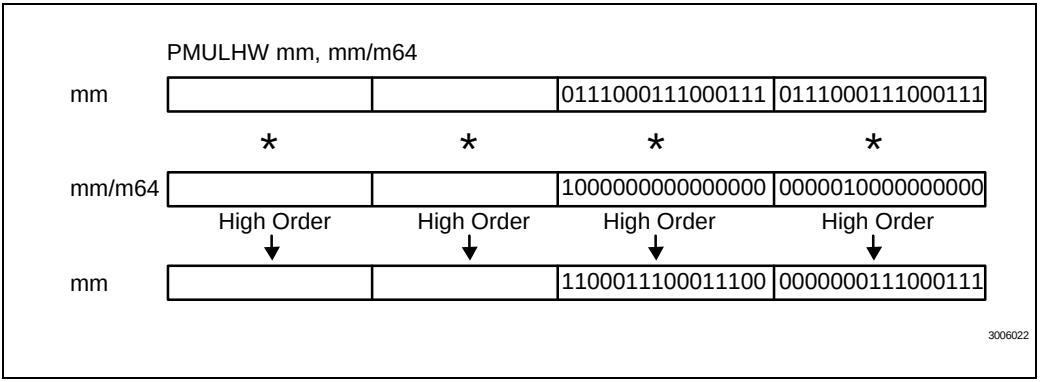
mm(15..0) ← (mm(15..0) * mm/m64(15..0) ) (31..16);
mm(31..16)← (mm(31..16) * mm/m64(31..16) ) (31..16);
mm(47..32) ← (mm(47..32) * mm/m64(47..32) ) (31..16);
mm(63..48) ← (mm(63..48) * mm/m64(63..48) ) (31..16);

## Description

The PMULHW instruction multiplies the four signed words of the destination operand with the four signed words of the source operand. The high-order 16 bits of the 32-bit intermediate results are written to the destination operand.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PMULLW—Packed Multiply Low

| Opcode | Instruction | Description |
|---|---|---|
| 0F D5 /r | PMULLW mm, mm/m64 | Multiply the packed word in MMX register with the packed word in MMX reg/memory, then store the low-order 16 bits of the results in MMX register. |

## Operation

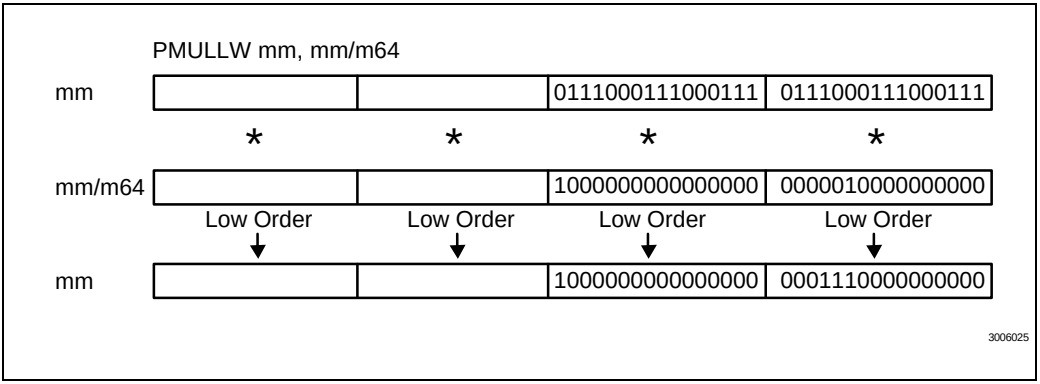
mm(15..0) ← (mm(15..0) * mm/m64(15..0) ) (15..0);
mm(31..16) ← (mm(31..16) * mm/m64(31..16) ) (15..0);
mm(47..32) ← (mm(47..32) * mm/m64(47..32) ) (15..0);
mm(63..48) ← (mm(63..48) * mm/m64(63..48) ) (15..0);

## Description

The PMULLW instruction multiplies the four signed or unsigned words of the destination operand with the four signed or unsigned words of the source operand. The low-order 16 bits of the 32-bit intermediate results are written to the destination operand.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# POR—Bitwise Logical Or

| Opcode | Instruction | Description |
|---|---|---|
| 0F EB /r | POR mm, mm/m64 | OR 64 bits from MMX reg/memory with MMX register. |

## Operation

mm ←mm OR mm/m64;

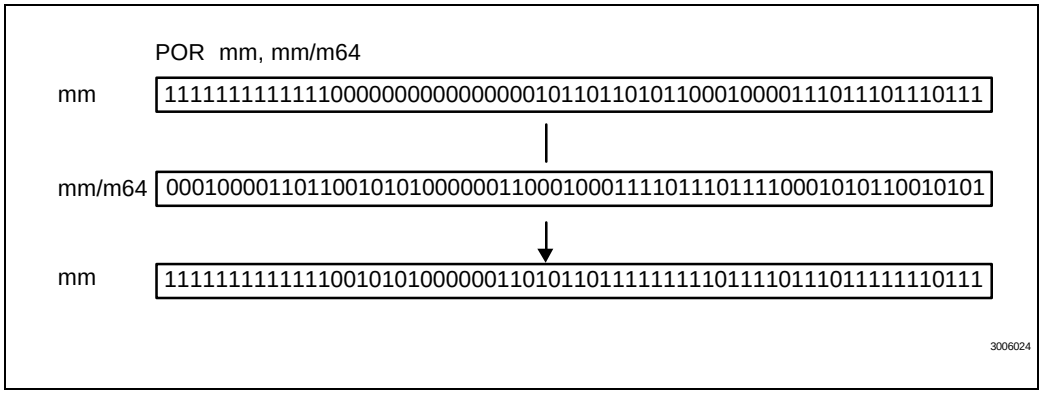
## Description

The POR instruction performs a bitwise logical OR on 64 bits of the destination and source operands, and writes the result to the destination register.

Each bit of the result is set to 0 if the corresponding bits of the two operands are 0. Otherwise, the bit is 1.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PSLLW/PSLLD/PSLLQ —Packed Shift Left Logical

| Opcode | Instruction | Description |
|---|---|---|
| 0F F1 /r | PSLLW mm, mm/m64 | Shift words in MMX register left by amount specified in MMX reg/memory, while shifting in zeros. |
| 0F 71 /6, ib | PSLLW mm, imm8 | Shift words in MMX register left by Imm8, while shifting in zeros. |
| 0F F2 /r | PSLLD mm, mm/m64 | Shift dwords in MMX register left by amount specified in MMX reg/memory, while shifting in zeros. |
| 0F 72 /6 ib | PSLLD mm, imm8 | Shift dwords in MMX register by Imm8, while shifting in zeros.. |
| 0F F3 /r | PSLLQ mm, mm/m64 | Shift MMX register left by amount specified in MMX reg/memory, while shifting in zeros. |
| 0F 73 /6 ib | PSLLQ mm, imm8 | Shift MMX register left by Imm8, while shifting in zeros. |

## Operation

```
IF the second operand is imm8
THEN
   temp ← imm8;
ELSE (* second operand is mm/m64 *)
   temp ← mm/m64;
IF instruction is PSLLW
THEN {
   mm(15..0) ← mm(15..0) << temp;
   mm(31..16) ← mm(31..16) << temp;
   mm(47..32) ← mm(47..32) << temp;
   mm(63..48) ← mm(63..48) << temp;
   }
ELSE IF instruction is PSLLD
THEN {
   mm(31..0) ← mm(31..0) << temp;
   mm(63..32) ← mm(63..32) << temp;
   }
ELSE  (* instruction is PSLLQ *)
   mm ← mm << temp;
```

## Description

The PSLL instructions shift the bits of the first operand to the left by the amount of bits specified in the count operand. The result of the shift operation is written to the destination register. The empty low-order bits are cleared (set to zero). If the value specified by the second operand is greater than 15 (for words), 31 (for doublewords), or 63 (for quadwords), then the destination is set to all zeros.

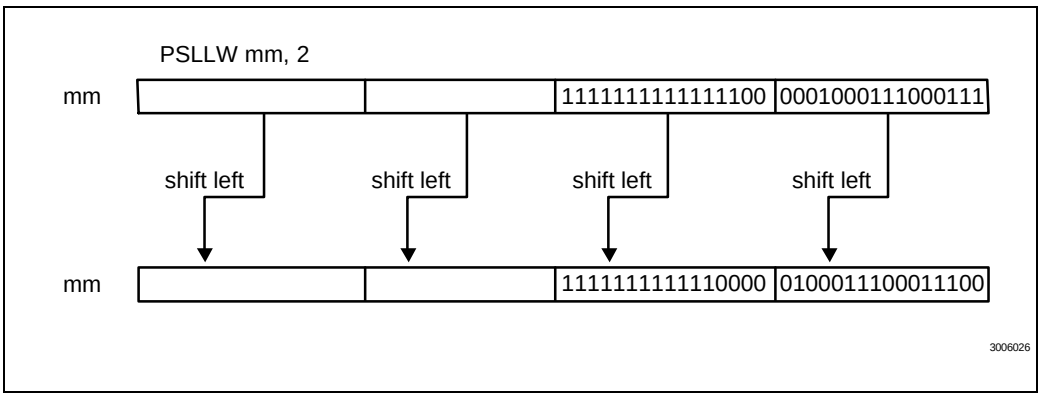
The destination operand is an MMX register. The count operand (source operand) can be either an MMX register, a 64-bit memory operand, or an immediate 8-bit operand.

The PSLLW instruction shifts each of the four words of the destination register to the left by the number of bits specified in the count operand. The low-order bit positions (of each word) are filled with zeros.

The PSLLD instruction shifts each of the two doublewords of the destination register to the left by the number of bits specified in the count operand. The low-order bit positions (of each doubleword) are filled with zeros.

The PSLLQ instruction shifts the 64-bit quadword in the destination register to the left by the number of bits specified in the count operand. The low-order bit positions are filled with zeros.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PSRAW/PSRAD —Packed Shift Right Arithmetic

| Opcode | Instruction | Description |
|---|---|---|
| 0F E1 /r | PSRAW mm, mm/m64 | Shift words in MMX register right by amount specified in MMX reg/memory while shifting in sign bits. |
| 0F 71 /4 ib | PSRAW mm, imm8 | Shift words in MMX register right by Imm8 while shifting in sign bits |
| 0F E2 /r | PSRAD mm, mm/m64 | Shift dwords in MMX register right by amount specified in MMX reg/memory while shifting in sign bits. |
| 0F 72 /4 ib | PSRAD mm, imm8 | Shift dwords in MMX register right by Imm8 while shifting in sign bits. |

## Operation

```
IF the second operand is imm8
THEN
   temp ← imm8;
ELSE (* second operand is mm/m64 *)
   temp ← mm/m64;
IF instruction is PSRAW
THEN {
   mm(15..0) ← SignExtend (mm(15..0) >>temp);
   mm(31..16) ← SignExtend (mm(31..16) >> temp);
   mm(47..32) ← SignExtend (mm(47..32) >> temp);
   mm(63..48) ← SignExtend (mm(63..48) >> temp);
   }
ELSE { (*instruction is PSRAD *)
   mm(31..0) ← SignExtend (mm(31..0) >> temp);
   mm(63..32) ← SignExtend (mm(63..32) >> temp);
   }
```

## Description

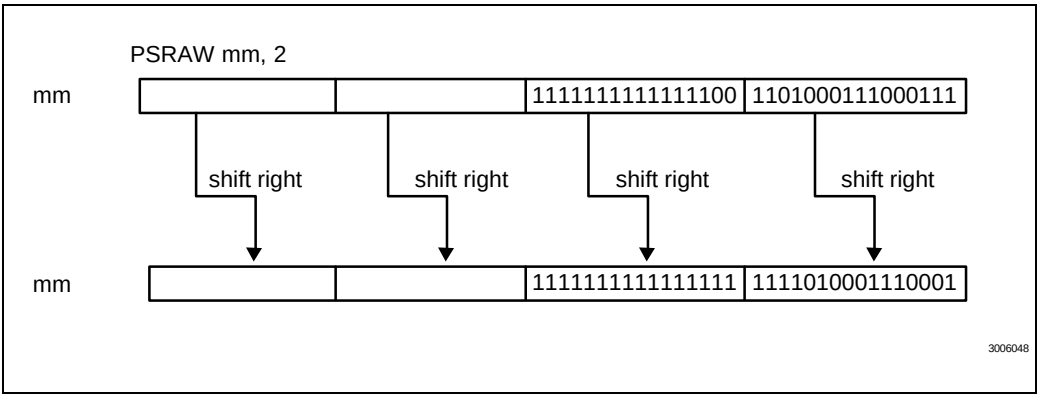
The PSRA instructions shift the bits of the first operand to the right by the amount of bits specified in the count operand. The result of the shift operation is written to the destination register. The empty high-order bits of each element are filled with the initial value of the sign bit of the data element. If the value specified by the second operand is greater than 15 (for words), or 31 (for doublewords), each destination element is filled with the initial value of the sign bit of the element.

The destination operand is an MMX register. The count operand (source operand) can be either an MMX register, a 64-bit memory operand, or an immediate 8-bit operand.

The PSRAW instruction shifts each of the four words in the destination register to the right by the number of bits specified in the count operand. The initial value of the sign bit of the data elements in the destination operand is copied into the most significant bits of the data element.

The PSRAD instruction shifts each of the two doublewords in the destination register to the right by the number of bits specified in the count operand. The initial value of the sign bit of the data elements in the destination operand is copied into the most significant bits of the data element.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PSRLW/PSRLD/PSRLQ —Packed Shift Right Logical

| Opcode | Instruction | Description |
|---|---|---|
| 0F D1 /r | PSRLW mm, mm/m64 | Shift words in MMX register right by amount specified in MMX reg/memory while shifting in zeros. |
| 0F 71 /2 ib | PSRLW mm, imm8 | Shift words in MMX register right by Imm8. |
| 0F D2 /r | PSRLD mm, mm/m64 | Shift dwords in MMX register right by amount specified in MMX reg/memory while shifting in zeros. |
| 0F 72 /2 ib | PSRLD mm, imm8 | Shift dwords in MMX register right by Imm8 . |
| 0F D3 /r | PSRLQ mm, mm/m64 | Shift MMX register right by amount specified in MMX reg/memory while shifting in zeros. |
| 0F 73 /2 ib | PSRLQ mm, imm8 | Shift MMX register right by Imm8 while shifting in zeros. |

## Operation

```
IF the second operand is imm8
THEN
   temp ← imm8;
ELSE (* second operand is mm/m64 *)
   temp ← mm/m64;
IF instruction is PSRLW
THEN {
   mm(15..0) ← mm(15..0) >> temp;
   mm(31..16) ← mm(31..16) >> temp;
   mm(47..32) ← mm(47..32) >> temp;
   mm(63..48) ← mm(63..48) >> temp;
   }
ELSE IF instruction is PSRLD
THEN {
   mm(31..0) ← mm(31..0) >> temp;
   mm(63..32) ← mm(63..32) >> temp;
   }
ELSE  (* instruction is PSRLQ *)
   mm ← mm >> temp;
```

## Description

The PSRL instructions shift the bits of the first operand to the right by the amount of bits specified in the count operand. The result of the shift operation is written to the destination register. The empty high-order bits are cleared (set to zero). If the value specified by the second operand is greater than 15 (for words), or 31 (for doublewords), or 63 (for quadwords), then the destination is set to all zeros.

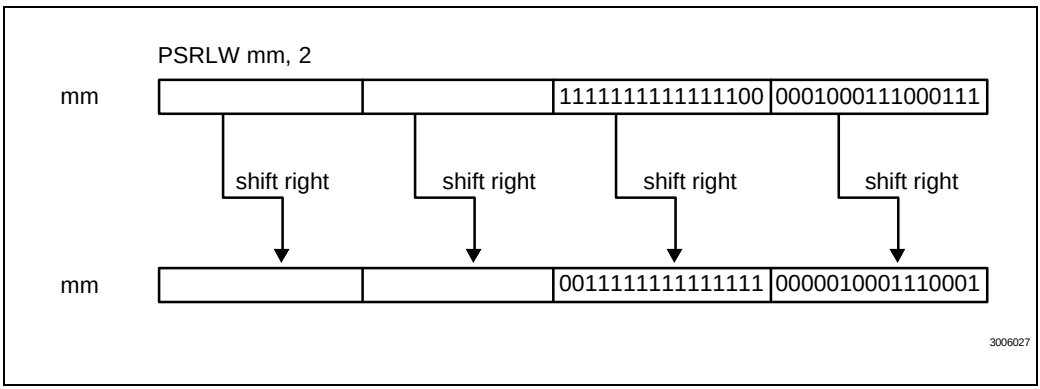
The destination operand is an MMX register. The count operand (source operand) can be either an MMX register, a 64-bit memory operand, or an immediate 8-bit operand.

The PSRLW instruction shifts each of the four words in the destination register to the right by the number of bits specified in the count operand. The empty high-order bits (of each word) are filled with zeros.

The PSLLD instruction shifts each of the two doublewords in the destination register to the right by the number of bits specified in the count operand. The empty high-order bits (of each doubleword) are filled with zeros.

The PSLLQ instruction shifts the 64-bit quadword in the destination register to the right by the number of bits specified in the count operand. The empty high-order bits are filled with zeros.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PSUBB/PSUBW/PSUBD —Packed Subtract

| Opcode | Instruction | Description |
|---|---|---|
| 0F F8 /r | PSUBB mm, mm/m64 | Subtract packed byte in MMX reg/memory from packed byte in MMX register. |
| 0F F9 /r | PSUBW mm, mm/m64 | Subtract packed word in MMX reg/memory from packed word in MMX register. |
| 0F FA /r | PSUBD mm, mm/m64 | Subtract packed dword in MMX reg/memory from packed dword in MMX register. |

## Operation

```
IF instruction is PSUBB
 THEN  {
   mm(7..0) ← mm(7..0) - mm/m64(7..0);
   mm(15..8) ← mm(15..8) - mm/m64(15..8);
   mm(23..16) ← mm(23..16) - mm/m64(23..16);
   mm(31..24) ← mm(31..24) - mm/m64(31..24);
   mm(39..32) ← mm(39..32) - mm/m64(39..32);
   mm(47..40) ← mm(47..40) - mm/m64(47..40);
   mm(55..48) ← mm(55..48) - mm/m64(55..48);
   mm(63..56) ← mm(63..56) - mm/m64(63..56);
   }
IF instruction is PSUBW
THEN {
   mm(15..0) ← mm(15..0) - mm/m64(15..0);
   mm(31..16) ← mm(31..16) - mm/m64(31..16);
   mm(47..32) ← mm(47..32) - mm/m64(47..32);
   mm(63..48) ← mm(63..48) - mm/m64(63..48);
   }
ELSE { (* instruction is PSUBD *)
   mm(31..0) ← mm(31..0) - mm/m64(31..0);
   mm(63..32) ← mm(63..32) - mm/m64(63..32);
   }
```

## Description

The PSUB instructions subtract the data elements of the source operand from the data elements of the destination operand. The result is written to the destination register. If the result is larger or smaller than the data-range limit for the data type, it wraps around.

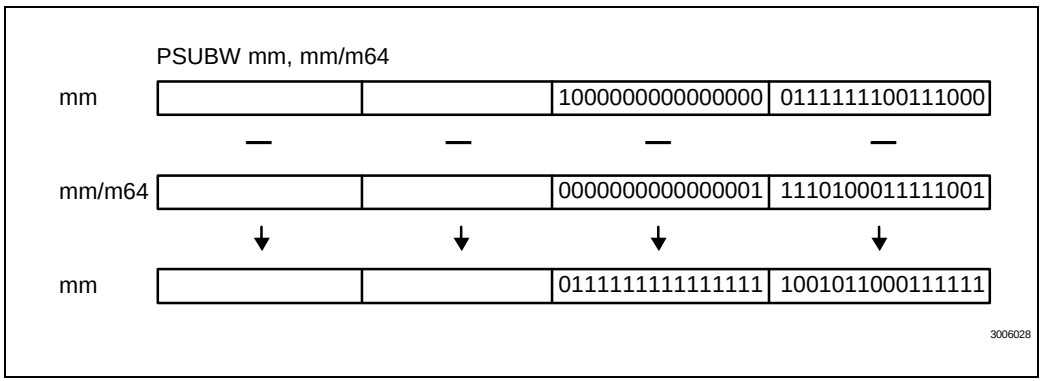
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PSUBB instruction subtracts the bytes of the source operand from the bytes of the destination operand. The result is written to the MMX register. When the result is too large or too small to be represented in a byte, the result wraps around and the lower 8 bits are written to the destination register.

The PSUBW instruction subtracts the words of the source operand from the words of the destination operand. The result is written to the MMX register. When the result is too large or too small to be represented in a word, the result wraps around and the lower 16 bits are written to the destination register.

The PSUBD instruction subtracts the doublewords of the source operand from the doublewords of the destination operand. The result is written to the MMX register. When the result is too large or too small to be represented in a doubleword, the result wraps around and the lower 32 bits are written to the destination register.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory.

# PSUBSB/PSUBSW —Packed Subtract with Saturation

| Opcode | Instruction | Description |
|---|---|---|
| 0F E8 /r | PSUBSB mm, mm/m64 | Subtract signed packed byte in MMX reg/memory from signed packed byte in MMX register and saturate. |
| 0F E9 /r | PSUBSW mm, mm/m64 | Subtract signed packed word in MMX reg/memory from signed packed word in MMX register and saturate. |

## Operation

```
IF instruction is PSUBSB
 THEN{
   mm(7..0) ← SaturateToSignedByte (mm(7..0) - mm/m64 (7..0));
   mm(15..8) ← SaturateToSignedByte (mm(15..8) - mm/m64(15..8));
   mm(23..16) ← SaturateToSignedByte (mm(23..16) - mm/m64(23..16));
   mm(31..24) ← SaturateToSignedByte (mm(31..24) - mm/m64(31..24));
   mm(39..32) ← SaturateToSignedByte (mm(39..32) - mm/m64(39..32));
   mm(47..40) ← SaturateToSignedByte (mm(47..40) - mm/m64(47..40));
   mm(55..48) ← SaturateToSignedByte (mm(55..48) - mm/m64(55..48));
   mm(63..56) ← SaturateToSignedByte (mm(63..56) - mm/m64(63..56))
   }
ELSE { (* instruction is PSUBSW *)
   mm(15..0) ← SaturateToSignedWord (mm(15..0) - mm/m64(15..0));
   mm(31..16) ← SaturateToSignedWord (mm(31..16) - mm/m64(31..16));
   mm(47..32) ← SaturateToSignedWord (mm(47..32) - mm/m64(47..32));
   mm(63..48) ← SaturateToSignedWord (mm(63..48) - mm/m64(63..48));
   }
```

## Description

The PSUBS instructions subtract the data elements of the source operand from the data elements of the destination operand. The results are saturated to the limits of a signed data element and written to the destination operand.

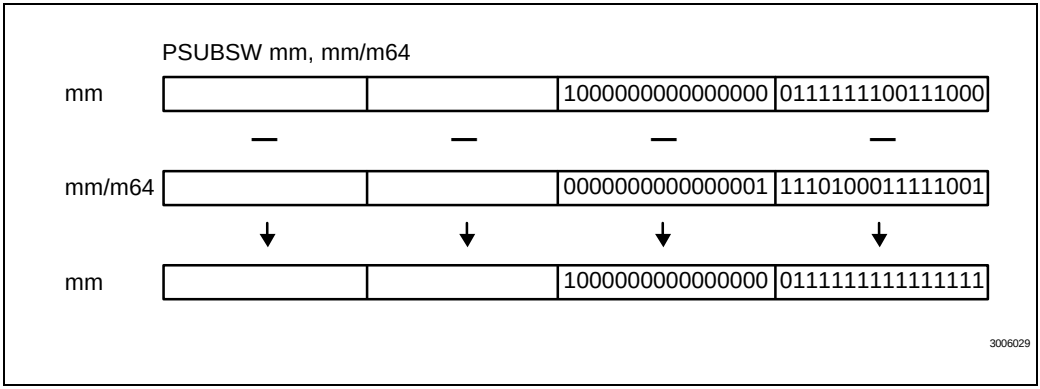
The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PSUBB instruction subtracts the signed bytes of the source operand from the signed bytes of the destination operand, and writes the results to the destination register. If the result is larger or smaller than the range of a signed byte, the value is saturated; in the case of an overflow - to 0x7F, and in the case of an underflow - to 0x80.

The PSUBW instruction subtracts the signed words of the source operand from the signed words of the destination operand and writes the results to the destination register. If the result

is larger or smaller than the range of a signed word, the value is saturated; in the case of an overflow to 0x7FFF, and in the case of an underflow to 0x8000.

### Example




### Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PSUBUSB/PSUBSW —Packed Subtract Unsigned with Saturation

| Opcode | Instruction | Description |
|---|---|---|
| 0F D8 /r | PSUBUSB mm, mm/m64 | Subtract unsigned packed byte in MMX reg/memory from unsigned packed byte in MMX register and saturate. |
| 0F D9 /r | PSUBUSW mm, mm/m64 | Subtract unsigned packed word in MMX reg/memory from unsigned packed word in MMX register and saturate. |

## Operation

```
IF instruction is PSUBUSB
 THEN{
   mm(7..0) ← SaturateToUnsignedByte (mm(7..0) - mm/m64 (7..0) );
   mm(15..8) ← SaturateToUnsignedByte ( mm(15..8) - mm/m64(15..8) );
   mm(23..16) ← SaturateToUnsignedByte (mm(23..16) - mm/m64(23..16) );
   mm(31..24) ← SaturateToUnsignedByte (mm(31..24) - mm/m64(31..24) );
   mm(39..32) ← SaturateToUnsignedByte (mm(39..32) - mm/m64(39..32) );
   mm(47..40) ← SaturateToUnsignedByte (mm(47..40) - mm/m64(47..40) );
   mm(55..48) ← SaturateToUnsignedByte (mm(55..48) - mm/m64(55..48) );
   mm(63..56) ← SaturateToUnsignedByte (mm(63..56) - mm/m64(63..56) );
   }
ELSE { (* instruction is PSUBUSW *)
   mm(15..0) ← SaturateToUnsignedWord (mm(15..0) - mm/m64(15..0) );
   mm(31..16) ← SaturateToUnsignedWord (mm(31..16) - mm/m64(31..16) );
   mm(47..32) ← SaturateToUnsignedWord (mm(47..32) - mm/m64(47..32) );
   mm(63..48) ← SaturateToUnsignedWord (mm(63..48) - mm/m64(63..48) );
   }
```

## Description

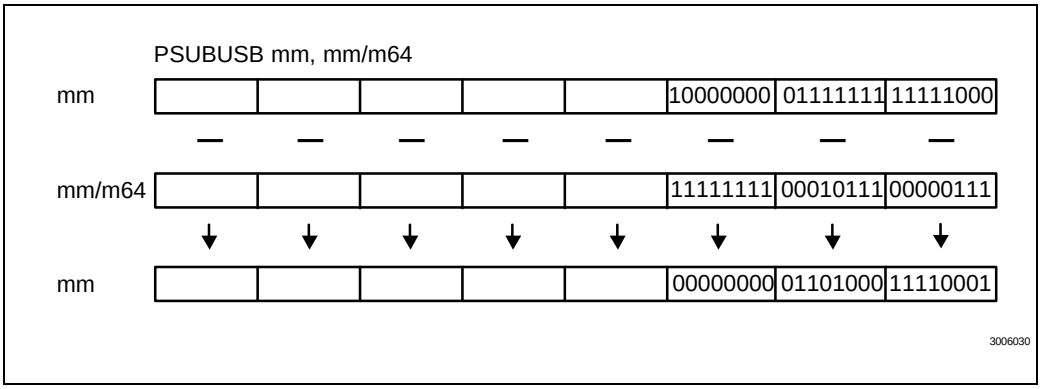
The PSUBUS instructions subtract the data elements of the source operand from the data elements of the destination register. The results are saturated to the limits of an unsigned data element and written to the destination operand.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

The PSUBUSB instruction subtracts the bytes of the source operand from the bytes of the destination operand and writes the results to the destination register. If the result element is less than zero (a negative value), it is saturated to 0x00.

The PSUBUSW instruction subtracts the words of the source operand from the words of the destination operand and writes the results to the destination register. If the result element is less than zero (a negative value), it is saturated to 0x0000.

Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

## Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ —Unpack High Packed Data

| Opcode | Instruction | Description |
|---|---|---|
| 0F 68 /r | PUNPCKHBW mm, mm/m64 | Interleave bytes from the high halves of MMX register and MMX reg/memory into MMX register. |
| 0F 69 /r | PUNPCKHWD mm, mm/m64 | Interleave words from the high halves of MMX register and MMX reg/memory into MMX register. |
| 0F 6A /r | PUNPCKHDQ mm, mm/m64 | Interleave dwords from the high halves of MMX register and MMX reg/memory into MMX register. |

## Operation

```
IF instruction is PUNPCKHBW
THEN {
   mm(63..56) ← mm/m64(63..56);
   mm(55..48) ← mm(63..56);
   mm(47..40) ← mm/m64(55..48);
   mm(39..32) ← mm(55..48);
   mm(31..24) ← mm/m64(47..40);
   mm(23..16) ← mm(47..40);
   mm(15..8) ← mm/m64(39..32);
   mm(7..0) ← mm(39..32);
ELSE IF instruction is PUNPCKHW
THEN {
   mm(63..48) ← mm/m64(63..48);
   mm(47..32) ← mm(63..48);
   mm(31..16) ← mm/m64(47..32);
   mm(15..0) ← mm(47..32);
   }
ELSE { (* instruction is PUNPCKHDQ *)
   mm(63..32) ← mm/m64(63..32);
   mm(31..0) ← mm(63..32)
   }
```

## Description

The PUNPCKH instructions unpack and interleave the high-order data elements of the destination and source operands into the destination operand. The low-order data elements are ignored.

The destination operand is an MMX register. The source operand can either be an MMX register or a 64-bit memory operand.

When unpacking from a memory operand, the full 64-bit operand is accessed from memory. The instruction uses only the high-order 32 bits.

The PUNPCKHBW instruction interleaves the four high-order bytes of the source operand and the four high-order bytes of the destination operand and writes them to the MMX register.

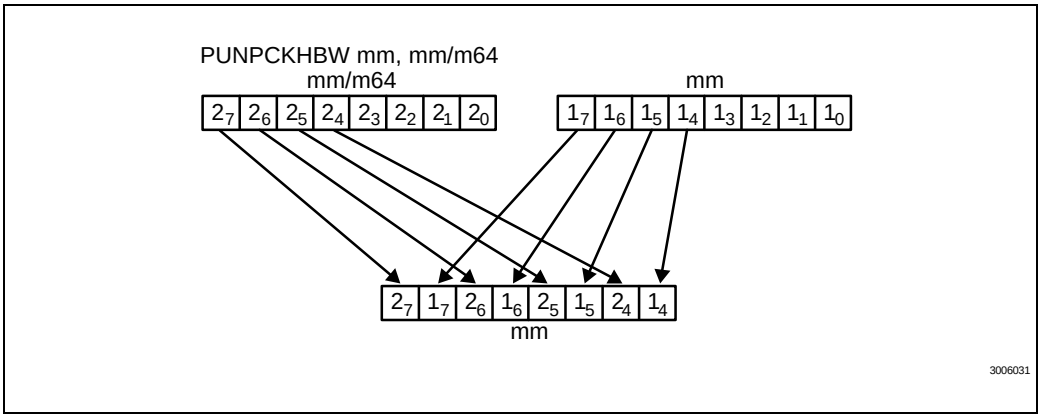
The PUNPCKHWD instruction interleaves the two high-order words of the source operand and the two high-order words of the destination operand and writes them to the MMX register.

The PUNPCKHDQ instruction interleaves the high-order 32 bits of the doubleword of the source operand and the high-order 32-bits of the doubleword of the destination operand and writes them to the MMX register.

### Note

If the source operand is all zeros, the result is a zero extension of the high order elements of the destination operand. When using the PUNPCKHBW instruction the bytes are zero extended, or unpacked into unsigned words. When using the PUNPCKHWD instruction, the words are zero exended, or unpacked into unsigned doublewords.

### Example




### Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ —Unpack Low Packed Data

| Opcode | Instruction | Description |
|---|---|---|
| 0F 60 /r | PUNPCKLBW mm, mm/m32 | Interleave bytes from the low halves of MMX register and MMX reg/memory into MMX register. |
| 0F 61 /r | PUNPCKLWD mm, mm/m32 | Interleave words from the low halves of MMX register and MMX reg/memory into MMX register. |
| 0F 62 /r | PUNPCKLDQ mm, mm/m32 | Interleave dwords from the low halves of MMX register and MMX reg/memory into MMX register. |

## Operation

```
IF instruction is PUNPCKLBW
THEN {
   mm(63..56) ←mm/m32(31..24);
   mm(55..48) ← mm(31..24);
   mm(47..40) ← mm/m32(23..16);
   mm(39..32) ← mm(23..16);
   mm(31..24) ← mm/m32(15..8);
   mm(23..16) ←mm(15..8);
   mm(15..8) ← mm/m32(7..0);
   mm(7..0) ← mm(7..0);
   }
ELSE IF instruction is PUNPCKLWD
THEN {
   mm(63..48) ← mm/m32(31..16);
   mm(47..32) ← mm(31..16);
   mm(31..16) ← mm/m32(15..0);
   mm(15..0) ← mm(15..0);
   }
ELSE{  (* instruction is PUNPCKLDQ *)
   mm(63..32) ← mm/m32(31..0);
   mm(31..0) ← mm(31..0);
   }
```

## Description

The PUNPCKL instructions unpack and interleave the low-order data elements of the destination and source operands into the destination operand.

The destination operand is an MMX register. The source operand can either be an MMX register or a 32-bit memory operand. When the source data comes from 64-bit registers, the upper 32 bits are ignored.

When unpacking from a memory operand, only 32 bits are accessed. The instruction uses all 32 bits.

The PUNPCKLBW instruction interleaves the four low-order bytes of the source operand and the four low-order bytes of the destination operand and writes them to the MMX register.

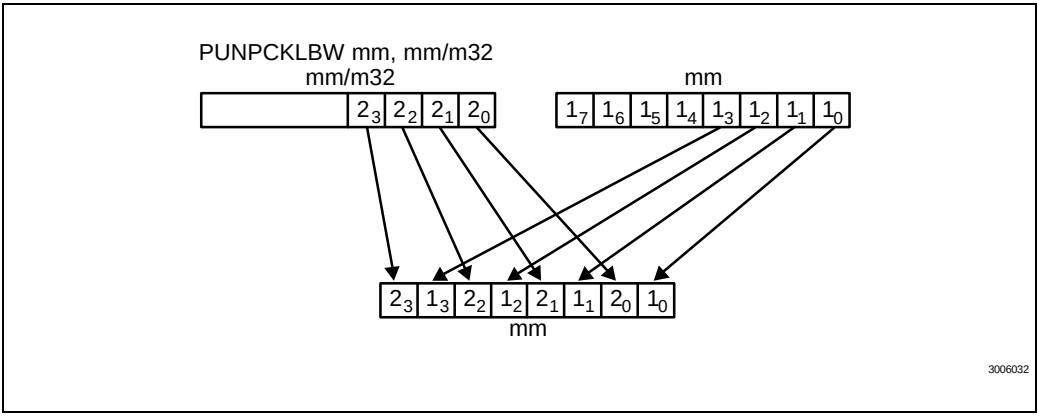
The PUNPCKLWD instruction interleaves the two low-order words of the source operand and the two low-order words of the destination operand and writes them to the MMX register.

The PUNPCKLDQ instruction interleaves the low-order doubleword of the source operand and the low-order doubleword of the destination operand and writes them to the MMX register.

### Note

If the source operand has a value of all zeros, the result is a zero extension of the low order elements of the destination operand. When using the PUNPCKLBW instruction the bytes are zero extended, or unpacked into unsigned words. When using the PUNPCKLWD instruction, the words are zero extended, or unpacked into unsigned doublewords.

### Example




### Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.

# PXOR—Bitwise Logical Exclusive OR

| Opcode | Instruction | Description |
|---|---|---|
| 0F EF /r | PXOR mm, mm/m64 | XOR 64 bits from MMX reg/memory to MMX register. |

## Operation

mm ←mm XOR mm/m64;

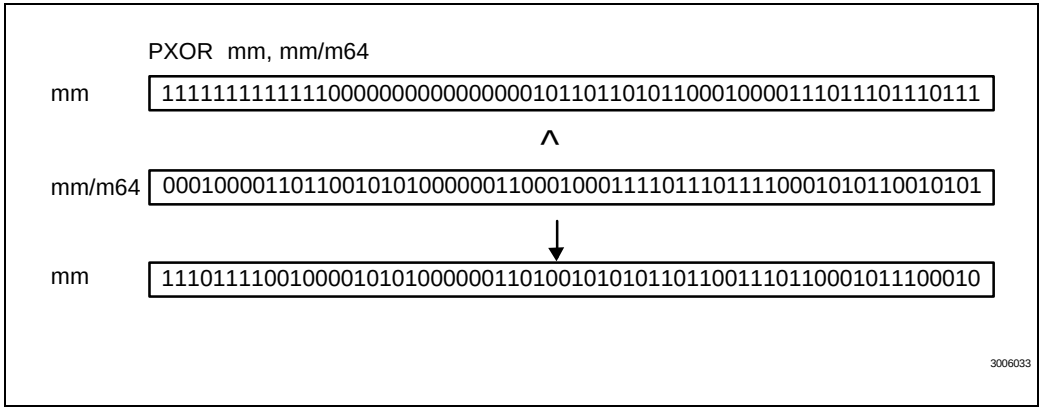
## Description

The PXOR instruction performs a bitwise logical XOR on the 64 bits of the destination with the source operands and writes the result to destination register.

Each bit of the result is 1 if the corresponding bits of the two operands are different. Each bit is 0 if the corresponding bits of the operands are the same.

The source operand can either be an MMX register or a 64 bit memory operand.

## Example




## Flags Affected

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; #AC for unaligned memory reference if the current privilege level is 3; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Real Address Mode Exceptions

Interrupt 13 if any part of the operand lies outside of the effective address space from 0 to 0FFFFH; #UD if CR0.EM = 1; #NM if TS bit in CR0 is set; #MF if there is a pending FPU exception.

### Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode; #PF(fault-code) for a page fault; #AC for unaligned memory reference.