Resumen de Instrucciones IA-32

Martín A. Miguel
m2.march@gmail.com

13 de mayo de 2010

Índice

Aclaraciones:
En este documento se evitan las referencias a modos de direccionamiento y operaciones para procesadores de 64-bits.
Las descripciones son extractos del manual. En algunos casos pueden tener referencias a partes del manual que no fueron copiadas en este documento. Esto fue hecho adrede para dar a conocer que hay información que no está siendo presentada.
Los formatos de direccionamiento presentados en los cuadros de cada instrucción son meramente la mención de los que acepta la instrucción como primer y segundo operando. Eso no implica que el procesador acepte el producto cartesiano de ambos conjuntos. En el caso de haber instrucciones agrupadas, tampoco puede asumirse que todos los formatos de instrucción vale para todas las instrucciones.

Parte I

Instrucciones de Proposito General

1. Informacion General

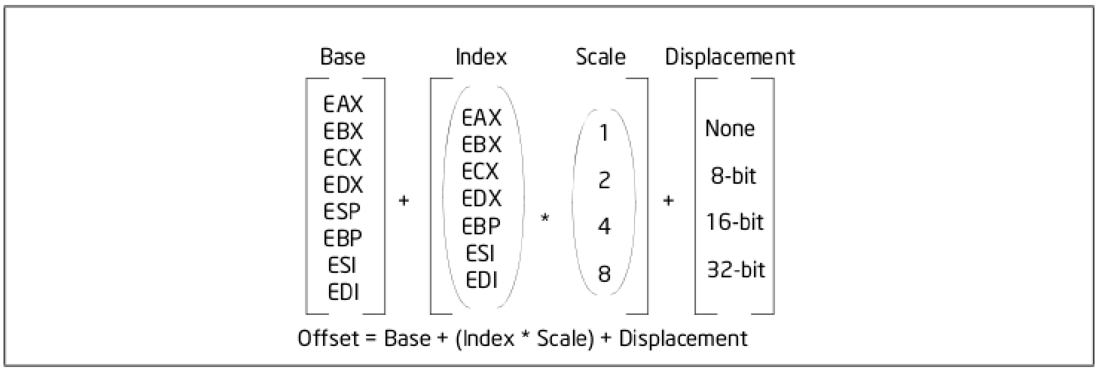
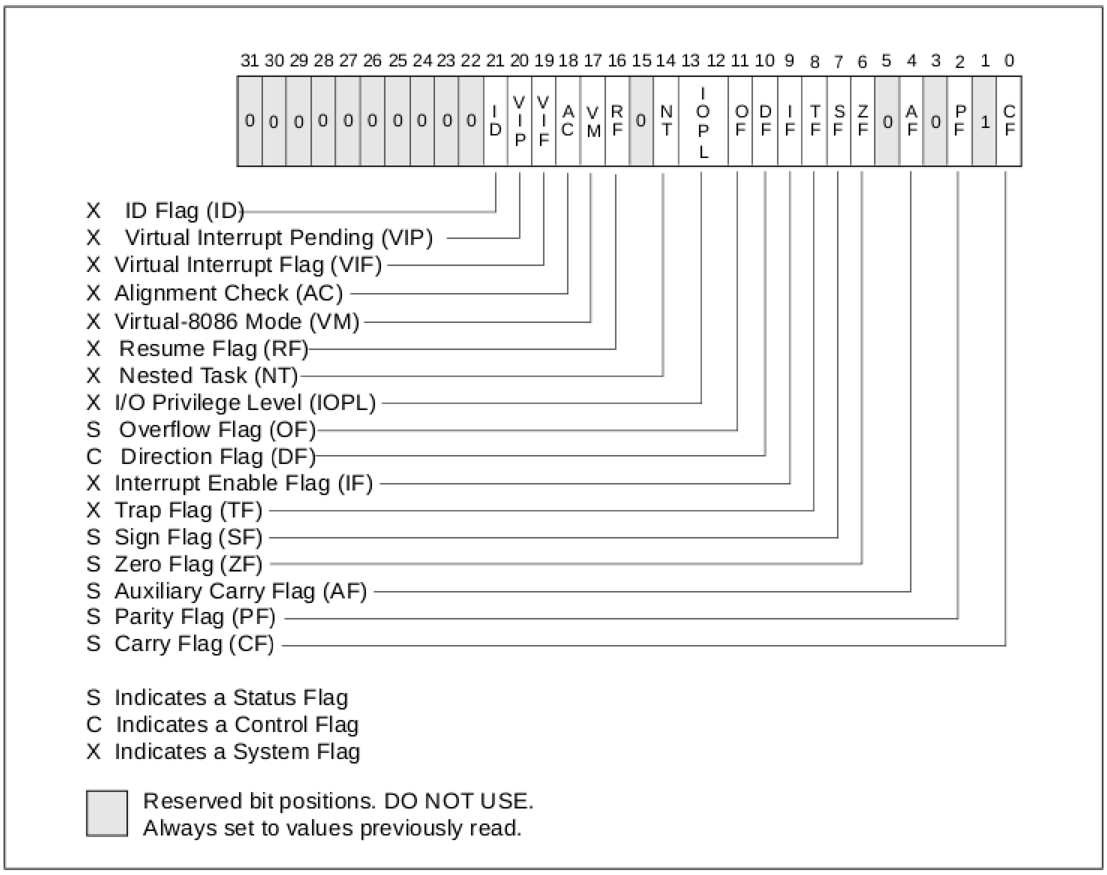


Figure 3-11. Offset (or Effective Address) Computation

1.1. Flags - Modo Protegido



31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE.
Always set to values previously read.

Figure 3-8. EFLAGS Register

1.2. Modos de Direccionamiento

1.3. Tabla de Condicionales

Unsigned Conditional Jumps		
A/NBE	(CF or ZF) = 0	Above/not below or equal
AE/NB	CF = 0	Above or equal/not below
B/NAE	CF = 1	Below/not above or equal
BE/NA	(CF or ZF) = 1	Below or equal/not above
C	CF = 1	Carry
E/Z	ZF = 1	Equal/zero
NC	CF = 0	Not carry
NE/NZ	ZF = 0	Not equal/not zero
NP/PO	PF = 0	Not parity/parity odd
P/PE	PF = 1	Parity/parity even
CXZ	CX = 0	Register CX is zero
ECXZ	ECX = 0	Register ECX is zero
Signed Conditional Jumps		
G/NLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
GE/NL	(SF xor OF) = 0	Greater or equal/not less
L/NGE	(SF xor OF) = 1	Less/not greater or equal
LE/NG	((SF xor OF) or ZF) = 1	Less or equal/not greater
NO	OF = 0	Not overflow
NS	SF = 0	Not sign (non-negative)
O	OF = 1	Overflow
S	SF = 1	Sign (negative)

2. Data Transfer Instructions

2.0.1. MOV-Move

Dest (First Op)	**Src (Second Op)**
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Copies the second operand (source operand) to the first operand (destination operand). The source operand can be an immediate value, general-purpose register, segment register, or memory location; the destination register can be a general- purpose register, segment register, or memory location. Both operands must be the same size, which can be a byte, a word, a doubleword, or a quadword.

Flags Affected None.

2.0.2. CMOVcc-Conditional Move

Dest (First Op)	**Src (Second Op)**
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description The CMOVcc instructions check the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and perform a move operation if the flags are in a specified state (or condition). A condition code (cc) is associated with each instruction to indicate the condition being tested for. If the condition is not satisfied, a move is not performed and execution continues with the instruction following the CMOVcc instruction. Ver Tabla de $SaltosCondicionales$.

Flags Affected None.

2.0.3. XCHG-Exchange Register/Memory

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32

Description Exchanges the contents of the destination (first) and source (second) operands. The operands can be two general-purpose registers or a register and a memory location.

Flags Affected None.

2.0.4. BSWAP-Byte Swap

Dest (First Op)	Src (Second Op)
r32	-

Description Reverses the byte order of a 32-bit or 64-bit (destination) register. This instruction is provided for converting little-endian values to big-endian format and vice versa.

Flags Affected None.

2.0.5. PUSH-Push Word, Doubleword or Quadword Onto the Stack

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32, imm8, imm16, imm32	-

Description Decrements the stack pointer and then stores the source operand on the top of the stack. The address-size attribute of the stack segment determines the stack pointer size (16, 32 or 64 bits). The operand-size attribute of the current code segment determines the amount the stack pointer is decremented (2, 4 or 8 bytes).

Flags Affected None.

2.0.6. PUSHA/PUSHAD-Push All General Purpose Registers

Dest (First Op)	Src (Second Op)
-	-

Description Pushes the contents of the general-purpose registers onto the stack. The registers are stored on the stack in the following order: EAX, ECX, EDX, EBX, ESP (original value), EBP, ESI, and EDI (if the current operand-size attribute is 32) and AX, CX, DX, BX, SP (original value), BP, SI, and DI (if the operand-size attribute is 16).

Flags Affected None.

2.0.7. POP-Pop a Value from the Stack

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	-

Description Loads the value from the top of the stack to the location specified with the destina- tion operand (or explicit opcode) and then increments the stack pointer. The destination operand can be a general-purpose register, memory location, or segment register. The address-size attribute of the stack segment determines the stack pointer size (16, 32, 64 bits) and the operand-size attribute of the current code segment determines the amount the stack pointer is incremented (2, 4, 8 bytes).

Flags Affected None.

2.0.8. CBW/CWDE/CDQE-Convert Byte to Word/Convert Word to Doubleword/Convert Doubleword to Quadword

Dest (First Op)	Src (Second Op)
-	-

Description Double the size of the source operand by means of sign extension. The CBW (convert byte to word) instruction copies the sign (bit 7) in the source operand into every bit in the AH register. The CWDE (convert word to doubleword) instruction copies the sign (bit 15) of the word in the AX register into the high 16 bits of the EAX register.

Flags Affected None.

2.0.9. CWD/CDQ/CQO-Convert Word to Doubleword/Convert Doubleword to Quadword

Dest (First Op)	Src (Second Op)
-	-

Description Doubles the size of the operand in register AX, EAX, or RAX (depending on the operand size) by means of sign extension and stores the result in registers DX:AX, EDX:EAX, or RDX:RAX, respectively.

Flags Affected None.

2.0.10. MOVSX/MOVSXD-Move with Sign-Extension

Dest (First Op)	Src (Second Op)
r16,r32	r/m8, r/m16

Description Copies the contents of the source operand (register or memory location) to the desti- nation operand (register) and sign extends the value to 16 or 32 bits.

Flags Affected None.

2.0.11. MOVZX-Move with Zero-Extend

Dest (First Op)	Src (Second Op)
r16,r32	r/m8, r/m16

Description Copies the contents of the source operand (register or memory location) to the destination operand (register) and zero extends the value. The size of the converted value depends on the operand-size attribute.

Flags Affected None.

3. Binary Arithmetic Instructions

3.0.12. ADD-Add

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Adds the destination operand (first operand) and the source operand (second operand) and then stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign- extended to the length of the destination operand format.

Operation DEST $\leftarrow$ DEST + SRC;

Flags Affected The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

3.0.13. ADC-Add with Carry

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Adds the destination operand (first operand), the source operand (second operand), and the carry (CF) flag and stores the result in the destination operand. [...] The ADC instruction does not distinguish between signed or unsigned operands. Instead, the processor evaluates the result for both data types and sets the OF and CF flags to indicate a carry in the signed or unsigned result, respectively.

Operation DEST $\leftarrow$ DEST + SRC + CF;

Flags Affected The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

3.0.14. SUB-Substract

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Subtracts the second operand (source operand) from the first operand (destination operand) and stores the result in the destination operand. [...] (Two memory operands cannot be used in one instruction.)

Operation DEST $\leftarrow$ (DEST - SRC);

Flags Affected The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

3.0.15. SBB-Integer Subtraction with Borrow

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Adds the source operand (second operand) and the carry (CF) flag, and subtracts the result from the destination operand (first operand). The result of the subtraction is stored in the destination operand. The SBB instruction does not distinguish between signed or unsigned operands. ($VerADC$)

Operation DEST ← (DEST - (SRC + CF));

Flags Affected The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

3.0.16. DEC-Decrement by 1

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	-

Description Subtracts 1 from the destination operand, while preserving the state of the CF flag.

Flags Affected The CF flag is not affected. The OF, SF, ZF, AF, and PF flags are set according to the result.

3.0.17. INC-Increment by 1

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	-

Description Adds 1 to the destination operand, while preserving the state of the CF flag.

Flags Affected The CF flag is not affected. The OF, SF, ZF, AF, and PF flags are set according to the result.

3.0.18. NEG-Two's Complement Negation

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	-

Description Replaces the value of operand (the destination operand) with its two's complement.

Flags Affected The CF flag set to 0 if the source operand is 0; otherwise it is set to 1. The OF, SF, ZF, AF, and PF flags are set according to the result.

3.0.19. CMP-Compare Two Operands

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Compares the first source operand with the second source operand and sets the status flags in the EFLAGS register according to the results. The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction.

Flags Affected The CF, OF, SF, ZF, AF, and PF flags are set according to the result.

3.0.20. MUL-Unsigned Multiply

Dest (First Op)	Src (Second Op)
-	r/m8, r/m16, r/m32

Description Performs an unsigned multiplication of the first operand (destination operand) and the second operand (source operand) and stores the result in the destination operand. The destination operand is an implied operand located in register AL, AX or EAX (depending on the size of the operand). The result is stored in register AX, register pair DX:AX, or register pair EDX:EAX (depending on the operand size), with the high-order bits of the product contained in register AH, DX, or EDX, respectively. If the high-order bits of the product are 0, the CF and OF flags are cleared; otherwise, the flags are set.

Flags Affected The OF and CF flags are set to 0 if the upper half of the result is 0; otherwise, they are set to 1. The SF, ZF, AF, and PF flags are undefined.

3.0.21. IMUL-Signed Multiply

Dest (First Op)	Src (Second Op)	(Third Op)
r16,r32	r/m8, r/m16, r/m32	imm8,imm16,imm32

Description

One-operand form This form is identical to that used by the MUL instruction. Here, the source operand (in a general-purpose register or memory location) is multiplied by the value in the AL, AX, EAX, or RAX register (depending on the operand size) and the product is stored in the AX, DX:AX, EDX:EAX, or RDX:RAX registers, respectively.

Two-operand form With this form the destination operand (the first operand) is multiplied by the source operand (second operand). The destination operand is a general-purpose register and the source operand is an immediate value, a general-purpose register, or a memory location. The product is then stored in the destination operand location.

Three-operand form This form requires a destination operand (the first operand) and two source operands (the second and the third operands). Here, the first source operand (which can be a general-purpose register or a memory location) is multiplied by the second source operand (an immediate value). The product is then stored in the destination operand (a general-purpose register).

Flags Affected For the one operand form of the instruction, the CF and OF flags are set when signif- icant bits are carried into the upper half of the result and cleared when the result fits exactly in the lower half of the result. For the two- and three-operand forms of the instruction, the CF and OF flags are set when the result must be truncated to fit in the destination operand size and cleared when the result fits exactly in the destination operand size. The SF, ZF, AF, and PF flags are undefined.

3.0.22. DIV-Unsigned Divide

Dest (First Op)	Src (Second Op)
-	r/m8, r/m16, r/m32

Description Divides unsigned the value in the AX, DX:AX, EDX:EAX, or RDX:RAX registers (divi- dend) by the source operand (divisor) and stores the result in the AX (AH:AL), DX:AX, EDX:EAX, or RDX:RAX registers.

Flags Affected The CF, OF, SF, ZF, AF, and PF flags are undefined.

3.0.23. IDIV-Signed Divide

Dest (First Op)	Src (Second Op)
-	r/m8, r/m16, r/m32

Description Divides (signed) the value in the AX, DX:AX, EDX:EAX, or RDX:RAX registers (divi- dend) by the source operand (divisor) and stores the result in the AX (AH:AL), DX:AX, EDX:EAX, or RDX:RAX registers.

Flags Affected The CF, OF, SF, ZF, AF, and PF flags are undefined.

3.0.24. DAA-Decimal Adjust AL after Addition

Dest (First Op)	Src (Second Op)
-	-

Description Adjusts the sum of two packed BCD values to create a packed BCD result. The AL register is the implied source and destination operand. The DAA instruction is only useful when it follows an ADD instruction that adds (binary addition) two 2-digit, packed BCD values and stores a byte result in the AL register. The DAA instruction then adjusts the contents of the AL register to contain the correct 2-digit, packed BCD result. If a decimal carry is detected, the CF and AF flags are set accordingly.

Flags Affected The CF and AF flags are set if the adjustment of the value results in a decimal carry in either digit of the result (see the âOperationâ section above). The SF, ZF, and PF flags are set according to the result. The OF flag is undefined.

3.0.25. DAS-Decimal Adjust AL after Subtraction

Dest (First Op)	Src (Second Op)
-	-

Description Adjusts the result of the subtraction of two packed BCD values to create a packed BCD result. The AL register is the implied source and destination operand. The DAS instruction is only useful when it follows a SUB instruction that subtracts (binary subtraction) one 2-digit, packed BCD value from another and stores a byte result in the AL register. The DAS instruction then adjusts the contents of the AL register to contain the correct 2-digit, packed BCD result. If a decimal borrow is detected, the CF and AF flags are set accordingly.

Flags Affected The CF and AF flags are set if the adjustment of the value results in a decimal carry in either digit of the result (see the âOperationâ section above). The SF, ZF, and PF flags are set according to the result. The OF flag is undefined.

4. Logical And Shift Instructions

4.0.26. AND-Logical AND

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Performs a bitwise AND operation on the destination (first) and source (second) operands and stores the result in the destination operand location.

Operation DEST $\leftarrow$ DEST & SRC;

Flags Affected The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

4.0.27. NOT-One's Complement Negation

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	-

Description Performs a bitwise NOT operation (each 1 is set to 0, and each 0 is set to 1) on the destination operand and stores the result in the destination operand location.

Operation DEST $\leftarrow$ NOT DEST;

Flags Affected None.

4.0.28. OR-Logical Inclusive OR

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Performs a bitwise inclusive OR operation between the destination (first) and source (second) operands and stores the result in the destination operand location.

Operation DEST $\leftarrow$ DEST | SRC;

Flags Affected The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

4.0.29. XOR-Logical Exclusive OR

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	r/m8, r/m16, r/m32, imm8, imm16, imm32

Description Performs a bitwise exclusive OR (XOR) operation on the destination (first) and source (second) operands and stores the result in the destination operand location.

Operation DEST $\leftarrow$ DEST XOR SRC;

Flags Affected The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

4.0.30. SAL/SAR/SHL/SHR-Shift

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	imm8, CL

Description Shifts the bits in the first operand (destination operand) to the left or right by the number of bits specified in the second operand (count operand). Shifts the bits in the first operand (destination operand) to the left or right by the number of bits specified in the second operand (count operand). The shift arithmetic left (SAL) and shift logical left (SHL) instructions perform the same operation. The shift arithmetic right (SAR) and shift logical right (SHR) instructions shift the bits of the destination operand to the right (toward less significant bit locations). In effect, the SAR instruction fills the empty bit positionâs shifted value with the sign of the unshifted value.

Flags Affected The CF flag contains the value of the last bit shifted out of the destination operand; it is undefined for SHL and SHR instructions where the count is greater than or equal to the size (in bits) of the destination operand. The OF flag is affected only for 1-bit shifts (see âDescriptionâ above); otherwise, it is undefined. The SF, ZF, and PF flags are set according to the result. If the count is 0, the flags are not affected. For a non-zero count, the AF flag is undefined.

4.0.31. RCL/RCR/ROL/ROR–Rotate

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	imm8, CL

Description Shifts (rotates) the bits of the first operand (destination operand) the number of bit positions specified in the second operand (count operand) and stores the result in the destination operand. The rotate left (ROL) and rotate through carry left (RCL) instructions shift all the bits toward more-significant bit positions, except for the most-significant bit, which is rotated to the least-significant bit location. The rotate right (ROR) and rotate through carry right (RCR) instructions shift all the bits toward less significant bit positions, except for the least-significant bit, which is rotated to the most-significant bit location. The RCL and RCR instructions include the CF flag in the rotation.

Flags Affected The CF flag contains the value of the bit shifted into it. The OF flag is affected only for single-bit rotates (see âDescriptionâ above); it is undefined for multi-bit rotates. The SF, ZF, AF, and PF flags are not affected.

5. Bit and Byte Instructions

5.0.32. BT/BTS/BTR/BTC-Bit Test and Set/Reset/Complement

Dest (First Op)	Src (Second Op)
r/m16, r/m32	r16, r32, imm8

Description Selects the bit in a bit string (specified with the first operand, called the bit base) at the bit-position designated by the bit offset (specified by the second operand) and stores the value of the bit in the CF flag.

Operation CF ← Bit(BitBase, BitOffset)
Bit(BitBase, BitOffset) ← 1 IF (BTS) ; 0 IF (BTR) ; NOT(Bit(BitBase, BitOffset)) IF (BTC)

Flags Affected The CF flag contains the value of the selected bit. The OF, SF, ZF, AF, and PF flags are undefined.

5.0.33. SETcc-Set Byte on Condition

Dest (First Op)	Src (Second Op)
r/m8	-

Description Sets the destination operand to 0 or 1 depending on the settings of the status flags (CF, SF, OF, ZF, and PF) in the EFLAGS register. Ver *Tabla de Condicionales*.

Flags Affected None.

5.0.34. TEST-Logical Compare

Dest (First Op)	Src (Second Op)
r/m8, r/m16, r/m32	imm8, r8, r16, r32

Description Computes the bit-wise logical AND of first operand (source 1 operand) and the second operand (source 2 operand) and sets the SF, ZF, and PF status flags according to the result. The result is then discarded.

Operation SF ← MSB(TEMP); PF ← BitwiseXNOR(TEMP[0:7]); CF ← 0; OF ← 0;

Flags Affected The OF and CF flags are set to 0. The SF, ZF, and PF flags are set according to the result (see the "Operation" section above). The state of the AF flag is undefined.

6. Control Transfer Instructions

Esta sección me la salteo porque la conocemos todos. También se saltean String Instructions, I/O Instructions, Segment Register y Miscelaneas (Exceptuando LEA)

7. Flag Control Instructions

7.0.35. STC/CLC-Set/Clear Carry Flag

Operation CF ← 1 IF (STC) ; O IF (CLC) ;

7.0.36. LAHF/SAHF-Load/Store Status Flags into AH Register

Dest (First Op)	Src (Second Op)
-	-

Operation IF (LAHF) AH ← EFLAGS(SF:ZF:0:AF:0:PF:1:CF);
IF (SAHF) EFLAGS(SF:ZF:0:AF:0:PF:1:CF) ← AH;

7.0.37. PUSHF/PUSHFD/POPF/POPFD-Push/Pop EFLAGS Register onto the Stack

Dest (First Op)	Src (Second Op)
-	-

Description Decrements the stack pointer by 4 (PUSHFD) and pushes the entire contents of the EFLAGS register onto the stack, or decrements the stack pointer by 2 (PUSHF) and pushes the lower 16 bits of the EFLAGS register (that is, the FLAGS register) onto the stack. These instructions reverse the operation of the POPF/POPFD instructions.

Flags Affected None.

7.0.38. LEA-Load Effective Address

Dest (First Op)	Src (Second Op)
r16	m

Description Computes the effective address of the second operand (the source operand) and stores it in the first operand (destination operand). The source operand is a memory address (offset part) specified with one of the processors addressing modes; the destination operand is a general-purpose register.

Flags Affected none

Parte II

Instrucciones x87 FPU

7.1. FPU Flags

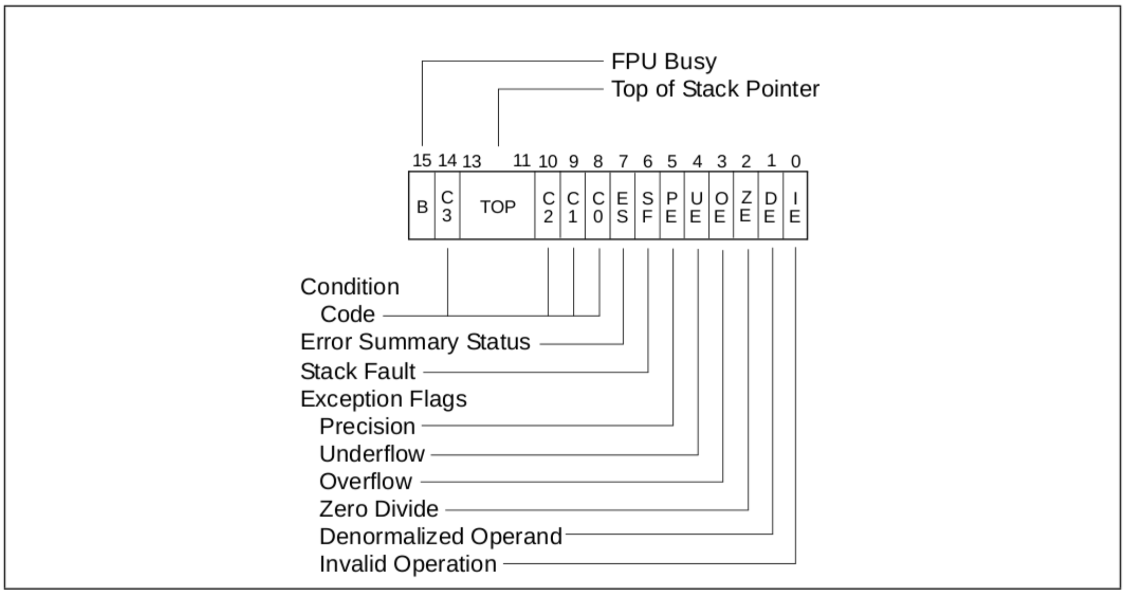
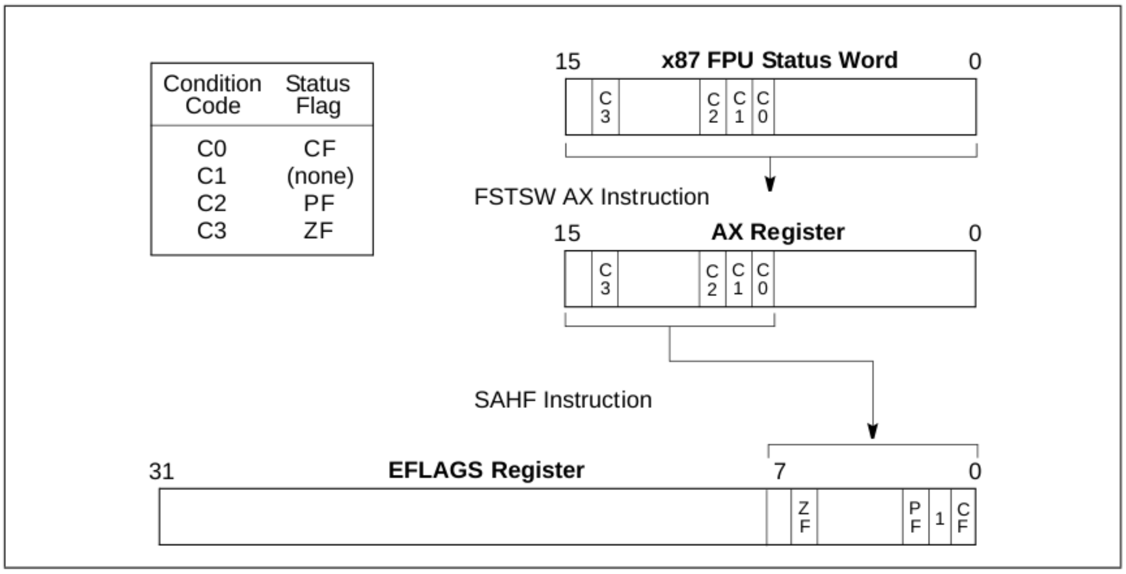


Figure 8-4. x87 FPU Status Word



Condition Code	Status Flag
C0	CF
C1	(none)
C2	PF
C3	ZF

Figure 8-5. Moving the Condition Codes to the EFLAGS Register

7.2. FPU Control Word

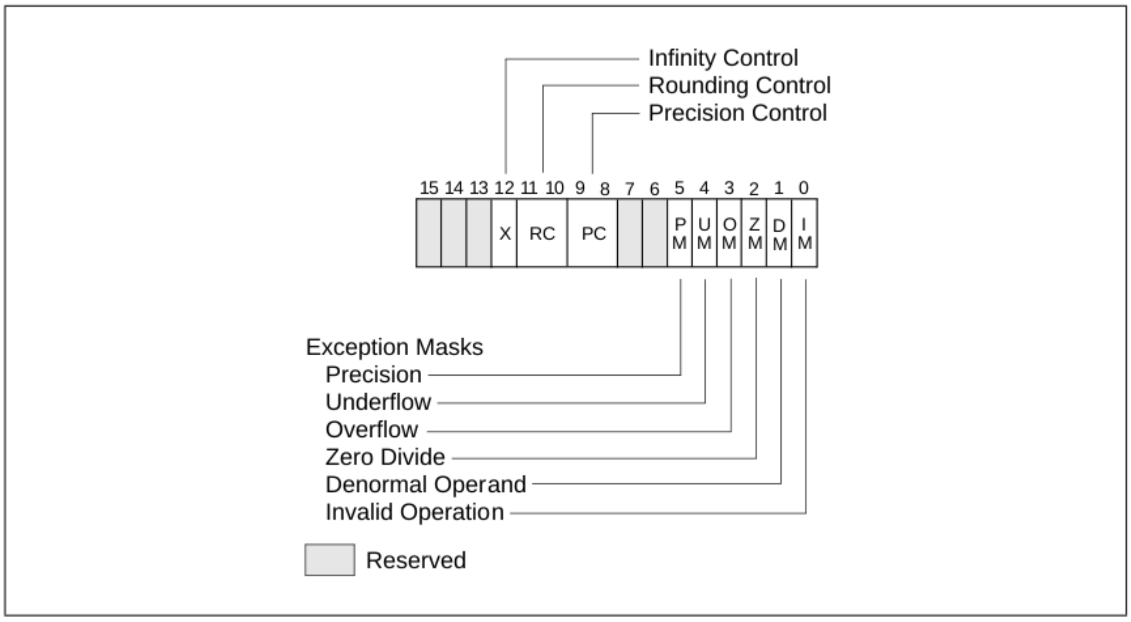


Figure 8-6. x87 FPU Control Word

7.3. Comparaciones FCOM

Comparison Results	ZF	PF	CF
ST0 >ST(i)	0	0	0
ST0 <ST(i)	0	0	1
ST0 = ST(i)	1	0	0
Unordered**	1	1	1

7.4. Comparaciones FTST

Condition	C3	C2	C0
ST(0) >0.0	0	0	0
ST(0) <0.0	0	0	1
ST(0) = 0.0	1	0	0
Unordered	1	1	1

8. x87 FPU Data Transfer Instructions

8.0.1. FLD-Load Floating Point Value

Dest (First Op)	**Src (Second Op)**
-	m32fp, m64fp, m80fp, ST(i)

Description Pushes the source operand onto the FPU register stack. If the source operand is in single-precision or double-precision floating-point format, it is automatically converted to the double extended-precision floating-point format before being pushed on the stack.

Flags Affected C1 Set to 1 if stack overflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

8.0.2. FST/FSTP-Store Floating Point Value

Dest (First Op)	Src (Second Op)
-	m32fp, m64fp, m80fp, ST(i)

Description The FST instruction copies the value in the ST(0) register to the destination operand, which can be a memory location or another register in the FPU register stack. When storing the value in memory, the value is converted to single-precision or double-precision floating-point format.The FSTP instruction performs the same operation as the FST instruction and then pops the register stack. If the destination size is single-precision or double-precision, the significand of the value being stored is rounded to the width of the destination (according to the rounding mode specified by the RC field of the FPU control word), and the exponent is converted to the width and bias of the destination format.

Flags Affected C1 Set to 0 if stack underflow occurred. C0, C2, C3 Undefined.

8.0.3. FILD/FIST/FISTP-Load/Store Integer (and Pop)

Dest (First Op)	Src (Second Op)
-	m32fp, m64fp, m80fp, ST(i) [ST(i) not for FIST and FSTIP]

Description FILD converts the signed-integer source operand into double extended-precision floating- point format and pushes the value onto the FPU register stack. The FIST instruction converts the value in the ST(0) register to a signed integer and stores the result in the destination operand. If the source value is a non-integral value, it is rounded to an integer value, according to the rounding mode specified by the RC field of the FPU control word. If the converted value is too large for the destination format, or if the source operand is an â, SNaN, QNAN, or is in an unsupported format, an invalid-arithmetic-operand condition is signaled.

Flags Affected C1 Set to 1 if stack overflow occurred; set to 0 otherwise. C0, C2, C3 Undefined.

8.0.4. FXCH-Exchange Register Contents

Dest (First Op)	Src (Second Op)
-	ST(i), -

Description Exchanges the contents of registers ST(0) and ST(i). If no source operand is speci- fied, the contents of ST(0) and ST(1) are exchanged.

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 1. C0, C2, C3 Undefined.

8.0.5. FCMOVcc-Floating-Point Conditional Move

Dest (First Op)	Src (Second Op)
ST(0)	ST(i)

Description Tests the status flags in the EFLAGS register and moves the source operand (second operand) to the destination operand (first operand) if the given test condition is true.
Solo son válidos los siguientes condicionales: B, E, BE, U, NB, NE, NBE, NU. Ver *Tabla de Condicionales*

Flags Affected C1 Set to 1 if stack overflow occurred; set to 0 otherwise. C0, C2, C3 Undefined.

9. x87 FPU Basic Arithmetic Instructions

9.0.6. FADD/FADDP/FIADD-Add

Dest (First Op)	Src (Second Op)
-, ST(0), ST(i)	-, m32fp, m64fp, ST(i), ST(0) [ST no válido para FIADD]

Description Adds the destination and source operands and stores the sum in the destination location. The destination operand is always an FPU register; the source operand can be a register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format or in word or doubleword integer format. The no-operand version of the instruction adds the contents of the ST(0) register to the ST(1) register. The one-operand version adds the contents of a memory location (either a floating-point or an integer value) to the contents of the ST(0) register. The two-operand version, adds the contents of the ST(0) register to the ST(i) register or vice versa. The FADDP instructions perform the additional operation of popping the FPU register stack after storing the result.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

9.0.7. FSUB/FSUBP/FISUB-Subtract

Dest (First Op)	Src (Second Op)
-, ST(0), ST(i)	-, m32fp, m64fp, ST(i), ST(0) [ST no válido para FISUB]

Description Subtracts the source operand from the destination operand and stores the difference in the destination location. The destination operand is always an FPU data register; the source operand can be a register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format or in word or doubleword integer format. The no-operand version of the instruction subtracts the contents of the ST(0) register from the ST(1) register and stores the result in ST(1). The one-operand version subtracts the contents of a memory location (either a floating-point or an integer value) from the contents of the ST(0) register and stores the result in ST(0). The two-operand version, subtracts the contents of the ST(0) register from the ST(i) register or vice versa. The FSUBP instructions perform the additional operation of popping the FPU register stack following the subtraction.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

9.0.8. FSUBR/FSUBRP/FISUBR-Reverse Subtract

Dest (First Op)	Src (Second Op)
-, ST(0), ST(i)	-, m32fp, m64fp, ST(i), ST(0) [ST no válido para FISUB]

Description Subtracts the destination operand from the source operand and stores the difference in the destination location. The destination operand is always an FPU register; the source operand can be a register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format or in word or doubleword integer format. These instructions perform the reverse operations of the FSUB, FSUBP, and FISUB instructions. They are provided to support more efficient coding. The no-operand version of the instruction subtracts the contents of the ST(1) register from the ST(0) register and stores the result in ST(1). The one-operand version subtracts the contents of the ST(0) register from the contents of a memory location (either a floating-point or an integer value) and stores the result in ST(0). The two- operand version, subtracts the contents of the ST(i) register from the ST(0) register or vice versa. The FSUBRP instructions perform the additional operation of popping the FPU register stack following the subtraction.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

9.0.9. FMUL/FMULP/FIMUL-Multiply

Dest (First Op)	Src (Second Op)
-, ST(0), ST(i)	-, m32fp, m64fp, ST(i), ST(0) [ST no válido para FIMUL]

Description Multiplies the destination and source operands and stores the product in the destina- tion location. The destination operand is always an FPU data register; the source operand can be an FPU data register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format or in word or doubleword integer format. The no-operand version of the instruction multiplies the contents of the ST(1) register by the contents of the ST(0) register and stores the product in the ST(1) register. The one-operand version multiplies the contents of the ST(0) register by the contents of a memory location (either a floating point or an integer value) and stores the product in the ST(0) register. The two-operand version, multiplies the contents of the ST(0) register by the contents of the ST(i) register, or vice versa, with the result being stored in the register specified with the first operand (the destination operand). The FMULP instructions perform the additional operation of popping the FPU register stack after storing the product.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

9.0.10. FDIV/FDIVP/FIDIV-Divide

Dest (First Op)	Src (Second Op)
-, ST(0), ST(i)	-, m32fp, m64fp, ST(i), ST(0) [ST no válido para FIMUL]

Description Divides the destination operand by the source operand and stores the result in the destination location. The destination operand (dividend) is always in an FPU register; the source operand (divisor) can be a register or a memory location. Source oper- ands in memory can be in single-precision or double-precision floating-point format, word or doubleword integer format. The no-operand version of the instruction divides the contents of the ST(1) register by the contents of the ST(0) register. The one-operand version divides the contents of the ST(0) register by the contents of a memory location (either a floating-point or an integer value). The two-operand version, divides the contents of the ST(0) register by the contents of the ST(i) register or vice versa. The FDIVP instructions perform the additional operation of popping the FPU register stack after storing the result.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

9.0.11. FDIVR/FDIVRP/FIDIVR-Reverse Divide

Dest (First Op)	Src (Second Op)
-, ST(0), ST(i)	-, m32fp, m64fp, ST(i), ST(0) [ST no válido para FIMUL]

Description Divides the source operand by the destination operand and stores the result in the destination location. The destination operand (divisor) is always in an FPU register; the source operand (dividend) can be a register or a memory location. Source oper- ands in memory can be in single-precision or double-precision floating-point format, word or doubleword integer format. These instructions perform the reverse operations of the FDIV, FDIVP, and FIDIV instructions. They are provided to support more efficient coding. The no-operand version of the instruction divides the contents of the ST(0) register by the contents of the ST(1) register. The one-operand version divides the contents of a memory location (either a floating-point or an integer value) by the contents of the ST(0) register. The two-operand version, divides the contents of the ST(i) register by the contents of the ST(0) register or vice versa. The FDIVRP instructions perform the additional operation of popping the FPU register stack after storing the result.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise. C0, C2, C3 Undefined.

9.0.12. FPREM-Partial Remainder

Dest (First Op)	Src (Second Op)
-	-

Description Computes the remainder obtained from dividing the value in the ST(0) register (the dividend) by the value in the ST(1) register (the divisor or modulus), and stores the result in ST(0). The remainder represents the following value: Remainder $\leftarrow ST(0) - (Q \times ST(1))$ Here, Q is an integer value that is obtained by truncating the floating-point number quotient of [ST(0) / ST(1)] toward zero. The sign of the remainder is the same as the sign of the dividend. The magnitude of the remainder is less than that of the modulus, unless a partial remainder was computed (as described below).

Flags Affected C0 Set to bit 2 (Q2) of the quotient.
C1 Set to 0 if stack underflow occurred; otherwise, set to least significant bit of quotient (Q0).
C2 Set to 0 if reduction complete; set to 1 if incomplete. C3 Set to bit 1 (Q1) of the quotient.

9.0.13. FPREM1-Partial Remainder

Dest (First Op)	Src (Second Op)
-	-

Description This instruction operates differently from the FPREM instruction in the way that it rounds the quotient of ST(0) divided by ST(1) to an integer (see the âOperationâ section below).

Flags Affected C0 Set to bit 2 (Q2) of the quotient.
C1 Set to 0 if stack underflow occurred; otherwise, set to least significant bit of quotient (Q0).
C2 Set to 0 if reduction complete; set to 1 if incomplete. C3 Set to bit 1 (Q1) of the quotient.

9.0.14. FABS-Absolute Value

Dest (First Op)	Src (Second Op)
-	-

Description Clears the sign bit of ST(0) to create the absolute value of the operand. The following table shows the results obtained when creating the absolute value of various classes of numbers.

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

9.0.15. FCHS-Change Sign

Dest (First Op)	Src (Second Op)
-	-

Description Complements the sign bit of ST(0). This operation changes a positive value into a negative value of equal magnitude or vice versa. The following table shows the results obtained when changing the sign of various classes of numbers.

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

9.0.16. FRNDINT-Round to Integer

Dest (First Op)	Src (Second Op)
-	-

Description Rounds the source value in the ST(0) register to the nearest integral value, depending on the current rounding mode (setting of the RC field of the FPU control word), and stores the result in ST(0).

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

9.0.17. FSCALE-Scale

Dest (First Op)	Src (Second Op)
-	-

Description Truncates the value in the source operand (toward 0) to an integral value and adds that value to the exponent of the destination operand. The destination and source operands are floating-point values located in registers ST(0) and ST(1), respectively.

Operation $ST(0) \leftarrow ST(0) \times 2^{RoundTowardZero(ST(1))};$

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

9.0.18. FSQRT-Square Root

Dest (First Op)	Src (Second Op)
-	-

Description Computes the square root of the source value in the ST(0) register and stores the result in ST(0).

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

9.0.19. FXTRACT-Extract Exponent and Significand

Dest (First Op)	Src (Second Op)
-	-

Description Separates the source value in the ST(0) register into its exponent and significand, stores the exponent in ST(0), and pushes the significand onto the register stack. ST(0) contains the value of the original significand expressed as a floating-point value. The ST(1) register contains the value of the original operandâs true (unbiased) exponent expressed as a floating- point value.

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Undefined.

9.0.20. FCOMI/FCOMIP/ FUCOMI/FUCOMIP-Compare Floating Point Values and Set EFLAGS

Dest (First Op)	Src (Second Op)
ST(0)	ST(i)

Description Performs an unordered comparison of the contents of registers ST(0) and ST(i) and sets the status flags ZF, PF, and CF in the EFLAGS register according to the results (Ver *Comparaciones FCOM*). The sign of zero is ignored for comparisons, so that -0.0 is equal to +0.0.

Flags Affected C1 Set to 0 if stack underflow occurred; otherwise, set to 0. C0, C2, C3 Not affected.

9.0.21. FTST-TEST

Dest (First Op)	Src (Second Op)
-	-

Description Compares the value in the ST(0) register with 0.0 and sets the condition code flags C0, C2, and C3 in the FPU status word according to the results (Ver *Tabla Comparaciones FTST*).

Flags Affected C1 Set to 0 if stack underflow occurred. C0, C2, C3 (Ver *Tabla Comparaciones FTST*).

9.0.22. FSIN-Sine

Dest (First Op)	Src (Second Op)
-	-

Description Computes the sine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$.If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C2 Set to 1 if outside range ($-2^{63} <$ src op $< +2^{63}$); otherwise, set to 0.
C0, C3 Undefined.

9.0.23. FCOS-Cosine

Dest (First Op)	Src (Second Op)
-	-

Description Computes the cosine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$.If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C2 Set to 1 if outside range ($-2^{63} <$ src op $< +2^{63}$); otherwise, set to 0.
C0, C3 Undefined.

9.0.24. FSINCOS-Sine and Cosine

Dest (First Op)	Src (Second Op)
-	-

Description Computes both the sine and the cosine of the source operand in register ST(0), stores the sine in ST(0), and pushes the cosine onto the top of the FPU register stack. (This instruction is faster than executing the FSIN and FCOS instructions in succession.) The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$.If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C2 Set to 1 if outside range ($-2^{63} <$ src op $< +2^{63}$); otherwise, set to 0.
C0, C3 Undefined.

9.0.25. FPTAN-Partial Tangent

Dest (First Op)	Src (Second Op)
-	-

Description Computes the tangent of the source operand in register ST(0), stores the result in ST(0), and pushes a 1.0 onto the FPU register stack. The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$.If the source operand is outside the acceptable range, the C2 flag in the FPU status word is set, and the value in register ST(0) remains unchanged.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C2 Set to 1 if outside range ($-2^{63} <$ src op $< +2^{63}$); otherwise, set to 0.
C0, C3 Undefined.

9.0.26. FPATAN-Partial Arctangent

Dest (First Op)	Src (Second Op)
-	-

Description Computes the arctangent of the source operand in register ST(1) divided by the source operand in register ST(0), stores the result in ST(1), and pops the FPU register stack. The result in register ST(0) has the same sign as the source operand ST(1) and a magnitude less than $+\pi$. The FPATAN instruction returns the angle between the X axis and the line from the origin to the point (X,Y), where Y (the ordinate) is ST(1) and X (the abscissa) is ST(0). The angle depends on the sign of X and Y independently, not just on the sign of the ratio Y/X. This is because a point (-X,Y) is in the second quadrant, resulting in an angle between $\pi/2$ and π, while a point (X,-Y) is in the fourth quadrant, resulting in an angle between 0 and -π/2. A point (-X,-Y) is in the third quadrant, giving an angle between $-\pi/2$ and $-\pi$.

Operation ST(1) ← arctan(ST(1) / ST(0)); PopRegisterStack;

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C0, C2, C3 Undefined.

9.0.27. F2XM1-Compute $2^x - 1$

Dest (First Op)	Src (Second Op)
-	-

Description Computes the exponential value of 2 to the power of the source operand minus 1. The source operand is located in register ST(0) and the result is also stored in ST(0). The value of the source operand must lie in the range -1.0 to +1.0. If the source value is outside this range, the result is undefined.
Values other than 2 can be exponentiated using the following formula: $x^y \leftarrow 2^{(y \times log_2 x)}$

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C0, C2, C3 Undefined.

9.0.28. FYL2X-Compute $y \times log_2 x$

Dest (First Op)	Src (Second Op)
-	-

Description Computes (ST(1) â log2 (ST(0))), stores the result in resister ST(1), and pops the FPU register stack. The source operand in ST(0) must be a non-zero positive number.

Operation ST(1) $\leftarrow ST(1) \times log_2 ST(0)$; PopRegisterStack;

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C0, C2, C3 Undefined.

9.0.29. FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ-Load Constant

Dest (First Op)	Src (Second Op)
-	-

Description Push one of seven commonly used constants (in double extended-precision floating- point format) onto the FPU register stack. The constants that can be loaded with these instructions include $+1{,}0, +0{,}0, log_2 10, log_2 e, \pi, log_{10} 2, and log_e 2$.

Flags Affected C1 Set to 0 if stack underflow occurred. Set if result was rounded up; cleared otherwise.
C0, C2, C3 Undefined.

9.0.30. FINCSTP/FDECSTP-Increment/Decrement Stack-Top Pointer

Dest (First Op)	Src (Second Op)
-	-

Description FINCSTP Adds one to the TOP field of the FPU status word (increments the top-of-stack pointer). If the TOP field contains a 7, it is set to 0. The effect of this instruction is to rotate the stack by one position. The contents of the FPU data registers and tag register are not affected. This operation is not equivalent to popping the stack, because the tag for the previous top-of-stack register is not marked empty. *FDECSTP hace lo análogo a FINCSTP decrementando el TOP en 1.*

Flags Affected The C1 flag is set to 0. The C0, C2, and C3 flags are undefined

9.0.31. FFREE-Free Floating-Point Register

Dest (First Op)	Src (Second Op)
-	-

Description Sets the tag in the FPU tag register associated with register ST(i) to empty (11B). The contents of ST(i) and the FPU stack-top pointer (TOP) are not affected.

Flags Affected C0, C1, C2, C3 undefined.

9.0.32. FINIT/FNINIT-Initialize Floating-Point Unit

Dest (First Op)	Src (Second Op)
-	-

Description Sets the FPU control, status, tag, instruction pointer, and data pointer registers to their default states. The FINIT instruction checks for and handles any pending unmasked floating-point exceptions before performing the initialization; the FNINIT instruction does not.

Flags Affected C0, C1, C2, C3 set to 0.

9.0.33. FINIT/FNINIT-Initialize Floating-Point Unit

Dest (First Op)	Src (Second Op)
-	-

Description Sets the FPU control, status, tag, instruction pointer, and data pointer registers to their default states. The FINIT instruction checks for and handles any pending unmasked floating-point exceptions before performing the initialization; the FNINIT instruction does not.

Flags Affected C0, C1, C2, C3 set to 0.

9.0.34. FSTCW/FNSTCW-Store x87 FPU Control Word

Dest (First Op)	Src (Second Op)
m2byte	-

Description Stores the current value of the FPU control word at the specified destination in memory. The FSTCW instruction checks for and handles pending unmasked floating- point exceptions before storing the control word; the FNSTCW instruction does not.

Flags Affected The C0, C1, C2, and C3 flags are undefined.

9.0.35. FLDCW-Load x87 FPU Control Word

Dest (First Op)	Src (Second Op)
-	m2byte

Description Loads the 16-bit source operand into the FPU control word. The source operand is a memory location. This instruction is typically used to establish or change the FPUâs mode of operation.

Flags Affected The C0, C1, C2, and C3 flags are undefined.

9.0.36. FSTSW/FNSTSW-Store x87 FPU Status Word

Dest (First Op)	Src (Second Op)
m2byte, AX	-

Description Stores the current value of the x87 FPU status word in the destination location. The destination operand can be either a two-byte memory location or the AX register. The FSTSW instruction checks for and handles pending unmasked floating-point excep- tions before storing the status word; the FNSTSW instruction does not.

Flags Affected The C0, C1, C2, and C3 flags are undefined.

9.0.37. FSTSW/FNSTSW-Store x87 FPU Status Word

Dest (First Op)	Src (Second Op)
m2byte, AX	-

Description Stores the current value of the x87 FPU status word in the destination location. The destination operand can be either a two-byte memory location or the AX register. The FSTSW instruction checks for and handles pending unmasked floating-point excep- tions before storing the status word; the FNSTSW instruction does not.

Flags Affected The C0, C1, C2, and C3 flags are undefined.

Parte III
Instrucciones MMX^{TM}

10. MMX Data Transfer Instructions

10.0.38. MOVD/MOVQ-Move Doubleword/Move Quadword

Dest (First Op)	Src (Second Op)
mm, r/m32, r/m64, xmm	mm, r/m32, r/m64, xmm

Description Copies a doubleword from the source operand (second operand) to the destination operand (first operand). The source and destination operands can be general-purpose registers, MMX technology registers, XMM registers, or 32-bit memory loca- tions. This instruction can be used to move a doubleword to and from the low double- word of an MMX technology register and a general-purpose register or a 32-bit memory location, or to and from the low doubleword of an XMM register and a general-purpose register or a 32-bit memory location. The instruction cannot be used to transfer data between MMX technology registers, between XMM registers, between general-purpose registers, or between memory locations.
When the destination operand is an MMX technology register, the source operand is written to the low doubleword of the register, and the register is zero-extended to 64 bits. When the destination operand is an XMM register, the source operand is written to the low doubleword of the register, and the register is zero-extended to 128 bits.

Flags Affected None.

11. MMX Conversion Instructions

11.0.39. PACKSSWB/PACKSSDW-Pack with Signed Saturation

Dest (First Op)	Src (Second Op)
mm1, xmm1	mm2, xmm2, m64, m128

Description Converts packed signed word integers into packed signed byte integers (PACKSSWB) or converts packed signed doubleword integers into packed signed word integers (PACKSSDW), using saturation to handle overflow conditions.
The PACKSSWB and PACKSSDW instructions operate on either 64-bit or 128-bit operands.

When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the desti- nation operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

11.0.40. PACKSSWB/PACKSSDW-Pack with Signed Saturation

Dest (First Op)	Src (Second Op)
mm1, xmm1	mm2, xmm2, m64, m128

Description Converts packed signed word integers into packed signed byte integers (PACKSSWB) or converts packed signed doubleword integers into packed signed word integers (PACKSSDW), using saturation to handle overflow conditions.
The PACKSSWB and PACKSSDW instructions operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the desti- nation operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

11.0.41. PACKUSWB-Pack with Unsigned Saturation

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Converts 4 or 8 signed word integers from the destination operand (first operand) and 4 or 8 signed word integers from the source operand (second operand) into 8 or 16 unsigned byte integers and stores the result in the destination operand.
The PACKUSWB instruction operates on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

11.0.42. PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ-Unpack High Data

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Unpacks and interleaves the high-order data elements (bytes, words, doublewords, or quadwords) of the destination operand (first operand) and source operand (second operand) into the destination operand.
The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register. When the source data comes from a 64-bit

memory operand, the full 64-bit operand is accessed from memory, but the instruction uses only the high-order 32 bits. When the source data comes from a 128-bit memory operand, an implementation may fetch only the appropriate 64 bits; however, alignment to a 16-byte boundary and normal segment checking will still be enforced.

Flags Affected None.

11.0.43. PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ-Unpack Low Data

Dest (First Op)	**Src (Second Op)**
mm, xmm	mm, xmm, m64, m128

Description Unpacks and interleaves the low-order data elements (bytes, words, doublewords, and quadwords) of the destination operand (first operand) and source operand (second operand) into the destination operand.
The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register. When the source data comes from a 64-bit memory operand, the full 64-bit operand is accessed from memory, but the instruction uses only the high-order 32 bits. When the source data comes from a 128-bit memory operand, an implementation may fetch only the appropriate 64 bits; however, alignment to a 16-byte boundary and normal segment checking will still be enforced.

Flags Affected None.

12. MMX Packed Arithmetic Instructions

12.0.44. PADDB/PADDW/PADDD-Add Packed Integers

Dest (First Op)	**Src (Second Op)**
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD add of the packed integers from the source operand (second operand) and the destination operand (first operand), and stores the packed integer results in the destination operand. Overflow is handled with wraparound.
These instructions can operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.
Note that the PADDB, PADDW, and PADDD instructions can operate on either unsigned or signed (two's complement notation) packed integers.

Flags Affected None.

12.0.45. PADDSB/PADDSW-Add Packed Signed Integers with Signed Saturation

Dest (First Op)	**Src (Second Op)**
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD add of the packed signed integers from the source operand (second operand) and the destination operand (first operand), and stores the packed integer results in the destination operand. Overflow is handled with signed saturation.
These instructions can operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

12.0.46. PADDUSB/PADDUSW-Add Packed Unsigned Integers with Unsigned Saturation

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD add of the packed unsigned integers from the source operand (second operand) and the destination operand (first operand), and stores the packed integer results in the destination operand. Overflow is handled with unsigned saturation.
These instructions can operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

12.0.47. PSUBB/PSUBW/PSUBD-Subtract Packed Integers

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD add of the packed signed integers from the source operand (second operand) and the destination operand (first operand), and stores the packed integer results in the destination operand. Overflow is handled with signed saturation.
These instructions can operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

12.0.48. PSUBSB/PSUBSW-Subtract Packed Signed Integers with Signed Saturation

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD subtract of the packed signed integers of the source operand (second operand) from the packed signed integers of the destination operand (first operand), and stores the packed integer results in the destination operand. Overflow is handled with signed saturation.
These instructions can operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

12.0.49. PSUBUSB/PSUBUSW-Subtract Packed Unsigned Integers with Unsigned Saturation

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD subtract of the packed unsigned integers of the source operand (second operand) from the packed unsigned integers of the destination operand (first operand), and stores the packed unsigned integer results in the destination operand Overflow is handled with unsigned saturation.
These instructions can operate on either 64-bit or 128-bit operands. When operating on 64-bit operands, the destination operand must be an MMX technology register and the source operand can be either an MMX technology register or a 64-bit memory location. When operating on 128-bit operands, the destination operand must be an XMM register and the source operand can be either an XMM register or a 128-bit memory location.

Flags Affected None.

12.0.50. PMULHW-Multiply Packed Signed Integers and Store High Result

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD signed multiply of the packed signed word integers in the destina- tion operand (first operand) and the source operand (second operand), and stores the high 16 bits of each intermediate 32-bit result in the destination operand. (Figure 4-3 shows this operation when using 64-bit operands.) The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX tech- nology register or an XMM register.

Flags Affected None.

12.0.51. PMULLW-Multiply Packed Signed Integers and Store Low Result

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD signed multiply of the packed signed word integers in the destina- tion operand (first operand) and the source operand (second operand), and stores the low 16 bits of each intermediate 32-bit result in the destination operand. (Figure 4-3 shows this operation when using 64-bit operands.) The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX tech- nology register or an XMM register.

Flags Affected None.

13. MMX Comparison Instructions

13.0.52. PCMPEQB/PCMPEQW/PCMPEQD- Compare Packed Data for Equal

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD compare for equality of the packed bytes, words, or doublewords in the destination operand (first operand) and the source operand (second operand). If a pair of data elements is equal, the corresponding data element in the destination operand is set to all 1s; otherwise, it is set to all 0s. The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register.

Flags Affected None.

13.0.53. PCMPGTB/PCMPGTW/PCMPGTD-Compare Packed Signed Integers for Greater Than

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD signed compare for the greater value of the packed byte, word, or doubleword integers in the destination operand (first operand) and the source operand (second operand). If a data element in the destination operand is greater than the corresponding date element in the source operand, the corresponding data element in the destination operand is set to all 1s; otherwise, it is set to all 0s. The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register.

Flags Affected None.

14. MMX Bitwise Logical and Shift/Rotate Instructions

14.0.54. PAND/PANDN/POR/PXOR-Logical AND/NAND/OR/XOR

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a bitwise logical AND/NAND/OR/XOR operation on the source operand (second operand) and the destination operand (first operand) and stores the result in the destination operand. The source operand can be an MMX technology register or a 64-bit memory location or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register.

Flags Affected None.

14.0.55. PSLLW/PSLLD/PSLLQ-Shift Packed Data Left Logical

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128, imm8

Description Shifts the bits in the individual data elements (words, doublewords, or quadword) in the destination operand (first operand) to the left by the number of bits specified in the count operand (second operand). As the bits in the data elements are shifted left, the empty low-order bits are cleared (set to 0). If the value specified by the count operand is greater than 15 (for words), 31 (for doublewords), or 63 (for a quadword), then the destination operand is set to all 0s.
The destination operand may be an MMX technology register or an XMM register; the count operand can be either an MMX technology register or an 64-bit memory loca- tion, an XMM register or a 128-bit memory location, or an 8-bit immediate. Note that only the first 64-bits of a 128-bit count operand are checked to compute the count.

Flags Affected None.

14.0.56. PSRLW/PSRLD/PSRLQ-Shift Packed Data Right Logical

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128, imm8

Description Shifts the bits in the individual data elements (words, doublewords, or quadword) in the destination operand (first operand) to the right by the number of bits specified in the count operand (second operand). As the bits in the data elements are shifted right, the empty high-order bits are cleared (set to 0). If the value specified by the count operand is greater than 15 (for words), 31 (for doublewords), or 63 (for a quadword), then the destination operand is set to all 0s.
The destination operand may be an MMX technology register or an XMM register; the count operand can be either an MMX technology register or an 64-bit memory loca- tion, an XMM register or a 128-bit memory location, or an 8-bit immediate. Note that only the first 64-bits of a 128-bit count operand are checked to compute the count.

Flags Affected None.

14.0.57. PSRAW/PSRAD-Shift Packed Data Right Arithmetic

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128, imm8

Description Shifts the bits in the individual data elements (words or doublewords) in the destination operand (first operand) to the right by the number of bits specified in the count operand (second operand). As the bits in the data elements are shifted right, the empty high-order bits are filled with the initial value of the sign bit of the data element. If the value specified by the count operand is greater than 15 (for words) or 31 (for doublewords), each destination data element is filled with the initial value of the sign bit of the element.

Flags Affected None.

Parte IV
Instrucciones SSE

15. SSE Data Transfer Instructions

15.0.58. MOVAPS/MOVUPS-Move Aligned/Unaligned Packed Single-Precision Floating-Point Values

Dest (First Op)	Src (Second Op)
xmm	xmm, m128

Description Moves a double quadword containing four packed single-precision floating-point values from the source operand (second operand) to the destination operand (first operand). This instruction can be used to load an XMM register from a 128-bit memory location, to store the contents of an XMM register into a 128-bit memory location, or to move data between two XMM registers. When the source or destina- tion operand is a memory operand, the operand must be aligned on a 16-byte boundary or a general-protection exception (GP) is generated. *MOVUPS es idéntico salvo que no genera la excepción*

15.0.59. MOVHPS/MOVLPS-Move High/Low Packed Single-Precision Floating-Point Values

Dest (First Op)	Src (Second Op)
xmm, m64	m64, xmm

Description Moves two packed single-precision floating-point values from the source operand (second operand) to the destination operand (first operand). The source and destina- tion operands can be an XMM register or a 64-bit memory location. This instruction allows two single-precision floating-point values to be moved to and from the high/low quadword of an XMM register and memory. It cannot be used for register to register or memory to memory moves. When the destination operand is an XMM register, the low quadword of the register remains unchanged.

15.0.60. MOVHLPS- Move Packed Single-Precision Floating-Point Values High to Low

Dest (First Op)	Src (Second Op)
xmm	xmm

Description Moves two packed single-precision floating-point values from the high/low quadword of the source operand (second operand) to the low/high quadword of the destination operand (first operand). The high quadword of the destination operand is left unchanged.

15.0.61. MOVMSKPS-Extract Packed Single-Precision Floating-Point Sign Mask

Dest (First Op)	**Src (Second Op)**
reg	xmm

Description Extracts the sign bits from the packed single-precision floating-point values in the source operand (second operand), formats them into a 4-bit mask, and stores the mask in the destination operand (first operand). The source operand is an XMM register, and the destination operand is a general-purpose register. The mask is stored in the 4 low-order bits of the destination operand. Zero-extend the upper bits of the destination operand.

15.0.62. MOVSS-Move Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm,m32

Description Moves a scalar single-precision floating-point value from the source operand (second operand) to the destination operand (first operand). The source and destination operands can be XMM registers or 32-bit memory locations. This instruction can be used to move a single-precision floating-point value to and from the low doubleword of an XMM register and a 32-bit memory location, or to move a single-precision floating-point value between the low doublewords of two XMM registers. The instruc- tion cannot be used to transfer data between memory locations. When the source and destination operands are XMM registers, the three high-order doublewords of the destination operand remain unchanged. When the source operand is a memory location and destination operand is an XMM registers, the three high-order doublewords of the destination operand are cleared to all 0s.

15.0.63. ADDPS/ADDSS-Add Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm,m128 [m32 para ADDSS]

Description Performs a SIMD add of the four packed single-precision floating-point values from the source operand (second operand) and the destination operand (first operand), and stores the packed single-precision floating-point results in the destination operand.
The source operand can be an XMM register or a 128-bit memory location. The destination operand is an XMM register.
ADDSS es idéntico a ADDPS salvo que solo trabaja con el primer single-precision fp value

15.0.64. SUBPS/SUBSS-Subtract Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para SUBSS]

Description Performs a SIMD subtract of the four packed single-precision floating-point values in the source operand (second operand) from the four packed single-precision floating- point values in the destination operand (first operand), and stores the packed single- precision floating-point results in the destination operand. The source operand can be an XMM register or a 128-bit memory location. The destination operand is an XMM register.
SUBSS es idéntico a SUBPS salvo que solo trabaja con el primer single-precision fp value

15.0.65. MULPS/MULSS-Multiply Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para MULSS]

Description Performs a SIMD multiply of the four packed single-precision floating-point values from the source operand (second operand) and the destination operand (first operand), and stores the packed single-precision floating-point results in the desti- nation operand. The source operand can be an XMM register or a 128-bit memory location. The destination operand is an XMM register.
MULSS es idéntico a MULPS salvo que solo trabaja con el primer single-precision fp value

15.0.66. DIVPS/DIVSS-Divide Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para DIVSS]

Description Performs a SIMD divide of the four packed single-precision floating-point values in the destination operand (first operand) by the four packed single-precision floating-point values in the source operand (second operand), and stores the packed single- precision floating-point results in the destination operand. The source operand can be an XMM register or a 128-bit memory location.
DIVSS es idéntico a DIVPS salvo que solo trabaja con el primer single-precision fp value

15.0.67. RCPPS/RCPSS-Compute Reciprocals of Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para RCPSS]

Description Performs a SIMD computation of the approximate reciprocals of the four packed single-precision floating-point values in the source operand (second operand) stores the packed single-precision floating-point results in the destination operand. The source operand can be an XMM register or a 128-bit memory location. The destina- tion operand is an XMM register.
RCPSS es idéntico a RCPPS salvo que solo trabaja con el primer single-precision fp value

15.0.68. SQRTPS-Compute Square Roots of Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para SQRTSS]

Description Performs a SIMD computation of the square roots of the four packed single-precision floating-point values in the source operand (second operand) stores the packed single-precision floating-point results in the destination operand. The source operand can be an XMM register or a 128-bit memory location. The destination operand is an XMM register.
SQRTSS es idéntico a SQRTPS salvo que solo trabaja con el primer single-precision fp value

15.0.69. RSQRTPS/RSQRTSS-Compute Reciprocals of Square Roots of Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para RSQRTSS]

Description Performs a SIMD computation of the approximate reciprocals of the square roots of the four packed single-precision floating-point values in the source operand (second operand) and stores the packed single-precision floating-point results in the destina- tion operand. The source operand can be an XMM register or a 128-bit memory loca- tion. The destination operand is an XMM register.
RSQRTSS es idéntico a RSQRTPS salvo que solo trabaja con el primer single-precision fp value

15.0.70. MAXPS/MAXSS-Return Maximum Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para MAXSS]

Description Performs a SIMD compare of the packed single-precision floating-point values in the destination operand (first operand) and the source operand (second operand), and returns the maximum value for each pair of values to the destination operand. The source operand can be an XMM register or a 128-bit memory location. The destina- tion operand is an XMM register.
MAXSS es idéntico a MAXPS salvo que solo trabaja con el primer single-precision fp value

15.0.71. MINPS/MINSS-Return Maximum Packed/Scalar Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**
xmm	xmm, m128 [m32 para MINSS]

Description Performs a SIMD compare of the packed single-precision floating-point values in the destination operand (first operand) and the source operand (second operand), and returns the minimum value for each pair of values to the destination operand. The source operand can be an XMM register or a 128-bit memory location. The destina- tion operand is an XMM register.
MINSS es idéntico a MINPS salvo que solo trabaja con el primer single-precision fp value

15.0.72. CMPPS-Compare Packed Single-Precision Floating-Point Values

Dest (First Op)	**Src (Second Op)**	**(Third Op)**
xmm	xmm, m128 [m32 para CMPSS]	imm8

Description Performs a SIMD compare of the four packed single-precision floating-point values in the source operand (second operand) and the destination operand (first operand) and returns the results of the comparison to the destination operand. The compar- ison predicate operand (third operand) specifies the type of comparison performed on each of the pairs of packed values. The result of each comparison is a doubleword mask of all 1s (comparison true) or all 0s (comparison false). The source operand can be an XMM register or a 128-bit memory location. The desti- nation operand is an XMM register. The comparison predicate operand is an 8-bit immediate, the first 3 bits of which define the type of comparison to be made (Ver abajo) Bits 3 through 7 of the immediate are reserved.

- Conditional Codes: EQ - 0 | LT - 1 | LE - 2 | UNORD - 3 | NEQ - 4 | NLT - 5 | NLE - 6 | ORD - 7 |

CMPSS es idéntico a CMPPS salvo que solo trabaja con el primer single-precision fp value

15.0.73. COMISS/UCOMISS-(Unordered)Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS

Dest (First Op)	Src (Second Op)
xmm	xmm,m32

Description Compares the single-precision floating-point values in the low doublewords of operand 1 (first operand) and operand 2 (second operand), and sets the ZF, PF, and CF flags in the EFLAGS register according to the result (unordered, greater than, less than, or equal). The OF, SF, and AF flags in the EFLAGS register are set to 0. The unordered result is returned if either source operand is a NaN (QNaN or SNaN). Operand 1 is an XMM register; Operand 2 can be an XMM register or a 32 bit memory location.
The COMISS instruction differs from the UCOMISS instruction in that it signals a SIMD floating-point invalid operation exception (I) when a source operand is either a QNaN or SNaN. The UCOMISS instruction signals an invalid numeric exception only if a source operand is an SNaN.

16. SSE Logical, Shuffle and Unpack Instructions

16.0.74. ANDPS/ANDNPS/ORPS/XORPS-Bitwise Logical AND/NAND/OR/XOR of Packed Single-Precision Floating-Point Values

Dest (First Op)	Src (Second Op)
xmm	xmm, m128

Description Performs a bitwise logical AND/NAND/OR/XOR of the four packed single-precision floating-point values from the source operand (second operand) and the destination operand (first operand), and stores the result in the destination operand. The source operand can be an XMM register or a 128-bit memory location. The desti- nation operand is an XMM register.

16.0.75. SHUFPS-Shuffle Packed Single-Precision Floating-Point Values

Dest (First Op)	Src (Second Op)	(Third Op)
xmm	xmm, m128	imm8

Description Moves two of the four packed single-precision floating-point values from the destina- tion operand (first operand) into the low quadword of the destination operand; moves two of the four packed single-precision floating-point values from the source operand (second operand) into to the high quadword of the destination operand (see Figure 4-15). The select operand (third operand) determines which values are moved to the destination operand.
The source operand can be an XMM register or a 128-bit memory location. The destination operand is an XMM register. The select operand is an 8-bit immediate: bits 0 and 1 select the value to be moved from the destination operand to the low double- word of the result, bits 2 and 3 select the value to be moved from the destination operand to the second doubleword of the result, bits 4 and 5 select the value to be moved from the source operand to the third doubleword of the result, and bits 6 and 7 select the value to be moved from the source operand to the high doubleword of the result.

16.0.76. UNPCKHPS-Unpack and Interleave High Packed Single-Precision Floating-Point Values

Dest (First Op)	Src (Second Op)
xmm	xmm, m128

Description Performs an interleaved unpack of the high-order single-precision floating-point values from the source operand (second operand) and the destination operand (first operand). See Figure 4-17. The source operand can be an XMM register or a 128-bit memory location; the destination operand is an XMM register.

Operation DEST[31:0] ← DEST[95:64]; DEST[63:32] ← SRC[95:64]; DEST[95:64] ← DEST[127:96]; DEST[127:96] ← SRC[127:96];

16.0.77. UNPCKLPS-Unpack and Interleave Low Packed Single-Precision Floating-Point Values

Dest (First Op)	Src (Second Op)
xmm	xmm, m128

Description Performs an interleaved unpack of the low-order single-precision floating-point values from the source operand (second operand) and the destination operand (first operand). See Figure 4-19. The source operand can be an XMM register or a 128-bit memory location; the destination operand is an XMM register.

Operation DEST[31:0] ← DEST[31:0]; DEST[63:32] ← SRC[31:0]; DEST[95:64] ← DEST[63:32]; DEST[127:96] ← SRC[63:32];

17. SSE Conversion Instructions

17.0.78. CVTPI2PS-Convert Packed Dword Integers to Packed Single-Precision FP Values

Dest (First Op)	Src (Second Op)
xmm	mm/m64

Description Converts two packed signed doubleword integers in the source operand (second operand) to two packed single-precision floating-point values in the destination operand (first operand). The source operand can be an MMX technology register or a 64-bit memory location. The destination operand is an XMM register. The results are stored in the low quad- word of the destination operand, and the high quadword remains unchanged. When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register.

17.0.79. CVTSI2SS-Convert Dword Integer to Scalar Single-Precision FP Value

Dest (First Op)	Src (Second Op)
xmm	r/m32, r/m64

Description Converts a signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the source operand (second operand) to a single-precision floating-point value in the destination operand (first operand). The source operand can be a general-purpose register or a memory location. The destination operand is an XMM register. The result is stored in the low doubleword of the destination operand, and the upper three doublewords are left unchanged. When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register.

17.0.80. CVTPS2PI-Convert Packed Single-Precision FP Values to Packed Dword Integers

Dest (First Op)	Src (Second Op)
mm	xmm/m64

Description Converts two packed single-precision floating-point values in the source operand (second operand) to two packed signed doubleword integers in the destination operand (first operand).
The source operand can be an XMM register or a 128-bit memory location. The destination operand is an MMX technology register. When the source operand is an XMM register, the two single-precision floating-point values are contained in the low quad- word of the register. When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register. If a converted result is larger than the maximum signed doubleword integer, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000H) is returned.

17.0.81. CVTTPS2PI-Convert with Truncation Packed Single-Precision FP Values to Packed Dword Integers

Dest (First Op)	Src (Second Op)
mm	xmm/m64

Description Converts two packed single-precision floating-point values in the source operand (second operand) to two packed signed doubleword integers in the destination operand (first operand). The source operand can be an XMM register or a 64-bit memory location. The destination operand is an MMX technology register. When the source operand is an XMM register, the two single-precision floating-point values are contained in the low quadword of the register. When a conversion is inexact, a truncated (round toward zero) result is returned. If a converted result is larger than the maximum signed doubleword integer, the floating- point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000H) is returned.

17.0.82. CVTSS2SI-Convert Scalar Single-Precision FP Value to Dword Integer

Dest (First Op)	Src (Second Op)
r32, r64	xmm, m32

Description Converts a single-precision floating-point value in the source operand (second operand) to a signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the destination operand (first operand). The source operand can be an XMM register or a memory location. The destination operand is a general-purpose register. When the source operand is an XMM register, the single-precision floating- point value is contained in the low doubleword of the register. When a conversion is inexact, the value returned is rounded according to the rounding control bits in the MXCSR register. If a converted result is larger than the maximum signed doubleword integer, the floating-point invalid exception is raised, and if this exception is masked, the indefinite integer value (80000000H) is returned.

17.0.83. CVTTSS2SI-Convert with Truncation Scalar Single-Precision FP Value to Dword Integer

Dest (First Op)	Src (Second Op)
r32, r64	xmm, m32

Description onverts a single-precision floating-point value in the source operand (second operand) to a signed doubleword integer (or signed quadword integer if operand size is 64 bits) in the destination operand (first operand). The source operand can be an XMM register or a 32-bit memory location. The destination operand is a general- purpose register. When the source operand is an XMM register, the single-precision floating-point value is contained in the low doubleword of the register. When a conversion is inexact, a truncated (round toward zero) result is returned. If a converted result is larger than the maximum signed doubleword integer, the floating- point invalid exception is raised. If this exception is masked, the indefinite integer value (80000000H) is returned.

18. SSE 64-Bit SIMD Integer Instructions

18.0.84. PAVGB/PAVGW-Average Packed Integers

Dest (First Op)	Src (Second Op)
mm, xmm	mm, m64, xmm, m128

Description Performs a SIMD average of the packed unsigned integers from the source operand (second operand) and the destination operand (first operand), and stores the results in the destination operand. For each corresponding pair of data elements in the first and second operands, the elements are added together, a 1 is added to the temporary sum, and that result is shifted right one bit position. The source operand can be an MMX technology register or a 64-bit memory location or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register.

Flags Affected None.

18.0.85. PEXTRW-Extract Word

Dest (First Op)	Src (Second Op)	(Third Op)
reg, m16	mm, xmm	imm8

Description Copies the word in the source operand (second operand) specified by the count operand (third operand) to the destination operand (first operand). The source operand can be an MMX technology register or an XMM register. The destination operand can be the low word of a general-purpose register or a 16-bit memory address. The count operand is an 8-bit immediate. When specifying a word location in an MMX technology register, the 2 least-significant bits of the count operand specify the location; for an XMM register, the 3 least-significant bits specify the loca- tion. The content of the destination register above bit 16 is cleared (set to all 0s).

Flags Affected None.

18.0.86. PINSRW-Insert Word

Dest (First Op)	Src (Second Op)	(Third Op)
mm, xmm	r32, m16	imm8

Description Copies a word from the source operand (second operand) and inserts it in the desti- nation operand (first operand) at the location specified with the count operand (third operand). (The other words in the destination register are left untouched.) The source operand can be a general-purpose register or a 16-bit memory location. (When the source operand is a general-purpose register, the low word of the register is copied.) The destination operand can be an MMX technology register or an XMM register. The count operand is an 8-bit immediate. When specifying a word location in an MMX technology register, the 2 least-significant bits of the count operand specify the location; for an XMM register, the 3 least-significant bits specify the location.

Flags Affected None.

18.0.87. PMAXSW/PMAXSW-Maximum/Minimum of Packed Signed Word Integers

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD compare of the packed signed word integers in the destination operand (first operand) and the source operand (second operand), and returns the maximum/minimum value for each pair of word integers to the destination operand. The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register.

Flags Affected None.

18.0.88. PMAXUB/PMINUB-Maximum/Minimum of Packed Unsigned Byte Integers

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD compare of the packed unsigned byte integers in the destination operand (first operand) and the source operand (second operand), and returns the maximum/minimum value for each pair of byte integers to the destination operand. The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX technology register or an XMM register.

Flags Affected None.

18.0.89. PMOVMSKB-Move Byte Mask

Dest (First Op)	Src (Second Op)
r32, r64, reg	mm, mm, xmm

Description Creates a mask made up of the most significant bit of each byte of the source operand (second operand) and stores the result in the low byte or word of the destination operand (first operand). The source operand is an MMX technology register or an XMM register; the destination operand is a general-purpose register. When oper- ating on 64-bit operands, the byte mask is 8 bits; when operating on 128-bit oper- ands, the byte mask is 16-bits.

Flags Affected None.

18.0.90. PMULHUW-Multiply Packed Unsigned Integers and Store High Result

Dest (First Op)	Src (Second Op)
mm, xmm	mm, xmm, m64, m128

Description Performs a SIMD unsigned multiply of the packed unsigned word integers in the destination operand (first operand) and the source operand (second operand), and stores the high 16 bits of each 32-bit intermediate results in the destination operand. (Figure 4-3 shows this operation when using 64-bit operands.) The source operand can be an MMX technology register or a 64-bit memory location, or it can be an XMM register or a 128-bit memory location. The destination operand can be an MMX tech- nology register or an XMM register.

Flags Affected None.

18.0.91. PSHUFW-Shuffle Packed Words

Dest (First Op)	Src (Second Op)	(Third Op)
mm	mm, m64	imm8

Description Copies words from the source operand (second operand) and inserts them in the destination operand (first operand) at word locations selected with the order operand (third operand). This operation is similar to the operation used by the PSHUFD instruction, which is illustrated in Figure 4-7. For the PSHUFW instruction, each 2-bit field in the order operand selects the contents of one word location in the destination operand. The encodings of the order operand fields select words from the source operand to be copied to the destination operand. The source operand can be an MMX technology register or a 64-bit memory location. The destination operand is an MMX technology register. The order operand is an 8-bit immediate. Note that this instruction permits a word in the source operand to be copied to more than one word location in the destination operand.

Flags Affected None.