Intel® Extended Memory 64 Technology Software Developer's Guide Volume 1 of 2

Revision 1.1

NOTE: This guide consists of volumes 1 and 2. Refer to both volumes when evaluating your design needs.

Order Number: 300834-002

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

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

Developers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Improper use of reserved or undefined features or instructions may cause unpredictable behavior or failure in developer's software code when running on an Intel processor. Intel reserves these features or instructions for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from their unauthorized use.

Intel® IA-32 architecture processors (e.g., Pentium® 4, Intel® XeonTM, and Pentium III processors) may contain design defects or errors known as errata. Current characterized errata are available on request.

Intel, Intel386, Intel486, Pentium, Intel Xeon are trademarks or registered trademarks of Intel Corporation and its subsidiaries in the United States and other countries.

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

COPYRIGHT © 1997-2004 INTEL CORPORATION

TABLE OF CONTENTS

The content listing reflects Volumes 1 and 2. Volume 1 houses the TOC, Chapters 1 and 2. Volume 2 houses the rest of the specification.

CHAPTI		
	UCTION	
1.1.	INTEL® EXTENDED MEMORY 64 TECHNOLOGY	1-
1.2.	OPERATING MODES	
1.2.1.	IA-32e Mode	
1.2.2.	64-Bit Mode	
1.2.3.	Compatibility Mode	
1.2.4.	Legacy Modes	
1.2.5.	System Management Mode	
1.3.	REGISTER-SET CHANGES	
1.3.1.	General-Purpose Registers (GPRs)	
1.3.2.	Streaming SIMD Extension (SSE) Registers	
1.3.3.	System Registers	1-
1.3.3.1.	Extended Feature Enable Register (IA32_EFER)	1-
1.3.3.2.	Control Registers	1-
1.3.3.3.	Descriptor Table Registers	
1.3.3.4.	Debug Registers	1-6
1.4.	INSTRUCTION-SET CHANGES	1-
1.4.1.	Address-Size and Operand-Size Prefixes	
1.4.2.	REX Prefixes	
1.4.2.1.	Encoding	
1.4.2.2.	REX Prefix Fields	
1.4.2.3. 1.4.2.4.	Displacement	
	Direct Memory-Offset MOVs	
1.4.2.5.	Immediates	
1.4.2.6. 1.4.2.7.	RIP-Relative Addressing	
1.4.2.7. 1.4.3.	New Encodings for Control and Debug Registers	1-1. 1-1:
1.4.3. 1.4.4.	New Instructions	1-14 1-14
1.4. 4 . 1.4.5.	Stack Pointer.	
1.4.6.	Branches	
1.5.	MEMORY ORGANIZATION.	
1.5.1.	Address Calculations in 64-Bit Mode	1-14
1.5.2.	Canonical Addressing	
1.6.	OPERATING SYSTEM CONSIDERATIONS	1-1!
1.6.1.	CPUID Instruction	
1.6.2.	Register Settings and IA-32e Mode	
1.6.3.	Processor Modes	
1.6.3.1.	IA-32e Mode 1	
1.6.3.2.	Activating IA-32e Mode 1	
1.6.3.3.	Virtual-8086 Mode	
1.6.3.4.	Compatibility Mode1	1-18
1.6.4.	Segmentation	
1.6.4.1.	Code Segments	
1.6.4.2.	Segment LOAD Instructions	1-19
1.6.4.3.	System Descriptors	1-20
1.6.5.	Linear Addressing and Paging	I-2
1.6.5.1.	Software Address Translations in 64-Bit Mode	
1.6.5.2.	Paging Data Structures	I-2
1.6.5.3.	Overall Page Protection	1-2
1.6.5.4.	Reserved Bit Checking 1	
1.6.6.	Enhanced Legacy-Mode Paging	
1.6.7.	CR2 and CR3	
1.6.8.	Address Translation	1-2
1.6.9.	Privilege-Level Transitions and Far Transfers	1-28



1.6.9.1.	Call Gates	
1.6.9.2.	Privilege-Level Changes and Stack Switching	1-30
1.6.9.3.	Fast System Calls	1-31
1.6.9.4.	Task State Segments	
1.6.10.	Interrupts	
1.6.10.1.	Gate Descriptor Format	1-33
1.6.10.2.	Stack Frame	1-34
1.6.10.3.	. IRET	1-34
1.6.10.4.	Stack Switching	1-35
1.6.10.5.	Interrupt Stack Table	1-35
1.6.10.6.		
1.6.10.7.		1-36
1.7.	GENERAL RULES FOR 64-BIT MODE	1-37
1.7.1.	Other Guidelines	
CHAPTE	=D 2	
	CTION SET REFERENCE (A-L)	
2.1.	INTERPRETING THE INSTRUCTION REFERENCE PAGES	0 -
2.1.		
	The Instruction Summary Table	
2.1.1.1.	Opcode Column in the Instruction Summary Table	
2.1.1.2.	Instruction Column in the Instruction Summary Table	2-2
2.1.1.3.	64-bit Mode Column in the Instruction Summary Table	2-4
2.1.1.4.	Compatibility/Legacy Mode Column in the Instruction Summary Table	
2.1.1.5.	Description Column in the Instruction Summary Table.	
2.1.2.	Description Section	
2.1.3.	Operation Section	
2.1.3.1.	IA-32e Mode Operation	2-8
2.1.4.	Flags Affected	
2.1.5.	FPU Flags Affected	
2.1.6.	Protected Mode Exceptions	
2.1.7.	Real-Address Mode Exceptions	
2.1.8.	Virtual-8086 Mode Exceptions	
2.1.9.	Floating-Point Exceptions	2-9
2.1.10.	SIMD Floating-Point Exceptions	2-10
2.2.	INSTRUCTION REFERENCE	2-10
	AAA—ASCII Adjust After Addition	2-1 ⁻¹
	AAD—ASCII Adjust AX Before Division	
	AAM—ASCII Adjust AX After Multiply	
	AAS—ASCII Adjust AL After Subtraction	
	ADC—Add with Carry	
	ADD—Add	
	ADDPD—Add Packed Double-Precision Floating-Point Values	
	ADDPS—Add Packed Single-Precision Floating-Point Values	
	ADDSD—Add Scalar Double-Precision Floating-Point Values	2-23
	ADDSS—Add Scalar Single-Precision Floating-Point Values	2-25
	ADDSUBPD—Packed Double-Precision Floating-Point Add/Subtract	
	ADDSUBPS—Packed Single-Precision Floating-Point Add/Subtract	
	AND—Logical AND	
	ANDPD—Bitwise Logical AND of Packed Double-Precision Floating-Point Values	
	ANDPS—Bitwise Logical AND of Packed Single-Precision Floating-Point Values	
	ANDNPD—Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values	
	ANDNPS—Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values	2-39
	ARPL—Adjust RPL Field of Segment Selector	2-4
	BOUND—Check Array Index Against Bounds	
	BSF—Bit Scan Forward	
	BSR—Bit Scan Reverse	
	BSWAP—Byte Swap	
	BT—Bit Test	
	BTC—Bit Test and Complement	
	BTR—Bit Test and Reset	2-5
	BTS—Bit Test and Set	2-53

CALL—Call Procedure	. 2-55
CBW/CWDE/CDQE—Convert Byte to Word/Convert Word to Doubleword/Convert Doubleword to	
Quadword	. 2-58
CDQ—Convert Double to Quad	. 2-59
CLC—Clear Carry Flag	
CLD—Clear Direction Flag	
CLFLUSH—Flush Cache Line.	
CLI—Clear Interrupt Flag	
CLTS—Clear Task-Switched Flag in CR0.	
CMC—Complement Carry Flag.	
CMOVcc—Conditional Move.	
CMP—Compare Two Operands	
CMPPD—Compare Packed Double-Precision Floating-Point Values.	
CMPPS—Compare Packed Single-Precision Floating-Point Values	
CMPS/CMPSB/CMPSD/CMPSQ—Compare String Operands	
CMPSD—Compare Scalar Double-Precision Floating-Point Values	
CMPSS—Compare Scalar Single-Precision Floating-Point Values	. 2-79
CMPXCHG—Compare and Exchange	. 2-81
CMPXCHG8B/CMPXCHG16B—Compare and Exchange 8 Bytes	. 2-83
COMISD—Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS	. 2-85
COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS	
CPUID—CPU Identification	
CVTDQ2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point	. – .
Values	. 2-99
CVTDQ2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point	00
Values	2-101
CVTPD2DQ—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword	2-10
-	0.100
	2-103
CVTPD2PI—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword	
Integers	2-105
CVTPD2PS—Covert Packed Double-Precision Floating-Point Values to Packed Single-Precision	
Floating-Point Values	2-107
CVTPI2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point	
Values	2-109
CVTPI2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point	
Values	2-111
CVTPS2DQ—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword	
Integers	2-113
CVTPS2PD—Covert Packed Single-Precision Floating-Point Values to Packed Double-Precision	
	2-115
CVTPS2PI—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword	
Integers	2-117
CVTSD2SI—Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer	
CVTSD2SS—Convert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision	
Floating-Point Value	2-121
CVTSI2SD—Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value	
CVTSI2SS—Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value	
	2-120
CVTSS2SD—Convert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision	0.40-
Floating-Point Value	
CVTSS2SI—Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer	2-129
CVTTPD2PI—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed	
Doubleword Integers	2-131
${\tt CVTTPD2DQ-Convert\ with\ Truncation\ Packed\ Double-Precision\ Floating-Point\ Values\ to\ Packed}$	
Doubleword Integers	2-133
CVTTPS2DQ—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed	
Doubleword Integers	2-135
CVTTPS2PI—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed	
Doubleword Integers	2-137



CV115D2SI—Convert with 1runcation Scalar Double-Precision Floating-Point value to Signed	
Doubleword Integer	2-139
CVTTSS2SI—Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword	
Integer	2-141
CWD/CDQ/CQQ—Convert Word to Doubleword/Convert Doubleword to Quadword/Convert	
Quadword to Double Quadword	2-143
DAA—Decimal Adjust AL after Addition	
DAS—Decimal Adjust AL after Subtraction.	
DEC—Decrement by 1	
DIV—Unsigned Divide	
DIVPD—Divide Packed Double-Precision Floating-Point Values	
DIVPS—Divide Packed Single-Precision Floating-Point Values	
DIVSD—Divide Scalar Double-Precision Floating-Point Values	
DIVSS—Divide Scalar Single-Precision Floating-Point Values	
EMMS—Empty MMX State	2-158
ENTER—Make Stack Frame for Procedure Parameters	2-159
F2XM1—Compute 2x–1	2-160
FABS—Absolute Value	
FADD/FADDP/FIADD—Add	
FBLD—Load Binary Coded Decimal	
FBSTP—Store BCD Integer and Pop	
FCHS—Change Sign	
FCLEX/FNCLEX—Clear Exceptions.	
FCMOVcc—Floating-Point Conditional Move	
FCOM/FCOMP/FCOMPP—Compare Floating Point Values	
FCOMI/FCOMIP/ FUCOMI/FUCOMIP—Compare Floating Point Values and Set EFLAGS	
FCOS—Cosine	
FDECSTP—Decrement Stack-Top Pointer	2-175
FDIV/FDIVP/FIDIV—Divide	
FDIVR/FDIVRP/FIDIVR—Reverse Divide	2-178
FFREE—Free Floating-Point Register	
FICOM/FICOMP—Compare Integer	
FILD—Load Integer	
FINCSTP—Increment Stack-Top Pointer	
FINIT/FNINIT—Initialize Floating-Point Unit	
FIST/FISTP—Store Integer	
FISTTP—Store Integer with Truncation	
FLD—Load Floating Point Value	
FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ—Load Constant	
FLDCW—Load x87 FPU Control Word	
FLDENV—Load x87 FPU Environment	
FMUL/FMULP/FIMUL—Multiply	2-198
FNOP—No Operation	2-200
FPATAN—Partial Arctangent	2-201
FPREM—Partial Remainder	2-202
FPREM1—Partial Remainder	
FPTAN—Partial Tangent.	
FRNDINT—Round to Integer	
FRSTOR—Restore x87 FPU State	
FSAVE/FNSAVE—Store x87 FPU State	
FSCALE—Scale	
	2-211
FSINCOS—Sine and Cosine.	
FSQRT—Square Root	
FST/FSTP—Store Floating Point Value	2-214
FSTCW/FNSTCW—Store x87 FPU Control Word	2-216
FSTENV/FNSTENV—Store x87 FPU Environment	2-218
FSTSW/FNSTSW—Store x87 FPU Status Word	
	-

	FSUB/FSUBP/FISUB—Subtract	
	FSUBR/FSUBRP/FISUBR—Reverse Subtract	2-224
	FTST—TEST	2-226
	FUCOM/FUCOMP/FUCOMPP—Unordered Compare Floating Point Values	
	FWAIT—Wait.	
	FXAM—Examine	
	FXCH—Exchange Register Contents	
	FXRSTOR—Restore x87 FPU, MMX, SSE, and SSE2 State	
	FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State	
	FXTRACT—Extract Exponent and Significand	
	FYL2X—Compute y * log2x.	
	FYL2XP1—Compute y * log2(x +1)	
	HADDPD—Horizontal Add Packed Double-Precision Floating-Point Values	
	HADDPS—Horizontal Add Packed Single-Precision Floating-Point Values	
	HLT—Halt	
	HSUBPD—Horizontal Subtract Packed Double-Precision Floating-Point Values	2-249
	HSUBPS—Horizontal Subtract Packed Single-Precision Floating-Point Values	2-251
	IDIV—Signed Divide	
	IMUL—Signed Multiply	
	IN—Input from Port	
	INC—Increment by 1	
	INS/INSB/INSW/INSD—Input from Port to String	
	INT n/INTO/INT 3—Call to Interrupt Procedure.	
	INVD—Invalidate Internal Caches	
	INVLPG—Invalidate TLB Entry	
	IRET/IRETD—Interrupt Return	
	Jcc—Jump if Condition Is Met	
	JMP—Jump	
	LAHF—Load Status Flags into AH Register	
	LAR—Load Access Rights Byte	
	LDDQU—Load Unaligned Double Quadword	2-277
	LDMXCSR—Load MXCSR Register	2-279
	LDS/LES/LFS/LGS/LSS—Load Far Pointer	2-281
	LEA—Load Effective Address	
	LEAVE—High Level Procedure Exit	
	LES—Load Full Pointer.	
	LFENCE—Load Fence	
	LFS—Load Full Pointer.	
	LGDT/LIDT—Load Global/Interrupt Descriptor Table Register	
	LGS—Load Full Pointer.	
	LLDT—Load Local Descriptor Table Register	
	LIDT—Load Interrupt Descriptor Table Register	
	LMSW—Load Machine Status Word	
	LOCK—Assert LOCK# Signal Prefix	
	LODS/LODSB/LODSW/LODSQ—Load String	
	LOOP/LOOPcc—Loop According to ECX Counter	2-297
	LSL—Load Segment Limit.	2-298
	LSS—Load Full Pointer	2-299
	LTR—Load Task Register	2-300
CHAPTER INSTRUCT	3 ION SET REFERENCE (M-Z)	
	MASKMOVDQU—Store Selected Bytes of Double Quadword	3-1
	MASKMOVQ—Store Selected Bytes of Quadword	
	MAXPD—Return Maximum Packed Double-Precision Floating-Point Values	
	MAXPS—Return Maximum Packed Single-Precision Floating-Point Values	
	MAXSD—Return Maximum Scalar Double-Precision Floating-Point Value	
	MAXSS—Return Maximum Scalar Single-Precision Floating-Point Value	৩-11



MFENCE—Memory Fence	
MINPD—Return Minimum Packed Double-Precision Floating-Point Values	3-14
MINPS—Return Minimum Packed Single-Precision Floating-Point Values	3-16
MINSD—Return Minimum Scalar Double-Precision Floating-Point Value	
MINSS—Return Minimum Scalar Single-Precision Floating-Point Value	
MONITOR—Setup Monitor Address	
MOV—Move	
MOV—Move to/from Control Registers	
MOV—Move to/from Debug Registers	
MOVAPD—Move Aligned Packed Double-Precision Floating-Point Values	
MOVAPS—Move Aligned Packed Single-Precision Floating-Point Values	
MOVD/MOVQ—Move Doubleword	
MOVDDUP—Move One Double-Precision Floating-Point Value and Duplicate	
MOVDQA—Move Aligned Double Quadword	
MOVDQU—Move Unaligned Double Quadword	
MOVDQ2Q—Move Quadword from XMM to MMX Register	
MOVHLPS— Move Packed Single-Precision Floating-Point Values High to Low	
MOVHPD—Move High Packed Double-Precision Floating-Point Value	
MOVHPS—Move High Packed Single-Precision Floating-Point Values	
MOVLHPS—Move Packed Single-Precision Floating-Point Values Low to High	
MOVLPD—Move Low Packed Double-Precision Floating-Point Value	3-48
MOVLPS—Move Low Packed Single-Precision Floating-Point Values	3-50
MOVMSKPD—Extract Packed Double-Precision Floating-Point Sign Mask	3-52
MOVMSKPS—Extract Packed Single-Precision Floating-Point Sign Mask	3-53
MOVNTDQ—Store Double Quadword Using Non-Temporal Hint	3-54
MOVNTI—Store Doubleword/Quadword Using Non-Temporal Hint	3-56
MOVNTPD—Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint	3-57
MOVNTPS—Store Packed Single-Precision Floating-Point Values Using Non-Temporal Hint	
MOVNTQ—Store of Quadword Using Non-Temporal Hint	3-61
MOVNTQ—Store of Quadword Using Non-Temporal Hint	
MOVQ—Move Quadword	3-63
MOVQ—Move Quadword	3-63 3-65
MOVQ—Move Quadword	3-63 3-65 3-66
MOVQ—Move Quadword	3-63 3-65 3-66 3-68
MOVQ—Move Quadword	3-63 3-65 3-66 3-68 3-70
MOVQ—Move Quadword	3-63 3-65 3-66 3-68 3-70 3-72
MOVQ—Move Quadword	3-63 3-65 3-66 3-68 3-70 3-72 3-74
MOVQ—Move Quadword	3-63 3-65 3-66 3-68 3-70 3-72 3-74 3-76
MOVQ—Move Quadword	3-63 3-65 3-66 3-68 3-70 3-72 3-74 3-76
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values.	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-78
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVUPS—Move with Zero-Extend	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-80 3-80
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-78 3-80 3-82 3-84
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-80 3-82 3-84 3-86
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-82 3-84 3-86 3-88
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend. MUL—Unsigned Multiply. MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-82 3-84 3-86 3-88 3-90
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-82 3-84 3-86 3-88 3-90 3-92
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait.	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-80 3-82 3-84 3-86 3-88 3-90 3-92 3-94
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend. MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-82 3-84 3-88 3-90 3-92 3-94 3-95
MOVQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait. NEG—Two's Complement Negation NOP—No Operation	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-83 3-84 3-86 3-88 3-90 3-92 3-94 3-95 3-97
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOP—No Operation NOT—One's Complement Negation	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-82 3-84 3-86 3-92 3-94 3-95 3-97 3-98
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend. MUL—Unsigned Multiply. MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-82 3-84 3-86 3-92 3-94 3-95 3-97 3-98 3-100
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MUAS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPD—Bitwise Logical OR of Double-Precision Floating-Point Values	3-63 3-65 3-66 3-70 3-72 3-74 3-76 3-83 3-84 3-86 3-92 3-94 3-95 3-97 3-98 3-100 3-102
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision FP Values Low and Duplicate MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPD—Bitwise Logical OR of Double-Precision Floating-Point Values ORPS—Bitwise Logical OR of Single-Precision Floating-Point Values	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-88 3-88 3-90 3-92 3-94 3-95 3-97 3-98 3-100 3-102 3-104
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULPS—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPD—Bitwise Logical OR of Double-Precision Floating-Point Values ORPS—Bitwise Logical OR of Single-Precision Floating-Point Values OUT—Output to Port	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-88 3-88 3-90 3-92 3-94 3-95 3-97 3-98 3-100 3-102 3-104 3-106
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULPS—Multiply Scalar Double-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSC—Two's Complement Negation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPD—Bitwise Logical OR of Double-Precision Floating-Point Values OUT—Output to Port. OUTS/OUTSB/OUTSW/OUTSD—Output String to Port.	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-78 3-88 3-88 3-90 3-92 3-94 3-95 3-100 3-102 3-104 3-106 3-107
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision FP Values Low and Duplicate MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSD—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPS—Bitwise Logical OR of Double-Precision Floating-Point Values OUTS/OUTSB/OUTSW/OUTSD—Output String to Port. PACKSSWB/PACKSSDW—Pack with Signed Saturation	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-78 3-88 3-88 3-90 3-95 3-95 3-95 3-100 3-102 3-104 3-106 3-107 3-109
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar SinglePrecision Floating-Point Values. MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULPS—Multiply Scalar Double-Precision Floating-Point Values MULSD—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MULSS—Multiply Scalar Single-Precision Floating-Point Values MUAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPD—Bitwise Logical OR of Double-Precision Floating-Point Values OUTS—Output to Port OUTS/OUTSB/OUTSW/OUTSD—Output String to Port. PACKSSWB/PACKSSDW—Pack with Signed Saturation PACKUSWB—Pack with Unsigned Saturation	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-78 3-80 3-82 3-84 3-86 3-95 3-95 3-95 3-95 3-100 3-102 3-104 3-106 3-107 3-109 3-111
MOVQ—Move Quadword MOVQ2DQ—Move Quadword from MMX to XMM Register. MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—Move Data from String to String MOVSD—Move Scalar Double-Precision Floating-Point Value MOVSHDUP—Move Packed Single-Precision FP Values High and Duplicate MOVSLDUP—Move Packed Single-Precision FP Values Low and Duplicate MOVSS—Move Scalar Single-Precision FP Values Low and Duplicate MOVSX/MOVSXD—Move with Sign-Extension MOVUPD—Move Unaligned Packed Double-Precision Floating-Point Values. MOVUPS—Move Unaligned Packed Single-Precision Floating-Point Values MOVZX—Move with Zero-Extend MUL—Unsigned Multiply MULPD—Multiply Packed Double-Precision Floating-Point Values MULPS—Multiply Packed Single-Precision Floating-Point Values MULSD—Multiply Scalar Double-Precision Floating-Point Values MULSD—Multiply Scalar Single-Precision Floating-Point Values MWAIT—Monitor Wait NEG—Two's Complement Negation NOP—No Operation NOP—No Operation NOT—One's Complement Negation OR—Logical Inclusive OR ORPS—Bitwise Logical OR of Double-Precision Floating-Point Values OUTS/OUTSB/OUTSW/OUTSD—Output String to Port. PACKSSWB/PACKSSDW—Pack with Signed Saturation	3-63 3-65 3-68 3-70 3-72 3-74 3-76 3-78 3-88 3-88 3-90 3-92 3-94 3-95 3-100 3-102 3-104 3-106 3-107 3-109 3-111 3-113

PADDSB/PADDSW—Add Packed Signed Integers with Signed Saturation	
PADDUSB/PADDUSW—Add Packed Unsigned Integers with Unsigned Saturation	
PAND—Logical AND	
PANDN—Logical AND NOT	
PAUSE—Spin Loop Hint	3-125
PAVGB/PAVGW—Average Packed Integers	3-126
PCMPEQB/PCMPEQW/PCMPEQD— Compare Packed Data for Equal	3-128
PCMPGTB/PCMPGTW/PCMPGTD—Compare Packed Signed Integers for Greater Than	3-130
PEXTRW—Extract Word	3-132
PINSRW—Insert Word	
PMADDWD—Multiply and Add Packed Integers	3-136
PMAXSW—Maximum of Packed Signed Word Integers	
PMAXUB—Maximum of Packed Unsigned Byte Integers	
PMINSW—Minimum of Packed Signed Word Integers	3-142
PMINUB—Minimum of Packed Unsigned Byte Integers	
PMOVMSKB—Move Byte Mask	
PMULHUW—Multiply Packed Unsigned Integers and Store High Result	
PMULHW—Multiply Packed Signed Integers and Store High Result	
PMULLW—Multiply Packed Signed Integers and Store Low Result	
PMULUDQ—Multiply Packed Unsigned Doubleword Integers	
POP—Pop a Value from the Stack	
POPA/POPAD—Pop All General-Purpose Registers	
POPF/POPFD—Pop Stack into EFLAGS Register	
POR—Bitwise Logical OR	3-159
PREFETCHh—Prefetch Data Into Caches	
PSADBW—Compute Sum of Absolute Differences	
PSHUFD—Shuffle Packed Doublewords	
PSHUFHW—Shuffle Packed High Words	
PSHUFLW—Shuffle Packed Low Words	
PSHUFW—Shuffle Packed Words	
PSLLDQ—Shift Double Quadword Left Logical	
PSLLW/PSLLD/PSLLQ—Shift Packed Data Left Logical	
PSRAW/PSRAD—Shift Packed Data Right Arithmetic	
PSRLDQ—Shift Double Quadword Right Logical	
PSRLW/PSRLD/PSRLQ—Shift Packed Data Right Logical	
PSUBB/PSUBW/PSUBD—Subtract Packed Integers	
PSUBQ—Subtract Packed Quadword Integers	
PSUBSB/PSUBSW—Subtract Packed Signed Integers with Signed Saturation	
PSUBUSB/PSUBUSW—Subtract Packed Unsigned Integers with Unsigned Saturation	
PUNPCKHBW/PUNPCKHWD/PUNPCKHQQ/PUNPCKHQQQ—	
Unpack High Data	3-188
PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ—	
Unpack Low Data	3-190
PUSH—Push Word or Doubleword Onto the Stack	3-192
PUSHA/PUSHAD—Push All General-Purpose Registers	
PUSHF/PUSHFD—Push EFLAGS Register onto the Stack	
PXOR—Logical Exclusive OR	
RCL/RCR/ROL/ROR-—Rotate	
RCPPS—Compute Reciprocals of Packed Single-Precision Floating-Point Values	
RCPSS—Compute Reciprocal of Scalar Single-Precision Floating-Point Values	
RDMSR—Read from Model Specific Register	
RDPMC—Read Performance-Monitoring Counters	
RDTSC—Read Time-Stamp Counter	
REP/REPE/REPZ/REPNE /REPNZ—Repeat String Operation Prefix	
RET—Return from Procedure	
ROL/ROR—Rotate	
RSM—Resume from System Management Mode	



RSQRTPS—Compute Reciprocals of Square Roots of Packed Single-Precision Floating-Point	
Values	
RSQRTSS—Compute Reciprocal of Square Root of Scalar Single-Precision Floating-Point Value .	3-21
SAHF—Store AH into Flags	3-21
SAL/SAR/SHL/SHR—Shift	3-21
SBB—Integer Subtraction with Borrow	3-22
SCAS/SCASB/SCASW/SCASD—Scan String	
SETcc—Set Byte on Condition	
SFENCE—Store Fence.	
SGDT/SIDT—Store Global/Interrupt Descriptor Table Register	2 22
SHL/SHR—Shift Instructions.	
SHLD—Double Precision Shift Left	
SHRD—Double Precision Shift Right	
SHUFPD—Shuffle Packed Double-Precision Floating-Point Values	
SHUFPS—Shuffle Packed Single-Precision Floating-Point Values	3-23
SIDT—Store Interrupt Descriptor Table Register	3-24
SLDT—Store Local Descriptor Table Register	3-24
SMSW—Store Machine Status Word	
SQRTPD—Compute Square Roots of Packed Double-Precision Floating-Point Values	
SQRTPS—Compute Square Roots of Packed Single-Precision Floating-Point Values	
SQRTSD—Compute Square Root of Scalar Double-Precision Floating-Point Value	
SQRTSS—Compute Square Root of Scalar Single-Precision Floating-Point Value	
STC—Set Carry Flag.	
STD—Set Direction Flag	
STI—Set Interrupt Flag	
STMXCSR—Store MXCSR Register State	
STOS/STOSB/STOSW/STOSQ—Store String	
STR—Store Task Register	3-26
SUB—Subtract	3-26
SUBPD—Subtract Packed Double-Precision Floating-Point Values	3-26
SUBPS—Subtract Packed Single-Precision Floating-Point Values	
SUBSD—Subtract Scalar Double-Precision Floating-Point Values	
SUBSS—Subtract Scalar Single-Precision Floating-Point Values	
SWAPGS—Swap GS Base Register	
SYSCALL—Fast System Call	
SYSENTER—Fast System Call	
SYSEXIT—Fast Return from Fast System Call	
SYSRET—Return From Fast System Call	
TEST—Logical Compare	
UCOMISD—Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS	3-28
UCOMISS—Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS.	3-28
UD2—Undefined Instruction	
UNPCKHPD—Unpack and Interleave High Packed Double-Precision Floating-Point Values	
UNPCKHPS—Unpack and Interleave High Packed Single-Precision Floating-Point Values	
UNPCKLPD—Unpack and Interleave Low Packed Double-Precision Floating-Point Values	
UNPCKLPS—Unpack and Interleave Low Packed Single-Precision Floating-Point Values	
VERR, VERW—Verify a Segment for Reading or Writing.	
WAIT/FWAIT—Wait.	
WBINVD—Write Back and Invalidate Cache	
WRMSR—Write to Model Specific Register	
XADD—Exchange and Add	
XCHG—Exchange Register/Memory with Register	3-30
XLAT/XLATB—Table Look-up Translation	
XOR—Logical Exclusive OR	
XORPD—Bitwise Logical XOR for Double-Precision Floating-Point Values	
XORPS—Bitwise Logical XOR for Single-Precision Floating-Point Values	3-30



CHAPTI	=:::::	
	ARE OPTIMIZATION GUIDELINES	
4.1.	INTRODUCTION	1-1
4.2.	64-BIT MODE OPTIMIZATION GUIDELINES	
4.2.1.	Coding Rules Affecting 64-bit Mode	1-1
4.2.1.1.	Use Legacy 32-Bit Instructions When The Data Size Is 32 Bits	
4.2.1.2.	Use the Extra Registers to Reduce Register Pressure	1-1
4.2.1.3.	Use 64-Bit by 64-Bit Multiplies That Produce 128-Bit Results Only When Necessary	
4.2.1.4.	Sign Extension to Full 64-Bits	
4.2.2.	Alternate Coding Rules for 64-Bit Mode	1-3
4.2.2.1.	Use 64-Bit Registers Instead of Two 32-Bit Registers for 64-Bit Arithmetic	1-3
4.2.3.	Other Coding Rules	1-4
4.2.3.1.	Use 32-Bit Versions of CVTSI2SS and CVTSI2SD When Possible	
4.2.3.2.	Using Software Prefetch	1-4
APPENI	DIX A SMRAM STATE SAVE MAP	
APPENI MACHIN B.1. B.2. B.3.	DIX B NE CHECK ARCHITECTURE SUPPORT MACHINE CHECK ARCHITECTURE 64-BIT MODE SPECIFIC EXTENSIONS/MODIFICATIONS INTERPRETING THE MCA ERROR CODES	B-1
APPENI DEBUG C.1. C.1.1. C.2. C.2.1.	DIX C SUPPORT LAST BRANCH RECORD STACK. 64-bit Mode Specific Extensions/Modifications. DEBUG - BRANCH TRACE STORE 64-bit Mode Extensions/Modifications.	C-1 C-1
APPENI PERFOI D.1.	DIX D RMANCE MONITORING SUPPORT 64-BIT MODE SPECIFIC EXTENSIONS/MODIFICATIONS	D-1
APPENI SMRAI E.1.	DIX E M STATE SAVE MAP SMRAM STATE SAVE MAP	E-1



CHAPTER 1 INTRODUCTION

1.1. INTEL® EXTENDED MEMORY 64 TECHNOLOGY

This document describes the enhancements made to Intel[®] IA-32 architecture to support 64-bit address extensions. Enhancements include new operating modes, new instructions, and enhanced instructions. Chapter 1 documents software-visible changes of the Intel[®] Extended Memory 64 Technology (Intel[®] EM64T). Chapters 2 and 3 document the instructions in various operating modes. Chapter 4 describes coding rules and optimizations that apply.

Intel EM64T is an enhancement to Intel IA-32 architecture. An IA-32 processor equipped with this technology is compatible with existing IA-32 software, enables software to access more memory address space, and allows for the co-existence of software written for 32-bit linear address space with software capable of accessing 64-bit linear address space.

1.2. OPERATING MODES

Intel EM64T introduces a new operating mode referred to as IA-32e mode. IA-32e mode includes two sub-modes. These are: (1) compatibility mode which enables a 64-bit operating system to run most legacy 32-bit software unmodified, (2) 64-bit mode which enables a 64-bit operating system to run applications written to access 64-bit address space.

In the 64-bit sub-mode of Intel EM64T, applications may access:

- 64-bit flat linear addressing
- 8 new general-purpose registers (GPRs)
- 8 new registers for streaming SIMD extensions (SSE, SSE2 and SSE3)
- 64-bit-wide GPRs and instruction pointers
- uniform byte-register addressing
- fast interrupt-prioritization mechanism
- a new instruction-pointer relative-addressing mode.

A processor with Intel EM64T can run in legacy IA-32 mode or IA-32e mode. Legacy IA-32 mode allows the processor to run in protected mode, real address mode or virtual 8086 mode. A processor with Intel EM64T initially operates in legacy, paging-enabled, protected mode. It then transitions to IA-32e mode when a bit in the IA32-EFER register is set and PAE mode is enabled (see Section 1.3.3.1.). Table 1-1 shows the supported operating modes and the differences between each mode.

Table 1-1 IA-32e Modes

Mode		Operating System Required	Application Recompile Required	Default Address Size (Bits)	Default Operand Size (bits)	Register Extension	GPR Width (bits)	Supported by SMM
IA-32e	64-Bit Mode	64-Bit OS	Yes	64	32	Yes	64	Yes*
mode	Compatibility		No	32	32	No	32	Yes
	Mode			16	16		16, 8	

^{*} SMM supports transitions to/from 64-bit OS and legacy OSs. However, PAE and 64-bit linear address are not available inside the SMM environment.

1.2.1. IA-32e Mode

IA-32e mode consists of two sub-modes: 64-bit mode and compatibility mode. IA-32e mode can only be entered by loading a 64-bit capable operating system. The procedure used to enter into IA-32e mode is described in Section 1.3.3.1. and Section 1.6.3.2.

1.2.2. 64-Bit Mode

64-bit mode is used by 64-bit applications running under a 64-bit operating system. It supports the following features:

- Architectural support for 64-bits of linear address; however IA-32 processors supporting Intel EM64T may implement less then 64-bits (see Section 1.3.3.3.and Section 1.5.2.)
- Register extensions accessible through a set of new opcode prefixes (REX)
- Existing general purpose registers widened to 64-bits (RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP)
- Eight new general purpose registers (R8–R15)
- Eight new 128-bit streaming SIMD extension registers (XMM8–XMM15)
- A 64-bit instruction pointer (RIP)
- A New RIP-relative data addressing mode
- The use of flat address space with single code, data, and stack space
- Extended and new instructions
- Physical address support greater than 64 GB; the actual physical address size of IA-32 processors supporting Intel EM64T is implementation specific
- A new interrupt priority control mechanism

64-bit mode is enabled by the operating system on a code-segment basis. Its default address size is 64 bits and its default operand size is 32 bits. These defaults can be overridden on an instruction-by-instruction basis using the new REX opcode prefixes. The REX prefix allows a 32-bit operand to be specified when operating in 64-bit mode. By using this mechanism, many existing instructions have been modified or redefined to allow usage of the larger 64-bit registers and 64-bit addresses.

1.2.3. Compatibility Mode

Compatibility mode permits most legacy 16-bit and 32-bit applications to run without recompilation under a 64-bit operating system. Note that legacy applications that run in Virtual 8086 mode or use hardware task management will not work in this mode. Like 64-bit mode, compatibility mode is enabled by the operating system on a code segment basis. This means that 64-bit applications may be running on the processor (in 64-bit mode) at the same time as legacy 32-bit applications (not recompiled for 64-bits).

Compatibility mode is like legacy protected mode. Applications only access the first 4GB of linear-address space, standard IA-32 instruction prefixes, and registers. REX prefixes do not apply in compatibility mode (encoding of the REX



prefixes are treated as a legacy IA-32 instruction). Compatibility mode uses 16-bit and 32-bit address and operand sizes. Like legacy protected mode, this mode allows applications to access up to 64GB of physical memory using PAE (Physical Address Extensions).

The following elements of legacy protected mode are not supported while in compatibility mode:

- Virtual 8086 mode, task switches and stack parameter copy features
- From the operating system's viewpoint: system data structures, address translation, interrupt and exception handling; use 64-bit mechanisms instead of 32-bit mechanisms to handle these structures or events

1.2.4. Legacy Modes

Legacy mode includes protected mode, real address mode, and virtual 8086 mode. Software written for these modes is fully compatible with processors supporting Intel EM64T.

1.2.5. System Management Mode

System management mode (SMM) in a processor supporting Intel EM64T provides the same execution environment for the system management interrupt (SMI) handler as was provided in the legacy IA-32 environment. Upon the delivery of an SMI, the processor switches to SMM and saves the processor state according to the SMRAM state save map.

SMM supports transitions from/to other operating modes (IA-32e and legacy modes). An SMI handler can access any physical memory pages using the PSE mechanism. The SMM environment does not support 64-bit linear address because PAE is not supported.



1.3. REGISTER-SET CHANGES

This section describes the register set changes. Table 1-2 compares the registers and data structures visible to applications in IA-32e mode with those visible to applications running in legacy IA-32 environments. Legacy environments include those present in existing IA-32 processors, legacy modes in a processor supporting Intel EM64T, and IA-32e compatibility mode.

Compatibility mode applications are not aware of 64-bit mode. Applications that need to run successfully in compatibility mode should be designed to run in legacy IA-32 protected mode environment.

Table 1-2 Register Set Changes

Software Visible	64-Bit Mode			Legacy and Compatibility Modes		
Register	Name	Number	Size (bits)	Name	Number	Size (bits)
General Purpose Registers	RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8-15	16	64	EAX, EBX, ECX, EDX, EBP, ESI, EDI, ESP	8	32
Instruction Pointer	RIP	1	64	EIP	1	32
Flags	EFLAGS	1	32	EFLAGS	1	32
FP Registers	ST0-7	8	80	ST0-7	8	80
Multi-Media Registers	MM0-7	8	64	MM0-7	8	64
Streaming SIMD Registers	XMM0-15	16	128	XMM0-7	8	128
Stack Width	-		64	-		16 or 32

1.3.1. General-Purpose Registers (GPRs)

There are eight general purpose registers (GPRs) in the IA-32 architecture when operating in legacy or compatibility modes. AX, BX, CX, DX, DI, SI, BP, SP are available when the operand size is 16 bits. EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP are available when the operand size is 32 bits.

In 64-bit mode, the default operand is 32-bits. However, GPRs are able to work with either 32-bit or 64-bit operands. If a 32-bit operand is specified; EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D are available. If a 64-bit operand is specified; RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15 are available. R8-R15 represent 8 new GPRs. All of these registers can be accessed at the byte, word, dword and qword level. The level of granularity is enabled by using the REX prefixes (see Section 1.4.2.).

In 64-bit mode, there are limitations on the byte registers that instructions can access. An instruction cannot reference legacy high-bytes (for example: AH, BH, CH, DH) and one of the new byte registers at the same time (for example: the low byte of the RAX register). However, instructions may reference legacy low-bytes (for example: AL, BL, CL or DL) and new byte registers at the same time (for example: the low byte of the R8 register, or RBP). The architecture will enforce the above limitation by changing high-byte references (AH, BH, CH, DH) to low byte references (BPL, SPL, DIL, SIL; these are the low 8 bits of RBP,R SP, RDI and RSI) for any instruction with an REX prefix.

When in 64-bit mode, the size of the operands determine the number of valid bits in the destination GPR:

- 64-bit operands generate a 64-bit result in the destination GPR.
- 32-bit operands generate a 32-bit result, zero-extended to a 64-bit result in the destination GPR.
- 8-bit and 16-bit operands generate an 8-bit or 16-bit result. The upper 56-bits or 48-bits (respectively) of the destination GPR are not be modified by the operation. If the result of an 8-bit or 16-bit operation is intended for 64-bit address calculation, explicitly sign-extend it to the full 64-bits.

Because the upper 32-bits of 64-bit GPR's are undefined in 32-bit modes, the upper 32-bits of any GPR are not preserved when switching from 64-bit mode to a 32-bit mode (for example, legacy mode or compatibility mode). Software must not depend on the upper, undefined bits to maintain a value after a 64-bit to 32-bit mode switch. These values may change from one hardware implementation to the next, or from one cycle to the next.

1.3.2. Streaming SIMD Extension (SSE) Registers

In compatibility and legacy modes, SSE registers consist of the eight 128-bit legacy registers (XMM0–XMM7). In 64-bit mode, eight additional 128-bit SSE registers are available (XMM8–XMM15). Access to these registers is controlled on an instruction-by-instruction basis by using the REX instruction prefix.

The XMM registers can be used with SSE, SSE2, and SSE3 in all modes.

1.3.3. System Registers

Intel EM64T introduces new registers and makes changes to existing system registers. These are:

- MSRs. The Extended Feature Enable MSR (IA32_EFER) contains bits for controlling, enabling and disabling features of the Intel EM64T. For information on the KernelGSbase MSR, see Section 1.4.4. For information on the STAR, LSTAR, CSTAR & FMASK MSRs, see Section 1.6.9.3. For information on the FS.base & GS.base MSRs, see Section 1.6.4.2.
- Control Registers. All control registers expand to 64 bits. A new control register, (the task priority register: CR8 or TPR) has been added.
- **Descriptor Table Registers.** The global descriptor table register (GDTR) and interrupt descriptor table register (IDTR) have been expanded to 10 bytes so that they can hold a full 64-bit base address. The local descriptor table register (LDTR) and the task register (TR) have also been expanded to hold a full 64-bit base address. See Table 1-6.
- **Debug Registers**. Debug registers have been expanded to 64 bits.

1.3.3.1. Extended Feature Enable Register (IA32_EFER)

The extended feature enable register (IA32_EFER) houses control bits. It is at address C0000080H. Table 1-3 summarizes IA32_EFER bits. Bit definitions are described in Table 1-4.

Table 1-3	Extended Feat	ure Enable M	SR (IA32_EFER)

63:11	10	9	8	7:1	0
Reserved	IA-32e mode Active (LMA)	Reserved	IA-32e mode Enable (LME)	Reserved	SysCall Enable (SCE)

1.3.3.2. Control Registers

In the Intel EM64T architecture:

- Control registers CR0-CR4 are expanded to 64 bits. MOV CRn instructions read or write 64 bits. Operand-size prefixes are ignored.
- In compatibility and legacy modes, control register writes fill the upper 32 bits with zeros and control register reads return only the lower 32 bits.
- In 64-bit mode, the upper 32 bits of CR0 and CR4 are reserved and must be written with zeros. Writing any non-zero value to any of the upper 32 bits results in a general-protection exception, #GP(0). All 64 bits of CR2 are writable by software. Bits 51:40 of CR3 are reserved, and must be 0. However, the MOV CRn instructions do not check that addresses written to CR2 or CR3 are within the linear-address or physical-address limitations of the implementation.
- There is a new control register, CR8, defined as the task priority register (TPR). Operating systems can use the TPR to control whether or not external interrupts are allowed to interrupt the processor, based on the interrupt's priority level. See Section 1.6.10.6. for details on TPR.



Table 1-4 IA32_EFER Bit Descriptions

Name	Description	Operation
LMA	IA-32e mode active (bit 10).	This bit is a read-only status bit. Any attempt to set LMA is silently ignored. It indicates that IA-32e mode is active.
		The processor sets LMA to 1 when both IA-32e mode and paging have been enabled. When LMA = 1, the processor is either in compatibility mode or 64-bit mode, depending on the values of the code segment descriptor's L and D bits as shown in Table 1-16.
		When LMA = 0, the processor is running in legacy mode. In this mode, the processor behaves like a legacy 32-bit IA-32 processor.
LME	IA-32e mode enable (bit 8).	Setting this bit to 1 enables the processor's ability to switch to IA-32e mode. IA-32e mode is not actually activated until software enables PAE mode paging.
		When PAE paging is enabled while LME is set to 1, the processor sets the IA32_EFER.LMA bit to 1, indicating that IA-32e mode is not only enabled but also active.
SCE	Syscall/Sysret enable (bit 0)	Setting this bit to 1 enables support for SYSCALL/SYSRET. SYSCALL/SYSRET is supported in 64-bit mode only. It is the responsibility of the OS to enable SYSCALL/SYSRET for 64-bit operation.
		All other bits in IA32_EFER are reserved and must be written with zeros (MBZ).

1.3.3.3. Descriptor Table Registers

The four system-descriptor-table registers (GDTR, IDTR, LDTR, and TR) are expanded in hardware to hold 64-bit base addresses. This allows operating systems running in IA-32e mode to locate system-descriptor tables anywhere in the linear-address space supported by the implementation.

Table 1-5 shows the GDTR and IDTR. Table 1-6 shows the LDTR and TR. In all cases, the base address must be in canonical form. The number of linear and physical address bits supported by an implementation can be determined by executing CPUID with EAX set to 80000008H.

See Chapter 2 for more CPUID details.

Table 1-5 GDTR & IDTR

QWord Offset	Bits 63:16	Bits 15:0
1		Limit
0	ВА	SE

Table 1-6 LDTR & TR

QWord Offset	Bits 63:20	Bits 19:16	Bits 15:0
3			Selector
2			Attributes
1		I	Limit
0		BASE	

1.3.3.4. Debug Registers

In 64 bit mode, debug registers DR0–DR7 are 64 bits. MOV DRn instructions read or write all 64 register bits. Operand-size prefixes are ignored.

In all 16-bit modes or 32-bit modes on an IA-32e platform (legacy or compatibility modes), writes to a debug register fill the upper 32 bits with zeros and reads from a debug register return only the lower 32 bits. In 64-bit mode, the upper



32 bits of DR6 and DR7 are reserved and must be written with zeros. Writing 1 to any of the upper 32 bits results in a #GP(0) exception.

All 64 bits of DR0–DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0–DR3 are in the linear-address limits of the implementation. Address matching is supported only on valid addresses generated by the processor implementation.

1.4. INSTRUCTION-SET CHANGES

1.4.1. Address-Size and Operand-Size Prefixes

In 64-bit mode, the default address size is 64 bits and the default operand size is 32 bits. Defaults can be overridden using a new set of instruction prefixes (REX). Address-size and operand-size prefixes allow mixing of 32-bit and 64-bit data and addresses on an instruction-by-instruction basis. Table 1-7 shows the instruction prefix requirements for address-size overrides in IA-32e operating modes.

Note that 16-bit addresses are not supported in 64-bit mode. In compatibility and legacy mode, address sizes function as they do in the IA-32 legacy architecture.

Table 1-7 IA-32e Mode Address-Size Overrides Requirements

IA-32e mode sub-mode	Default Address Size (bits)	Effective Address Size (bits)	Address-Size Prefix Required
64-bit	64	64	No
		32	Yes
Compatibility	32	32	No
		16	Yes
	16	32	Yes
		16	No

Table 1-8 shows valid combinations of the 66H instruction prefix and the REX.W prefix that may be used to specify operand-size overrides in IA-32e operating modes.

In 64-bit mode, the default operand size is 32 bits. REX prefixes include 4 bit fields that form 16 different values. The W bit field in the REX prefixes is referred to as REX.W. REX.W = 1 prefix specifies a 64-bit operand size. Note that software can still use the operand-size 66H prefix to toggle to a 16-bit operand size. However, the REX.W = 1 prefix takes precedence over the operand-size prefix (66H) when both are used.

In the case of SSE/SSE2/SSE3 SIMD instructions; the 66H, F2H, and F3H prefixes are used as opcode extensions and are considered to be part of the opcode. In such cases, there is no interaction between a valid REX.W prefix and the 66H opcode extension prefix.



Table 1-8 64-Bit Extensions Operand-Size Overrides

IA-32e sub-mode	Default	Effective Operand Size (bits)	Instruction Prefix		
	Operand Size (bits)		66H	REX.W = 1	
64-bit	32	64	Х	Yes	
		32	No	No	
		16	Yes	No	
Compatibility	32	32	No	Not Applicable	
		16	Yes		
	16	32	Yes		
		16	No		

1.4.2. REX Prefixes

The REX prefixes are a new family of instruction-prefix bytes used in 64-bit mode. They do the following:

- Specify the new GPRs and SSE registers
- Specify a 64-bit operand size
- Specify extended control registers (used by system software)

Not all instructions require a REX prefix. The prefix is necessary only if an instruction references one of the extended registers or uses a 64-bit operand. If a REX prefix is used when it has no meaning, it is ignored.

Only one REX prefix is allowed in an instruction. The prefix, if used, must immediately precede the opcode byte or the two-byte opcode escape prefix (if present). Other placements of an REX prefix are ignored.

The instruction-size limit of 15 bytes still applies to instructions that contain a REX prefix. Figure 1-1 shows how a REX prefix fits within the byte-order of instructions.

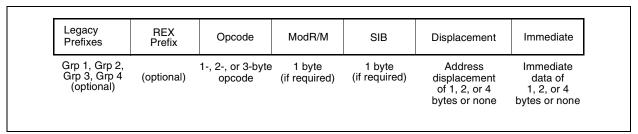


Figure 1-1. Prefix Ordering in 64-bit Mode

The legacy prefixes noted in Figure 1-1 include 66H, 67H, F2H and F3H. Group 1, Group 2, Group 3 and Group 4 prefix refers to Section 2.2 of IA-32 Intel[®] Architecture Software Developer's Manual, Volume 2A.

1.4.2.1. **Encoding**

IA-32 instruction formats specify up to three registers by using 3-bit fields in the instruction encoding, depending on the format:

- ModRM: the reg and r/m fields of the ModRM byte
- ModRM with SIB: the reg field of the ModRM byte and the base and index fields of the SIB (scale, index, base) byte
- Instructions without the ModRM: the reg field of the opcode



In 64-bit mode, these fields and formats do not change. All bits needed to extend fields for 64-bit are provided by the addition of the REX prefixes.

1.4.2.2. REX Prefix Fields

REX prefixes are a set of 16 opcodes that span one row of the opcode map and occupy entries 40H to 4FH. These opcodes represent valid instructions (INC or DEC) in legacy IA-32 operating modes and in compatibility mode. In 64-bit mode, these same opcodes represent the instruction prefix REX and are not treated as individual instructions.

The functionality of the single-byte-opcode INC/DEC instruction is no longer available in 64-bit mode. INC/DEC functionality is still available with ModRM forms of the same instructions (opcodes FF/0 and FF/1). Table 1-9, and Figure 1-2 though Figure 1-5, show the prefix fields and their uses. Some combinations of REX prefix fields result in an operation that is not valid. In those cases, the REX prefix is ignored.

Figure 1-2 through Figure 1-5 show four examples of how the R, X, and B bits of REX prefixes are combined with fields from the ModRM byte, SIB byte, and opcode to specify register and memory addressing. The R, X, and B bits are described in Table 1-9.

Field Name	Bit Position	Definition
-	7:4	0100
W	3	0 = Default Operand Size
		1 = 64 Bit Operand Size
R	2	Extension of the MODRM reg field
X	1	Extension of the SIB index field
В	0	Extension of the MODRM r/m field, SIB base field, or Opcode reg field

Table 1-9 REX Prefix Fields

Some additional information:

- Setting the REX.W bit is used to determine the operand size, but doesn't solely determine the operand width. Like the existing 66H operand size prefix, the REX 64-bit operand size override has no effect on byte-specific operations.
- For non-byte operations, an REX operand-size override takes precedence over the 66H prefix. If a 66H prefix is used with a REX prefix (REX.W = 1), the 66H prefix is ignored. If a 66H override is used with REX and REX.W = 0, the operand size is 16 bits.
- REX.R modifies the ModRM reg field when that field encodes a GPR, SSE, control or debug register. REX.R is ignored when ModRM specifies other registers or includes an extended opcode.
- REX.X bit modifies the SIB index field.
- REX.B either modifies the base in the ModRM r/m field or SIB base field; or it modifies the opcode reg field used for accessing GPRs.

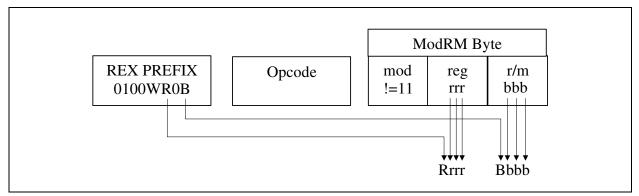


Figure 1-2. Memory Addressing Without a SIB Byte, REX.X is Not Used, ModRM Reg Field!

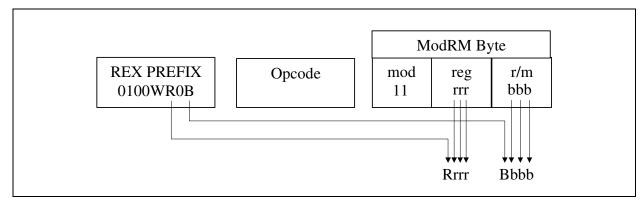


Figure 1-3. Register-Register Addressing (No Memory Operand), REX.X is Not Used

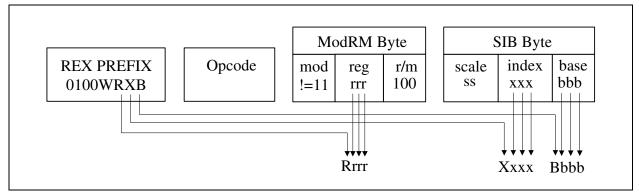


Figure 1-4. Memory Addressing With a SIB Byte

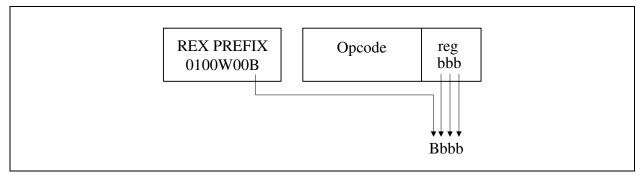


Figure 1-5. Register Operand Coded in Opcode Byte, REX.X and REX.R Are Not Used

In the legacy IA-32 architecture, byte registers (AH, AL, BH, BL, CH, CL, DH, and DL) are encoded in the ModRM byte's reg field, the r/m field or the opcode reg field as registers 0 through 7. The REX prefixes provide an additional addressing capability to byte-registers that makes the least-significant byte of GPRs available for byte operations.

Certain combinations of the fields of the ModRM byte and the SIB byte have special meaning for register encodings. For some combinations, instruction fields expanded by the REX prefix are not decoded. Table 1-10 describes how each case behaves.

Table 1-10 Special Cases of REX Encodings

ModRM or SIB	Sub-field Encodings	Compatibility Mode Operation	Compatibility Mode Implications	Additional Implications
ModRM Byte	mod != 11	SIB byte present.	SIB byte required for ESP-based addressing	REX prefix adds a fourth bit (b) which is not decoded (don't care)
	r/m == b*100(ESP)		LSF-based addressing	SIB byte also required for R12- based addressing
ModRM Byte	mod == 0	Base register not used	EBP without a displacement must be	REX prefix adds a fourth bit (b) which is not decoded (don't care)
	r/m == b*101(EBP)	useu	done using mod = 01 with displacement of 0	Using RBP or R13 without displacement must be done using mod = 01 with a displacement of 0
SIB Byte	index == 0100(ESP)	Index register not used	ESP cannot be used as an index register	REX prefix adds a fourth bit (b) which is decoded.
				There are no additional implications. The expanded index field allows distinguishing RSP from R12, therefore R12 can be used as an index.
SIB Byte	base == 0101(EBP)	Base register is unused if	Base register depends on mod encoding	REX prefix adds a fourth bit (b) which is decoded.
	mod = 0		There are no additional implications. The expanded base field is used to distinguish RBP from R13, allowing R13 to be used as a SIB base regardless of the mod.	
* Don't care abo	ut the value of b (REX.B).	<u>'</u>	'	1

1.4.2.3. Displacement

Addressing in 64-bit mode uses existing 32-bit ModRM and SIB encodings. In particular, the ModRM and SIB displacement sizes do not change. They remain 8 bits or 32 bits and are sign-extended to 64 bits.



1.4.2.4. Direct Memory-Offset MOVs

In 64-bit mode, direct memory-offset forms of the MOV instruction (Table 1-11) are extended to specify a 64-bit immediate absolute address. This address is called a moffset. No prefix is needed to specify this 64-bit memory offset. For these MOV instructions, the size of the memory offset follows the address-size default (64 bits in 64-bit mode).

Table 1-11 Direct Memory Offset Form of MOV

Opcode	Instruction
A0	MOV AL, moffset
A1	MOV EAX, moffset
A2	MOV moffset, AL
A3	MOV moffset, EAX

1.4.2.5. Immediates

In 64-bit mode, the typical size of immediate operands remains 32 bits. When the operand size is 64 bits, the processor sign-extends all immediates to 64 bits prior to their use.

Support for 64-bit immediate operands is accomplished by expanding the semantics of the existing move (MOV reg, imm16/32) instructions. These instructions (opcodes B8H – BFH) move 16-bits or 32-bits of immediate data (depending on the effective operand size) into a GPR. When the effective operand size is 64 bits, these instructions can be used to load an immediate into a GPR. A REX prefix is needed to override the 32-bit default operand size to a 64-bit operand size.

For example:

48 B8 8877665544332211 MOV RAX,1122334455667788H

1.4.2.6. RIP-Relative Addressing

A new addressing form, RIP-relative (relative instruction-pointer) addressing, is implemented in 64-bit mode. An effective address is formed by adding displacement to the 64-bit RIP of the next instruction.

In legacy IA-32 architecture, addressing relative to the instruction pointer is available only with control-transfer instructions. In 64-bit mode, instructions that use ModRM addressing can use RIP-relative addressing. Without RIP-relative addressing, all ModRM instruction modes address memory relative to zero.

RIP-relative addressing allows specific ModRM modes to address memory relative to the 64-bit RIP using a signed 32-bit displacement. This provides an offset range of ±2GB from the RIP. Table 1-12 shows the ModRM and SIB encodings for RIP-relative addressing. Redundant forms of 32-bit displacement-addressing exist in the current ModRM and SIB encodings. There is one ModRM encoding and there are several SIB encodings. RIP-relative addressing is encoded using a redundant form.

In 64-bit mode, the ModRM Disp32 (32-bit displacement) encoding is re-defined to be RIP+Disp32 rather than displacement-only. See Table 1-12.

Table 1-12 RIP-Relative Addressing

	Table 1 12 1111 Holative Addressing						
ModRM and SIB Sub-field Encodings		Compatibility Mode Operation	64-bit Mode Operation	Additional Implications in 64-bit mode			
ModRM Byte	mod == 00	Disp32	RIP + Disp32	Must use SIB form with normal			
	r/m == 101(none)			(zero-based) displacement addressing			
SIB Byte	base == 101 (none)	if mod = 00, Disp32	Same as legacy	None			
	index == 100(none)						
	scale = 0, 1, 2, 4						



The ModRM encoding for RIP-relative addressing does not depend on using a REX prefix. Specifically, the r/m bit field encoding of 101B, used to select RIP-relative addressing, is not affected by the REX prefix. For example, selecting R13 (REX.B = 1, r/m = 101B) with mod = 00B still results in RIP-relative addressing. The 4-bit r/m field of REX.B combined with ModRM is not fully decoded. In order to address R13 with no displacement, software must encode it as R13 + 0 using a 1-byte displacement of zero.

RIP-relative addressing is enabled by 64-bit mode, not by a 64-bit address-size. The use of the address-size prefix does not disable RIP-relative addressing. The effect of the address-size prefix is to truncate and zero-extend the computed effective address to 32 bits.

1.4.2.7. Default 64-Bit Operand Size

In 64-bit mode, two groups of instructions have a default operand size of 64 bits (do not need a REX prefix for this operand size). These are:

- Near branches
- All instructions, except far branches, that implicitly reference the RSP

1.4.3. New Encodings for Control and Debug Registers

In 64-bit mode, new encodings for control and debug registers are available. The REX.R bit is used to modify the ModRM reg field when that field encodes a control or debug register (see Table 1-9). These encodings enable the processor to address CR8-CR15 and DR8- DR15.

One additional control register (CR8) is defined in 64-bit mode. CR8 becomes the Task Priority Register (TPR). In the first implementation of the IA-32e technology, CR9-CR15 and DR8-DR15 are not implemented. Any attempt to access unimplemented registers results in an invalid-opcode exception (#UD).

1.4.4. New Instructions

The following instructions are being introduced in 64-bit mode with 64-bit extensions. They are discussed in detail in Chapter 2.

- SWAPGS
- SYSCALL and SYSRET
- CDQE
- CMPSO
- CMPXCHG16B
- LODSQ
- MOVSQ
- MOVZX (64-bits)
- STOSQ

1.4.5. Stack Pointer

In 64-bit mode, the stack pointer size is 64 bits. The stack size is not controlled by a bit in the SS descriptor (as it is in compatibility or legacy mode) nor can it be overridden by an instruction prefix.

In implicit stack references, address-size overrides are ignored. Except for far branches, all instructions that implicitly reference the RSP default to 64-bit operand size in 64-bit mode. Instructions affected include: PUSH, POP, PUSHF, POPF, ENTER, and LEAVE. Pushes and pops of 32-bit values on the stack are not possible in 64-bit mode with these instructions. 16-bit pushes and pops are supported by using the 66H operand-size prefix.



The 64-bit default operation-size eliminates the need for a REX prefix to precede these instructions when registers RAX – RSP are used as operands. REX is still required if the R8–R15 registers are used. This is because the prefix is required to address the new extended registers.

1.4.6. Branches

The 64-bit extensions expand two branching mechanisms to accommodate branches in 64-bit linear-address space. These are:

- Near-branch semantics are redefined in 64-bit mode
- In both 64-bit mode and compatibility mode, 64-bit call-gate descriptors for far calls

In 64-bit mode, the operand size for all near branches (CALL, RET, JCC, JCXZ, JMP, and LOOP) is forced to 64 bits. These instructions update the 64-bit RIP without the need for a REX operand-size prefix. The following aspects of near branches are controlled by the effective operand size:

- Truncation of the size of the instruction pointer
- Size of a stack pop or push, due to a CALL or RET
- Size of a stack-pointer increment or decrement, due to a CALL or RET
- Indirect-branch operand size

In 64-bit mode, all of the above actions are forced to 64 bits regardless of operand size prefixes (operand size prefixes are silently ignored). However, the displacement field for relative branches is still limited to 32 bits and the address size for near branches is not forced in 64-bit mode.

Address sizes affect the size of RCX used for JCXZ and LOOP; they also impact the address calculation for memory indirect branches. Such addresses are 64 bits by default; but they can be overridden to 32 bits by an address size prefix.

Software typically uses far branches to change privilege levels. Legacy IA-32 architecture provides the call-gate mechanism to allow software to branch from one privilege level to another, although call gates can also be used for branches that do not change privilege levels. When call gates are used, the selector portion of the direct or indirect pointer references a gate descriptor (the offset in this instruction is ignored). The offset to the destination's code segment is taken from the call-gate descriptor. IA-32e mode redefines the type value of a 32-bit call-gate descriptor type to a 64-bit call gate descriptor and expands the size of the 64-bit descriptor to hold a 64-bit offset. The 64-bit mode call-gate descriptor allows far branches that reference any location in the supported linear-address space. These call gates also hold the target code selector (CS), allowing changes to privilege level and default size as a result of the gate transition.

Because immediates are generally specified up to 32 bits, the only way to specify a full 64-bit absolute RIP in 64-bit mode is with an indirect branch. For this reason, direct far branches are eliminated from the instruction set in 64-bit mode.

IA-32e mode expands the semantics of the SYSENTER and SYSEXIT instructions so that they operate within a 64-bit memory space. In Chapter 2, see "SYSENTER—Fast System Call" and "SYSEXIT—Fast Return from Fast System Call" for details on enhancements. IA-32e mode also introduces two new instructions: SYSCALL and SYSRET which are valid only in 64-bit mode. In Chapter 2, see "SYSCALL—Fast System Call" and "SYSRET—Return From Fast System Call".

1.5. MEMORY ORGANIZATION

1.5.1. Address Calculations in 64-Bit Mode

In 64-bit mode (if there is no address-size override), the size of effective address calculations is 64 bits. An effective-address calculation uses a 64-bit base and index registers and sign-extend displacements to 64 bits.

Due to the flat address space in 64-bit mode, linear addresses are equal to effective addresses. In the event that FS or GS segments are used with a non-zero base, this rule does not hold. In 64-bit mode, the effective address components



are added and the effective address is truncated before adding the full 64-bit segment base. The base is never truncated, regardless of addressing mode in 64-bit mode.

In IA-32e mode, the instruction pointer is extended to 64 bits to support 64-bit code offsets. The 64-bit instruction pointer is called the RIP. Table 1-13 shows the relationship between RIP, EIP, and IP.

Table 1-13 Instruction Pointer

QWord offset	Bits 63:32	Bits 15:0			
2	Not Modified	IP			
1	Zero Extension EIP				
0	RIP				

Generally, displacements and immediates in 64-bit mode are not extended to 64 bits. They are still limited to 32 bits and sign-extended during effective-address calculations. In 64-bit mode, however, support is provided for 64-bit displacement and immediate forms of the MOV instruction.

All 16-bit and 32-bit address calculations are zero-extended in IA-32e mode to form 64-bit addresses. Address calculations are first truncated to the effective address size of the current mode (64-bit mode or compatibility mode), as over-ridden by any address-size prefix. The result is then zero-extended to the full 64-bit address width. Because of this, 16-bit and 32-bit applications running in compatibility mode can access only the low 4GBytes of the 64-bit mode effective addresses. Likewise, a 32-bit address generated in 64-bit mode can access only the low 4GBytes of the 64-bit mode effective addresses.

1.5.2. Canonical Addressing

An address considered to be in canonical form has address bit 63 through to the most-significant implemented bit by the micro architecture set to either all ones or all zeros.

IA-32e mode defines a 64-bit linear address. Implementations, however, can support less. The first implementation of IA-32 processors with Intel EM64T will support 48 bits of linear address. This means a canonical address must have bits 63 thru 48 set to zeros or one's (depending whether bit 47 is a zero or one).

Although implementations may not use all 64 bits of the linear address, they should check bits 63 through the most-significant implemented bit to see if the address is in canonical form. If a linear-memory reference is not in canonical form, the implementation should generate an exception. In most cases, a general-protection exception (#GP) is generated. However, in the case of explicit or implied stack references, a stack fault (#SS) is generated.

Instructions that have implied stack references, by default, use the SS segment register. These include PUSH/POP-related instructions and instructions using RSP/RBP as base registers. In these cases, the canonical fault is #SF. If an instruction uses RSP/RBP as base registers and has a segment override specifying a non-SS segment, a canonical fault will generate #GP. The check for canonical address form is performed after privilege checks but before paging and alignment checks.

1.6. OPERATING SYSTEM CONSIDERATIONS

1.6.1. CPUID Instruction

The CPUID instruction reports the presence of processor features and capabilities. Operating systems must rely on CPUID to report the availability of IA-32 mode, the IA32_EFER MSR, and instructions that are available in 64-bit mode. See the reference page of CPUID in Section 2.2. for feature information that pertains to IA-32e mode.



1.6.2. Register Settings and IA-32e Mode

The operation of 64-bit mode and compatibility mode are governed by various control-bits in the IA32_EFER MSR and CS descriptor. Table 1-14 shows the control-bit settings for IA-32e mode and legacy IA-32 mode. The default address and data sizes are shown for legacy and IA-32e modes. For more information on SMM and register extensions, see Table 1-1.

Table 1-14 Processor Modes

Mode		Encoding			Default	Default
		IA32_EFER.LMA	CS.L	CS.D	Address Size	Operand Size
Legacy Mode		0	Not	1	32	32
			applicable	0	16	16
IA-32e mode	64-Bit Mode	1	1	0	64	32
	Compatibility Mode		0	1	32	32
				0	16	16

X: Not applicable.

1.6.3. Processor Modes

1.6.3.1. IA-32e Mode

IA-32e mode uses two code segment-descriptor bits (CS.L and CS.D, see Table 1-14) to control the sub operating mode. If the processor is running in 64-bit mode, CS.L = 1 and CS.D = 0. With this encoding, the default operand size is 32 bits and default address size is 64 bits. Using instruction prefixes, the operand size can be changed to 64 bits or 16 bits; address size can be changed to 32 bits.

When IA-32e mode is active and CS.L = 0, the processor is in compatibility mode. In this mode, CS.D controls default operand and address sizes exactly as it does in the legacy IA-32 architecture. Setting CS.D = 1 specifies default operand and address size as 32 bits. Clearing CS.D to 0 specifies default operand and address size as 16 bits.

The CS.L = 1 and CS.D = 1 combination is reserved for future use.

1.6.3.2. Activating IA-32e Mode

Operating systems should follow this sequence to activate IA-32e mode:

- 1. Starting from page-enabled protected mode, disable paging by setting CR0.PG = 0. Use the MOV CR0 instruction to disable paging (it must be located in an identity-mapped page).
- 2. Enable physical-address extensions by setting CR4.PAE = 1. Failure to enable PAE will result in a #GP fault when an attempt is made to enable IA-32e mode.
- 3. Load CR3 with the physical base address of the Level 4 page map table (PML4).
- 4. Enable IA-32e mode by setting IA32_EFER.LME = 1.
- 5. Enable paging by setting CR0.PG = 1. This causes the processor to set the LMA bit to 1. The MOV CR0 instruction that enables paging and the following instructions must be located in an identity-mapped page, until such time that a branch to non-identity mapped pages can be effected.

To return from IA-32e mode to legacy paged-protected mode, deactivate and disable IA-32e mode. Use the following sequence:

- 1. Must be in compatibility sub mode.
- 2. Deactivate IA-32e mode by clearing CR0.PG = 0. This causes the processor to set IA32_EFER.LMA = 0. The MOV CR0 instruction used to disable paging and subsequent instructions must be located in an identity-mapped page.

- 3. Load CR3 with the physical base address of the legacy page-table-directory base address.
- 4. Disable IA-32e mode by setting IA32_EFER.LME = 0.
- 5. Enable legacy paged-protected mode by setting CR0.PG = 1
- 6. A branch instruction must follow the MOV CR0 that enables paging. Both the MOV CR0 and the branch instruction must be located in an identity-mapped page.

Immediately after activating IA-32e mode, the system-descriptor-table registers (GDTR, LDTR, IDTR, TR) continue to reference legacy descriptor tables. The tables referenced by the descriptors all reside in the lower 4GBytes of linear-address space. After activating IA-32e mode, 64-bit operating-system should use the LGDT, LLDT, LIDT, and LTR instructions to load the system-descriptor-table registers with references to 64-bit descriptor tables.

Software must not allow exceptions or interrupts to occur between the time IA-32e mode is activated and the subsequent update of the interrupt-descriptor-table register (IDTR) that establishes references to the 64-bit interrupt-descriptor table (IDT). This is because the IDT remains in its legacy form immediately after IA-32e mode is activated. If an interrupt or exception occurs prior to updating the IDTR, a legacy 32-bit interrupt gate will be referenced and interpreted as a 64-bit interrupt gate with unpredictable results. External interrupts can be disabled by the CLI instruction. Non-maskable interrupts (NMI) must be disabled using external hardware.

64-bit mode paging tables must be located in the first 4GBytes of physical-address space prior to activating IA-32e mode. This is necessary because the MOV CR3 instruction used to initialize the page-directory base must be executed in legacy mode prior to activating IA-32e mode (setting CR0.PG = 1 to enable paging). Because the MOV CR3 is executed in legacy mode, only the lower 32 bits of the register are written, limiting the table location to the low 4GBytes of memory. Software can relocate the page tables anywhere in physical memory after IA-32e mode is activated.

The processor performs 64-bit mode consistency checks whenever software attempts to modify any of the enable bits directly involved in activating IA-32e mode (IA32_EFER.LME, CR0.PG, and CR4.PAE). The processor generates a general protection fault (#GP) when consistency checks fail. 64-bit mode consistency checks ensure that the processor does not enter an undefined mode or state with unpredictable behavior.

64-bit mode consistency checks fail in the following circumstances:

- An attempt is made to enable or disable IA-32e mode while paging is enabled.
- IA-32e mode is enabled and an attempt is made to enable paging prior to enabling physical-address extensions (PAE).
- IA-32e mode is active and an attempt is made to disable physical-address extensions (PAE).
- If the current CS has the L-bit set on an attempt to activate IA-32e mode.
- The TR must be contain a 16-bit TSS.

Table 1-15 summarizes the 64-bit mode consistency checks.

Table 1-15 IA-32e Mode Consistency Checks

Register	Bit	Check
EFER	LME $0 \rightarrow 1$	if (CR0.PG ==1) then #GP(0)
	LME 1 \rightarrow 0	if (CR0.PG == 1) then #GP(0)
CR0	PG 0 → 1	if ((IA32_EFER.LME == 1) & (CR4.PAE-0)) then #GP(0)
CR4	PAE 1 \rightarrow 0	if (IA32_EFER.LMA == 1) then #GP(0)

1.6.3.3. Virtual-8086 Mode

Virtual-8086 mode is not supported when the processor is operating in IA-32e mode. When IA-32e mode is enabled, any attempt to set the EFLAGS.VM bit is silently ignored.



1.6.3.4. Compatibility Mode

Compatibility mode, within IA-32e mode, maintains binary compatibility with legacy IA-32 16-bit and 32-bit applications. Legacy 16-bit or 32-bit applications that run in Virtual 8086 mode or use hardware task management are not supported in compatibility mode.

Compatibility mode execution is selected on a code-segment basis. It allows legacy applications to coexist under a 64-bit operating system along with 64-bit applications running in 64-bit mode. An operating system running in IA-32e mode can execute existing 16-bit and 32-bit applications by clearing their code-segment descriptor's CS.L bit to 0.

When CS.L = 0, legacy IA-32 meanings of the CS.D bit and the address-size and operand-size prefixes are observed. Segmentation is enabled. From the application's viewpoint, the processor is in a legacy 16-bit or 32-bit (depending on CS.D) operating environment even though IA-32e mode is activated.

In compatibility mode, the following system-level mechanisms continue to operate using the IA-32e-mode architectural semantics:

- Linear-to-physical address translation use the 64-bit mode extended page-translation mechanism.
- Interrupts and exceptions are handled using the 64-bit mode mechanisms.
- System calls (calls through call gates and SYSENTER/SYSEXIT) are handled using the IA-32e mode mechanisms.

1.6.4. Segmentation

In IA-32e mode, the effects of segmentation depend on whether the processor is running in compatibility mode or 64-bit mode. In compatibility mode, segmentation functions just as it does in legacy IA-32 mode, using legacy 16-bit or 32-bit protected mode semantics.

In 64-bit mode, segmentation is generally (but not completely) disabled, creating a flat 64-bit linear-address space. Specifically, the processor treats the segment base of CS, DS, ES, SS as zero in 64-bit mode, creating a linear address that is equal to the effective address. The exceptions are the FS and GS segments, whose segment registers (which hold the segment base) can be used as an additional base register in linear address calculations. This facilitates addressing local data and certain operating system data structures.

Note that even though segmentation is generally disabled, segment register loads may cause the processor to perform segment access assists. Also, segment register loads still perform all legacy checks on the values, even if the values may not be applicable while in 64-bit mode. Checks are needed because a segment register may be loaded in 64-bit mode for use by an application in compatibility mode.

1.6.4.1. Code Segments

In 64-bit mode, some code-segment (CS) descriptor content (e.g the base address and limit fields) are ignored; the remaining fields functions normally (with the possible exception of the readable bit in the type field). Code segments continue to exist in 64-bit mode. Code segments and their associated descriptors and selectors are needed to establish the processor's operating mode as well as execution privilege-level. The L (long), D (default operation size), and DPL (descriptor privilege level) specify the segment's operating mode and its privilege level.

For address calculations in 64-bit mode, the segment base is treated as if it is zero. IA-32e mode uses a previously unused bit in the CS descriptor. Bit 53 is defined as the long (L) bit and is used to select between 64-bit and compatibility modes when IA-32e mode is activated (IA32_EFER.LMA = 1).

Table 1-16 shows a legacy CS descriptor with the L bit added.

Table 1-16 Code Segment Descriptor

						Bit Position	on				
DW Offset	31:24						15 14:13		12	11:8	7:0
1	Base Address 31:24	G	G D L AVL			Segment Limit 19:16	Р	DPL	1	Туре	Base Address 23:16
0	Base Address 15:0			Segm	nent Limit 1	5:0					

The CS descriptor's D bit selects the default operand and address sizes. When the CS.L bit is 1, the only valid setting of CS.D is 0. This setting corresponds to a default operand size of 32 bits and a default address size of 64 bits. Note that the CS.L = 1 and CS.D = 1 bit combination is reserved for future use and a #GP fault will be generated on an attempt to use a code segment with these bits set in IA-32e mode.

If CS.L = 0 while IA-32e mode is activated, the processor is running in compatibility mode. In this case, CS.D selects the default size for both data and addresses as it does in legacy mode. If CS.D = 0, the default data and address sizes are 16 bits. CS.D = 1 selects a default data and address size of 32 bits.

In IA-32e mode, the CS descriptor's DPL is used for execution privilege checks just as in legacy mode.

1.6.4.2. Segment LOAD Instructions

The following list of behaviors can be expected for Segment Load instructions and registers while in IA-32e mode.

- ES, DS, and SS segment registers are not used. Their fields (base, limit, and attribute) in the corresponding segment descriptor registers are ignored.
- Some forms of segment load instructions are also invalid (for example, LDS, POP ES).
- In 64-bit mode, address calculations that reference the ES, DS or SS segments are treated as if the segment base is zero. SS DPL is modified such that it is always equal to CPL. This will be true even if it is the only field in the SS descriptor that is modified.
- The processor checks that all linear-address references are in canonical form instead of performing limit checks. Mode switching does not change the contents of the segment registers or the associated descriptor registers. These registers are also not changed during 64-bit mode execution, unless explicit segment loads are performed.
- In order to set up compatibility mode for an application, segment-load instructions (MOV to Sreg, POP Sreg) work normally in 64-bit mode. An entry is read from the system descriptor table (GDT or LDT) and is loaded in the hidden portion of the segment descriptor register. The descriptor-register base, limit, and attribute fields are all loaded. However, the contents of data and stack segment selector and descriptor registers are ignored.
- The FS and GS segment registers are treated differently in 64-bit mode. When FS and GS segment overrides are used, their respective base addresses are used in the linear address calculation: (FS or GS).base + index + displacement. FS.base and GS.base are then expanded to the full linear-address size supported by the implementation. The resulting effective address calculation is allowed to wrap across positive and negative addresses, and the resulting linear address must be canonical.
- In 64-bit mode, memory accesses using FS-segment and GS-segment overrides are not checked for a runtime limit nor subjected to attribute-checking. Normal segment loads (MOV to Sreg and POP Sreg) into FS and GS load a standard 32-bit base value in the hidden portion of the segment descriptor register. The base address bits above the standard 32 bits are cleared to 0 to allow consistency for implementations that use less than 64 bits.
- The hidden descriptor register fields for FS.base and GS.base are physically mapped to MSRs to load all address bits supported by a 64-bit implementation. Software with CPL = 0 (Privileged software) can load all supported linear-address bits into FS.base or GS.base using a single WRMSR instruction.
- The FS.base MSR index is C0000100H while the GS.base index is C0000101H. Addresses written into the 64-bit FS.base and GS.base registers must be in canonical form. A WRMSR instruction that attempts to write a non-canonical address to those registers will generate a general-protection exception, #GP.



- When in compatibility mode, FS and GS overrides operate as defined by the legacy IA-32 architecture regardless of the value loaded into the upper 32 linear-address bits of the hidden descriptor register base field. Compatibility mode ignores the upper 32 bits when calculating an effective address.
- A new 64-bit mode instruction, SWAPGS can be used to load GS base. SWAPGS exchanges the kernel data structure pointer from the KernelGSbase MSR with the GS base register. The kernel can then use the GS prefix on normal memory references to access the kernel data structures. (An attempt to write a non-canonical value via WRMSR to the KernelGSBase MSR will cause a #GP fault.)

1.6.4.3. **System Descriptors**

In certain modes, system descriptors are expanded by 64 bits to handle 64-bit base addresses. Where this size-expansion occurs depends on the purpose served by the descriptor. Note the following:

- Descriptors and pseudo-descriptors that are loaded into the GDTR, IDTR, LDTR, and TR registers are used to define system tables. These descriptors are expanded in 64-bit mode but not in compatibility mode.
- In IA-32e mode, descriptors that populate system tables and are referenced by application programs are expanded. These descriptors include call gates, interrupt gates, and trap gates (task gates are not supported in IA-32e mode).
- The GDTR, LDTR, IDTR, and TR system descriptor registers (used by the processor to locate the GDT, LDT, and IDT system-descriptor tables and TSS of the current process) are changed in IA-32e mode to support expanded memory addressing.
- The base addresses of the LDT and TSS are specified by their associated descriptors. These descriptor registers are expanded to hold 64-bit addresses (see Section 1.3.3.3.). GDT and IDT, on the other hand, do not use descriptors. Instead, their bases are loaded by the LGDT and LIDT instructions. For 64-bit mode (but not compatibility or legacy modes), the operands size for these instructions increases to specify a 64-bit base.
- The processor checks descriptor-table limits in IA-32e mode. The limit-field size in all four descriptor-table registers are unchanged from their legacy IA-32 sizes. The GDTR, and IDTR limits remain 16 bits and the LDTR, TR limits are the normal 20/32 bits. The size of the segment-attribute fields in the LDTR and TR registers are also unchanged in IA-32e mode.
- The existing LDT type field (02H) and the existing 32-bit TSS type field (09H) are redefined in 64-bit mode for use as the 64-bit LDT and TSS types. LDT and TSS system descriptors are expanded by 64 bits to be 16 bytes (Table 1-17), allowing them to hold 64-bit base addresses.

Bit Position 31:24 23 22:20 19:16 15 14:13 12:8 7:0 Bytes 15:12 00000 Reserved Reserved Base 63:32 Bytes 11:8 Limit 19:16 DPL Bytes 7:4 Base Address G Reserved Туре Base 23:16 31:24 Bytes 3:0 Base Address 15:0 Limit 15:0

Table 1-17 LDT & TSS Descriptors in 64-bit Mode

Bytes 11:8 hold the upper 32 bits of the base address in canonical form. A second type field, used for consistency checking, is defined in bits 12:8 of the highest dword (Bytes 15:12). This entire field must be cleared to 0, indicating an illegal type. This illegal type (00H) serves to generate a general-protection exception (#GP) if an attempt is made to access the upper half of a 64-bit-mode descriptor as a legacy IA-32 descriptor.

For the existing type field (bits 12:8 of the lowest dword), some of the existing type codes are redefined in 64-bit mode. For example, a 32-bit LDT type (02H) and 32-bit TSS type (09H) are redefined in 64-bit mode for use as a 64-bit LDT and 64-bit TSS types. In compatibility mode, a 02H type continues to refer to a 32-bit LDT, and a 09H type continues to refer to be a 32-bit TSS. No other type-field codes are defined or redefined. The 64-bit base address specified in the descriptor must be in canonical form. If it is not, a general-protection exception, #GP (selector), is generated.



LGDT and LIDT instructions load a pseudo-descriptor into the GDTR or the IDTR register. The first two bytes loaded in all modes (legacy, compatibility, and 64-bit) are a 16-bit limit. The next bytes loaded depend on the mode, as follows:

- In any 16-bit or 32-bit mode (legacy or compatibility mode), the next 4 bytes loaded are the base, for a total of 6 bytes.
- In 64-bit mode, the next 8 bytes loaded are the base, for a total of 10 bytes.

Operand-size prefixes are ignored by the LGDT and LIDT instructions. In 64-bit mode, the 64-bit base address loaded into the GDTR and IDTR registers must be in canonical form, otherwise a general-protection exception, #GP(0), is generated.

LLDT and LTR instructions load a system descriptor into the processor's internal LDTR and TR segment descriptor registers (hidden portion). In 64-bit mode, the expanded descriptor format and redefined descriptor types give rise to the following restrictions on the descriptors that these instructions can load:

- A general-protection exception, #GP (selector), is generated if an attempt is made to load the second type field (bits 12:8 of the highest dword) with a value other than 00H.
- The 64-bit base address loaded by an LLDT or LTR must be in canonical form, otherwise a general-protection exception, #GP (selector), is generated.
- A general-protection exception, #GP (selector), is generated if an LTR instruction references either a busy or 16bit TSS.

In 64-bit mode, the LTR instruction still changes a task's state to busy (descriptor type set to 0BH). Because IA-32e mode does not support task switches, a task descriptor's busy bit is never automatically cleared. If the operating system has previously loaded the task descriptor using the LTR instruction, the operating system is responsible for clearing the task's busy bit (setting descriptor type to 09H). The actual order of the memory accesses generated by the LLDT and LTR instructions is implementation specific.

1.6.5. Linear Addressing and Paging

When IA-32e mode is enabled, the linear address to physical address translation is different than in legacy protected mode. With the introduction of a new page mapping table, page map level 4 (PML4), 64-bit addresses are translated into physical addresses using conversions described in the following paragraphs.

1.6.5.1. Software Address Translations in 64-Bit Mode

Using the flat address space in 64-bit mode, linear addresses are equal to effective addresses. As mentioned in the memory section above: when FS or GS segments are used with a non-zero base, linear and effective addresses are not the same.

Generally, displacements and immediates in 64-bit mode are not extended to 64 bits. They are still limited to 32 bits and sign-extended during effective-address calculations. In 64-bit mode, however, some of the 64-bit displacement and immediate forms of the MOV instruction are supported.

All 16-bit and 32-bit address calculations are zero-extended in IA-32e mode to form 64-bit addresses. Address calculations are first truncated to the effective address size of the current mode (64-bit mode or compatibility mode) as overridden by any address-size prefix. The result is then zero-extended to the full 64-bit address width. Because of this, 16-bit and 32-bit applications running in compatibility mode can access only the low 4GBytes of the 64-bit mode effective addresses. Likewise, a 32-bit address generated in 64-bit mode can access only the low 4GBytes of the 64-bit mode effective-address space.

1.6.5.2. Paging Data Structures

The 64-bit extensions architecture expands physical address extension (PAE) paging structures to potentially support mapping a 64-bit linear address to a 52-bit physical address. In the first implementation of the Intel EM64T, PAE paging structures are extended to support translation of a 48-bit linear address into a 40-bit physical address.



Prior to activating IA-32e mode, PAE must be enabled by setting CR4.PAE = 1. PAE expands the size of an individual page-directory entry (PDE) and page-table entry (PTE) from 32 bits to 64 bits to support physical-address sizes of greater than 32 bits. Attempting to activate IA-32e mode prior to enabling PAE results in a general-protection exception (#GP).

64-bit extensions architecture adds a new table, called the page map level 4 table (PML4), to the linear-address translation hierarchy. The PML4 table sits above the page directory pointer (PDP) table in the page-translation hierarchy. The PML4 contains 512 eight-byte entries, with each entry pointing to a PDP table. Nine linear-address bits are used to index into the PML4.

PML4 tables are used in page translation only when IA-32e mode is activated. They are not used when IA-32e mode is disabled, regardless of whether or not PAE is enabled. The existing page-directory pointer table is expanded by the 64-bit extensions to 512 eight-byte entries from four entries. As a result, nine bits of the linear address are used to index into a PDP table rather than two bits. The size of both page-directory entry (PDE) tables and page-table entry (PTE) tables remains 512 eight-byte entries, each indexed by nine linear-address bits. The total of linear-address index bits into the collection of paging data structures (PML4 + PDP + PDE + PTE + page offset) defined above is 48. The method for translating the high-order 16 linear-address bits into a physical address is currently reserved.

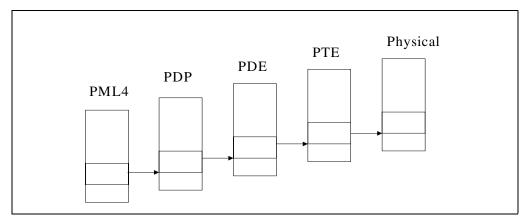


Figure 1-6. Paging Data Structures

The PS flag in the page directory entry (PDE.PS) selects between 4KByte and 2MByte page sizes. Because PDE.PS is used to control large page selection, the CR4.PSE bit is ignored. Table 1-18 through Table 1-21 shows the 64-bit mode PML4, PDP, PDE, and PTE formats when 4KByte pages are enabled.

Table 1-18 IA-32e Mode Page Map Level 4 Entry (PML4 - 4K Pages)

	Table 1-10 IA	-32e IVIO	ue raye map	Level 4 Li	LIY (PIVIL	4 - 4	N Fa	iges	<u> </u>		
63	62:52	62:52 51:40		40 39:12 11:9		5	4	3	2	1	0
Reserved	Available	Reserved	Page Directory Pointer Base Address	Available	Reserved	A	PCD	PWT	U/S	R/W	ס

Table 1-19 IA-32e Mode Page Directory Pointer Table Entry (PDPTE - 4K Pages)

			<u> </u>	,	· · · / /	· J · · /					
63	62:52	51:40	39:12	11:9	8:6	5	4	3	2	1	0
Reserved	Available	Reserved	Page Directory Base Address	Available	Reserved	А	PCD	TWP	S/N	M/H	Р

Table 1-20 IA-32e Mode Page Directory Entry (PDE - 4K Pages)

63	62:52	51:40	39:12	11:9	8	7	6	5	4	3	2	1	0
Reserved	Available	Reserved	Page Table Base Address	Available	Reserved	0	Resvd	Α	PCD	PWT	U/S	R/W	Р

Table 1-21 IA-32e Mode Page Table Entry (PTE - 4K Pages

63	62:52	51:40	39:12	11:9	8	7	6	5	4	3	2	1	0
Reserved	Available	Reserved	Page Base Address	Available	G	PAT	D	Α	PCD	PWT	U/S	R/W	Р

The physical base-address field in all four table entry formats is extended by 64-bit extensions to bits 51:12. This allows paging tables to be located anywhere in the physical memory supported by a 64-bit implementation. Implementations that do not support the maximum physical-address size reserve the unsupported high-order bits and require that they be cleared to zeros. The physical base-address field in the first implementation of the IA-32e technology is specified by bits 39:12. Software should take care not to locate paging tables above 4G in memory if it is anticipated that mode changes to legacy mode will be needed.

Bit 63 is reserved. Bits 62:52 in all page-table entry formats are available for use by system software. In the 64-bit extensions architecture, future implementations leave bits 62:52 available for software use. Other than the extensions made to the base-address field and the addition of the software-available field at bits 62:52, all other PDE and PTE fields are the same as in legacy mode.

Fields within the PDP table entry are similar to legacy mode PDP table entries, with the following exceptions. The exceptions reflect changes necessary to indicate that a higher-level paging structure (PML4) now references the PDP tables:

- Bit 0 is no longer reserved. IA-32e mode defines this bit as the present (P) flag to indicate whether or not the PDE table referenced by the PDP entry is currently stored in physical memory. A page-fault exception (#PF) is generated when the processor accesses a PDP entry with the P flag cleared to 0.
- Bit 1 is no longer reserved. IA-32e mode defines this bit as the read/write (R/W) flag.
- Bit 2 is no longer reserved. IA-32e mode defines this bit as the user/supervisor (U/S) flag.
- Bit 5 is no longer reserved. IA-32e mode defines this bit as the accessed (A) flag.
- The base-address field extensions, as specified above.
- Bits 62:52 are available to software, as specified above. The format of a PML4 table entry is identical to the 64-bit mode PDP table-entry format.

Table 1-22 through Table 1-24 shows the 64-bit mode PML4, PDP, and PDE formats when 2MByte pages are enabled. As with legacy mode, 2MByte pages are enabled by setting the PDE page-size bit to 1 (PDE.PS = 1). Control of 2M page sizes is not dependent on CR4.PSE.

Table 1-22 IA-32e Mode Page Map Level 4 Entry (PML4 - 2MB Pages)

63	62:52	51:40	39:12	11:9	8:6	5	4	3	2	1	0
Reserved	Available	Reserved	Page Directory Pointer Base Address	Available	Reserved	Α	PCD	PWT	U/S	R/W	Р

Table 1-23 IA-32e Mode Page Directory Pointer Table Entry (PDPTR - 2MB Pages)

63	63:52	51:40	39:12	11:9	8:6	5	4	3	2	1	0
Reserved	Available	Reserved	Page Directory Base Address	Available	Resvd	Α	PCD	PWT	U/S	R/W	Р

Table 1-24 IA-32e Mode Page Directory Entry (PDE - 2MB Pages)

					, , , (
63	62:52	51:40	39:21	20:13	12	11:9	8	7	6	5	4	თ	2	1	0
Reserved	Available	Reserved	Page Base Address	Reserved	PAT	Available	G	1	D	Α	PCD	PWT	S/U	M/H	Р

The physical base-address field in all three table-entry formats is extended by the 64-bit architecture to bits 51:12. This allows paging tables to be located anywhere in the physical memory supported by a 64-bit mode implementation. Implementations that do not support the maximum physical-address size reserve the unsupported high-order bits and require that they be cleared to zeros.

The physical base-address field in the first implementation of the Intel EM64T is specified by bits 39:12. Bits 63:52 in all page-table entry formats are available for use by system software. In the 64-bit extensions, future implementations will leave bits 63:52 available for software use. When 2MByte pages are selected, the PDE points directly to the physical page, and not to a PTE. Other than the extensions made to the base-address field and the addition of the software-available field at bits 63:52, all other PDE fields are the same as in legacy mode.

Fields within the PDP table entry are similar to legacy-mode PDP table entries, with the following exceptions. The exceptions reflect changes necessary to indicate that a higher-level paging structure (PML4) now references the PDP tables:

- Bit 0 is no longer reserved. IA-32e mode defines this bit as the present (P) flag to indicate whether or not the PDE table referenced by the PDP entry is currently stored in physical memory. A page-fault exception (#PF) is generated when the processor accesses a PDP entry with the P flag cleared to 0.
- Bit 1 is no longer reserved. IA-32e mode defines this bit as the read/write (R/W) flag.
- Bit 2 is no longer reserved. IA-32e mode defines this bit as the user/supervisor (U/S) flag.
- Bit 5 is no longer reserved. IA-32e mode defines this bit as the accessed (A) flag.
- The base-address field extensions, as specified above.
- Bits 62:52 available to software, as specified above.

The format of a PML4 table entry is identical to the 64-bit mode PDP table-entry format.

1.6.5.3. Overall Page Protection

The addition of incremental layers of paging necessitates changes by applying the same IA-32 methodology on the R/W and U/S flags to four levels of the page tables. Table 1-25 shows IA-32e mode paging protection.

Table 1-25 IA-32e Mode Page Level Protection Matrix

Privilege (U/S bit, bit 2)			Access Type (R/W bit, bit 1				Combined Effects		
PML4	PDP	PDE	PTE	PML4	PDP	PDE	PTE		
User	User	User	User	RO	*	*	*	User	RO
User	User	User	User	*	RO	*	*	User	RO
User	User	User	User	*	*	RO	*	User	RO
User	User	User	User	*	*	*	RO	User	RO
User	User	User	User	R/W	R/W	R/W	R/W	User	R/W
Super	*	*	*	*	*	*	*	Super	R/W
*	Super	*	*	*	*	*	*	Super	R/W
*	*	Super	*	*	*	*	*	Super	R/W
*	*	*	Super	*	*	*	*	Super	R/W

There are two basic rules for combining the privilege levels and access types from multiple levels of page entries within a page walk. These are:

- 1. The most privilege mode of the page entries within the page walk will be used. (for example, a page has a supervisor privilege level if the U/S bit is cleared in any of its page entries).
- 2. The most restrictive access type of the page entries with the page walk will be used (for example, a page has a read-only access type if the R/W bit is cleared in any of the page entries).

1.6.5.4. Reserved Bit Checking

The processor will enforce reserved bit checking on the following paging mode specific bits. Table 1-26 shows the reserved bits that are checked. In legacy paging modes of Table 1-26:

- Non-PAE 4KB paging: 4KB-page only paging (CR4.PAE = 0, CR4.PSE = 0)
- PSE-36: 4KB and 4MB pages (CR4.PAE = 0, CR4.PSE = 1)
- PAE: 4KB and 2MB pages (CR4.PAE = 1, CR4.PSE = x)

Table 1-26 Reserved Bit Checking

Mode	Paging Mode	Check Bits	
Legacy	4KB Paging (Non-PAE)	no reserved bits checked	
	PSE36 - PDE, 4MB page	bit [21]	
	PSE36 - PDE, 4KB page	no reserved bits checked	
	PSE36 - PTE	no reserved bits checked	
	PAE - PDP table entry	bits [63:40] & [8:5] & [2:1]	
	PAE - PDE, 2MB page	bits [63:40] & [20:13]	
	PAE - PDE, 4KB page	bits [63:40]	
	PAE - PTE	bits [63:40]	
64-bit	PML4E	bit [63], bits [51:40]	
	PDPTE	bit [63], bits [51:40]	
	PDE, 2MB page	bit [63], bits [51:40] & [20:13]	
	PDE, 4KB page	bit [63], bits [51:40]	
	PTE	bit [63], bits [51:40]	

1.6.6. Enhanced Legacy-Mode Paging

Some changes made to paging data structures to support the larger physical address sizes used in IA-32e mode are available in legacy mode operating systems. Legacy-mode operating system can take advantage of the enhancements made to the physical address extension (PAE) support and page size extension (PSE) support (more physical address bits). However, the four-level page translation mechanism introduced by IA-32e mode is not available to legacy mode software.

As previously described, setting CR4.PAE = 1 expands the size of an individual PDE and PTE from 32 bits to 64 bits. This allows physical-address sizes of greater than 32 bits. Previous legacy IA-32 implementations limit physical-address size to 36-bits.

The architecture of processors with Intel EM64T support potentially allows legacy-mode software to load up to 52-bit physical addresses in the PDE and PTE. Unsupported physical-address bits are reserved and must be cleared to zero. In the first implementation of the Intel EM64T, legacy-mode software can use up to 40 bits of the physical address in PDE and PTE entries. Software must clear bits 62:40 to 0. Bit 63 is reserved.

Legacy-mode page-size extensions (PSE) are enabled by setting the CR4 page-size enable bit to 1 (CR4.PSE = 1). PSE modifies the original 4-byte PDE format to support 4MByte pages in addition to legacy 4KByte pages. 4MByte pages are selected by setting the PDE page size bit to 1 (PDE.PS = 1); clearing the bit selects 4KByte pages (PDE.PS = 0).

When PDE.PS = 1, the processor combines PDE bits 31:22 with linear address bits 21:0 to form a 32-bit physical address into a 4MByte page. Legacy PTEs are not used in a 4MByte page translation. Because legacy PTEs are not used, PDE bits 21:12 are reserved in the original PSE mode definition. Updates to PSE mode change the 4-byte PDE format so that it also supports 36-bit physical addresses without requiring the 8-byte format used by PAE. This is accomplished by using previously reserved PDE bits 16:13 to hold four additional high-order physical address bits. Bits 21:17 are reserved.

The architecture of IA-32 processors that support Intel EM64T further modifies the 4-byte PDE format in PSE mode to increase physical address size support to 40 bits. This is accomplished by defining previously reserved PDE bits 20:17 to hold four additional high-order physical address bits. Bit 21 is reserved and must be cleared to 0. Table 1-27 shows the format of the PDE when PSE mode is enabled. The high-order physical address bits 39:32 are located in PDE[20:13], while physical address bits 31:22 are located in PDE[31:22].

Table 1-27 Legacy Mode Page Directory Entry for 4MB Pages

31:22	21	20:13	12	11:9	8	7	6	5	4	3	2	1	0
Base Address 31:22	0	Page Base Address 39:32	PAT	Available	G	1	D	Α	PCD	PWT	U/S	R/W	Р

1.6.7. CR2 and CR3

The size of CR2 (page-fault address register) is increased to 64 bits by 64-bit extensions to hold 64-bit linear addresses. It also increases the size of CR3 (page-directory base register) to 64 bits. This allows first level of the paging structures to be located anywhere in physical memory, subject to the implementation-dependent physical-address size limits.

Table 1-28 shows the 64-bit mode format of CR3. The Base Address field specifies the most-significant bits of the page-directory base address above bit 11. The Page-Directory Base field holds the most-significant physical-address bits of the top-level paging structure. Bits 51:12 of CR3 define the maximum base address allowed by the 64-bit mode architecture, but specific implementations can support smaller physical-address spaces. The lower 12 bits (11:0) of the base address are always assumed to be zero. This forces the top-level paging structure to be aligned on a 4KByte boundary.

Table 1-28 IA-32e Mode CR3

63:52	51:40	39:32	32:12	11:5	4	3	2:0
Reserved	Reserved	Base Address	Page Directory Base	Reserved	PCD	PWT	Reserved

CR3[12] through to the bit position that represents the maximum limit of physical addressability supported by the processor (CR3[39] if the implementation supports 40 bits of physical addressing) specify the top-level paging-structure (PML4) base address. CR3[51] down to the bit position that represents the maximum limit of physical addressability supported by the processor are reserved and must be cleared to zero (for example, CR3[51:40] if the implementation supports 40 bits of physical addressing).

CR3[63:52] are reserved. CR3[51:40] are reserved for future expansion of the page-directory base address. In the first implementation of the Intel EM64T, the processor checks that these bits are written as zeros and generates a general-protection exception, #GP(0), if they are not.

The MOV to CR3 instruction is not affected by operand size in IA-32e mode. In 64-bit mode, all 64 bits of CR3 are loaded from the source register. In compatibility mode, only the lower 32 bits of CR3 are loaded from the source register and the upper 32 bits are cleared to 0.

1.6.8. Address Translation

When paging is used in IA-32e mode, the processor divides the linear address into a collection of table and physical-page offsets, much like in legacy mode. However, the 64-bit architecture extends how the processor divides the linear address to support the 64-bit linear-address size and its deeper paging data-structure hierarchy.

4KByte pages are enabled by clearing the PDE page-size flag (PDE.PS = 0). As the first implementation of the IA-32e technology supports a maximum 48-bit linear address, this paging option supports 2^{36} 4KByte pages spanning a linear-address space of 2^{48} bytes (256 terabytes).

The 48-bit linear address is broken into five fields to index into the 4-level paging structure, as follows:

- Bits 47:39 index into the 512-entry page map level-4 table (PML4).
- Bits 38:30 index into the 512-entry page-directory pointer table (PDP).



- Bits 29:21 index into the 512-entry page-directory table (PDE).
- Bits 20:12 index into the 512-entry page table (PTE).
- Bits 11:0 provide the byte offset into the physical page.

Table 1-29 4KB Page Translation

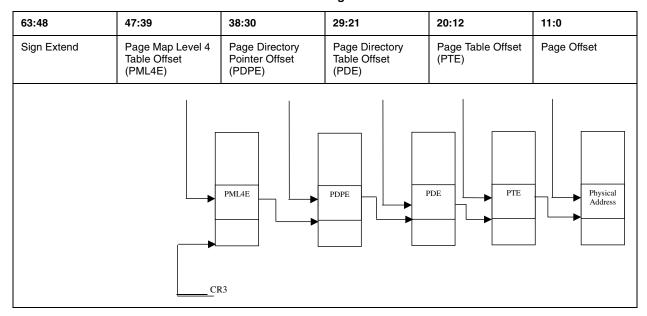


Table 1-30 CR3 Page Directory Pointer Offset

63:40	39:12	11:0
	Page Directory Pointer Offset	

2MByte pages are enabled by setting the PDE page size flag to 1 (PDE.PS = 1). Because the first implementation of the Intel EM64T supports a maximum 48 bits of linear address, this paging option supports 2^{27} 2MByte pages spanning a linear-address space of 2^{48} bytes (256 terabytes).

The 48-bit linear address is broken up into four fields to index into the 3-level paging structure, as follows:

- Bits 47:39 index into the 512-entry page map level-4 table.
- Bits 38:30 index into the 512-entry page-directory pointer table.
- Bits 29:21 index into the 512-entry page-directory table.
- Bits 20:0 provide the byte offset into the physical page

1.6.9. Privilege-Level Transitions and Far Transfers

The 64-bit extensions provide three mechanisms for changing privilege levels:

- Call gates and interrupt gates
- SYSCALL and SYSRET instructions
- SYSENTER and SYSEXIT instructions

Table 1-31 2MB Page Translation

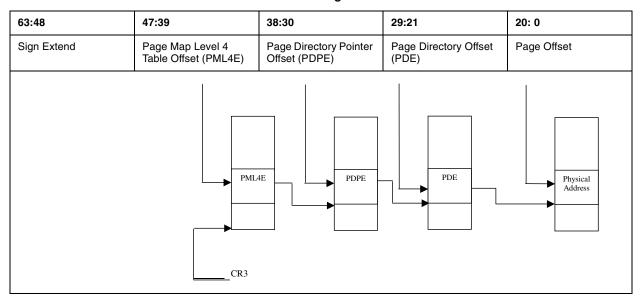


Table 1-32 CR3

63:40	39:12	11:0
	Page Directory Pointer Offset	

1.6.9.1. Call Gates

The call-gate mechanism provides a public entry point into the operating system. It also provides a means for changing privilege levels and stacks when calling the operating system.

Legacy IA-32 call-gate descriptors provide a 32-bit offset for the instruction pointer (EIP). 64-bit extensions double the size of legacy call gates to provide a 64-bit offset for the instruction pointer (RIP). Table 1-33 shows the layout of a call-gate descriptor in IA-32e mode. Table 1-34 describes the fields in a 64-bit mode call gate.

Table 1-33 Call Gates in IA-32e Mode

	Bit Position					
DW Offset	31:16	15	14:13	12:8	7:0	
3	Reserved			00000	Reserved	
2		Offset 63:32				
1	Offset 31:16	Р	DPL	0CH (type)	00000000	
0	Target Segment Selector			Offset 15:0		

The first eight bytes (bytes 7:0) of a 64-bit mode call gate are similar but not identical to legacy 32-bit call gates. The parameter copy count field has been removed. Bytes 11:8 hold the upper 32 bits of the target-segment offset in canonical form. A general-protection exception (#GP) is generated if software attempts to use a call gate with a target offset that is not in canonical form.

The target code segment referenced by the call gate must be a 64-bit code segment (CS.L = 1, CS.D = 0). If it is not, a general-protection exception, #GP (selector), is generated with the target CS selector reported as the error code. The double-sized descriptors can reside in the same descriptor table as 16-bit and 32-bit legacy descriptors. A second type field, used for consistency checking, is defined in bits 12:8 of the highest dword and must be cleared to zero. This illegal type (00H) results in a general-protection exception (#GP) if an attempt is made to access the upper half of the 64-bit mode descriptor as a legacy descriptor.



Only 64-bit mode call gates can be referenced in IA-32e mode (64-bit mode and compatibility mode). The legacy 32-bit call gate type (0CH) is redefined in IA-32e mode as the 64-bit call-gate type. No 32-bit call-gate type exists in IA-32e mode. If a far call references a 16-bit call gate type (04H), a general-protection exception (#GP) is generated.

Table 1-34 IA-32e Mode Call Gate Fields

DWord Offset	Gate Field	Function
3	+12(31:13)	Unused
	+12(12:8)	Must Be 0
	+12(7:0)	Unused
2	+8(31:0)	Offset in canonical form
1	+4(31:16)	Offset 31:16
	+4(15:13)	Present and Descriptor Privilege Level
	+4(12:8)	64-bit Mode Call Gate Type (0Ch)
	+4(7:0)	Unused
0	+0(31:16)	Target Segment Selector
	+0(15:0)	Offset 15:0

When a CALL references a 64-bit mode call gate, the actions taken are identical to those taken in legacy calls through a 32-bit gate, with these exceptions:

- Stack pushes are performed in eight-byte increments.
- A 64-bit RIP is pushed onto the stack.
- Parameter copying is not performed.

Software should use the matching far-return instruction size for correct operation (for example, a return from a 64-bit call must be performed with a 64-bit operand-sized return in order to process the stack correctly).

1.6.9.2. Privilege-Level Changes and Stack Switching

A call gate can be used to change to a more-privileged code segment. Although the protection-check rules for call gates are unchanged in IA-32e mode from legacy mode, the associated stack-switch changes slightly in IA-32e mode.

In IA-32e mode, the target of any call gate must be a 64-bit code segment. 64-bit mode does not use segmentation. Stack pointers consist solely of the 64-bit stack pointer (RSP) and the SS segment register is ignored.

When stacks are switched as part of a 64-bit mode privilege-level change through a call gate, a new SS descriptor is not loaded. IA-32e mode only loads an inner-level RSP from the TSS. The new SS is forced to null and the SS selector's RPL field is forced to the new CPL. The new SS is set to null in order to handle nested far transfers (CALLF, INTn, interrupts and exceptions). The old SS and RSP are saved on the new stack (Table 1-35). On the subsequent RETF the old SS is popped from the stack and loaded into the SS register.

Table 1-35 64-Bit-Mode Stack Layout After CALLF with CPL Change

Legacy Mode					IA-32e mode
Old SS Selector	+12			+24	Old SS Selector
Old ESP	+8			+16	Old RSP
CS Selector	+4			+8	Old CS Selector
EIP	0	ESP	RSP	0	RIP
< 4 Bytes >					< 8 Bytes >

In summary, a stack switch in IA-32e mode works like the legacy stack switch except that a new SS selector is not loaded from the TSS. Instead, the new SS is forced to null. All 64-bit mode stack operations resulting from a privilege-level-changing far call or far return are eight-bytes wide and change the RSP by eight.

IA-32e mode does not support the automatic parameter-copy feature found in legacy mode. The call-gate count field is ignored by IA-32e mode. Software can access the old stack, if necessary, by referencing the old stack-segment selector and stack pointer saved on the new process stack.

In IA-32e mode, RETF is allowed to load a null SS under certain conditions. If the target mode is 64-bit mode and the target CPL <> 3, IRET allows SS to be loaded with a null selector.

As part of the stack switch mechanism, an interrupt or exception sets the new SS to null instead of fetching a new SS selector from the TSS and loading the corresponding descriptor from the GDT or LDT. The new SS selector is set to null in order to properly handle returns from subsequent nested far transfers. If the called procedure itself is interrupted, the null SS is pushed on the stack frame. On the subsequent RETF, the null SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

1.6.9.3. Fast System Calls

The SYSCALL and SYSRET instructions are designed for operating systems that use a flat memory model where segmentation is not used. These instructions (along with SYSENTER/SYSEXIT) are ideally suited for IA-32e mode operation. SYSCALL and SYSRET are not supported in compatibility mode. See CPUID for more details about detecting the availability of SYSCALL and SYSRET.

The semantics of SYSCALL and SYSRET specify a 64-bit code offset. In IA-32e mode, the clearing of bits in the EFLAGS is programmable rather than fixed; SYSCALL and SYSRET save and restore the EFLAGS register.

The legacy System Target-Address Register (STAR) cannot be expanded to provide a 64-bit target RIP address because the upper 32 bits of that MSR already contain the target CS and SS selectors. IA-32e mode provides two new STAR registers — Long STAR (LSTAR) and Compatibility STAR (CSTAR) — that hold a 64-bit target RIP. LSTAR holds the target RIP used by a SYSCALL when IA-32e mode is activated and the calling program is in 64-bit mode. CSTAR holds the target RIP used by a SYSCALL when IA-32e mode is activated and the calling program is in compatibility mode. In the initial implementation of IA-32e, CSTAR is not available because SYSCALL and SYSRET is not supported in compatibility mode.

The SYSCALL and SYSRET CS and SS selectors used in IA-32e mode and legacy mode are stored in the STAR. LSTAR and CSTAR are written by the WRMSR instruction. The addresses written to LSTAR and CSTAR are first checked by the WRMSR instruction to ensure they are in canonical form. If they are not, a general protection exception (#GP) is generated. Table 1-36 shows the layout and MSR numbers for the STAR, LSTAR, CSTAR and FMASK registers.

	rubic 1 00 CTAIT, and COTAIT model opcome regional (morte)								
		63:48	47:32	31:0					
STAR	C0000081H	SYSRET CS and SS	Reserved	Reserved					
LSTAR	C0000082H	Target RIP for 64-bit Mode 0	Target RIP for 64-bit Mode Calling Programs						
CSTAR	C0000083H	Target RIP for Compatibility	Mode Calling Programs						
FMASK	C0000084H	Reserved		SYSCALL EFLAGS Mask					

Table 1-36 STAR, LSTAR, and CSTAR Model-Specific Registers (MSRs)

See details on the operation of SYSCALL and SYSRET in Chapter 4 of Volume 2.

1.6.9.4. Task State Segments

The legacy IA-32 task-switching architecture is not supported in IA-32e mode. IA-32e mode requires that task management and switching be performed by software. The processor issues a general-protection exception (#GP) if any of the following is attempted in IA-32e mode:

- A control transfer to a TSS or a task gate via a JMP, CALL, INTn, or interrupt
- An IRET with EFLAGS.NT (nested task) set to 1



Although the hardware task-switching mechanism is not supported in IA-32e mode, a 64-bit task state segment (TSS) must still exist. Table 1-37 shows the format of a 64-bit TSS. This 64-bit field holds information important to IA-32e mode and is not directly related to the task-switch mechanism. These are:

- RSPn; the full 64-bit canonical forms of the stack pointers (RSP) for privilege levels 0–2
- ISTn; the full 64-bit canonical forms of the interrupt stack table (IST, see Section 1.6.10.4.) pointers
- I/O Map Base Address; the 16-bit offset to the I/O permission bit map from the 64-bit TSS base

The operating system must create at least one 64-bit TSS after activating IA-32e mode. It must execute the LTR instruction (in 64-bit mode) to load the TR register with a pointer to the 64-bit TSS responsible for both 64-bit-mode programs and compatibility-mode programs.

Table 1-37 TSS Format in IA-32e Mode

Byte Offset	31:16	15:0
+64H	I/O Map Base	Reserved
+60H	Reserved	
+5CH	Reserved	
+58H	IST7 (upper 32-bits)	
+54H	IST7 (lower 32-bits)	
+50H	IST6 (upper 32-bits)	
+4CH	IST6 (lower 32-bits)	
+48H	IST5 (upper 32-bits)	
+44H	IST5 (lower 32-bits)	
+40H	IST4 (upper 32-bits)	
+3CH	IST4 (lower 32-bits)	
+38H	IST3 (upper 32-bits)	
+34H	IST3 (lower 32-bits)	
+30H	IST2 (upper 32-bits)	
+2CH	IST2 (lower 32-bits)	
+28H	IST1 (upper 32-bits)	
+24H	IST1 (lower 32-bits)	
+20H	Reserved	
+1CH	Reserved	
+18H	RSP2 (upper 32-bits)	
+14H	RSP2 (lower 32-bits)	
+10H	RSP1 (upper 32-bits)	
+0CH	RSP1 (lower 32-bits)	
+08H	RSP0 (upper 32-bits)	
+04H	RSP0 (lower 32-bits)	
00H	Reserved	

1.6.10. Interrupts

Interrupts and exceptions force control transfers from the currently executing program to an interrupt service routine that handles the particular interrupt. The interrupt-handling and exception-handling mechanism saves the interrupted program's execution state, transfers control to the interrupt service routine, and ultimately returns to the interrupted program.

Throughout this section, the term "interrupt" covers both asynchronous events generated external to the processor and synchronous events related to instruction execution (exceptions, faults and traps). 64-bit extensions expand the legacy IA-32 interrupt-processing and exception-processing mechanism to support 64-bit operating systems and applications. These changes include:

- All interrupt handlers pointed by the IDT are 64-bit code (does not apply to the SMI handler).
- The size of interrupt-stack pushes is fixed at 64 bits. The processor uses 8-byte, zero extended stores.
- The stack pointer (i.e. SS:RSP) is pushed unconditionally on interrupts. In legacy environments, this push is conditional and based on a change in current privilege level (CPL).
- The new SS is set to null if there is a change in CPL.
- IRET behavior changes.
- There is a new interrupt stack-switch mechanism.
- The alignment of interrupt stack frame is different.

1.6.10.1. Gate Descriptor Format

The interrupt descriptor table (IDT) contains gate descriptors that are used to locate the service routine for each interrupt vector. Legacy interrupt-gate descriptors provide a 32-bit offset for the instruction pointer (EIP). 64-bit extensions double the size of legacy interrupt gates from eight bytes to 16 bytes in order to provide a 64-bit offset for the instruction pointer (RIP). The 64-bit RIP referenced by an interrupt-gate descriptor allows an interrupt service routine to be located anywhere in the linear-address space.

Table 1-38 shows the layout of 64-bit mode interrupt-gate and trap-gate descriptors. Table 1-39 describes the fields in a 64-bit interrupt and trap gate.

31:16 15 14 13 12:8 7:3 2:0 Reserved Offset 63:32 Р DPL 000000 Offset 31:16 Type IST Target Segment Offset 15:0 Selector

Table 1-38 Interrupt and Trap Gate in IA-32e Mode

In legacy mode, the IDT index is formed by scaling the interrupt vector by eight. In IA-32e mode, the IDT index is formed by scaling the interrupt vector by 16. The first eight bytes (bytes 7:0) of a 64-bit mode interrupt gate are similar but not identical to legacy 32-bit interrupt gates. Bytes 11:8 hold the upper 32 bits of the target RIP (interrupt segment offset) in canonical form. A general-protection exception (#GP) is generated if software attempts to reference an interrupt gate with a target RIP that is not in canonical form.

The target code segment referenced by the interrupt gate must be a 64-bit code segment (CS.L = 1, CS.D = 0). If the target is not a 64-bit code segment, a general-protection exception (#GP) is generated with the IDT vector number reported as the error code.



Table 1-39 IA-32e Mode Interrupt and Trap Gate Fields

DWord Offset	Gate Field	Function
3	+12(31:0)	Unused
2	+8(31:0))	Offset bits 63:32
1	+4(31:16)	Offset bits 31:16
	+4(15:13)	Present and Descriptor Privilege Level
	+4(12:8)	64-bit Interrupt or Trap Gate Type (0Eh)
	+4(7:3)	Unused
	+4(2:0)	IST Index
0	+0(31:16)	Target Segment Selector
	+0(15:0)	Offset 15:0

Only 64-bit interrupt and trap gates can be referenced in IA-32e mode (64-bit mode and compatibility mode). The legacy 32-bit interrupt or trap gate types (0EH or 0FH) are redefined in IA-32e mode as the 64-bit interrupt and trapgate types. No 32-bit interrupt or trap gate type exists in IA-32e mode. If a reference is made to a 16-bit interrupt or trap gate (06H or 07H), a general-protection exception (#GP(0)) is generated.

1.6.10.2. Stack Frame

In legacy mode, the size of an IDT entry (either 16 bits or 32 bits) determines the size of interrupt-stack-frame pushes. SS:ESP is pushed only on a CPL change. In 64-bit mode, the size of interrupt stack-frame pushes is fixed at eight bytes. This is because only 64-bit mode gates can be referenced. 64-bit mode also pushes SS:RSP unconditionally, rather than pushing only on a CPL change.

Aside from error codes, pushing SS:RSP unconditionally presents operating systems with a consistent interrupt-stack-frame size across all interrupts. Interrupt service-routine entry points that handle interrupts generated by the INTn instruction or external INTR# signal can push an additional error code place-holder to maintain consistency.

In legacy mode, the stack pointer is at any alignment when an interrupt or exception causes a stack frame to be pushed. Thus, the stack frame and succeeding pushes done by the interrupt handler are at arbitrary alignments. In IA-32e mode, the RSP is aligned to a 16-byte boundary before pushing the stack frame. The stack frame itself will be aligned on a 16-byte boundary when the interrupt handler is entered. The processor can arbitrarily realign the new RSP on interrupts because the previous (possibly unaligned) RSP is unconditionally saved on the newly aligned stack. The previous RSP will be automatically restored by the subsequent IRET.

Aligning the stack permits exception and interrupt frames to be aligned on a 16-byte boundary before interrupts are reenabled. This allows the stack to be laid out for optimal storage of 16-byte XMM registers. This enables the interrupt handler to use the faster 16-byte aligned loads and stores (MOVAPS) rather than unaligned accesses (MOVUPS) to save and restore XMM registers. Efficiently saving and restoring the XMM registers becomes more important as SSE is emphasized over x87 for floating point. Although the RSP alignment is done in all cases when LMA = 1, it is only of consequence for the kernel-mode case where there is no stack switch or IST used. For a stack switch or IST, the OS would have presumably put suitably aligned RSP values in the TSS.

1.6.10.3. IRET

IRET semantics change in IA-32e mode. Execute IRET with an 8-byte operand size. There is nothing that forces this requirement; but the stack is formatted such that for typical actions where IRET is required, an 8-byte IRET operand size works correctly.

In 64-bit mode, SS:RSP pops unconditionally. In compatibility and legacy modes, SS:RSP is popped only if the CPL changes. Because interrupt stack-frame pushes are always eight bytes in IA-32e mode, an IRET must pop eight byte items off the stack. This is accomplished by preceding the IRET with a 64-bit operand-size prefix. The size of the pop is determined by the address size of the instruction. The SS/ESP/RSP size adjustment is determined by the stack size.

IRET pops SS:RSP unconditionally off the interrupt stack frame only when it is executed in 64-bit mode. In compatibility mode, IRET pops SS:RSP off the stack only if there is a CPL change. This allows legacy applications to execute properly in compatibility mode when using the IRET instruction.

64-bit interrupt service routines that exit with an IRET unconditionally pop SS:RSP off of the interrupt stack frame, even if the target code segment is running in 64-bit mode or at CPL = 0. This is because the original interrupt always pushes SS:RSP.

In IA-32e mode, IRET is allowed to load a null SS under certain conditions. If the target mode is 64-bit mode and the target CPL <> 3, IRET allows SS to be loaded with a null selector. As part of the stack switch mechanism, an interrupt or exception sets the new SS to null, instead of fetching a new SS selector from the TSS and loading the corresponding descriptor from the GDT or LDT. The new SS selector is set to null in order to properly handle returns from subsequent nested far transfers. If the called procedure itself is interrupted, the null SS is pushed on the stack frame. On the subsequent IRET, the null SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

1.6.10.4. Stack Switching

The legacy IA-32 architecture provides a mechanism to automatically switch stack frames in response to an interrupt. The 64-bit extensions implement a slightly modified version of the legacy stack-switching mechanism and an alternative stack-switching mechanism called the interrupt stack table (IST).

In legacy mode, the legacy IA-32 stack-switch mechanism is unchanged. In IA-32e mode, the legacy stack-switch mechanism is modified. When stacks are switched as part of a 64-bit mode privilege-level change resulting from an interrupt, a new SS descriptor is not loaded. IA-32e mode only loads an inner-level RSP from the TSS. The new SS selector is forced to null and the SS selector's RPL field is set to the new CPL. The new SS is set to null in order to handle nested far transfers (CALLF, INT, interrupts and exceptions). The old SS and RSP are saved on the new stack (Table 1-40). On the subsequent IRET, the old SS is popped from the stack and loaded into the SS register.

In summary, a stack switch in IA-32e mode works like the legacy stack switch, except that a new SS selector is not loaded from the TSS. Instead, the new SS is forced to null.

Legacy Mode			64-bit Mode		
Content	Byte Offset	-		Byte Offset	Content
Old SS	+20			+40	Old SS
Old ESP	+16			+32	Old RSP
EFLAGS	+12			+24	RF Flags
CS	+8			+16	CS
EIP	+4			+8	RIP
Error Code	0	ESP	RSP	0	Error Code
< 4 Bytes >					< 8 Bytes >

Table 1-40 IA-32e Mode Stack Layout After Interrupt With CPL Change

1.6.10.5. Interrupt Stack Table

In IA-32e mode, a new interrupt stack table (IST) mechanism is available as an alternative to the modified legacy stack-switching mechanism described above. This IST mechanism unconditionally switches stacks when it is enabled. It can be enabled on an individual interrupt-vector basis via a field in the IDT entry. Thus, some interrupt vectors can use the modified legacy mechanism and others can use the IST mechanism. The IST mechanism is only available in IA-32e mode. It is part of the 64-bit mode TSS shown in Table 1-37. The primary motivation for the IST mechanism is to provide a method for specific interrupts, such as NMI, double-fault, and machine-check, to always execute on a known good stack. In legacy mode, interrupts can use the task-switch mechanism to set up a known-good stack by accessing the interrupt service routine through a task gate located in the IDT. However, the legacy task-switch mechanism is not supported in IA-32e mode. The IST mechanism is part of the 64-bit mode task state segment (TSS) shown in Table 1-37. It provides up to seven IST pointers located in the TSS. The pointers are referenced by an interrupt-gate descriptor in the interrupt-descriptor table (IDT), as shown in Table 1-38. The gate descriptor contains a 3-bit IST index field that provides an offset into the IST section of the TSS.



If the IST index for an interrupt gate is not zero, the IST pointer corresponding to the index is loaded into the RSP when an interrupt occurs. The new SS selector is forced to null, and the SS selector's RPL field is set to the new CPL. The old SS, RSP, RFLAGS, CS, and RIP are pushed onto the new stack. Interrupt processing then proceeds as normal. If the IST index is zero, the modified legacy stack-switching mechanism described above is used.

1.6.10.6. Task Priority

64-bit extensions build on the 15 external interrupt-priority classes defined by the APIC specification. Priority class 1 is the lowest and 15 is the highest. How external interrupts are mapped into these priority classes is platform-dependent. Operating systems can use the TPR to temporarily block specific (generally low-priority) interrupts from interrupting a high-priority task. This is done by loading TPR with a value corresponding to the highest-priority interrupt that is to be blocked. For example, loading TPR with a value of 8 (01000B) blocks all interrupts with a priority of 8 or less, while allowing all interrupts with a priority of 9 or more to be recognized. Loading the TPR with 0 enables all external interrupts. Loading TPR with 15 (01111B) disables all external interrupts. The TPR is cleared to 0 on reset.

Software can read and write the TPR using a MOV CR8 instruction. The new priority level is established when the MOV CR8 instruction completes execution. Software does not need to force serialization after loading TPR.

Use of the MOV CRn instruction requires a privilege level of 0. Programs running at privilege level greater than 0 cannot read or write the TPR. An attempt to do so results in a general-protection exception, #GP(0).

The TPR is abstracted from the interrupt controller (IC), which prioritizes and manages external interrupt delivery to the processor. The IC can be an external device, such as an APIC or 8259. Typically, the IC provides a priority mechanism similar, if not identical to, the TPR. The IC, however, is considered implementation-dependent with the underlying priority mechanisms subject to change. The TPR, by contrast, is part of 64-bit architecture. Software can depend on this definition remaining unchanged. Table 1-41 shows the TPR. Only the low four bits are used. The remaining 60 bits are reserved and must be written with zeros, failure to do so results in a general-protection exception, #GP(0).

Table 1-41 Task Priority Register - CR8

63:4	3:0
Reserved	Task Priority Register (TPR)

1.6.10.7. CR8 Interactions with APIC

The first implementation of Intel EM64T includes a local advanced programmable interrupt controller (APIC) that is similar to the APIC used with many IA-32 processors. Some aspects of the local APIC affect the operation of the architecturally defined task priority register (CR8.TPR).

Notable CR8 and APIC interactions are:

- The processor powers up with the local APIC enabled.
- The APIC must be enabled for CR8 to function as the TPR. The interaction between the CR8 and the APIC is the following: Writes to CR8 are reflected into the APIC's Task Priority Register.
- APIC.TPR.7:4 = CR8.3:0, APIC.TPR.3:0 = 0. Reads of CR8 return APIC.TPR.7:4, zero is extended to 64 bits.
- There are no ordering mechanisms between direct updates of the APIC.TPR and CR8. It is expected that operating software will implement either direct APIC TPR updates or CR8 style TPR updates but will not mix them. Software can use a serializing instruction (e.g. CPUID) to serialize updates between MOV CR8 and stores to the APIC.

1.7. GENERAL RULES FOR 64-BIT MODE

In 64-bit mode, the following general rules apply to changes in instructions and their operands:

- If an instruction's operand size in legacy mode (16-bit or 32-bit) depends on the effective operand size (dependent on CS.D and prefix overrides); operand-size choices are extended in 64-bit mode from 16-bit and 32-bit to include 64 bits, or the operand size is fixed at a size that supports 64-bit operands. Such instructions are said to be 'promoted to 64 bits'. However, byte-operand opcodes of such instructions are not promoted.
- As stated above, the byte-operand opcodes of promoted instructions are not usually promoted. Those instructions
 continue to operate only on bytes.
- If an instruction's operand size is fixed in legacy mode (independent of CS.D and prefix overrides), that operand size is usually fixed at the same size in 64-bit mode. For example, CPUID operates on the same-size operands in legacy mode and 64-bit mode. There are some exceptions.
- Operations on 32-bit operands in 64-bit mode zero-extend the high 32 bits of 64-bit GPR destination registers.
- When operating on 8-bit (or 16-bit) operands in 64-bit mode, the high 56 (or 48) bits of 64-bit GPR destination registers are unchanged.
- When the operand size is 64 bits, the shift and rotate instructions use one additional bit (6 bits total) to specify shift-count or rotate-count, allowing 64-bit shifts and rotates.
- The maximum size of immediate operands remains 32 bits, except that 64-bit immediates can be moved to 64-bit GPRs. When the operand size is 64 bits, immediates are sign-extended to 64 bits prior to using them.
- Branch-address displacements remains 8 bits or 32 bits, but they are sign-extended to 64 bits prior to using them.
- When the processor makes transitions from 64-bit mode to compatibility or legacy modes, it does not preserve
 the upper 32 bits of the 64-bit GPRs. In compatibility or legacy mode, only the lower 32 bits of the GPRS are
 defined.

1.7.1. Other Guidelines

- In the initial implementation of Intel EM64T, an operand-size prefix (66H) is ignored when used in 64-bit mode with a near branch. In 64-bit mode, a near branch uses 32-bit displacement (the instruction pointer is advanced to a linear address that is the next sequential instruction offset by a 32 bit displacement, sign extended to 64-bit). Software must not rely on this behavior as future implementations may be different.
- In 64-bit mode, when the value of the source operand for BSR/BSF is 0, the upper 32 bits of the registers are cleared.



CHAPTER 2 INSTRUCTION SET REFERENCE (A-L)

Chapter 2, *Instruction Set Reference (A-L)*, starts an alphabetical discussion of IA-32 instructions (A-L). This discussion is continued in Chapter 3, *Intel*[®] *Extended Memory 64 Technology Software Developer's Guide, Volume 2*.

This chapter describes new instructions in Intel[®] EM64T and the existing IA-32 instruction set in IA-32e modes. This description includes general-purpose, x87 FPU, MMX[™], SSE, SSE2, SSE3 and system instructions. Instruction descriptions are arranged in alphabetical order.

For each instruction, forms are given for each operand combination. This includes the opcode, operands required, and a description. Also given for each instruction:

- a description of the instruction and its operands
- an operational description
- a description of the effect of the instructions on flags in the EFLAGS register
- a summary of the exceptions that can be generated

2.1. INTERPRETING THE INSTRUCTION REFERENCE PAGES

This section describes the presentation format of this chapter. It covers notational conventions and abbreviations. The following is an example of the format used in the remainder of the chapter.

2.1.1. The Instruction Summary Table

The presentation of each instruction opens with an "Instruction Summary Table" like the following example.

CMC—Complement Carry Flag (example section with explanation...)

Opcode	Instruction	64-bit Mode	Compat/Leg Mode	Description
F5	CMC	Valid	Valid	Complement carry flag

The information in each column is described below.

2.1.1.1. Opcode Column in the Instruction Summary Table

The "Opcode" column gives the object code produced for each form of the instruction. When possible, the codes are given as hexadecimal bytes, in the same order in which they appear in memory. Definitions of entries other than hexadecimal bytes are as follows:

- **REX.W** Indicates the use of a REX prefix that affects operand size or instruction semantics. The ordering of the REX prefix and other optional/mandatory instruction prefix is discussed in Chapter 1, Figure 1-1. The use of a REX prefix that results in promoting the legacy instruction behavior to 64-bits is not listed explicitly in the opcode column.
- /digit A digit 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 an extension 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.
- **cb, cw, cd, cp, co, ct** A 1-byte (cb), 2-byte (cw), 4-byte (cd), 6-byte (cp), 8-byte (co) or 10-byte (ct) value following the opcode that is used to specify a code offset and possibly a new value for the code segment register.



- **ib, iw, id, io** A 1-byte (ib), 2-byte (iw), 4-byte (id) or 8-byte (io) immediate operand to the instruction that follows the opcode, ModR/M bytes or scale-indexing bytes. The opcode determines if the operand is a signed value. All words and doublewords are given with the low-order byte first.
- **+rb**, **+rw**, **+rd**, **+ro** A register code, from 0 through 7, added to the hexadecimal byte given at the left of the plus sign to form a single opcode byte. The register codes are given in Table 2-1.
- +i A number used in floating-point instructions when one of the operands is ST(i) from the FPU register stack. The number i (which can range from 0 to 7) is added to the hexadecimal byte given at the left of the plus sign to form a single opcode byte.

Table 2-1 Register Codes

Table 2-1 negister Codes											
Register	REX.R	Reg Field	Register	REX.R	Reg Field	Register	REX.R	Reg Field	Register	REX.R	Reg Field
AL	0	0	AX	0	0	EAX	0	0	RAX	0	0
CL	0	1	СХ	0	1	ECX	0	1	RCX	0	1
DL	0	2	DX	0	2	EDX	0	2	RDX	0	2
BL	0	3	вх	0	3	EBX	0	3	RBX	0	3
AH	No REX prefix	4	SP	No REX prefix	4	ESP	No REX prefix	4	N/A	N/A	N/A
СН	No REX prefix	5	BP	No REX prefix	5	EBP	No REX prefix	5	N/A	N/A	N/A
DH	No REX prefix	6	SI	No REX prefix	6	ESI	No REX prefix	6	N/A	N/A	N/A
BH	No REX prefix	7	DI	No REX prefix	7	EDI	No REX prefix	7	N/A	N/A	N/A
SPL	Any REX Prefix	4	SP	0	4	ESP	0	4	RSP	0	4
BPL	Any REX Prefix	5	BP	0	5	EBP	0	5	RBP	0	5
SIL	Any REX Prefix	6	SI	0	6	ESI	0	6	RSI	0	6
DIL	Any REX Prefix	7	DI	0	7	EDI	0	7	RDI	0	7
R8L	1	0	R8W	1	0	R8D	1	0	R8	1	0
R9L	1	1	R9W	1	1	R9D	1	1	R9	1	1
R10L	1	2	R10W	1	2	R10D	1	2	R10	1	2
R11L	1	3	R11W	1	3	R11D	1	3	R11	1	3
R12L	1	4	R12W	1	4	R12D	1	4	R12	1	4
R13L	1	5	R13W	1	5	R13D	1	5	R13	1	5
R14L	1	6	R14W	1	6	R14D	1	6	R14	1	6
R15L	1	7	R15W	1	7	R15D	1	7	R15	1	7

2.1.1.2. Instruction Column in the Instruction Summary Table

The "Instruction" column gives the syntax of the instruction statement as it would appear in an ASM386 program. The following is a list of the symbols used to represent operands in the instruction statements:

- **rel8** A relative address in the range from 128 bytes before the end of the instruction to 127 bytes after the end of the instruction.
- rel16 and rel32 A relative address within the same code segment as the instruction assembled. The rel16 symbol applies to instructions with an operand-size attribute of 16 bits; the rel32 symbol applies to instructions with an operand-size attribute of 32 bits.

- **rel64** A relative address within the same code segment as the instruction assembled. The rel64 symbol applies to instructions with an operand-size attribute of 64 bits.
- **ptr16:16, ptr16:32 and ptr16:64** A far pointer, typically in a code segment different from that of the instruction. The notation *16:16* indicates that the value of the pointer has two parts. The value to the left of the colon is a 16-bit selector or value destined for the code segment register. The value to the right corresponds to the offset within the destination segment. The ptr16:16 symbol is used when the instruction's operand-size attribute is 16 bits; the ptr16:32 symbol is used when the operand-size attribute is 64 bits.
- r8 One of the byte general-purpose registers: AL, CL, DL, BL, AH, CH, DH, BH, R8l R15l, BPL, SPL, DIL and SIL
- r16 One of the word general-purpose registers: AX, CX, DX, BX, SP, BP, SI, DI, R8-R15.
- r32 One of the doubleword general-purpose registers: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, R8D -R15D.
- **r64** One of the quadword general-purpose registers: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8–R15 registers
- imm8 An immediate byte value. The imm8 symbol is a signed number between -128 and +127 inclusive. For instructions in which imm8 is combined with a word or doubleword operand, the immediate value is sign-extended to form a word or doubleword. The upper byte of the word is filled with the topmost bit of the immediate value.
- imm16 An immediate word value used for instructions whose operand-size attribute is 16 bits. This is a number between -32,768 and +32,767 inclusive.
- imm32 An immediate doubleword value used for instructions whose operand-size attribute is 32 bits. It allows the use of a number between +2,147,483,647 and -2,147,483,648 inclusive.
- **imm64** An immediate quadword value used for instructions whose operand-size attribute is 64 bits. It allows the use of a number between +9,223,372,036,854,775,807 and -9,223,372,036,854,775,808 inclusive.
- r/m8 A byte operand that is either the contents of a byte general-purpose register (AL, CL, DL, BL, AH, CH, DH, BH, R81 R151, BPL, SPL, DIL and SIL), or a byte from memory.
- r/m16 A word general-purpose register or memory operand used for instructions whose operand-size attribute is 16 bits. The word general-purpose registers are: AX, CX, DX, BX, SP, BP, SI, DI, R8-R15. The contents of memory are found at the address provided by the effective address computation.
- r/m32 A doubleword general-purpose register or memory operand used for instructions whose operand-size attribute is 32 bits. The doubleword general-purpose registers are: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI, R8D R15D. The contents of memory are found at the address provided by the effective address computation.
- **r/m64** A quadword general-purpose register or memory operand used for instructions whose operand-size attribute is 64 bits. The quadword general-purpose registers are: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8–R15. The contents of memory are found at the address provided by the effective address computation.
- **m** A 16-, 32- or 64-bit operand in memory.
- **m8** A byte operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. This nomenclature is used only with the string instructions and the XLAT instruction.
- **m16** A word operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. This nomenclature is used only with the string instructions.
- m32 A doubleword operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. This nomenclature is used only with the string instructions.
- **m64** A memory quadword operand in memory.
- m128 A memory double quadword operand in memory. This nomenclature is used only with the SSE and SSE2 instructions.

- m16:16, m16:32 & m16:64 —A memory operand containing a far pointer composed of two numbers. The
 number to the left of the colon corresponds to the pointer's segment selector. The number to the right corresponds
 to its offset.
- m16&32, m16&16, m32&32, m16&64 A memory operand consisting of data item pairs whose sizes are indicated on the left and the right side of the ampersand. All memory addressing modes are allowed. The m16&16 and m32&32 operands are used by the BOUND instruction to provide an operand containing an upper and lower bounds for array indices. The m16&32 operand is used by LIDT and LGDT to provide a word with which to load the limit field, and a doubleword with which to load the base field of the corresponding GDTR and IDTR registers. The m16&64 operand is used by LIDT and LGDT to provide a word with which to load the limit field, and a quadword with which to load the base field of the corresponding GDTR and IDTR registers.
- moffs8, moffs16, moffs32, moffs64 A simple memory variable (memory offset) of type byte, word, or doubleword used by some variants of the MOV instruction. The actual address is given by a simple offset relative to the segment base. No ModR/M byte is used in the instruction. The number shown with moffs indicates its size, which is determined by the address-size attribute of the instruction.
- Sreg A segment register. The segment register bit assignments are ES = 0, CS = 1, SS = 2, DS = 3, FS = 4, and GS = 5.
- m32fp, m64fp, m80fp A single-precision, double-precision, and double extended-precision (respectively) floating-point operand in memory. These symbols designate floating-point values that are used as operands for x87 FPU floating-point instructions.
- **m16int, m32int, m64int** A word, doubleword, and quadword integer (respectively) operand in memory. These symbols designate integers that are used as operands for x87 FPU integer instructions.
- ST or ST(0) The top element of the FPU register stack.
- **ST(i)** The ith element from the top of the FPU register stack ($i \leftarrow 0$ through 7).
- mm An MMX register. The 64-bit MMX registers are: MM0 through MM7.
- mm/m32 The low order 32 bits of an MMX register or a 32-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- mm/m64 An MMX register or a 64-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- xmm An XMM register. The 128-bit XMM registers are: XMM0 through XMM15.
- xmm/m32 An XMM register or a 32-bit memory operand. The 128-bit XMM registers are XMM0 through XMM15. The contents of memory are found at the address provided by the effective address computation.
- xmm/m64 An XMM register or a 64-bit memory operand. The 128-bit SIMD floating-point registers are XMM0 through XMM15. The contents of memory are found at the address provided by the effective address computation.
- xmm/m128 An XMM register or a 128-bit memory operand. The 128-bit XMM registers are XMM0 through XMM15. The contents of memory are found at the address provided by the effective address computation.

2.1.1.3. 64-bit Mode Column in the Instruction Summary Table

The "64-bit Mode" column indicates whether the opcode sequence is supported in 64-bit mode. The column uses the following notation:

- "Valid" supported
- "Inv." not supported
- "N.E." indicates an instruction syntax is not encodable; not applicable in 64-bit mode, but may represent part of a sequence of valid instructions.
- "N.P." indicates the REX prefix does not affect the legacy instruction in 64-bit mode.
- "N.I." indicates the opcode is treated as a new instruction in 64-bit mode.

• "N.S." — indicates an instruction syntax that requires an address override prefix in 64-bit mode and is not supported. Using an address override prefix in 64-bit mode may results in model-specific execution behavior.

2.1.1.4. Compatibility/Legacy Mode Column in the Instruction Summary Table

The "Compatibility/Legacy Mode" column provides information on the opcode sequence in either the compatibility mode or other legacy IA-32 modes. The column uses the following notation:

- "Valid" supported
- "Inv." not supported
- "N.E." indicates an instruction syntax that is not encodable; the opcode sequence is not applicable as an individual instruction in compatibility mode nor legacy IA-32 modes but may represent a valid sequence of legacy IA-32 instructions

2.1.1.5. Description Column in the Instruction Summary Table

The "Description" column briefly explains forms of the instruction.

2.1.2. Description Section

Each instruction is then described by number of information sections. The "Description" section describes the purpose of the instructions and the required operands in more detail. It also discusses the effect of the instruction on flags.

2.1.3. Operation Section

The "Operation" section contains an algorithm description (written in pseudo-code) of the instruction. The pseudo-code uses notation similar to the Algol or Pascal language. Algorithms are composed of the following elements:

- Comments are enclosed within the symbol pairs "(*" and "*)".
- Compound statements are enclosed in keywords, such as IF, THEN, ELSE, and FI for an if statement, DO and OD for a do statement, or CASE... OF and ESAC for a case statement.
- A register name implies the contents of the register. A register name enclosed in brackets implies the contents of the location whose address is contained in that register. For example, ES:[DI] indicates the contents of the location whose ES segment relative address is in register DI. [SI] indicates the contents of the address contained in register SI relative to the SI register's default segment (DS) or overridden segment.
- Parentheses around the "E" or "R" in a general-purpose register name, such as (E)SI, (R)SI, or the presence of a 64-bit register definition i.e. RSI indicates that an offset is read from the SI register if the current address-size attribute is 16 or is read from the ESI register if the address-size attribute is 32 or from the 64-bit RSI register if the address-size attribute is 64.
- Brackets are also used for memory operands, where they mean that the contents of the memory location is a segment-relative offset. For example, [SRC] indicates that the contents of the source operand is a segment-relative offset.
- A \leftarrow 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 of A is equal to B; otherwise it is FALSE.
- The expression "<< COUNT" and ">> COUNT" indicates that the destination operand should be shifted left or right, respectively, by the number of bits indicated by the count operand.

The following identifiers are used in the algorithmic descriptions:

• OperandSize and AddressSize — The OperandSize identifier represents the operand-size attribute of the instruction, which is either 16, 32 or 64-bits. The AddressSize identifier represents the address-size attribute,



which is either 16, 32 or 64-bits. For example, the following pseudo-code indicates that the operand-size attribute depends on the form of the MOV instruction used.

```
 \begin{split} \text{IF instruction} \leftarrow \text{MOVW} \\ \text{THEN OperandSize} \leftarrow \text{16}; \\ \text{ELSE} \\ \text{IF instruction} \leftarrow \text{MOVD} \\ \text{THEN OperandSize} \leftarrow \text{32}; \\ \text{ELSE} \\ \text{IF instruction} \leftarrow \text{MOVQ} \\ \text{THEN OperandSize} \leftarrow \text{64}; \\ \text{FI}; \\ \text{FI}; \\ \text{FI}; \end{split}
```

See "Operand-Size and Address-Size Attributes" in Chapter 3 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1* for general guidelines on how these attributes are determined.

- StackAddrSize Represents the stack address-size attribute associated with the instruction, which has a value of 16, 32 or 64-bits (see "Address-Size Attribute for Stack" in Chapter 6 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*).
- **SRC** Represents the source operand.
- **DEST** Represents the destination operand.

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 the operand-size attribute is 32, zero extending a byte value of -10 converts the byte from F6H to a doubleword value of 000000F6H. If the value passed to the ZeroExtend function 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 the operand-size attribute is 32, sign extending a byte containing the value –10 converts the byte from F6H to a doubleword value of FFFFFF6H. If the value passed to the SignExtend function 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 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **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 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- 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 (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **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 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **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 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- 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 (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **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 (00H); if it is greater than 65535, it is represented by the saturated value 65535 (FFFFH).

- **LowOrderWord(DEST * SRC)** Multiplies a word operand by a word operand and stores the least significant word of the doubleword result in the destination operand.
- **HighOrderWord(DEST * SRC)** Multiplies a word operand by a word operand and stores the most significant word of the doubleword result in the destination operand.
- **Push(value)** Pushes a value onto the stack. The number of bytes pushed is determined by the operand-size attribute of the instruction. See the "Operation" section in "PUSH—Push Word or Doubleword Onto the Stack" in this chapter for more information on the push operation.
- **Pop**() removes the value from the top of the stack and returns it. The statement EAX ← Pop(); assigns to EAX the 32-bit value from the top of the stack. Pop will return either a word, a doubleword or a quadword depending on the operand-size attribute. See the "Operation" section in Chapter 3, "POP—Pop a Value from the Stack" for more information on the pop operation.
- PopRegisterStack Marks the FPU ST(0) register as empty and increments the FPU register stack pointer (TOP) by 1.
- **Switch-Tasks** Performs a task switch.
- **Bit(BitBase, BitOffset)** Returns the value of a bit within a bit string, which is a sequence of bits in memory or a register. Bits are numbered from low-order to high-order within registers and within memory bytes. If the base operand is a register, the offset can be in the range 0..64 depending on register size. This offset addresses a bit within the indicated register. An example, the function Bit[EAX, 21] is illustrated in Figure 2-1.

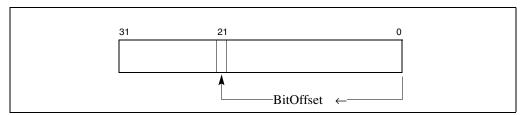


Figure 2-1. Bit Offset for BIT[EAX,21]

If BitBase is a memory address, BitOffset can range from –2 GBits to 2 GBits. The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)), where DIV is signed division with rounding towards negative infinity, and MOD returns a positive number. This operation is illustrated in Figure 2-2.

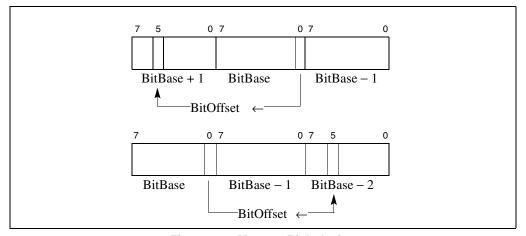


Figure 2-2. Memory Bit Indexing

Since instruction behavior for legacy modes can be found in IA-32 Intel Architecture Software Developer's Manual, Volume 2A and 2B, Instruction References. They are not repeated in this document.

2.1.3.1. IA-32e Mode Operation

The section "IA-32e Mode Operation" summarizes instruction behavior in 64-bit mode that differs from legacy IA-32 modes.

2.1.4. Flags Affected

The "Flags Affected" section lists the flags in the EFLAGS register that are affected by the instruction. When a flag is cleared, it is equal to 0; when it is set, it is equal to 1. The arithmetic and logical instructions usually assign values to the status flags in a uniform manner (see Appendix A, EFLAGS Cross-Reference, in the IA-32 Intel Architecture Software Developer's Manual, Volume 1). Non-conventional assignments are described in the "Operation" section. The values of flags listed as **undefined** may be changed by the instruction in an indeterminate manner. Flags that are not listed are unchanged by the instruction.

2.1.5. FPU Flags Affected

The floating-point instructions have an "FPU Flags Affected" section that describes how each instruction can affect the four condition code flags of the FPU status word.

2.1.6. Protected Mode Exceptions

The "Protected Mode Exceptions" section lists the exceptions that can occur when the instruction is executed in protected mode and the reasons for the exceptions. Each exception is given a mnemonic that consists of a pound sign (#) followed by two letters and an optional error code in parentheses. For example, #GP(0) denotes a general protection exception with an error code of 0. Table 2-2 associates each two-letter mnemonic with the corresponding interrupt vector number and exception name. See Chapter 5, *Interrupt and Exception Handling*, in the *IA-32 Intel Architecture Software Developer's Manual, Volume 3*, for a detailed description of the exceptions.

Application programmers should consult the documentation provided with their operating systems to determine the actions taken when exceptions occur.

Vector No.	Name	Source	Protected Mode	Real Address Mode	Virtual 8086 Mode	64-Bit Mode
0	#DE—Divide Error	DIV and IDIV instructions.	Yes	Yes	Yes	Yes
1	#DB—Debug	Any code or data reference.	Yes	Yes	Yes	Yes
3	#BP—Breakpoint	INT 3 instruction.	Yes	Yes	Yes	Yes
4	#OF—Overflow	INTO instruction.	Yes	Yes	Yes	Yes
5	#BR—BOUND Range Exceeded	BOUND instruction.	Yes	Yes	Yes	Reserved
6	#UD—Invalid Opcode (Undefined Opcode)	UD2 instruction or reserved opcode.	Yes	Yes	Yes	Yes
7	#NM—Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes	Yes
8	#DF—Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.	Yes	Yes	Yes	Yes
10	#TS—Invalid TSS	Task switch or TSS access.	Yes	Reserved	Yes	Yes
11	#NP—Segment Not Present	Loading segment registers or accessing system segments.	Yes	Reserved	Yes	Yes

Table 2-2 Interrupt Vectors

Table 2-2 Interrupt Vectors (Contd.)

Vector No.	Name	Source	Protected Mode	Real Address Mode	Virtual 8086 Mode	64-Bit Mode		
12	#SS—Stack Segment Fault	Stack operations and SS register loads.	Yes	Yes	Yes	Yes		
13	#GP—General Protection*	Any memory reference and other protection checks.	Yes	Yes	Yes	Yes		
14	#PF—Page Fault	Any memory reference.	Yes	Reserved	Yes	Yes		
16	#MF—Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes	Yes		
17	#AC—Alignment Check	Any data reference in memory.	Yes	Reserved	Yes	Yes		
18	#MC—Machine Check	Model dependent machine check errors.	Yes	Yes	Yes	Yes		
19	#XF—SIMD Floating-Point Numeric Error SSE and SSE2 floating-point instructions.		Yes	Yes	Yes	Yes		
* In the	In the real-address mode, vector 13 is the segment overrun exception.							

2.1.7. Real-Address Mode Exceptions

The "Real-Address Mode Exceptions" section lists the exceptions that can occur when the instruction is executed in real-address mode.

2.1.8. Virtual-8086 Mode Exceptions

The "Virtual-8086 Mode Exceptions" section lists the exceptions that can occur when the instruction is executed in virtual-8086 mode.

2.1.9. Floating-Point Exceptions

The "Floating-Point Exceptions" section lists exceptions that can occur when an x87 FPU floating-point instruction is executed. All of these exception conditions result in a floating-point error exception (#MF, vector number 16) being generated. Table 2-3 associates a one- or two-letter mnemonic with the corresponding exception name. See Chapter 8 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1* for a detailed description of these exceptions.

Table 2-3 Exception Names

Mnemonic	Name	Source
#IS #IA	Floating-point invalid operation: - Stack overflow or underflow - Invalid arithmetic operation	- x87 FPU stack overflow or underflow - Invalid FPU arithmetic operation
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result (precision)	Inexact result (precision)



2.1.10. SIMD Floating-Point Exceptions

The "SIMD Floating-Point Exceptions" section lists exceptions that can occur when an SSE and SSE2 floating-point instruction is executed. All of these exception conditions result in a SIMD floating-point error exception (#XF, vector number 19) being generated. Table 2-4 associates a one-letter mnemonic with the corresponding exception name. For a detailed description of these exceptions, refer to "SSE and SSE2 Exceptions" in Chapter 11 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*.

Table 2-4 Floating Point Exception Names

Mnemonic	Name	Source	
#1	Floating-point invalid operation	Invalid arithmetic operation or source operand	
#Z	Floating-point divide-by-zero	Divide-by-zero	
#D	Floating-point denormal operand	Source operand that is a denormal number	
#O	Floating-point numeric overflow	orflow Overflow in result	
3,1		Underflow in result	
		Inexact result (precision)	

2.2. INSTRUCTION REFERENCE

The remainder of this chapter provides detailed descriptions of each of the IA-32 instructions.

AAA—ASCII Adjust After Addition

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
37	AAA	Inv.	Valid	ASCII adjust AL after addition	

Flags Affected

The AF and CF flags are set to 1 if the adjustment results in a decimal carry; otherwise they are cleared to 0. The OF, SF, ZF, and PF flags are undefined.

IA-32e Mode Operation

Instruction is invalid in 64-bit mode.

Protected Mode Exceptions

None

Real-Address Mode Exceptions

None

Virtual-8086 Mode Exceptions

None

Compatibility Mode Exceptions

None

64-Bit Mode Exceptions



AAD—ASCII Adjust AX Before Division

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D5 0A	AAD	Inv.	Valid	ASCII adjust AX before division
D5 ib	(No mnemonic)	Inv.	Valid	Adjust AX before division to number base imm8

Flags Affected

The SF, ZF, and PF flags are set according to the resulting binary value in the AL register; the OF, AF, and CF flags are undefined.

IA-32e Mode Operation

Instruction is invalid in 64-bit mode.

Protected Mode Exceptions

None

Real-Address Mode Exceptions

None

Virtual-8086 Mode Exceptions

None

Compatibility Mode Exceptions

None

64-Bit Mode Exceptions

AAM—ASCII Adjust AX After Multiply

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D4 0A	AAM	Inv.	Valid	ASCII adjust AX after multiply
D4 ib	(No mnemonic)	Inv.	Valid	Adjust AX after multiply to number base imm8

Flags Affected

The SF, ZF, and PF flags are set according to the resulting binary value in the AL register. The OF, AF, and CF flags are undefined.

IA-32e Mode Operation

Instruction is invalid in 64-bit mode.

Protected Mode Exceptions

#DE If an immediate value of 0 is used.

Real-Address Mode Exceptions

#DE If an immediate value of 0 is used.

Virtual-8086 Mode Exceptions

#DE If an immediate value of 0 is used.

Compatibility Mode Exceptions

#DE If an immediate value of 0 is used.

64-Bit Mode Exceptions



AAS—ASCII Adjust AL After Subtraction

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
3F	AAS	Inv.	Valid	ASCII adjust AL after subtraction

Flags Affected

The AF and CF flags are set to 1 if there is a decimal borrow; otherwise, they are cleared to 0. The OF, SF, ZF, and PF flags are undefined.

IA-32e Mode Operation

Instruction is invalid in 64-bit mode.

Protected Mode Exceptions

None

Real-Address Mode Exceptions

None

Virtual-8086 Mode Exceptions

None

Compatibility Mode Exceptions

None

64-Bit Mode Exceptions

ADC—Add with Carry

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
14 <i>ib</i>	ADC AL,imm8	Valid	Valid	Add with carry imm8 to AL
15 <i>iw</i>	ADC AX,imm16	Valid	Valid	Add with carry imm16 to AX
15 <i>id</i>	ADC EAX,imm32	Valid	Valid	Add with carry imm32 to EAX
REX.W + 15 id	ADC RAX,imm32	Valid	N.E.	Add with carry <i>imm32 sign extended to 64-bits</i> to RAX
80 /2 ib	ADC r/m8,imm8	Valid	Valid	Add with carry imm8 to r/m8
REX + 80 /2 ib	ADC r/m8*,imm8	Valid	N.E.	Add with carry imm8 to r/m8
81 /2 <i>iw</i>	ADC r/m16,imm16	Valid	Valid	Add with carry imm16 to r/m16
81 /2 id	ADC r/m32,imm32	Valid	Valid	Add with CF imm32 to r/m32
REX.W + 81 /2 id	ADC r/m64,imm32	Valid	N.E.	Add with CF <i>imm32</i> sign extended to 64-bits to r/m64
83 /2 ib	ADC r/m16,imm8	Valid	Valid	Add with CF sign-extended imm8 to r/m16
83 /2 ib	ADC r/m32,imm8	Valid	Valid	Add with CF sign-extended imm8 into r/m32
REX.W + 83 /2 ib	ADC r/m64,imm8	Valid	N.E.	Add with CF sign-extended imm8 into r/m64
10 / <i>r</i>	ADC r/m8,r8	Valid	Valid	Add with carry byte register to r/m8
REX + 10 /r	ADC r/m8*,r8*	Valid	N.E.	Add with carry byte register to r/m64
11 / <i>r</i>	ADC r/m16,r16	Valid	Valid	Add with carry r16 to r/m16
11 / <i>r</i>	ADC r/m32,r32	Valid	Valid	Add with CF r32 to r/m32
REX.W + 11 /r	ADC r/m64,r64	Valid	N.E.	Add with CF r64 to r/m64
12 /r	ADC r8,r/m8	Valid	Valid	Add with carry r/m8 to byte register
REX + 12 /r	ADC r8*,r/m8*	Valid	N.E.	Add with carry r/m64 to byte register
13 /r	ADC r16,r/m16	Valid	Valid	Add with carry r/m16 to r16
13 /r	ADC r32,r/m32	Valid	Valid	Add with CF r/m32 to r32
REX.W + 13 /r	ADC r64,r/m64	Valid	N.E.	Add with CF r/m64 to r64

^{*} In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination is located in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

ADD—Add

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
04 <i>ib</i>	ADD AL,imm8	Valid	Valid	Add imm8 to AL
05 <i>iw</i>	ADD AX,imm16	Valid	Valid	Add imm16 to AX
05 id	ADD EAX,imm32	Valid	Valid	Add imm32 to EAX
REX.W + 05 id	ADD RAX,imm32	Valid	N.E.	Add imm32 sign-extended to 64-bits to RAX
80 /0 <i>ib</i>	ADD r/m8,imm8	Valid	Valid	Add imm8 to r/m8
REX + 80 /0 ib	ADD r/m8*,imm8	Valid	N.E.	Add sign-extended imm8 to r/m64
81 /0 <i>iw</i>	ADD r/m16,imm16	Valid	Valid	Add imm16 to r/m16
81 /0 <i>id</i>	ADD r/m32,imm32	Valid	Valid	Add imm32 to r/m32
REX.W + 81 /0 id	ADD r/m64,imm32	Valid	N.E.	Add imm32 sign-extended to 64-bits to r/m64
83 /0 <i>ib</i>	ADD r/m16,imm8	Valid	Valid	Add sign-extended imm8 to r/m16
83 /0 <i>ib</i>	ADD r/m32,imm8	Valid	Valid	Add sign-extended imm8 to r/m32
REX.W + 83 /0 ib	ADD r/m64,imm8	Valid	N.E.	Add sign-extended imm8 to r/m64
00 /r	ADD r/m8,r8	Valid	Valid	Add r8 to r/m8
REX + 00 /r	ADD r/m8*,r8*	Valid	N.E.	Add r8 to r/m8
01 / <i>r</i>	ADD r/m16,r16	Valid	Valid	Add r16 to r/m16
01 /r	ADD r/m32,r32	Valid	Valid	Add r32 to r/m32
REX.W + 01 /r	ADD r/m64,r64	Valid	N.E.	Add r64 to r/m64
02 /r	ADD r8,r/m8	Valid	Valid	Add r/m8 to r8
REX + 02 /r	ADD r8*,r/m8*	Valid	N.E.	Add r/m8 to r8
03 /r	ADD r16,r/m16	Valid	Valid	Add r/m16 to r16
03 /r	ADD r32,r/m32	Valid	Valid	Add r/m32 to r32
REX.W + 03 /r	ADD r64,r/m64	Valid	N.E.	Add r/m64 to r64

^{*} In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination is located in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

ADDPD—Add Packed Double-Precision Floating-Point Values

Opcode	e Instruction	64-Bit Mod	de Compat/Le Mode	g Description
66 0F 5	8 /r ADDPD xmm1, x	mm2/m128 Valid	Valid	Add packed double-precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

ADDPS—Add Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 58 /r	ADDPS xmm1, xmm2/m128	Valid	Valid	Add packed single-precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

ADDSD—Add Scalar Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 58 /r	ADDSD xmm1, xmm2/m64	Valid	Valid	Add the low double-precision floating-point value from <i>xmm2/m64</i> to <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

ADDSS—Add Scalar Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 58 /r	ADDSS xmm1, xmm2/m32	Valid	Valid	Add the low single-precision floating-point value from xmm2/m32 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

ADDSUBPD—Packed Double-Precision Floating-Point Add/Subtract

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F D0 /r	ADDSUBPD xmm1, xmm2/ m128	Valid	Valid	Add/subtract double-precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

ADDSUBPS—Packed Single-Precision Floating-Point Add/Subtract

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F D0 /r	ADDSUBPS xmm1, xmm2/ m128	Valid	Valid	Add/subtract single-precision floating-point values from <i>xmm2/m128</i> to <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

AND—Logical AND

Opcode	Instruction	64-Bit Mode	Comp/Leg'y Mode	Description
24 ib	AND AL,imm8	Valid	Valid	AL AND imm8
25 <i>iw</i>	AND AX,imm16	Valid	Valid	AX AND i <i>mm16</i>
25 id	AND EAX,imm32	Valid	Valid	EAX AND imm32
REX.W + 25 id	AND RAX,imm32	Valid	N.E.	RAX AND imm32 sign-extended to 64-bits
80 /4 <i>ib</i>	AND r/m8,imm8	Valid	Valid	r/m8 AND imm8
REX + 80 /4 ib	AND r/m8*,imm8	Valid	N.E.	r/m64 AND imm8 (sign-extended)
81 /4 <i>iw</i>	AND r/m16,imm16	Valid	Valid	r/m16 AND imm16
81 /4 <i>id</i>	AND r/m32,imm32	Valid	Valid	r/m32 AND imm32
REX.W + 81 /4 id	AND r/m64,imm32	Valid	N.E.	r/m64 AND imm32 sign extended to 64-bits
83 /4 <i>ib</i>	AND r/m16,imm8	Valid	Valid	r/m16 AND imm8 (sign-extended)
83 /4 <i>ib</i>	AND r/m32,imm8	Valid	Valid	r/m32 AND imm8 (sign-extended)
REX.W + 83 /4 ib	AND r/m64,imm8	Valid	N.E.	r/m64 AND imm8 (sign-extended)
20 /r	AND r/m8,r8	Valid	Valid	r/m8 AND r8
REX + 20 /r	AND r/m8*,r8*	Valid	N.E.	r/m64 AND r8 (sign-extended)
21 /r	AND r/m16,r16	Valid	Valid	r/m16 AND r16
21 /r	AND r/m32,r32	Valid	Valid	r/m32 AND r32
REX.W + 21 /r	AND r/m64,r64	Valid	N.E.	r/m64 AND r32
22 /r	AND r8,r/m8	Valid	Valid	<i>r8</i> AND <i>r/m8</i>
REX + 22 /r	AND r8*,r/m8*	Valid	N.E.	r/m64 AND r8 (sign-extended)
23 /r	AND r16,r/m16	Valid	Valid	r16 AND r/m16
23 /r	AND r32,r/m32	Valid	Valid	r32 AND r/m32
REX.W + 23 /r	AND r64,r/m64	Valid	N.E.	r64 AND r/m64

^{*} In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

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.

IA-32e Mode Operation

Promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination operand points to a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

64-Bit Mode Exceptions

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

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

#GP(0) If the destination is located in a non-writable segment.

If the memory address is in a non-canonical form.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

ANDPD—Bitwise Logical AND of Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 54 /r	ANDPD xmm1, xmm2/m128	Valid	Valid	Bitwise logical AND of xmm2/m128 and xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.



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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

ANDPS—Bitwise Logical AND of Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 54 /r	ANDPS xmm1, xmm2/m128	Valid	Valid	Bitwise logical AND of xmm2/m128 and xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.



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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

ANDNPD—Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 55 /r	ANDNPD xmm1, xmm2/m128	Valid	Valid	Bitwise logical AND NOT of xmm2/m128 and xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.



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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

ANDNPS—Bitwise Logical AND NOT of Packed Single-Precision Floating-Point Values

Opco	de Instruction	64-Bit	Mode Comp Mode	npat/Leg Description e
0F 55	7 ANDNPS xm	m1, xmm2/m128 Valid	Valid	Bitwise logical AND NOT of xmm2/m128 and xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.



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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

ARPL—Adjust RPL Field of Segment Selector

Opcod	e Instruction	64-Bit Mode	Compat/Leg Mode	Description
63 /r	ARPL r/m16,r16	N. I.	Valid	Adjust RPL of <i>r/m16</i> to not less than RPL of <i>r16</i>

IA-32e Mode Operation

Instruction is invalid in 64-bit mode. The opcode is treated as part of the instruction MOVSXD.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#UD The ARPL instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The ARPL instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

None



BOUND—Check Array Index Against Bounds

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
62 /r	BOUND r16, m16&16	Inv.	Valid	Check if <i>r16</i> (array index) is within bounds specified by <i>m16&16</i>
62 /r	BOUND <i>r32, m32&32</i>	Inv.	Valid	Check if <i>r32</i> (array index) is within bounds specified by <i>m16&16</i>

Flags Affected

None.

IA-32e Mode Operation

Instruction is invalid in 64-bit mode.

Protected Mode Exceptions

#BR If the bounds test fails.

#UD If second operand is not a memory location.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#BR If the bounds test fails.

#UD If second operand is not a memory location.

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#BR If the bounds test fails.

#UD If second operand is not a memory location.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

#UD

If in 64-bit mode.

BSF—Bit Scan Forward

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F BC	BSF r16,r/m16	Valid	Valid	Bit scan forward on r/m16
0F BC	BSF r32,r/m32	Valid	Valid	Bit scan forward on r/m32
REX.W + 0F BC	BSF r64,r/m64	Valid	N.E.	Bit scan forward on r/m64

Flags Affected

The ZF flag is set to 1 if all the source operand is 0; otherwise, the ZF flag is cleared. The CF, OF, SF, AF, and PF, flags are undefined.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

BSR—Bit Scan Reverse

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F BD	BSR r16,r/m16	Valid	Valid	Bit scan reverse on r/m16
0F BD	BSR r32,r/m32	Valid	Valid	Bit scan reverse on r/m32
REX.W + 0F BD	BSR r64,r/m64	Valid	N.E.	Bit scan reverse on r/m64

Flags Affected

The ZF flag is set to 1 if all the source operand is 0; otherwise, the ZF flag is cleared. The CF, OF, SF, AF, and PF, flags are undefined.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



BSWAP—Byte Swap

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F C8+rd	BSWAP r32	Valid	Valid	Reverses the byte order of a 32-bit register.
REX.W + 0F C8+rd	BSWAP r64	Valid	N.E.	Reverses the byte order of a 64-bit register.

Flags Affected

None.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

None

Real-Address Mode Exceptions

None

Virtual-8086 Mode Exceptions

None

Compatibility Mode Exceptions

None

64-Bit Mode Exceptions

None

BT—Bit Test

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F A3	BT r/m16,r16	Valid	Valid	Store selected bit in CF flag
0F A3	BT r/m32,r32	Valid	Valid	Store selected bit in CF flag
REX.W + 0F A3	BT r/m64,r64	Valid	N.E.	Store selected bit in CF flag
0F BA /4 <i>ib</i>	BT <i>r/m16,imm8</i>	Valid	Valid	Store selected bit in CF flag
0F BA /4 <i>ib</i>	BT <i>r/m32,imm8</i>	Valid	Valid	Store selected bit in CF flag
REX.W + 0F BA /4 <i>ib</i>	BT r/m64,imm8	Valid	N.E.	Store selected bit in CF flag

Flags Affected

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

IA-32e Mode Operation

Instruction is promoted to 64-bits. Default Operation Size is 32 bits. Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.



BTC—Bit Test and Complement

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F BB	BTC r/m16,r16	Valid	Valid	Store selected bit in CF flag and complement
0F BB	BTC r/m32,r32	Valid	Valid	Store selected bit in CF flag and complement
REX.W + 0F BB	BTC r/m64,r64	Valid	N.E.	Store selected bit in CF flag and complement
0F BA /7 <i>ib</i>	BTC r/m16,imm8	Valid	Valid	Store selected bit in CF flag and complement
0F BA /7 <i>ib</i>	BTC r/m32,imm8	Valid	Valid	Store selected bit in CF flag and complement
REX.W + 0F BA /7 <i>ib</i>	BTC r/m64,imm8	Valid	N.E.	Store selected bit in CF flag and complement

Flags Affected

The CF flag contains the value of the selected bit before it is complemented. The OF, SF, ZF, AF, and PF flags are undefined.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination operand points to a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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



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

#PF(fault-code) If a page fault occurs.

If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3. #AC(0)



BTR—Bit Test and Reset

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F B3	BTR r/m16,r16	Valid	Valid	Store selected bit in CF flag and clear
0F B3	BTR r/m32,r32	Valid	Valid	Store selected bit in CF flag and clear
REX.W + 0F B3	BTR r/m64,r64	Valid	N.E.	Store selected bit in CF flag and clear
0F BA /6 <i>ib</i>	BTR r/m16,imm8	Valid	Valid	Store selected bit in CF flag and clear
0F BA /6 <i>ib</i>	BTR r/m32,imm8	Valid	Valid	Store selected bit in CF flag and clear
REX.W + 0F BA /6 <i>ib</i>	BTR r/m64,imm8	Valid	N.E.	Store selected bit in CF flag and clear

Flags Affected

The CF flag contains the value of the selected bit before it is cleared. The OF, SF, ZF, AF, and PF flags are undefined.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination operand points to a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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



#PF(fault-code) If a page fault occurs.

If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3. #AC(0)

BTS—Bit Test and Set

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F AB	BTS r/m16,r16	Valid	Valid	Store selected bit in CF flag and set
0F AB	BTS r/m32,r32	Valid	Valid	Store selected bit in CF flag and set
REX.W + 0F AB	BTS r/m64,r64	Valid	N.E.	Store selected bit in CF flag and set
0F BA /5 <i>ib</i>	BTS r/m16,imm8	Valid	Valid	Store selected bit in CF flag and set
0F BA /5 <i>ib</i>	BTS r/m32,imm8	Valid	Valid	Store selected bit in CF flag and set
REX.W + 0F BA /5 <i>ib</i>	BTS r/m64,imm8	Valid	N.E.	Store selected bit in CF flag and set

Flags Affected

The CF flag contains the value of the selected bit before it is set. The OF, SF, ZF, AF, and PF flags are undefined.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination operand points to a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CALL—Call Procedure

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
E8 <i>cw</i>	CALL rel16	N.S.	Valid	Call near, relative, displacement relative to next instruction. Not supported in 64-bit mode.
E8 <i>cd</i>	CALL rel32	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.
FF /2	CALL r/m16	N.E.	Valid	Call near, absolute indirect, address given in r/m16.
FF /2	CALL r/m32	N.E.	Valid	Call near, absolute indirect, address given in r/m32. 32-bit displacement sign extended to 64-bits in 64-bit mode
FF /2	CALL r/m64	Valid	N.E.	Call near, absolute indirect, address given in r/m64.
9A <i>cd</i>	CALL ptr16:16	Inv.	Valid	Call far, absolute, address given in operand
9A <i>cp</i>	CALL ptr16:32	Inv.	Valid	Call far, absolute, address given in operand
FF /3	CALL <i>m16:16</i>	Valid	Valid	Call far, absolute indirect, address given in <i>m16:16</i> In 32-bit mode if selector points to a gate then RIP = 32-bit zero extended displacement taken from gate else RIP = zero extended 16-bit offset from far pointer referenced in the instruction.
FF /3	CALL <i>m16:32</i>	Valid	Valid	In 64-bit mode of operation If selector points to a gate then RIP = 64-bit displacement taken from gate else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.
FF /3	CALL <i>m16:64</i>	Valid	N.E.	In 64-bit mode of operation If selector points to a gate then RIP = 64-bit displacement taken from gate else RIP = 64-bit offset from far pointer from far pointer referenced in the instruction.

Flags Affected

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

64-bit Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32-bits.

32 bit displacements are sign-extended to 64 bits.

Protected Mode Exceptions

#GP(0) If target offset in destination operand is beyond the new code segment limit.

If the segment selector in the destination operand is null.

If the code segment selector in the gate is null.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#GP(selector) If code segment or gate or TSS selector index is outside descriptor table limits.

If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or

task state segment.



If the DPL for a nonconforming-code segment is not equal to the CPL or the RPL for the segment's segment selector is greater than the CPL.

If the DPL for a conforming-code segment is greater than the CPL.

If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector.

If the segment descriptor for a segment selector from a call gate does not indicate it is a code segment.

If the segment selector from a call gate is beyond the descriptor table limits.

If the DPL for a code-segment obtained from a call gate is greater than the CPL.

If the segment selector for a TSS has its local/global bit set for local.

If a TSS segment descriptor specifies that the TSS is busy or not available.

#SS(0) If pushing the return address, parameters, or stack segment pointer onto the stack exceeds

the bounds of the stack segment, when no stack switch occurs.

If a memory operand effective address is outside the SS segment limit.

#SS(selector) If pushing the return address, parameters, or stack segment pointer onto the stack exceeds

the bounds of the stack segment, when a stack switch occurs.

If the SS register is being loaded as part of a stack switch and the segment pointed to is

marked not present.

If stack segment does not have room for the return address, parameters, or stack segment

pointer, when stack switch occurs.

#NP(selector) If a code segment, data segment, stack segment, call gate, task gate, or TSS is not present.

#TS(selector) If the new stack segment selector and ESP are beyond the end of the TSS.

If the new stack segment selector is null.

If the RPL of the new stack segment selector in the TSS is not equal to the DPL of the code

segment being accessed.

If DPL of the stack segment descriptor for the new stack segment is not equal to the DPL

of the code segment descriptor.

If the new stack segment is not a writable data segment.

If segment-selector index for stack segment is outside descriptor table limits.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the target offset is beyond the code segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the target offset is beyond the code segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as 64-bit mode exceptions.

64-Bit Mode Exceptions

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

If target offset in destination operand is non-canonical.

If target offset in destination operand is beyond the new code segment limit.

If the segment selector in the destination operand is null.

If the code segment selector in the 64-bit gate is null.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.

#GP(selector) If code segment or 64-bit call gate is outside descriptor table limits.

If code segment or 64-bit call gate overlaps non-canonical space.

If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, 64-bit call gate.

If the segment descriptor pointed to by the segment selector in the destination operand is a code segment, and has both the D-bit and the L-bit set.

If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL.

If the DPL for a conforming-code segment is greater than the CPL.

If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate.

If the upper type field of a 64-bit call gate is not 0x0.

If the segment selector from a 64-bit call gate is beyond the descriptor table limits.

If the DPL for a code-segment obtained from a 64-bit call gate is greater than the CPL.

If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear.

If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment.

it is a code segment.

If pushing the return offset or CS selector onto the stack exceeds the bounds of the stack

segment, when no stack switch occurs.

If a memory operand effective address is outside the SS segment limit.

If the stack address is in a non-canonical form.

#SS(selector) If pushing the old values of SS selector, stack pointer, EFLAGS, CS selector, offset, or error

code onto the stack violates the canonical boundary, when a stack switch occurs.

#NP(selector) If a code segment or 64-bit call gate is not present.

#TS(selector) If the load of the new RSP exceeds the limit of the TSS.

#UD (64-bit mode only) If a far call is direct to an absolute address in memory.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

#SS(0)

CBW/CWDE/CDQE—Convert Byte to Word/Convert Word to Doubleword/Convert Doubleword to Quadword

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
98	CBW	Valid	Valid	AX ← sign-extend of AL
98	CWDE	Valid	Valid	EAX ← sign-extend of AX
REX.W + 98	CDQE	Valid	N.E.	RAX ← sign-extend of EAX

Flags Affected

None.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is size of destination register.

Exceptions (All Operating Modes)

None.

CDQ—Convert Double to Quad

See entry for CWD/CDQ — Convert Word to Doubleword/Convert Doubleword to Quadword.



CLC—Clear Carry Flag

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F8	CLC	Valid	Valid	Clear CF flag

Flags Affected

The CF flag is cleared to 0. The OF, ZF, SF, AF, and PF flags are unaffected.

IA-32e Mode Operation

Same as legacy mode.

Exceptions (All Operating Modes)

None.

CLD—Clear Direction Flag

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
FC	CLD	Valid	Valid	Clear DF flag

Flags Affected

The DF flag is cleared to 0. The CF, OF, ZF, SF, AF, and PF flags are unaffected.

IA-32e Mode Operation

Same as legacy mode.

Exceptions (All Operating Modes)

None.



CLFLUSH—Flush Cache Line

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F AE/7	CLFLUSH m8	Valid	Valid	Flushes cache line containing m8.

IA-32e Mode Operation

Same as Legacy.

Intel® C/C++ Compiler Intrinsic Equivalents

CLFLUSH void_mm_clflush(void const *p)

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

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

#UD If CPUID feature flag CLFSH is 0.

Real-Address Mode Exceptions

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

#UD If CPUID feature flag CLFSH is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

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

#UD If CPUID feature flag CLFSH is 0.

CLI—Clear Interrupt Flag

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
FA	CLI	Valid	Valid	Clear interrupt flag; interrupts disabled when interrupt flag cleared

Flags Affected

The IF is cleared to 0 if the CPL is equal to or less than the IOPL; otherwise, it is not affected. The other flags in the EFLAGS register are unaffected.

IA-32e Mode Operation

Same as legacy mode.

Protected Mode Exceptions

#GP(0) If the CPL is greater (has less privilege) than the IOPL of the current program or procedure.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

#GP(0) If the CPL is greater (has less privilege) than the IOPL of the current program or procedure.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

#GP(0) If the CPL is greater (has less privilege) than the IOPL of the current program or procedure.



CLTS—Clear Task-Switched Flag in CR0

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 06	CLTS	Valid	Valid	Clears TS flag in CR0

Flags Affected

The TS flag in CR0 register is cleared.

IA-32e Mode Operation

Same as legacy mode.

Protected Mode Exceptions

#GP(0) If the CPL is greater than 0.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

#GP(0) If the CPL is greater than 0.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

#GP(0) If the CPL is greater than 0.

CMC—Complement Carry Flag

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F5	CMC	Valid	Valid	Complement CF flag

Flags Affected

The CF flag contains the complement of its original value. The OF, ZF, SF, AF, and PF flags are unaffected.

IA-32e Mode Operation

Same as legacy mode.

Exceptions (All Operating Modes)

None.



CMOVcc—Conditional Move

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 47 /r	CMOVA r16, r/m16	Valid	Valid	Move if above (CF=0 and ZF=0)
0F 47 /r	CMOVA r32, r/m32	Valid	Valid	Move if above (CF=0 and ZF=0)
REX.W + 0F 47 /r	CMOVA r64, r/m64	Valid	N.E.	Move if above (CF=0 and ZF=0)
0F 43 /r	CMOVAE <i>r16, r/m16</i>	Valid	Valid	Move if above or equal (CF=0)
0F 43 /r	CMOVAE <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if above or equal (CF=0)
REX.W + 0F 43 /r	CMOVAE r64, r/m64	Valid	N.E.	Move if above or equal (CF=0)
0F 42 /r	CMOVB r16, r/m16	Valid	Valid	Move if below (CF=1)
0F 42 /r	CMOVB r32, r/m32	Valid	Valid	Move if below (CF=1)
REX.W + 0F 42 /r	CMOVB r64, r/m64	Valid	N.E.	Move if below (CF=1)
0F 46 /r	CMOVBE r16, r/m16	Valid	Valid	Move if below or equal (CF=1 or ZF=1)
0F 46 /r	CMOVBE r32, r/m32	Valid	Valid	Move if below or equal (CF=1 or ZF=1)
REX.W + 0F 46 /r	CMOVBE r64, r/m64	Valid	N.E.	Move if below or equal (CF=1 or ZF=1)
0F 42 /r	CMOVC r16, r/m16	Valid	Valid	Move if carry (CF=1)
0F 42 /r	CMOVC r32, r/m32	Valid	Valid	Move if carry (CF=1)
REX.W + 0F 42 /r	CMOVC r64, r/m64	Valid	N.E.	Move if carry (CF=1)
0F 44 /r	CMOVE r16, r/m16	Valid	Valid	Move if equal (ZF=1)
0F 44 /r	CMOVE r32, r/m32	Valid	Valid	Move if equal (ZF=1)
REX.W + 0F 44 /r	CMOVE r64, r/m64	Valid	N.E.	Move if equal (ZF=1)
0F 4F /r	CMOVG r16, r/m16	Valid	Valid	Move if greater (ZF=0 and SF=OF)
0F 4F /r	CMOVG r32, r/m32	Valid	Valid	Move if greater (ZF=0 and SF=OF)
REX.W + 0F 4F /r	CMOVG r64, r/m64	Valid	N.E.	Move if greater (ZF=0 and SF=OF)
0F 4D /r	CMOVGE r16, r/m16	Valid	Valid	Move if greater or equal (SF=OF)
0F 4D /r	CMOVGE r32, r/m32	Valid	Valid	Move if greater or equal (SF=OF)
REX.W + 0F 4D /r	CMOVGE r64, r/m64	Valid	N.E.	Move if greater or equal (SF=OF)
0F 4C /r	CMOVL r16, r/m16	Valid	Valid	Move if less (SF<>OF)
0F 4C /r	CMOVL r32, r/m32	Valid	Valid	Move if less (SF<>OF)
REX.W + 0F 4C /r	CMOVL r64, r/m64	Valid	N.E.	Move if less (SF<>OF)
0F 4E /r	CMOVLE r16, r/m16	Valid	Valid	Move if less or equal (ZF=1 or SF<>OF)
0F 4E /r	CMOVLE r32, r/m32	Valid	Valid	Move if less or equal (ZF=1 or SF<>OF)
REX.W + 0F 4E /r	CMOVLE r64, r/m64	Valid	N.E.	Move if less or equal (ZF=1 or SF<>OF)
0F 46 /r	CMOVNA r16, r/m16	Valid	Valid	Move if not above (CF=1 or ZF=1)
0F 46 /r	CMOVNA r32, r/m32	Valid	Valid	Move if not above (CF=1 or ZF=1)
REX.W + 0F 46 /r	CMOVNA r64, r/m64	Valid	N.E.	Move if not above (CF=1 or ZF=1)
0F 42 /r	CMOVNAE r16, r/m16	Valid	Valid	Move if not above or equal (CF=1)
0F 42 /r	CMOVNAE r32, r/m32	Valid	Valid	Move if not above or equal (CF=1)
REX.W + 0F 42 /r	CMOVNAE r64, r/m64	Valid	N.E.	Move if not above or equal (CF=1)
0F 43 /r	CMOVNB r16, r/m16	Valid	Valid	Move if not below (CF=0)
0F 43 /r	CMOVNB r32, r/m32	Valid	Valid	Move if not below (CF=0)
REX.W + 0F 43 /r	CMOVNB <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if not below (CF=0)
0F 47 /r	CMOVNBE r16, r/m16	Valid	Valid	Move if not below or equal (CF=0 and ZF=0)
0F 47 /r	CMOVNBE <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if not below or equal (CF=0 and ZF=0)
REX.W + 0F 47 /r	CMOVNBE <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if not below or equal (CF=0 and ZF=0)
0F 43 /r	CMOVNC <i>r16</i> , <i>r/m16</i>	Valid	Valid	Move if not carry (CF=0)
0F 43 /r	CMOVNC <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if not carry (CF=0)
REX.W + 0F 43 /r	CMOVNC <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if not carry (CF=0)
0F 45 /r	CMOVNE <i>r16</i> , <i>r/m16</i>	Valid	Valid	Move if not equal (ZF=0)
0F 45 /r	CMOVNE <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if not equal (ZF=0)
REX.W + 0F 45 /r	CMOVNE <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if not equal (ZF=0)
0F 4E /r	CMOVNG <i>r16</i> , <i>r/m16</i>	Valid	Valid	Move if not greater (ZF=1 or SF<>OF)
01 7L/1	5.VIG V 14G 7 10, 1/11/10	valia	valia .	move if flot greater (21 -1 of of <>of)



Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 4E /r	CMOVNG r32, r/m32	Valid	Valid	Move if not greater (ZF=1 or SF<>OF)
REX.W + 0F 4E $/r$	CMOVNG r64, r/m64	Valid	N.E.	Move if not greater (ZF=1 or SF<>OF)
0F 4C /r	CMOVNGE r16, r/m16	Valid	Valid	Move if not greater or equal (SF<>OF)
0F 4C /r	CMOVNGE r32, r/m32	Valid	Valid	Move if not greater or equal (SF<>OF)
REX.W + 0F 4C /r	CMOVNGE r64, r/m64	Valid	N.E.	Move if not greater or equal (SF<>OF)
0F 4D /r	CMOVNL r16, r/m16	Valid	Valid	Move if not less (SF=OF)
0F 4D /r	CMOVNL r32, r/m32	Valid	Valid	Move if not less (SF=OF)
REX.W + 0F 4D $/r$	CMOVNL r64, r/m64	Valid	N.E.	Move if not less (SF=OF)
0F 4F /r	CMOVNLE r16, r/m16	Valid	Valid	Move if not less or equal (ZF=0 and SF=OF)
0F 4F /r	CMOVNLE r32, r/m32	Valid	Valid	Move if not less or equal (ZF=0 and SF=OF)
REX.W + 0F 4F /r	CMOVNLE r64, r/m64	Valid	N.E.	Move if not less or equal (ZF=0 and SF=OF)
0F 41 /r	CMOVNO r16, r/m16	Valid	Valid	Move if not overflow (OF=0)
0F 41 /r	CMOVNO r32, r/m32	Valid	Valid	Move if not overflow (OF=0)
REX.W + 0F 41 /r	CMOVNO r64, r/m64	Valid	N.E.	Move if not overflow (OF=0)
0F 4B /r	CMOVNP r16, r/m16	Valid	Valid	Move if not parity (PF=0)
0F 4B /r	CMOVNP r32, r/m32	Valid	Valid	Move if not parity (PF=0)
REX.W + 0F 4B /r	CMOVNP r64, r/m64	Valid	N.E.	Move if not parity (PF=0)
0F 49 /r	CMOVNS r16, r/m16	Valid	Valid	Move if not sign (SF=0)
0F 49 /r	CMOVNS r32, r/m32	Valid	Valid	Move if not sign (SF=0)
REX.W + 0F 49 /r	CMOVNS r64, r/m64	Valid	N.E.	Move if not sign (SF=0)
0F 45 /r	CMOVNZ r16, r/m16	Valid	Valid	Move if not zero (ZF=0)
0F 45 /r	CMOVNZ r32, r/m32	Valid	Valid	Move if not zero (ZF=0)
REX.W + 0F 45 /r	CMOVNZ r64, r/m64	Valid	N.E.	Move if not zero (ZF=0)
0F 40 /r	CMOVO r16, r/m16	Valid	Valid	Move if overflow (OF=0)
0F 40 /r	CMOVO r32, r/m32	Valid	Valid	Move if overflow (OF=0)
REX.W + 0F 40 /r	CMOVO <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if overflow (OF=0)
0F 4A /r	CMOVP r16, r/m16	Valid	Valid	Move if parity (PF=1)
0F 4A /r	CMOVP <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if parity (PF=1)
REX.W + 0F 4A /r	CMOVP <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if parity (PF=1)
0F 4A /r	CMOVPE <i>r16</i> , <i>r/m16</i>	Valid	Valid	Move if parity even (PF=1)
0F 4A /r	CMOVPE <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if parity even (PF=1)
REX.W + 0F 4A /r	CMOVPE <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if parity even (PF=1)
0F 4B /r	CMOVPO <i>r16</i> , <i>r/m16</i>	Valid	Valid	Move if parity odd (PF=0)
0F 4B /r	CMOVPO <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if parity odd (PF=0)
REX.W + 0F 4B /r	CMOVPO <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if parity odd (PF=0)
0F 48 /r	CMOVS <i>r16</i> , <i>r/m16</i>	Valid	Valid	Move if sign (SF=1)
0F 48 /r	CMOVS <i>r32</i> , <i>r/m32</i>	Valid	Valid	Move if sign (SF=1)
REX.W + 0F 48 /r	CMOVS <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if sign (SF=1)
0F 44 /r	CMOVZ <i>r16, r/m16</i>	Valid	Valid	Move if zero (ZF=1)
0F 44 /r	CMOVZ 176, 1/11176 CMOVZ r32, r/m32	Valid	Valid	Move if zero (ZF=1)
REX.W + 0F 44 /r	CMOVZ <i>r64</i> , <i>r/m64</i>	Valid	N.E.	Move if zero (ZF=1)

IA-32e Mode Operation:

```
temp <-- DEST
IF condition TRUE
  THEN
    DEST <-- SRC
ELSIF (osize = 32)
    DEST <-- temp AND 0x00000000FFFFFFFF
FI;</pre>
```



```
ELSE
  DEST <-- temp
FI;
FI;</pre>
```

Flags Affected

None.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



CMP—Compare Two Operands

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
3C ib	CMP AL, imm8	Valid	Valid	Compare imm8 with AL
3D iw	CMP AX, imm16	Valid	Valid	Compare imm16 with AX
3D id	CMP EAX, imm32	Valid	Valid	Compare imm32 with EAX
REX.W + 3D id	CMP RAX, imm32	Valid	N.E.	Compare imm32 sign-extended to 64-bits with RAX
80 /7 ib	CMP r/m8, imm8	Valid	Valid	Compare imm8 with r/m8
REX + 80 /7 ib	CMP r/m8*, imm8	Valid	N.E.	Compare imm8 with r/m8
81 /7 <i>iw</i>	CMP r/m16, imm16	Valid	Valid	Compare imm16 with r/m16
81 /7 id	CMP r/m32,imm32	Valid	Valid	Compare imm32 with r/m32
REX.W + 81 /7 id	CMP r/m64,imm32	Valid	N.E.	Compare imm32 sign-extended to 64-bits with r/m64
83 /7 ib	CMP r/m16,imm8	Valid	Valid	Compare imm8 with r/m16
83 /7 ib	CMP r/m32,imm8	Valid	Valid	Compare imm8 with r/m32
REX.W + 83 /7 ib	CMP r/m64,imm8	Valid	N.E.	Compare imm8 with r/m64
38 /r	CMP r/m8,r8	Valid	Valid	Compare r8 with r/m8
REX + 38 /r	CMP r/m8*,r8*	Valid	N.E.	Compare r8 with r/m8
39 /r	CMP r/m16,r16	Valid	Valid	Compare r16 with r/m16
39 /r	CMP r/m32,r32	Valid	Valid	Compare r32 with r/m32
REX.W + 39 /r	CMP r/m64,r64	Valid	N.E.	Compare r64 with r/m64
3A /r	CMP r8,r/m8	Valid	Valid	Compare r/m8 with r8
REX + 3A /r	CMP r8*,r/m8*	Valid	N.E.	Compare r/m8 with r8
3B /r	CMP r16,r/m16	Valid	Valid	Compare r/m16 with r16
3B /r	CMP r32,r/m32	Valid	Valid	Compare r/m32 with r32
REX.W + 3B /r	CMP r64,r/m64	Valid	N.E.	Compare r/m64 with r64

In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CMPPD—Compare Packed Double-Precision Floating-Point Values

Opcod	de	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F	C2 /r ib	CMPPD xmm1, xmm2/m128, imm8	Valid	Valid	Compare packed double-precision floating-point values in <i>xmm2/m128</i> and <i>xmm1</i> using imm8 as comparison predicate.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid if SNaN operand; invalid if QNaN and predicate as listed in Table 3-6 of *IA-32 Intel Architecture Software Developer's Manual*, Volume 2A; denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



#GP(0)

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CMPPS—Compare Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F C2 /r ib	CMPPS xmm1, xmm2/m128, imm8	Valid	Valid	Compare packed single-precision floating-point values in xmm2/mem and xmm1 using imm8 as comparison predicate.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid if SNaN operand; invalid if QNaN and predicate as listed in Table 3-6 of IA-32 Intel Architecture Software Developer's Manual, Volume 2; denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

CMPS/CMPSB/CMPSW/CMPSD/CMPSQ—Compare String Operands

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
A6	CMPS m8, m8	Valid	Valid	For legacy mode, compare byte at address ES:(E)SI; For 64-bit mode compare byte at address (R)SI. The status flags are set accordingly.
A7	CMPS m16, m16	Valid	Valid	For legacy mode, compare word at address ES:(E)SI; For 64-bit mode compare word at address (R)SI. The status flags are set accordingly.
A7	CMPS m32, m32	Valid	Valid	For legacy mode, compare dword at address ES:(E)SI; For 64-bit mode compare dword at address (R)SI. The status flags are set accordingly.
REX.W + A7	CMPS m64, m64	Valid	N.E.	Compares quadword at address RSI with quadword at address RDI and sets the status flags accordingly
A6	CMPSB	Valid	Valid	For legacy mode, compare byte at address ES:(E)SI; For 64-bit mode compare byte at address (R)SI. The status flags are set accordingly.
A7	CMPSW	Valid	Valid	For legacy mode, compare word at address ES:(E)SI; For 64-bit mode compare word at address (R)SI. The status flags are set accordingly.
A7	CMPSD	Valid	Valid	For legacy mode, compare dword at address ES:(E)SI; For 64-bit mode compare dword at address (R)SI. The status flags are set accordingly.
REX.W + A7	CMPSD	Valid	N.E.	Compares quadword at address RSI with quadword at address RDI and sets the status flags accordingly

Flags Affected

The CF, OF, SF, ZF, AF, and PF flags are set according to the temporary result of the comparison.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.



Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CMPSD—Compare Scalar Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F C2 /r ib	CMPSD xmm1, xmm2/m64, imm8	Valid	Valid	Compare low double-precision floating-point value in <i>xmm2/m64</i> and <i>xmm1</i> using <i>imm8</i> as comparison predicate.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid if SNaN operand; invalid if QNaN and predicate as listed in Table 3-6 of *IA-32 Intel Architecture Software Developer's Manual*, Volume 2A; denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CMPSS—Compare Scalar Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F C2 /r ib	CMPSS xmm1, xmm2/m32, imm8	Valid	Valid	Compare low single-precision floating-point value in <i>xmm2/m32</i> and <i>xmm1</i> using <i>imm8</i> as comparison predicate.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid if SNaN operand; invalid if QNaN and predicate as listed in Table 3-6 of *IA-32 Intel Architecture Software Developer's Manual*, Volume 2A; denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CMPXCHG—Compare and Exchange

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F B0/ <i>r</i>	CMPXCHG r/m8,r8	Valid	Valid	Compare AL with <i>r/m8</i> . If equal, ZF is set and <i>r8</i> is loaded into <i>r/m8</i> . Else, clear ZF and load <i>r/m8</i> into AL.
REX + 0F B0/r	CMPXCHG r/m8*,r8*	Valid	N.E.	Compare AL with <i>r/m8</i> . If equal, ZF is set and <i>r8</i> is loaded into <i>r/m8</i> . Else, clear ZF and load <i>r/m8</i> into AL.
0F B1/r	CMPXCHG r/m16,r16	Valid	Valid	Compare AX with r/m16. If equal, ZF is set and r16 is loaded into r/m16. Else, clear ZF and load r/m16 into AX
0F B1/r	CMPXCHG r/m32,r32	Valid	Valid	Compare EAX with <i>r/m32</i> . If equal, ZF is set and <i>r32</i> is loaded into <i>r/m32</i> . Else, clear ZF and load <i>r/m32</i> into EAX
REX.W + 0F B1/r	CMPXCHG r/m64,r64	Valid	N.E.	Compare RAX with <i>r/m64</i> . If equal, ZF is set and <i>r64</i> is loaded into <i>r/m64</i> . Else, clear ZF and load <i>r/m64</i> into AL

^{*} In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

The ZF flag is set if the values in the destination operand and register AL, AX, or EAX are equal; otherwise it is cleared. The CF, PF, AF, SF, and OF flags are set according to the results of the comparison operation.

IA-32e Mode Operation

Promoted to 64-bits

Default Operation Size is 32-bits.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CMPXCHG8B/CMPXCHG16B—Compare and Exchange 8 Bytes

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F C7 /1 m64	CMPXCHG8B m64	Valid	Valid	Compare EDX:EAX with <i>m64</i> . If equal, set ZF and load ECX:EBX into <i>m64</i> . Else, clear ZF and load <i>m64</i> into EDX:EAX.
REX.W + 0F C7 /1 m128	CMPXCHG16B m128	Valid	N.E.	Compare RDX:RAX with <i>m128</i> . If equal, set ZF and load RCX:RBX into <i>m128</i> . Else, clear ZF and load <i>m128</i> into RDX:RAX.

Flags Affected

The ZF flag is set if the destination operand and EDX:EAX are equal; otherwise it is cleared. The CF, PF, AF, SF, and OF flags are unaffected.

IA-32e Mode Operation

Promoted to 64-bits

Default Operation Size is 64-bits.

CMPXCHG16B requires that the destination (memory) operand be 16-byte-aligned.

Protected Mode Exceptions

#UD If the destination operand is not a memory location.
#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#UD If the destination operand is not a memory location.

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#UD If the destination operand is not a memory location.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand for CMPXCHG16B is not aligned on a 16-byte boundary.

If CPUID feature flag CMPXCHG16B is 0.

#UD If the destination operand is not a memory location.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

COMISD—Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 2F /r	COMISD xmm1, xmm2/m64	Valid	Valid	Compare low double-precision floating-point values in <i>xmm1</i> and <i>xmm2/mem64</i> and set the EFLAGS flags accordingly.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid (if SNaN or QNaN operands), Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

COMISS—Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 2F /r	COMISS xmm1, xmm2/m32	Valid	Valid	Compare low single-precision floating-point values in <i>xmm1</i> and <i>xmm2/mem32</i> and set the EFLAGS flags accordingly.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid (if SNaN or QNaN operands), Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CPUID—CPU Identification

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F A2	CPUID	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, according to the input value entered initially in the EAX register

Description

When the input value in the EAX register is 0, the processor returns the highest value the CPUID instruction recognizes in the EAX register for returning basic CPUID information (see Table 2-5). A vendor identification string is returned in the EBX, EDX, and ECX registers. For Intel[®] processors, the vendor identification string is "GenuineIntel" as follows:

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

Table 2-5 CPUID Function Leaf

Initial EAX Value	Information Provided about the Processor
ОН	EAX Maximum Input Value for Basic CPUID Information (see Table 2-6). EBX "Genu" ECX "ntel" EDX "inel"
1H	EAX Version Information (Type, Family, Model, and Stepping ID) Bits 7-0: Brand Index Bits 15-8: CLFLUSH line size. (Value returned * 8 = cache line size) Bits 23-16: # of logical processors per physical package Bits 31-24: Processor's initial local APIC ID ECX Feature Information (see Figure 2-8) EDX Feature Information (see Figure 2-8)
2H	EAX Cache and TLB Information EBX Cache and TLB Information ECX Cache and TLB Information EDX Cache and TLB Information
3H	EAX Reserved EBX Reserved ECX Reserved EDX Reserved

Table 2-5 CPUID Function Leaf (Contd.)

Initial EAX	Table 2-5 CPUID Function Leaf (Contd.)
Value	Information Provided about the Processor
4H	Deterministic Cache Parameters Leaf Bits 4-0: Cache Type** Bits 7-5: Cache Level (starts at 1) Bits 8: Self Initializing cache level (does not need SW initialization) Bits 9: Fully Associative cache Bits 13-10: Reserved Bits 25-14: Number of threads sharing this cache* Bits 31-26: Number of processor cores on this die (Multicore)*. EBX Bits 11-00: L = System Coherency Line Size*
	Bits 21-12: P = Physical Line partitions* Bits 31-22: W = Ways of associativity*. ECX Bits 31-00: S = Number of Sets* EDX Reserved = 0
	* Add one to the value in the register file to get the number. For example, the number of processor cores is EAX[31:26]+1. ** Cache Types fields 0 = Null - No more caches 1 = Data Cache 2 = Instruction Cache 3 = Unified Cache 4-31 = Reserved
	NOTE: CPUID leaves > 3 < 80000000 are only visible when IA32_CR_MISC_ENABLES.BOOT NT4 (bit 22) is clear (Default)
5H	MONITOR/MWAIT Leaf Bits 15-00: Smallest monitor-line size in bytes (default is processor's monitor granularity) Bits 31-16: Reserved = 0. Bits 15-00: Largest monitor-line size in bytes (default is processor's monitor granularity) EBX Bits 31-16: Reserved = 0. ECX Reserved = 0 EDX Reserved = 0
	Extended Function CPUID Information
80000000H	EAX Maximum Input Value for Extended Function CPUID Information (see Table 2-6). EBX Reserved. ECX Reserved. EDX Reserved.
8000001H	Extended Processor Signature and Extended Feature Bits. Reserved. ECX Reserved EDX Bits 10-0: Reserved Bit 11: SYSCALL/SYSRET available Bits 19-12: Reserved Bits 28-21: Reserved Bit 29: Intel EM64T available Bits 31-30: Reserved.
80000002H	EAX Processor Brand String. EBX Processor Brand String Continued. ECX Processor Brand String Continued. EDX Processor Brand String Continued.
80000003H	EAX Processor Brand String Continued. EBX Processor Brand String Continued. ECX Processor Brand String Continued. EDX Processor Brand String Continued.
80000004H	EAX Processor Brand String Continued. EBX Processor Brand String Continued. ECX Processor Brand String Continued. EDX Processor Brand String Continued.

Table 2-5 CPUID Function Leaf (Contd.)

Initial EAX Value		Information Provided about the Processor
80000005H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000006H	EAX EBX ECX	Reserved = 0 Reserved = 0 Bits 7-0: Cache Line size Bits 15-12: L2 Associativity Bits 31-16: Cache size in 1K units In initial implementation ECX = 0x04008040 Reserved = 0
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000008H	EBX ECX EDX	Virtual/Physical Address size Bits 7:0: #Physical Address Bits Bits 15:8: #Virtual Address Bits, Bits 31:16: reserved. In initial implementation EAX = 0x3028 Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0

Table 2-6 Highest CPUID Source Operand for Processor Supporting Intel EM64T

14 co P	Highest Value in EAX		
IA-32 Processors	Basic Information	Extended Function Information	
	5H	8000008H	

When the input value is 1, the processor returns version information in the EAX register (see Figure 2-3). The version information consists of an IA-32 processor family identifier, a model identifier, a stepping ID, and a processor type. The model, family, and processor type for the first processor supporting Intel EM64T is as follows:

- Model—0011B
- Family—1111B
- Processor Type—00B

The available processor types are given in Table 2-7. Intel releases information on stepping IDs as needed.

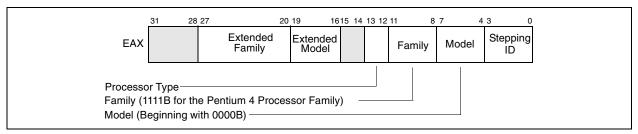


Figure 2-3. Version Information in the EAX Register



Table 2-7 Processor Type Field

Туре	Encoding
Original OEM Processor	00B
Intel [®] OverDrive [®] Processor	01B
Dual processor*	10B
Intel reserved.	11B

^{*} Not applicable to Intel486™ processors.

The Extended Family ID and Extended Model ID need be examined only if the Family ID reaches 0FH. Always display processor information as a combination of family, model, and stepping.

Integrate the ID fields into a display as:

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

Compute the displayed model from the Model ID and the Extended Model ID as:

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

See AP-485, *Intel Processor Identification and the CPUID Instruction* (Order Number 241618) and Chapter 13 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for more information on identifying earlier IA-32 processors.

When the input value in EAX is 1, four unrelated pieces of information are returned to the EBX register:

- Brand index EBX[7:0]—this number provides an entry into a brand string table that contains brand strings for IA-32 processors. See "Brand Identification" later in the description of this instruction for information about the intended use of brand indices. This field was introduced in the Pentium[®] III XeonTM processors.
- CLFLUSH instruction cache line size EBX[15:8]—this number indicates the size of the cache line flushed with CLFLUSH instruction in 8-byte increments. This field was introduced with the Pentium 4 processors.
- The number of threads contained in the package is indicated in EBX[23:16].
- Initial APIC ID EBX[31:24]—this number is the 8-bit physical ID that is assigned to the local APIC on the processor during power up. This field was introduced with the Pentium 4 processors.

When the input value in EAX is 1, feature information is returned to the EDX register (see Figure 2-8). The feature bits permit operating system or application code to determine which IA-32 architectural features are available in the processor. Table 2-8 shows the encoding of the feature flags in the EDX register. For all the feature flags currently returned in EDX, a 1 indicates that the corresponding feature is supported. Software should identify Intel as the vendor to properly interpret the feature flags. (Software should not depend on a 1 indicating the presence of a feature for future feature flags.)

Table 2-8 CPUID Feature Information

Reg.Bit #	Mnemonic	Description
EDX.0	FPU	Floating Point Unit On-Chip. The processor contains an x87 FPU.
EDX.1	VME	Virtual 8086 Mode Enhancements. Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
EDX.2	DE	Debugging Extensions. Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.

Table 2-8 CPUID Feature Information (Contd.)

Table 2-8 CPUID Feature Information (Contd.) Reg.Bit # Mnemonic Description			
EDX.3	PSE	Page Size Extension. Large pages of size 4Mbyte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.	
EDX.4	TSC	Time Stamp Counter. The RDTSC instruction is supported, including CR4.TSD for controlling privilege.	
EDX.5	MSR	Model Specific Registers RDMSR and WRMSR Instructions. The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.	
EDX.6	PAE	Physical Address Extension. Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2 Mbyte pages are supported instead of 4 Mbyte pages if PAE bit is 1. The actual number of address bits beyond 32 is not defined, and is implementation specific.	
EDX.7	MCE	Machine Check Exception. Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.	
EDX.8	CX8	CMPXCHG8B Instruction. The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).	
EDX.9	APIC	APIC On-Chip. The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).	
EDX.10	Reserved	Reserved	
EDX.11	SEP	SYSENTER and SYSEXIT Instructions. The SYSENTER and SYSEXIT and associated MSRs are supported.	
EDX.12	MTRR	Memory Type Range Registers. MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.	
EDX.13	PGE	PTE Global Bit. The global bit in page directory entries (PDEs) and page table entries (PTEs) is supported, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.	
EDX.14	MCA	Machine Check Architecture. The Machine Check Architecture, which provides a compatible mechanism for error reporting in Pentium 4 processors, P6 family processors, and future processors, is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.	
EDX.15	CMOV	Conditional Move Instructions. The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported	
EDX.16	PAT	Page Attribute Table. Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory on a 4K granularity through a linear address.	
EDX.17	PSE-36	36-Bit Page Size Extension. Extended 4-MByte pages that are capable of addressing physical memory beyond 4 GBytes are supported. This feature indicates that the upper four bits of the physical address of the 4-MByte page is encoded by bits 13-16 of the page directory entry.	
EDX.18	PSN	Processor Serial Number. The processor supports the 96-bit processor identification number feature and the feature is enabled. (Only Pentium III processor supports this feature.)	
EDX.19	CLFSH	CLFLUSH Instruction. CLFLUSH Instruction is supported.	
EDX.20	Reserved	Reserved	



Table 2-8 CPUID Feature Information (Contd.)

Reg.Bit #	Mnemonic	Description
EDX.21	DS	Debug Store. The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 14, <i>Debugging and Performance Monitoring</i> , in the <i>IA-32 Intel Architecture Software Developer's Manual, Volume 3</i>).
EDX.22	ACPI	Thermal Monitor and Software Controlled Clock Facilities. The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
EDX.23	MMX	Intel [®] MMX [™] Technology. The processor supports the Intel MMX technology.
EDX.24	FXSR	FXSAVE and FXRSTOR Instructions. The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions
EDX.25	SSE	SSE. The processor supports the SSE extensions.
EDX.26	SSE2	SSE2. The processor supports the SSE2 extensions.
EDX.27	SS	Self Snoop. The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus
EDX.28	НТ	Hyper-Threading Technology. The processor supports Hyper-Threading Technology.
EDX.29	ТМ	Thermal Monitor. The processor implements the thermal monitor automatic thermal control circuitry (TCC).
EDX.30	Reserved	Reserved
EDX.31	FERR	FERR# Signalling change
ECX.0	SSE3	SSE3 available
ECX.1	Reserved	Reserved
ECX.2	Reserved	Reserved
ECX.3	Monitor	Monitor/Mwait Instructions
ECX.4	DS_CPL	CPL qualified debug store available
ECX.5	Reserved	Reserved
ECX.6	Reserved	Reserved
ECX.7	EST	Enhanced Intel [®] SpeedStep [®] Technology
ECX.8	TM2	Thermal Monitor 2 available
ECX.9	Reserved	Reserved
ECX.10	CNXT-ID	L1 context ID available
ECX.11	Reserved	Reserved
ECX.12	Reserved	Reserved
ECX.13	CMPXCHG16B	CMPXCHG16B available
ECX.31:13	Reserved	Reserved

When the input value is 2, the processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers. The encoding of these registers is as follows:

• The least-significant byte in register EAX (register AL) indicates the number of times the CPUID instruction must be executed with an input value of 2 to get a complete description of the processor's caches and TLBs. The first member of the family of Pentium 4 processors will return a 1.

- The most significant bit (bit 31) of each register indicates whether the register contains valid information (cleared to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1 byte descriptors. Table 2-9 shows the encoding of these descriptors. Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache or TLB types. The descriptors may appear in any order.

Table 2-9 Cache Parameter Table

Descriptor Value	Description			
22h	512K Third-Level Cache, 4-way, 64byte line size, 128 byte sector size			
23h	1MB Third-Level Cache, 8-way, 64byte line size, 128 byte sector size			
25h	2MB Third-Level Cache, 8-way, 64byte line size, 128 byte sector size			
29h	4MB Third-Level Cache, 8-way, 64byte line size, 128 byte sector size			
40h	No Third-Level Cache			
50h	64 entries for 4KB pages & 2MB/4MB pages (ITLB)			
51h	128 entries for 4KB pages & 2MB/4MB pages (ITLB)			
52h	256 entries for 4KB pages & 2MB/4MB pages (ITLB)			
5Bh	64 entries for 4KB pages & 4MB pages (DTLB)			
5Ch	128 entries for 4KB pages & 4MB pages (DTLB)			
5Dh	256 entries for 4KB pages & 4MB pages (DTLB)			
60h	16KB First-Level Data Cache, 8-way set associative, 64 byte line size			
66h	8KB First-Level Data Cache, 4-way set associative, 64 byte line size			
67h	16KB First-Level Data Cache, 4-way set associative, 64 byte line size			
68h	32KB First-Level Data Cache, 4-way set associative, 64 byte line size			
70h	12K uops, Trace Cache, 8-way set associative			
71h	16K uops, Trace Cache, 8-way set associative			
72h	32K uops, Trace Cache, 8-way set associative			
78H	2nd-level cache: 1Mbyte, 8-way set associative, 64 byte line size, 128 byte sector size			
79h	128KB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size			
7Ah	256KB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size			
7Bh	512KB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size			
7Ch	1MB Second-Level Cache, 8-way set associative, 64byte line size, 128 byte sector size			

The first member of the family of Pentium 4 processors will return the following information about caches and TLBs when the CPUID instruction is executed with an input value of 2:

EAX	66 5B 50 01H
EBX	0H
ECX	0H
EDX	00 7A 70 40H

These values are interpreted as follows:

- The least-significant byte (byte 0) of register EAX is set to 01H, indicating that the CPUID instruction needs to be executed only once with an input value of 2 to retrieve complete information about the processor's caches and TLBs.
- The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.



- Bytes 1, 2, and 3 of register EAX indicate that the processor contains the following:
 - 50H—A 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.
 - 5BH—A 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.
 - 66H—An 8-KByte 1st level data cache, 4-way set associative, with a 64-byte cache line size.
- The descriptors in registers EBX and ECX are valid, but contain null descriptors.
- Bytes 0, 1, 2, and 3 of register EDX indicate that the processor contains the following:
 - 00H—Null descriptor.
 - 70H—A 12-KByte 1st level code cache, 4-way set associative, with a 64-byte cache line size.
 - 7AH—A 256-KByte 2nd level cache, 8-way set associative, with a 128-byte cache line size.
 - 00H—Null descriptor.

Brand Identification

To facilitate brand identification of IA-32 processors with the CPUID instruction, two features are provided: brand index and brand string.

The brand index was added to the CPUID instruction with the Pentium III Xeon processor and will be included on all future IA-32 processors, including the Pentium 4 processors. The brand index provides an entry point into a brand identification table that is maintained in memory by system software and is accessible from system- and user-level code. In this table, each brand index is associate with an ASCII brand identification string that identifies the official Intel[®] family and model number of a processor (for example, "Intel Pentium III processor").

When executed with a value of 1 in the EAX register, the CPUID instruction returns the brand index to the low byte in EBX. Software can then use this index to locate the brand identification string for the processor in the brand identification table. The first entry (brand index 0) in this table is reserved, allowing for backward compatibility with processors that do not support the brand identification feature. Table 2-10 shows those brand indices that currently have processor brand identification strings associated with them.

It is recommended that (1) all reserved entries included in the brand identification table be associated with a brand string that indicates that the index is reserved for future Intel processors and (2) that software be prepared to handle reserved brand indices gracefully.

Table 2-10 Brand String Offsets

Brand Index	Brand String			
0	This processor does not support the brand identification feature			
1	Celeron [®] processor*			
2	Pentium [®] III processor [*]			
3	Intel [®] Pentium [®] III Xeon™ processor			
4	Intel [®] Pentium [®] III processor			
6	Mobile Intel [®] Pentium [®] III processor -M			
7	Mobile Intel [®] Celeron [®] processor			
8	Intel [®] Pentium [®] 4 processor If processor signature ≥ 00000F13H; then Intel [®] Genuine processor			
9	Intel® Pentium® 4 processor			
10	Intel [®] Celeron [®] processor			
11	Intel [®] Xeon [™] processor If processor signature < 00000F13H; then Intel [®] Xeon [™] processor MP			
12	Intel [®] Xeon™ processor MP			
14	Mobile Intel [®] Pentium [®] 4 processor -M If processor signature < 00000F13H; then Intel [®] Xeon [™] processor			



Table 2-10 Brand String Offsets (Contd.)

15	Mobile Intel [®] Celeron [®] processor
16 – 255	Reserved for future processor

^{*} Indicates versions of these processors that were introduced after the Pentium III Xeon processor.

The brand string feature is an extension to the CPUID instruction introduced in the Pentium 4 processors. With this feature, the CPUID instruction returns the ASCII brand identification string and the maximum operating frequency of the processor to the EAX, EBX, ECX, and EDX registers. (Note that the frequency returned is the maximum operating frequency that the processor has been qualified for and not the current operating frequency of the processor.)

To use the brand string feature, the CPUID instructions must be executed three times, once with an input value of 80000002H in the EAX register, and a second time an input value of 80000003, and a third time with a value of 80000004H.

The brand string is architecturally defined to be 48 byte long: the first 47 bytes contain ASCII characters and the 48th byte is defined to be null (0). The string may be right justified (with leading spaces) for implementation simplicity. For each input value (EAX is 80000002H, 80000003H, or 80000004H), the CPUID instruction returns 16 bytes of the brand string to the EAX, EBX, ECX, and EDX registers. Processor implementations may return less than the 47 ASCII characters, in which case the string will be null terminated and the processor will return valid data for each of the CPUID input values of 80000002H, 80000003H, and 80000004H.

Table 2-11 shows the brand string that is returned by the first processor in the family of Pentium 4 processors.

NOTE

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

The following procedure can be used for detection of the brand string feature:

- 1. Execute the CPUID instruction with input value in EAX of 80000000H.
- 2. If ((EAX_Return_Value) AND (80000000H) ≠ 0) then the processor supports the extended CPUID functions and EAX contains the largest extended function input value supported.
- 3. If EAX_Return_Value ≥ 80000004H, then the CPUID instruction supports the brand string feature.

Table 2-11 Processor Brand String Returned

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

Future processors may return a frequency in GHz rather than MHz. To identify an IA-32 processor using the CPUID instruction, brand identification software should use the following brand identification techniques ordered by decreasing priority:

- Processor brand string
- Processor brand index and a software supplied brand string table.



 Table based mechanism using type, family, model, stepping, and cache information returned by the CPUID instruction.

IA-32 Architecture Compatibility

The CPUID instruction is not supported in early models of the Intel486 processor or in any IA-32 processor earlier than the Intel486 processor.

Operation

Flags Affected

None.

Exceptions (All Operating Modes)

None.

NOTE

In earlier IA-32 processors that do not support the CPUID instruction, execution of the instruction results in an invalid opcode (#UD) exception being generated.

CVTDQ2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F E6	CVTDQ2PD xmm1, xmm2/m64	Valid	Valid	Convert two packed signed doubleword integers from <i>xmm2/m128</i> to two packed double-precision floating-point values in <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTDQ2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 5B /r	CVTDQ2PS xmm1, xmm2/m128	Valid	Valid	Convert four packed signed doubleword integers from <i>xmm2/m128</i> to four packed single-precision floating-point values in <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Precision.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTPD2DQ—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F E6	CVTPD2DQ xmm1, xmm2/ m128	Valid	Valid	Convert two packed double-precision floating-point values from xmm2/m128 to two packed signed doubleword integers in xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

segments.segments.

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTPD2PI—Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 2D /r	CVTPD2PI mm, xmm/m128	Valid	Valid	Convert two packed double-precision floating-point values from xmm/m128 to two packed signed doubleword integers in mm.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#MF If there is a pending x87 FPU exception.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#MF If there is a pending x87 FPU exception.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTPD2PS—Covert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 5A /r	CVTPD2PS xmm1, xmm2/ m128	Valid	Valid	Convert two packed double-precision floating- point values in <i>xmm2/m128</i> to two packed single-precision floating-point values in <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTPI2PD—Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 2A /r	CVTPI2PD xmm, mm/m64	Valid	Valid	Convert two packed signed doubleword integers from <i>mm/mem64</i> to two packed double-precision floating-point values in <i>xmm</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTPI2PS—Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 2A /r	CVTPI2PS xmm, mm/ m64	Valid	Valid	Convert two signed doubleword integers from mm/m64 to two single-precision floating-point values in xmm

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Precision.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTPS2DQ—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 5B /r	CVTPS2DQ xmm1, xmm2/m128	Valid	Valid	Convert four packed single-precision floating-point values from xmm2/m128 to four packed signed doubleword integers in xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTPS2PD—Covert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 5A /r	CVTPS2PD xmm1, xmm2/ m64	Valid	Valid	Convert two packed single-precision floating-point values in <i>xmm2/m64</i> to two packed double-precision floating-point values in <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTPS2PI—Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

C	Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0)F 2D /r	CVTPS2PI mm, xmm/m64	Valid	Valid	Convert two packed single-precision floating-point values from <i>xmm/m64</i> to two packed signed doubleword integers in <i>mm</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

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.

#MF If there is a pending x87 FPU exception.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTSD2SI—Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 2D /r	CVTSD2SI r32, xmm/ m64	Valid	Valid	Convert one double-precision floating- point value from <i>xmm/m64</i> to one signed doubleword integer <i>r32</i> .
REX.W + F2 0F 2D /r	CVTSD2SI r64, xmm/ m64	Valid	N.E.	Convert one double-precision floating- point value from xmm/m64 to one signed quadword integer sign-extended into r64.

IA-32e Mode Operation

Promoted to 64-bits.

Enables access to XMM8-XMM15.

Enables access to new registers R8-R15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode



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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTSD2SS—Convert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 5A /r	CVTSD2SS xmm1, xmm2/m64	Valid	Valid	Convert one double-precision floating- point value in <i>xmm2/m64</i> to one single- precision floating-point value in <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTSI2SD—Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 2A /r	CVTSI2SD <i>xmm</i> , r/m32	Valid	Valid	Convert one signed doubleword integer from <i>r/m32</i> to one double-precision floating-point value in <i>xmm</i> .
REX.W + F2 0F 2A /r	CVTSI2SD <i>xmm</i> , r/m64	Valid	N.E.	Convert one signed quadword integer from <i>r/m64</i> to one double-precision floating-point value in <i>xmm</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

Promoted to 64 bits.

Enables access to new registers R8-R15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode



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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTSI2SS—Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
F3 0F 2A /r	CVTSI2SS xmm, r/m32	Valid	Valid	Convert one signed doubleword integer from r/m32 to one single-precision floating-point value in xmm.
REX.W + F3 0F 2A /r	CVTSI2SS xmm, r/m64	Valid	N.E.	Convert one signed quadword integer from r/ m64 to one single-precision floating-point value in xmm.

IA-32e Mode Operation

Enables access to XMM8-XMM15. XMMn[31:0] = CVT(reg/mem64), XMMn[127:32] = unchanged.

Promoted to 64 bits.

Enables access to new registers R8-R15.

SIMD Floating-Point Exceptions

Precision.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode



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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTSS2SD—Convert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 5A /r	CVTSS2SD xmm1, xmm2/m32	Valid	Valid	Convert one single-precision floating-point value in <i>xmm2/m32</i> to one double-precision floating-point value in <i>xmm1</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15. XMMn[63:0] = CVT(reg/mem32), XMMn[127:64] = unchanged.

SIMD Floating-Point Exceptions

Invalid, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTSS2SI—Convert Scalar Single-Precision Floating-Point Value to Doubleword Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 2D /r	CVTSS2SI r32, xmm/ m32	Valid	Valid	Convert one single-precision floating-point value from <i>xmm/m32</i> to one signed doubleword integer in <i>r32</i> .
REX.W + F3 0F 2D /r	CVTSS2SI r64, xmm/ m32	Valid	N.E.	Convert one single-precision floating-point value from <i>xmm/m32</i> to one signed quadword integer in <i>r64</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

Promoted to 64-bits.

Enables access to new registers R8-R15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.
If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.



Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTTPD2PI—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 2C /r	CVTTPD2PI mm, xmm/m128	Valid	Valid	Convert two packer double-precision floating-point values from <i>xmm/m128</i> to two packed signed doubleword integers in <i>mm</i> using truncation.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

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

#MF If there is a pending x87 FPU exception.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTTPD2DQ—Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F E6	CVTTPD2DQ xmm1, xmm2/m128	Valid	Valid	Convert two packed double-precision floating-point values from xmm2/m128 to two packed signed doubleword integers in xmm1 using truncation.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.



Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTTPS2DQ—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 5B /r	CVTTPS2DQ xmm1, xmm2/m128	Valid	Valid	Convert four single-precision floating-point values from xmm2/m128 to four signed doubleword integers in xmm1 using truncation.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

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

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



Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

CVTTPS2PI—Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 2C /r	CVTTPS2PI mm, xmm/ m64	Valid	Valid	Convert two single-precision floating-point values from <i>xmm/m64</i> to two signed doubleword signed integers in <i>mm</i> using truncation.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Invalid, Precision.

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.

#MF If there is a pending x87 FPU exception.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.



Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

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

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTTSD2SI—Convert with Truncation Scalar Double-Precision Floating-Point Value to Signed Doubleword Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 2C /r	CVTTSD2SI <i>r32</i> , <i>xmm/</i> <i>m64</i>	Valid	Valid	Convert one double-precision floating-point value from <i>xmm/m64</i> to one signed doubleword integer in <i>r32</i> using truncation.
REX.W + F2 0F 2C /r	CVTTSD2SI r64, xmm/ m64	Valid	N.E.	Convert one double precision floating-point value from xmm/m64 to one signed quadword integer in r64 using truncation.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

Promoted to 64-bits.

Enables access to new registers R8-R15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.



Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CVTTSS2SI—Convert with Truncation Scalar Single-Precision Floating-Point Value to Doubleword Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 2C /r	CVTTSS2SI r32, xmm/ m32	Valid	Valid	Convert one single-precision floating-point value from <i>xmm/m32</i> to one signed doubleword integer in <i>r32</i> using truncation.
REX.W + F3 0F 2C /r	CVTTSS2SI r64, xmm/ m32	Valid	N.E.	Convert one single-precision floating-point value from <i>xmm/m32</i> to one signed quadword integer in <i>r64</i> using truncation.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

Promoted to 64-bits.

Enables access to new registers R8-R15.

SIMD Floating-Point Exceptions

Invalid, Precision.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.



Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

CWD/CDQ/CQQ—Convert Word to Doubleword/Convert Doubleword to Quadword/Convert Quadword to Double Quadword

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
99	CWD	Valid	Valid	$DX \mathpunct{:} AX \leftarrow sign \textup{-} extend \ of \ AX$
99	CDQ	Valid	Valid	EDX:EAX ← sign-extend of EAX
REX.W + 99	CQO	Valid	N.E.	RDX:RAX← sign-extend of RAX

Flags Affected

None.

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is size of destination register.

Exceptions (All Operating Modes)

None.

See entry for CBW/CWDE—Convert Byte to Word/Convert Word to Doubleword.



DAA—Decimal Adjust AL after Addition

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
27	DAA	Inv.	Valid	Decimal adjust AL after addition

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 "Operation" section above). The SF, ZF, and PF flags are set according to the result. The OF flag is undefined.

IA-32e Mode Operation

Invalid in 64 bit Mode

Protected Mode Exceptions

None.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

None.

Compatibility Mode Exceptions

None.

64-Bit Mode Exceptions

#UD If in 64-bit mode.

DAS—Decimal Adjust AL after Subtraction

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
2F	DAS	Inv.	Valid	Decimal adjust AL after subtraction	

Flags Affected

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

IA-32e Mode Operation

Invalid in 64 bit Mode

Protected Mode Exceptions

None.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

None.

Compatibility Mode Exceptions

None.

64-Bit Mode Exceptions

#UD If in 64-bit mode.



DEC—Decrement by 1

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
FE /1	DEC r/m8	Valid	Valid	Decrement r/m8 by 1
REX + FE /1	DEC r/m8*	Valid	N.E.	Decrement r/m8 by 1
FF /1	DEC r/m16	Valid	Valid	Decrement r/m16 by 1
FF /1	DEC r/m32	Valid	Valid	Decrement r/m32 by 1
REX.W + FF /1	DEC r/m64	Valid	N.E.	Decrement r/m64 by 1
48+rw	DEC r16	N.E.	Valid	Decrement r16 by 1
48+rd	DEC <i>r32</i>	N.E.	Valid	Decrement r32 by 1

In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32-bits.

Opcode 48H through 4FH are REX prefixes in 64-bit mode.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination operand is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



DIV—Unsigned Divide

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F6 /6	DIV r/m8	Valid	Valid	Unsigned divide AX by $r/m8$, with result stored in AL \leftarrow Quotient, AH \leftarrow Remainder
REX + F6 /6	DIV <i>r/m8*</i>	Valid	N.E.	Unsigned divide AX by $r/m8$, with result stored in AL \leftarrow Quotient, AH \leftarrow Remainder
F7 /6	DIV <i>r/m16</i>	Valid	Valid	Unsigned divide DX:AX by $r/m16$, with result stored in AX \leftarrow Quotient, DX \leftarrow Remainder
F7 /6	DIV <i>r/m32</i>	Valid	Valid	Unsigned divide EDX:EAX by $r/m32$, with result stored in EAX \leftarrow Quotient, EDX \leftarrow Remainder
REX.W + F7 /6	DIV <i>r/m64</i>	Valid	N.E.	Unsigned divide RDX:RAX by $r/m64$, with result stored in RAX \leftarrow Quotient, RDX \leftarrow Remainder

In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Instruction is promoted to 64-bits.

Default Operation Size is 32 bits

With 64-bit operation RAX contains the 64-bit quotient and RDX the 64-bit remainder

Enables access to new registers R8-R15.

Protected Mode Exceptions

#DE If the source operand (divisor) is 0

If the quotient is too large for the designated register.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#DE If the source operand (divisor) is 0.

If the quotient is too large for the designated register.

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#DE If the source operand (divisor) is 0.

If the quotient is too large for the designated register.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#DE If the source operand (divisor) is 0

If the quotient is too large for the designated register.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

DIVPD—Divide Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 5E /r	DIVPD xmm1, xmm2/m128	Valid	Valid	Divide packed double-precision floating-point values in <i>xmm1</i> by packed double-precision floating-point values <i>xmm2/m128</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Divide-by-Zero, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.



DIVPS—Divide Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 5E /r	DIVPS xmm1, xmm2/m128	Valid	Valid	Divide packed single-precision floating-point values in <i>xmm1</i> by packed single-precision floating-point values <i>xmm2/m128</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Divide-by-Zero, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.



DIVSD—Divide Scalar Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 5E /r	DIVSD xmm1, xmm2/m64	Valid	Valid	Divide low double-precision floating-point value n <i>xmm1</i> by low double-precision floating-point value in <i>xmm2/mem64</i> .

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Divide-by-Zero, Precision, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.



64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE2 is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

DIVSS—Divide Scalar Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F3 0F 5E /r	DIVSS xmm1, xmm2/m32	Valid	Valid	Divide low single-precision floating-point value in <i>xmm1</i> by low single-precision floating-point value in <i>xmm2/m32</i>

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Divide-by-Zero, Precision, Denormal.

Protected Mode Exceptions

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

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

EMMS—Empty MMX State

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 77	EMMS	Valid	Valid	Set the x87 FPU tag word to empty.

Flags Affected

None.

IA-32e Mode Operation

Same as Legacy mode

Protected Mode Exceptions

#UD If EM in CR0 is set.
#NM If TS in CR0 is set.

#MF If there is a pending FPU exception.

Real-Address Mode Exceptions

Same as for protected mode exceptions.

Virtual-8086 Mode Exceptions

Same as for protected mode exceptions.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

Same as for protected mode exceptions.

ENTER—Make Stack Frame for Procedure Parameters

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
C8 iw 00	ENTER imm16,0	Valid	Valid	Create a stack frame for a procedure
C8 iw 01	ENTER imm16,1	Valid	Valid	Create a nested stack frame for a procedure
C8 <i>iw</i> ib	ENTER imm16,imm8	Valid	Valid	Create a nested stack frame for a procedure

Flags Affected

None.

IA-32e Mode Operation

Default Operation Size is 64 bits

In 64-bit mode a 32-bit operation size cannot be encoded.

Protected Mode Exceptions

#SS(0) If the new value of the SP or ESP register is outside the stack segment limit.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#SS(0) If the new value of the SP or ESP register is outside the stack segment limit.

Virtual-8086 Mode Exceptions

#SS(0) If the new value of the SP or ESP register is outside the stack segment limit.

#PF(fault-code) If a page fault occurs.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#PF(fault-code) If a page fault occurs.



F2XM1—Compute 2^x-1

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
D9 F0	F2XM1	Valid	Valid	Replace ST(0) with (2 ^{ST(0)} – 1)	

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow$ not

roundup; $1 \leftarrow roundup$.

C0, C2, C3 Undefined.

IA-32e Mode Operation

Same as legacy mode.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value or unsupported format.

#D Result is a denormal value.

#U Result is too small for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

Same as for protected mode exceptions.

FABS—Absolute Value

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
D9 E1	FABS	Valid	Valid	Replace ST with its absolute value.	

FPU Flags Affected

C1 Set to 0 if stack underflow occurred; otherwise, cleared to 0.

C0, C2, C3 Undefined.

IA-32e Mode Operation

Same as legacy mode.

Floating-Point Exceptions

#IS Stack underflow occurred.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

Same as for protected mode exceptions.



FADD/FADDP/FIADD—Add

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /0	FADD m32fp	Valid	Valid	Add m32fp to ST(0) and store result in ST(0)
DC /0	FADD m64fp	Valid	Valid	Add m64fp to ST(0) and store result in ST(0)
D8 C0+i	FADD ST(0), ST(i)	Valid	Valid	Add ST(0) to ST(i) and store result in ST(0)
DC C0+i	FADD ST(i), ST(0)	Valid	Valid	Add ST(i) to ST(0) and store result in ST(i)
DE C0+i	FADDP ST(i), ST(0)	Valid	Valid	Add ST(0) to ST(i), store result in ST(i), and pop the register stack
DE C1	FADDP	Valid	Valid	Add ST(0) to ST(1), store result in ST(1), and pop the register stack
DA /0	FIADD m32int	Valid	Valid	Add m32int to ST(0) and store result in ST(0)
DE /0	FIADD m16int	Valid	Valid	Add <i>m16int</i> to ST(0) and store result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow$ roundup.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Operand is an SNaN value or unsupported format.

Operands are infinities of unlike sign.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



FBLD—Load Binary Coded Decimal

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /4	FBLD m80 dec	Valid	Valid	Convert BCD value to floating-point and push onto the FPU stack.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 1 if stack overflow occurred; otherwise, cleared to 0.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack overflow occurred.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FBSTP—Store BCD Integer and Pop

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /6	FBSTP m80bcd	Valid	Valid	Store ST(0) in m80bcd and pop ST(0).

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact exception (#P) is generated: 0 = not roundup;

 $1 \leftarrow \text{roundup}$.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is empty; contains a NaN, ±∞, or unsupported format; or contains value that

exceeds 18 BCD digits in length.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a segment register is being loaded with a segment selector that points to a nonwritable

segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FCHS—Change Sign

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 E0	FCHS	Valid	Valid	Complements sign of ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred; otherwise, cleared to 0.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FCLEX/FNCLEX—Clear Exceptions

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B DB E2	FCLEX	Valid	Valid	Clear floating-point exception flags after checking for pending unmasked floating-point exceptions.
DB E2	FNCLEX*	Valid	Valid	Clear floating-point exception flags without checking for pending unmasked floating-point exceptions.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

The PE, UE, OE, ZE, DE, IE, ES, SF, and B flags in the FPU status word are cleared. The C0, C1, C2, and C3 flags are undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FCMOVcc—Floating-Point Conditional Move

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DA C0+i	FCMOVB ST(0), ST(i)	Valid	Valid	Move if below (CF=1)
DA C8+i	FCMOVE ST(0), ST(i)	Valid	Valid	Move if equal (ZF=1)
DA D0+i	FCMOVBE ST(0), ST(i)	Valid	Valid	Move if below or equal (CF=1 or ZF=1)
DA D8+i	FCMOVU ST(0), ST(i)	Valid	Valid	Move if unordered (PF=1)
DB C0+i	FCMOVNB ST(0), ST(i)	Valid	Valid	Move if not below (CF=0)
DB C8+i	FCMOVNE ST(0), ST(i)	Valid	Valid	Move if not equal (ZF=0)
DB D0+i	FCMOVNBE ST(0), ST(i)	Valid	Valid	Move if not below or equal (CF=0 and ZF=0)
DB D8+i	FCMOVNU ST(0), ST(i)	Valid	Valid	Move if not unordered (PF=0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

Integer Flags Affected

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FCOM/FCOMP/FCOMPP—Compare Floating Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /2	FCOM m32fp	Valid	Valid	Compare ST(0) with m32fp.
DC /2	FCOM m64fp	Valid	Valid	Compare ST(0) with <i>m64fp</i> .
D8 D0+i	FCOM ST(i)	Valid	Valid	Compare ST(0) with ST(i).
D8 D1	FCOM	Valid	Valid	Compare ST(0) with ST(1).
D8 /3	FCOMP m32fp	Valid	Valid	Compare ST(0) with <i>m32fp</i> and pop register stack.
DC /3	FCOMP m64fp	Valid	Valid	Compare ST(0) with m64fp and pop register stack.
D8 D8+i	FCOMP ST(i)	Valid	Valid	Compare ST(0) with ST(i) and pop register stack.
D8 D9	FCOMP	Valid	Valid	Compare ST(0) with ST(1) and pop register stack.
DE D9	FCOMPP	Valid	Valid	Compare ST(0) with ST(1) and pop register stack twice.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Cleared to 0. C0, C2, C3 See table below.

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

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA One or both operands are NaN values or have unsupported formats.

Register is marked empty.

#D One or both operands are denormal values.



Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FCOMI/FCOMIP/FUCOMI/FUCOMIP—Compare Floating Point Values and Set EFLAGS

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DB F0+i	FCOMI ST, ST(i)	Valid	Valid	Compare ST(0) with ST(i) and set status flags accordingly
DF F0+i	FCOMIP ST, ST(i)	Valid	Valid	Compare ST(0) with ST(i), set status flags accordingly, and pop register stack
DB E8+i	FUCOMI ST, ST(i)	Valid	Valid	Compare ST(0) with ST(i), check for ordered values, and set status flags accordingly
DF E8+i	FUCOMIP ST, ST(i)	Valid	Valid	Compare ST(0) with ST(i), check for ordered values, set status flags accordingly, and pop register stack

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Cleared to 0. C0, C2, C3 Not affected.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA (FCOMI or FCOMIP instruction) One or both operands are NaN values or have unsup-

ported formats.

(FUCOMI or FUCOMIP instruction) One or both operands are SNaN values (but not QNaNs) or have undefined formats. Detection of a QNaN value does not raise an invalid-

operand exception.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#MF If there is a pending x87 FPU exception.

#NM EM or TS in CR0 is set.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FCOS—Cosine

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 FF	FCOS	Valid	Valid	Replace ST(0) with its cosine

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

 $roundup; \ 1 \leftarrow roundup.$

Undefined if C2 is 1.

C2 Set to 1 if source operand is outside the range -2^{63} to $+2^{63}$; otherwise, cleared to 0.

C0, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value, ∞, or unsupported format.

#D Result is a denormal value.

#U Result is too small for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FDECSTP—Decrement Stack-Top Pointer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	Ī
D9 F6	FDECSTP	Valid	Valid	Decrement TOP field in FPU status word.	

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

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

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FDIV/FDIVP/FIDIV—Divide

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /6	FDIV m32fp	Valid	Valid	Divide ST(0) by m32fp and store result in ST(0)
DC /6	FDIV m64fp	Valid	Valid	Divide ST(0) by m64fp and store result in ST(0)
D8 F0+i	FDIV ST(0), ST(i)	Valid	Valid	Divide ST(0) by ST(i) and store result in ST(0)
DC F8+i	FDIV ST(i), ST(0)	Valid	Valid	Divide ST(i) by ST(0) and store result in ST(i)
DE F8+i	FDIVP ST(i), ST(0)	Valid	Valid	Divide ST(i) by ST(0), store result in ST(i), and pop the register stack
DE F9	FDIVP	Valid	Valid	Divide ST(1) by ST(0), store result in ST(1), and pop the register stack
DA /6	FIDIV m32int	Valid	Valid	Divide ST(0) by m32int and store result in ST(0)
DE /6	FIDIV m16int	Valid	Valid	Divide ST(0) by <i>m64int</i> and store result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow$ roundup.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Operand is an SNaN value or unsupported format.

 $\pm \infty / \pm \infty$; $\pm 0 / \pm 0$

#D Result is a denormal value.

#Z DEST / ±0, where DEST is not equal to ±0.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#MF If there is a pending x87 FPU exception.

#NM EM or TS in CR0 is set. #PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



FDIVR/FDIVRP/FIDIVR—Reverse Divide

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /7	FDIVR m32fp	Valid	Valid	Divide m32fp by ST(0) and store result in ST(0)
DC /7	FDIVR m64fp	Valid	Valid	Divide m64fp by ST(0) and store result in ST(0)
D8 F8+i	FDIVR ST(0), ST(i)	Valid	Valid	Divide ST(i) by ST(0) and store result in ST(0)
DC F0+i	FDIVR ST(i), ST(0)	Valid	Valid	Divide ST(0) by ST(i) and store result in ST(i)
DE F0+i	FDIVRP ST(i), ST(0)	Valid	Valid	Divide ST(0) by ST(i), store result in ST(i), and pop the register stack
DE F1	FDIVRP	Valid	Valid	Divide ST(0) by ST(1), store result in ST(1), and pop the register stack
DA /7	FIDIVR m32int	Valid	Valid	Divide m32int by ST(0) and store result in ST(0)
DE /7	FIDIVR m16int	Valid	Valid	Divide m16int by ST(0) and store result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow$ roundup.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Operand is an SNaN value or unsupported format.

 $\pm \infty / \pm \infty$; $\pm 0 / \pm 0$

#D Result is a denormal value.

#Z SRC / ±0, where SRC is not equal to ±0.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FFREE—Free Floating-Point Register

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DD C0+i	FFREE ST(i)	Valid	Valid	Sets tag for ST(i) to empty

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C0, C1, C2, C3 undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FICOM/FICOMP—Compare Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DE /2	FICOM m16int	Valid	Valid	Compare ST(0) with m16int
DA /2	FICOM m32int	Valid	Valid	Compare ST(0) with m32int
DE /3	FICOMP m16int	Valid	Valid	Compare ST(0) with <i>m16int</i> and pop stack register
DA /3	FICOMP m32int	Valid	Valid	Compare ST(0) with <i>m32int</i> and pop stack register

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0.

C0, C2, C3 See table on FCOM instruction.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA One or both operands are NaN values or have unsupported formats.

#D One or both operands are denormal values.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.



Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FILD—Load Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF/0	FILD m16int	Valid	Valid	Push m16int onto the FPU register stack.
DB /0	FILD m32int	Valid	Valid	Push m32int onto the FPU register stack.
DF /5	FILD m64int	Valid	Valid	Push <i>m64int</i> onto the FPU register stack.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 1 if stack overflow occurred; cleared to 0 otherwise.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack overflow occurred.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FINCSTP—Increment Stack-Top Pointer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F7	FINCSTP	Valid	Valid	Increment the TOP field in the FPU status register

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

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

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FINIT/FNINIT—Initialize Floating-Point Unit

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B DB E3	FINIT	Valid	Valid	Initialize FPU after checking for pending unmasked floating-point exceptions.
DB E3	FNINIT*	Valid	Valid	Initialize FPU without checking for pending unmasked floating-point exceptions.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C0, C1, C2, C3 cleared to 0.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FIST/FISTP—Store Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /2	FIST m16int	Valid	Valid	Store ST(0) in m16int
DB /2	FIST m32int	Valid	Valid	Store ST(0) in m32int
DF /3	FISTP m16int	Valid	Valid	Store ST(0) in m16int and pop register stack
DB /3	FISTP m32int	Valid	Valid	Store ST(0) in m32int and pop register stack
DF /7	FISTP m64int	Valid	Valid	Store ST(0) in m64int and pop register stack

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction of if the inexact exception (#P) is generated: $0 \leftarrow$ not roundup;

 $1 \leftarrow \text{roundup}$.

Cleared to 0 otherwise.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is too large for the destination format

Source operand is a NaN value or unsupported format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.



Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FISTTP—Store Integer with Truncation

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DF /1	FISTTP m16int	Valid	Valid	Store ST(0) in m16int with truncation
DB /1	FISTTP m32int	Valid	Valid	Store ST(0) in m32int with truncation
DD /1	FISTTP m64int	Valid	Valid	Store ST(0) in <i>m64int</i> with truncation

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Cleared to 0. C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is too large for the destination format

Source operand is a NaN value or unsupported format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

For an illegal memory operand effective address in the CS, DS, ES, FS, or GS segment.

#SS(0) For an illegal address in the SS segment.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

#UD If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.



Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FLD—Load Floating Point Value

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 /0	FLD m32fp	Valid	Valid	Push m32fp onto the FPU register stack.
DD /0	FLD m64fp	Valid	Valid	Push m64fp onto the FPU register stack.
DB /5	FLD m80fp	Valid	Valid	Push m80fp onto the FPU register stack.
D9 C0+i	FLD ST(i)	Valid	Valid	Push ST(i) onto the FPU register stack.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 1 if stack overflow occurred; otherwise, cleared to 0.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack overflow occurred.

#IA Source operand is an SNaN value or unsupported format. Does not occur if the source

operand is in double extended-precision floating-point format.

#D Source operand is a denormal value. Does not occur if the source operand is in double

extended-precision floating-point format.

Protected Mode Exceptions

#GP(0) If destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.



Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ—Load Constant

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 E8	FLD1	Valid	Valid	Push +1.0 onto the FPU register stack.
D9 E9	FLDL2T	Valid	Valid	Push log ₂ 10 onto the FPU register stack.
D9 EA	FLDL2E	Valid	Valid	Push log ₂ e onto the FPU register stack.
D9 EB	FLDPI	Valid	Valid	Push π onto the FPU register stack.
D9 EC	FLDLG2	Valid	Valid	Push log ₁₀ 2 onto the FPU register stack.
D9 ED	FLDLN2	Valid	Valid	Push log _e 2 onto the FPU register stack.
D9 EE	FLDZ	Valid	Valid	Push +0.0 onto the FPU register stack.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 1 if stack overflow occurred; otherwise, cleared to 0.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack overflow occurred.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

Same as for protected mode exceptions.

IA-32 Architecture Compatibility

When the RC field is set to round-to-nearest, the FPU produces the same constants that is produced by the Intel 8087 and Intel 287 math coprocessors.



FLDCW—Load x87 FPU Control Word

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 /5	FLDCW m2byte	Valid	Valid	Load FPU control word from m2byte.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C0, C1, C2, C3 undefined.

Floating-Point Exceptions

None; however, this operation might unmask a pending exception in the FPU status word. That exception is then generated upon execution of the next "waiting" floating-point instruction.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FLDENV—Load x87 FPU Environment

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 /4	FLDENV m14/28byte	Valid	Valid	Load FPU environment from <i>m14byte</i> or <i>m28byte</i> .

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

The C0, C1, C2, C3 flags are loaded.

Floating-Point Exceptions

None; however, if an unmasked exception is loaded in the status word, it is generated upon execution of the next "waiting" floating-point instruction.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



FMUL/FMULP/FIMUL—Multiply

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /1	FMUL m32fp	Valid	Valid	Multiply ST(0) by m32fp and store result in ST(0)
DC /1	FMUL m64fp	Valid	Valid	Multiply ST(0) by m64fp and store result in ST(0)
D8 C8+i	FMUL ST(0), ST(i)	Valid	Valid	Multiply ST(0) by ST(i) and store result in ST(0)
DC C8+i	FMUL ST(i), ST(0)	Valid	Valid	Multiply ST(i) by ST(0) and store result in ST(i)
DE C8+i	FMULP ST(i), ST(0)	Valid	Valid	Multiply ST(i) by ST(0), store result in ST(i), and pop the register stack
DE C9	FMULP	Valid	Valid	Multiply ST(1) by ST(0), store result in ST(1), and pop the register stack
DA /1	FIMUL m32int	Valid	Valid	Multiply ST(0) by m32int and store result in ST(0)
DE /1	FIMUL m16int	Valid	Valid	Multiply ST(0) by m16int and store result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) fault is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow$ roundup.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Operand is an SNaN value or unsupported format.

One operand is ± 0 and the other is $\pm \infty$.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



FNOP—No Operation

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 D0	FNOP	Valid	Valid	No operation is performed.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C0, C1, C2, C3 undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FPATAN—Partial Arctangent

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
D9 F3	FPATAN	Valid	Valid	Replace $ST(1)$ with $arctan(ST(1)/ST(0))$ and pop the register stack	

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow \text{roundup}$.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value or unsupported format.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FPREM—Partial Remainder

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F8	FPREM	Valid	Valid	Replace ST(0) with the remainder obtained from dividing ST(0) by ST(1)

IA-32e Mode Operation

Same as legacy mode.

FPU 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.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value, modulus is 0, dividend is ∞, or unsupported format.

#D Source operand is a denormal value.

#U Result is too small for destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FPREM1—Partial Remainder

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F5	FPREM1	Valid	Valid	Replace ST(0) with the IEEE remainder obtained from dividing ST(0) by ST(1)

IA-32e Mode Operation

Same as legacy mode.

FPU 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.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value, modulus (divisor) is 0, dividend is ∞, or unsupported

format.

#U Source operand is a denormal value.

#U Result is too small for destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FPTAN—Partial Tangent

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F2	FPTAN	Valid	Valid	Replace ST(0) with its tangent and push 1 onto the FPU stack.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred; set to 1 if stack overflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow \text{roundup}$.

C2 Set to 1 if source operand is outside the range -2^{63} to $+2^{63}$; otherwise, cleared to 0.

C0, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow or overflow occurred.

#IA Source operand is an SNaN value, ∞, or unsupported format.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FRNDINT—Round to Integer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 FC	FRNDINT	Valid	Valid	Round ST(0) to an integer.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow \text{roundup}$.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value or unsupported format.

#D Source operand is a denormal value.

#P Source operand is not an integral value.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FRSTOR—Restore x87 FPU State

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DD /4	FRSTOR m94/108byte	Valid	Valid	Load FPU state from m94byte or m108byte.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

The C0, C1, C2, C3 flags are loaded.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FSAVE/FNSAVE—Store x87 FPU State

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B DD /6	FSAVE m94/108byte	Valid	Valid	Store FPU state to <i>m94byte</i> or <i>m108byte</i> after checking for pending unmasked floating-point exceptions. Then re-initialize the FPU.
DD /6	FNSAVE* m94/108byte	Valid	Valid	Store FPU environment to m94byte or m108byte without checking for pending unmasked floating-point exceptions. Then re-initialize the FPU.

FPU Flags Affected

The C0, C1, C2, and C3 flags are saved and then cleared.

IA-32e Mode Operation

Same as legacy mode.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0) If destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FSCALE—Scale

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 FD	FSCALE	Valid	Valid	Scale ST(0) by ST(1).

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow \text{roundup}$.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value or unsupported format.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FSIN—Sine

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
D9 FE	FSIN	Valid	Valid	Replace ST(0) with its sine.	

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow roundup$.

C2 Set to 1 if source operand is outside the range -2^{63} to $+2^{63}$; otherwise, cleared to 0.

C0, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value, ∞, or unsupported format.

#D Source operand is a denormal value.

#P Value cannot be represented exactly in destination format.

#U Result is too small for destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FSINCOS—Sine and Cosine

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 FB	FSINCOS	Valid	Valid	Compute the sine and cosine of ST(0); replace ST(0) with the sine, and push the cosine onto the register stack.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred; set to 1 of stack overflow occurs.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

roundup; $1 \leftarrow \text{roundup}$.

C2 Set to 1 if source operand is outside the range -2^{63} to $+2^{63}$; otherwise, cleared to 0.

C0, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow or overflow occurred.

#IA Source operand is an SNaN value, ∞, or unsupported format.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FSQRT—Square Root

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 FA	FSQRT	Valid	Valid	Computes square root of ST(0) and stores the result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if inexact-result exception (#P) is generated: $0 \leftarrow$ not roundup;

 $1 \leftarrow \text{roundup}$.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value or unsupported format.

Source operand is a negative value (except for -0).

#D Source operand is a denormal value.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FST/FSTP—Store Floating Point Value

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 /2	FST m32fp	Valid	Valid	Copy ST(0) to m32fp
DD /2	FST m64fp	Valid	Valid	Copy ST(0) to m64fp
DD D0+i	FST ST(i)	Valid	Valid	Copy ST(0) to ST(i)
D9 /3	FSTP m32fp	Valid	Valid	Copy ST(0) to m32fp and pop register stack
DD /3	FSTP m64fp	Valid	Valid	Copy ST(0) to m64fp and pop register stack
DB /7	FSTP m80fp	Valid	Valid	Copy ST(0) to m80fp and pop register stack
DD D8+i	FSTP ST(i)	Valid	Valid	Copy ST(0) to ST(i) and pop register stack

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction of if the floating-point inexact exception (#P) is generated: 0

 \leftarrow not roundup; $1 \leftarrow$ roundup.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Source operand is an SNaN value or unsupported format. Does not occur if the destination

operand is in double extended-precision floating-point format.

#U Result is too small for the destination format. Does not occur if the destination operand is

in double extended-precision floating-point format.

#O Result is too large for the destination format. Does not occur if the destination operand is in

double extended-precision floating-point format.

#P Value cannot be represented exactly in destination format. Does not occur if the destination

operand is in double extended-precision floating-point format.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FSTCW/FNSTCW—Store x87 FPU Control Word

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B D9 /7	FSTCW m2byte	Valid	Valid	Store FPU control word to <i>m2byte</i> after checking for pending unmasked floating-point exceptions.
D9 /7	FNSTCW* m2byte	Valid	Valid	Store FPU control word to <i>m2byte</i> without checking for pending unmasked floating-point exceptions.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

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

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FSTENV/FNSTENV—Store x87 FPU Environment

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B D9 /6	FSTENV m14/28byte	Valid	Valid	Store FPU environment to <i>m14byte</i> or <i>m28byte</i> after checking for pending unmasked floating-point exceptions. Then mask all floating-point exceptions.
D9 /6	FNSTENV* m14/28byte	Valid	Valid	Store FPU environment to <i>m14byte</i> or <i>m28byte</i> without checking for pending unmasked floating-point exceptions. Then mask all floating-point exceptions.

FPU Flags Affected

The C0, C1, C2, and C3 are undefined.

Floating-Point Exceptions

None.

IA-32e Mode Operation

Same as legacy mode.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FSTSW/FNSTSW—Store x87 FPU Status Word

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9B DD /7	FSTSW m2byte	Valid	Valid	Store FPU status word at <i>m2byte</i> after checking for pending unmasked floating-point exceptions.
9B DF E0	FSTSW AX	Valid	Valid	Store FPU status word in AX register after checking for pending unmasked floating-point exceptions.
DD /7	FNSTSW* m2byte	Valid	Valid	Store FPU status word at <i>m2byte</i> without checking for pending unmasked floating-point exceptions.
DF E0	FNSTSW* AX	Valid	Valid	Store FPU status word in AX register without checking for pending unmasked floating-point exceptions.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

The C0, C1, C2, and C3 are undefined.

Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0) If the destination is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

FSUB/FSUBP/FISUB—Subtract

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /4	FSUB m32fp	Valid	Valid	Subtract <i>m32fp</i> from ST(0) and store result in ST(0)
DC /4	FSUB m64fp	Valid	Valid	Subtract <i>m64fp</i> from ST(0) and store result in ST(0)
D8 E0+i	FSUB ST(0), ST(i)	Valid	Valid	Subtract ST(i) from ST(0) and store result in ST(0)
DC E8+i	FSUB ST(i), ST(0)	Valid	Valid	Subtract ST(0) from ST(i) and store result in ST(i)
DE E8+i	FSUBP ST(i), ST(0)	Valid	Valid	Subtract ST(0) from ST(i), store result in ST(i), and pop register stack
DE E9	FSUBP	Valid	Valid	Subtract ST(0) from ST(1), store result in ST(1), and pop register stack
DA /4	FISUB m32int	Valid	Valid	Subtract <i>m32int</i> from ST(0) and store result in ST(0)
DE /4	FISUB m16int	Valid	Valid	Subtract <i>m16int</i> from ST(0) and store result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) fault is generated: $0 \leftarrow \text{not}$

 $roundup; \ 1 \leftarrow roundup.$

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Operand is an SNaN value or unsupported format.

Operands are infinities of like sign.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D8 /5	FSUBR m32fp	Valid	Valid	Subtract ST(0) from <i>m32fp</i> and store result in ST(0)
DC /5	FSUBR m64fp	Valid	Valid	Subtract ST(0) from <i>m64fp</i> and store result in ST(0)
D8 E8+i	FSUBR ST(0), ST(i)	Valid	Valid	Subtract ST(0) from ST(i) and store result in ST(0)
DC E0+i	FSUBR ST(i), ST(0)	Valid	Valid	Subtract ST(i) from ST(0) and store result in ST(i)
DE E0+i	FSUBRP ST(i), ST(0)	Valid	Valid	Subtract ST(i) from ST(0), store result in ST(i), and pop register stack
DE E1	FSUBRP	Valid	Valid	Subtract ST(1) from ST(0), store result in ST(1), and pop register stack
DA /5	FISUBR m32int	Valid	Valid	Subtract ST(0) from <i>m32int</i> and store result in ST(0)
DE /5	FISUBR m16int	Valid	Valid	Subtract ST(0) from $m16int$ and store result in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) fault is generated: $0 \leftarrow \text{not}$

 $roundup; \ 1 \leftarrow roundup.$

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Operand is an SNaN value or unsupported format.

Operands are infinities of like sign.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the



FTST—TEST

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 E4	FTST	Valid	Valid	Compare ST(0) with 0.0.

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred; otherwise, cleared to 0.

C0, C2, C3 See above table.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA The source operand is a NaN value or is in an unsupported format.

#D The source operand is a denormal value.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FUCOM/FUCOMP/FUCOMPP—Unordered Compare Floating Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DD E0+i	FUCOM ST(i)	Valid	Valid	Compare ST(0) with ST(i)
DD E1	FUCOM	Valid	Valid	Compare ST(0) with ST(1)
DD E8+i	FUCOMP ST(i)	Valid	Valid	Compare ST(0) with ST(i) and pop register stack
DD E9	FUCOMP	Valid	Valid	Compare ST(0) with ST(1) and pop register stack
DA E9	FUCOMPP	Valid	Valid	Compare ST(0) with ST(1) and pop register stack twice

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

C0, C2, C3 See table on previous page.

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA One or both operands are SNaN values or have unsupported formats. Detection of a QNaN

value in and of itself does not raise an invalid-operand exception.

#D One or both operands are denormal values.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FWAIT—Wait

See entry for WAIT/FWAIT—Wait.

FXAM—Examine

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 E5	FXAM	Valid	Valid	Classify value or number in ST(0)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Sign of value in ST(0).

C0, C2, C3 See table above.

Class	С3	C2	C0
Unsupported	0	0	0
NaN	0	0	1
Normal finite number	0	1	0
Infinity	0	1	1
Zero	1	0	0
Empty	1	0	1
Denormal number	1	1	0

Floating-Point Exceptions

None.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FXCH—Exchange Register Contents

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 C8+i	FXCH ST(i)	Valid	Valid	Exchange the contents of ST(0) and ST(i)
D9 C9	FXCH	Valid	Valid	Exchange the contents of ST(0) and ST(1)

IA-32e Mode Operation

Same as legacy mode.

FPU Flags Affected

C1 Cleared to 0. C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FXRSTOR—Restore x87 FPU, MMX, SSE, and SSE2 State

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
OF AE /1	FXRSTOR m512byte	Valid	Valid	Restore the x87 FPU, MMX, XMM, and MXCSR register state from <i>m512byte</i> .

x87 FPU and SIMD Floating-Point Exceptions

None.

IA-32e Mode Operation

See FXSAVE for IA-32e Mode save/restore format.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment. (See align-

ment check exception [#AC] below.)

If attempting to set a reserved bit in MXCSR.

#SS(0) For an illegal address in the SS segment.

#MF If there is a pending x87 FPU exception.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If CPUID feature flag FXSR is 0.

If instruction is preceded by a LOCK prefix.

#AC If this exception is disabled a general protection exception (#GP) is signaled if the memory

operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte

misalignments).

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#NM If TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

#UD If EM in CR0 is set.

If CPUID feature flag FXSR is 0.

If instruction is preceded by a LOCK override prefix.



Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

#AC For unaligned memory reference if the current privilege level is 3.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#MF If there is a pending x87 FPU exception.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If CPUID feature flag FXSR is 0.

If instruction is preceded by a LOCK prefix.

#AC If this exception is disabled a general protection exception (#GP) is signaled if the memory

operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte

misalignments).

FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F AE /0	FXSAVE m512byte	Valid	Valid	Save the x87 FPU, MMX, XMM, and MXCSR register state to <i>m512byte</i> .

Description

Saves the current state of the x87 FPU, MMX, XMM, and MXCSR registers to a 512-byte memory location specified in the destination operand. Table 2-12 shows the layout of the state information in memory in legacy mode.

Table 2-12 Layout of the Legacy FXSAVE Map

			1	Table	2-12 1	Layout	of the Legac	, y 1 A3/	AVE IVI	aμ				
15 14	13	12	11	10	9	8	7 6	5	4	3	2	1	0	
Reserved	C	S		FPU	J IP		FOP		FTW	FSV	٧	F	CW	0
MXCSR	R_MASK			MXC	CSR		Reserved		s		FPU	J DP		16
	Rese	rved						ST0/	MM0					32
	Rese	rved						ST1/	MM1					48
	Rese	rved						ST2/	MM2					64
	Rese	rved						ST3/	′ММЗ					80
	Rese	rved						ST4/	MM4					96
	Rese	rved						ST5/	MM5					112
	Rese	rved						ST6/	MM6					128
	Rese	rved						ST7/	MM7					144
						XM	IM0							160
						XM	IM1							176
						XM	IM2							192
						XM	IM3							208
	XMM4							224						
						XM	IM5							240
						XM	IM6							256
						XM	IM7							272
						Rese	erved							288
						Rese	erved							304
						Rese	erved							320
						Rese	erved							336
						Rese	erved							352
						Rese	erved							368
						Rese	erved							384
						Rese	erved							400
	Reserved							416						
						Rese	erved							432
						Rese	erved							448
	Reserved									464				
						Rese	erved							480
						Rese	erved							496



The destination operand contains the first byte of the memory image, and it must be aligned on a 16-byte boundary. A misaligned destination operand will result in a general-protection (#GP) exception being generated (or in some cases, an alignment check exception [#AC]).

The FXSAVE instruction is used when an operating system needs to perform a context switch or when an exception handler needs to save and examine the current state of the x87 FPU, MMX, and/or XMM and MXCSR registers. The fields in Table 2-12 are as follows:

FCW x87 FPU Control Word (16 bits). See Figure 8-6 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for the layout of the x87 FPU control word.

FSW x87 FPU Status Word (16 bits). See Figure 8-4 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for the layout of the x87 FPU status word.

x87 FPU Tag Word (8 bits). The tag information saved here is abridged, as described in the following paragraphs. See Figure 8-7 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for the layout of the x87 FPU tag word.

x87 FPU Opcode (16 bits). The lower 11 bits of this field contain the opcode, upper 5 bits are reserved. See Figure 8-8 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for the layout of the x87 FPU opcode field.

x87 FPU Instruction Pointer Offset (32 bits). The contents of this field differ depending on the current addressing mode (32-bit or 16-bit) of the processor when the FXSAVE instruction was executed:

- 32-bit mode—32-bit IP offset.
- 16-bit mode—low 16 bits are IP offset; high 16 bits are reserved.

See "x87 FPU Instruction and Operand (Data) Pointers" in Chapter 8 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for a description of the x87 FPU instruction pointer.

x87 FPU Instruction Pointer Selector (16 bits).

x87 FPU Instruction Operand (Data) Pointer Offset (32 bits). The contents of this field differ depending on the current addressing mode (32-bit or 16-bit) of the processor when the FXSAVE instruction was executed:

- 32-bit mode—32-bit IP offset.
- 16-bit mode—low 16 bits are IP offset; high 16 bits are reserved.

See "x87 FPU Instruction and Operand (Data) Pointers" in Chapter 8 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for a description of the x87 FPU operand pointer.

x87 FPU Instruction Operand (Data) Pointer Selector (16 bits).

MXCSR Register State (32 bits). See Figure 10-3 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for the layout of the MXCSR register. If the OSFXSR bit in control register CR4 is not set, the FXSAVE instruction may not save this register. This behavior is implementation dependent.

MXCSR_MASK (32 bits). This value can be used to adjust a value to be written to the MXCSR register to ensure that all reserved bits are set to 0. Setting the reserved bits to 0 prevents a general-protection exception (#GP) from being generated when writing to the MXCSR register with an FXRSTOR or LDMXCSR instruction. See "Guidelines for Writing to the MXCSR Register" in Chapter 11 of the *IA-32 Intel Architecture Software Developer's Manual, Volume 1*, for instructions for how to determine and use the MXCSR_MASK value.

x87 FPU or MMX registers. These 80-bit fields contain the x87 FPU data registers or the MMX registers, depending on the state of the processor prior to the execution of the FXSAVE instruction. If the processor had been executing x87 FPU instruction prior to the FXSAVE instruction, the x87 FPU data registers are saved; if it had been executing MMX instructions (or SSE or SSE2 instructions that operated on the MMX registers), the MMX registers are saved. When the MMX registers are saved, the high 16-bits of the field are reserved.

CS

FPU DP

FTW

FOP

FPU IP

DS

MXCSR_ MASK

MXCSR

through ST7/ MM7

ST0/MM0

XMM0 through XMM registers (128 bits per field). If the OSFXSR bit in control register CR4 is not set, the XMM7 FXSAVE instruction may not save these registers. This behavior is implementation dependent.

The FXSAVE instruction saves an abridged version of the x87 FPU tag word in the FTW field (unlike the FSAVE instruction, which saves the complete tag word). The tag information is saved in physical register order (R0 through R7), rather than in top-of-stack (TOS) order. With the FXSAVE instruction, however, only a single bit (1 for valid or 0 for empty) is saved for each tag. For example, assume that the tag word is currently set as follows:

R7	R6	R5	R4	R3	R2	R1	R0
11	XX	XX	XX	11	11	11	11

Here, 11B indicates empty stack elements and "xx" indicates valid (00B), zero (01B), or special (10B).

For this example, the FXSAVE instruction saves only the following 8-bits of information:

R7	R6	R5	R4	R3	R2	R1	R0
0	1	1	1	0	0	0	0

Here, a 1 is saved for any valid, zero, or special tag, and a 0 is saved for any empty tag.

The operation of the FXSAVE instruction differs from that of the FSAVE instruction, the as follows:

- FXSAVE instruction does not check for pending unmasked floating-point exceptions. (The FXSAVE operation in this regard is similar to the operation of the FNSAVE instruction).
- After the FXSAVE instruction has saved the state of the x87 FPU, MMX, XMM, and MXCSR registers, the processor retains the contents of the registers. Because of this behavior, the FXSAVE instruction cannot be used by an application program to pass a "clean" x87 FPU state to a procedure, since it retains the current state. To clean the x87 FPU state, an application must explicitly execute an FINIT instruction after an FXSAVE instruction to reinitialize the x87 FPU state.
- The format of the memory image saved with the FXSAVE instruction is the same regardless of the current addressing mode (32-bit or 16-bit) and operating mode (protected, real address, or system management). This behavior differs from the FSAVE instructions, where the memory image format is different depending on the addressing mode and operating mode. Because of the different image formats, the memory image saved with the FXSAVE instruction cannot be restored correctly with the FRSTOR instruction, and likewise the state saved with the FSAVE instruction cannot be restored correctly with the FXRSTOR instruction.

Note that The FSAVE format for FTW can be recreated from the FTW valid bits and the stored 80-bit FP data (assuming the stored data was not the contents of MMX registers) using the following table:

Table 2-13 State Save Map

Exponent all 1's	Exponent all 0's	Fraction all 0's	J and M bits	FTW valid bit	x87 F	TW
0	0	0	0x	1	Special	10
0	0	0	1x	1	Valid	00
0	0	1	00	1	Special	10
0	0	1	10	1	Valid	00
0	1	0	0x	1	Special	10
0	1	0	1x	1	Special	10
0	1	1	00	1	Zero	01
0	1	1	10	1	Special	10
1	0	0	1x	1	Special	10
1	0	0	1x	1	Special	10
1	0	1	00	1	Special	10
1	0	1	10	1	Special	10
For all legal combir	nations above		•	0	Empty	11

The J-bit is defined to be the 1-bit binary integer to the left of the decimal place in the significand. The M-bit is defined to be the most significant bit of the fractional portion of the significand (i.e., the bit immediately to the right of the decimal place).

When the M- bit is the most significant bit of the fractional portion of the significand, it must be 0 if the fraction is all 0's.

Operation

DEST ← Save(x87 FPU, MMX, XXM7-XMM0, MXCSR);

IA-32e Mode Operation

In IA-32e mode there are two fxsave/fxrstor formats which differ depending on whether the processor is in compatibility mode or 64-bit mode. In 64-bit mode, all of the SSE registers, XMM0 through XMM15, are saved. In compatibility mode, legacy SSE registers, XMM0 through XMM7, are saved according to the legacy FXSAVE map. In 64-bit mode, the layout of the 64-bit FXSAVE map has two flavors, depending on the value of the REX.W bit. The difference of these two flavors is in the FPU IP and FPU DP pointers. When REX.W = 0, the FPU IP is saved as CS with the 32 bit IP, and the FPU DP is saved as DS with the 32 bit DP. When REX.W = 1, the FPU IP and FPU DP are both 64 bit values without and segment selectors. The IA-32e Mode save formats are shown in Table 2-14 and Table 2-15 listed below.

Table 2-14 Layout of the 64-bit FXSAVE Map When REX.W Is Set

15 14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
		FPl	J IP				FC	FOP FTW FSW FCW						CW	0
MXCSR	MXCSR_MASK MXC				CSR	SR FPU DP							16		
	Rese	eserved ST0/MM0					ST0/MM0								32
	Reserved				ST1/MM1							48			
	Reserved								ST2/	MM2					64
	Reserved								ST3/	′ММЗ					80
	Reserved					ST4/MM4							96		
	Reserved								ST5/	MM5					112

Table 2-14 Layout of the 64-bit FXSAVE Map When REX.W Is Set (Contd.)

15 14	1	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Reserve				-	-	1			MM6			<u> </u>		128
	Reserve									MM7					144
						XIV	1M0								160
							1M1								176
							1M2								192
							1M3								208
							1M4								224
							1M5								240
							1M6								256
							1M7								272
							1M8								288
							1M9								304
							M10								320
							M11								336
							M12								352
							M13								368
							M14								384
							M15								400
							erved								416
							erved								432
							erved								448
							erved								464
							erved								480
							erved								496
						1 1030	J. 700								400

Table 2-15 Layout of the 64-bit FXSAVE Map When REX.W Is Clear

								•								
15 14	13 12	11	10	9	8	7	6	5	4	3	2	1	0			
Reserved	CS		FP	J IP	FOP)P		FTW	FSW		FCW		0		
MXCSF	R_MASK		MX	CSR	R Reserved DS						16					
	Reserved							ST0/	MM0					32		
	Reserved							ST1/	MM1					48		
	Reserved							ST2/	MM2					64		
	Reserved ST3/MM3								ST3/MM3							
	Reserved							ST4/	MM4					96		
	Reserved							ST5/	MM5					112		
	Reserved				ST6/MM6									128		
	Reserved	Reserved						ST7/	MM7					144		
					XIV	IM0								160		
	-				XMM1									176		
					XMM2								192			
					XIV	IM3								208		



Table 2-15 Layout of the 64-bit FXSAVE Map When REX.W Is Clear

15 14							- 1					, .				
XMM5 240 XMM6 256 XMM7 272 XMM8 288 XMM9 304 XMM10 320 XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 448 Reserved 448 Reserved 464 Reserved 480		0	1	2	3	4	5	6	7	8	9	10	11	12	13	15 14
XMM6 256 XMM7 272 XMM8 288 XMM9 304 XMM10 320 XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 464 Reserved 480	224		-						M4	XM	,					
XMM7 272 XMM8 288 XMM9 304 XMM10 320 XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 464 Reserved 480	240		XMM5													
XMM8 288 XMM9 304 XMM10 320 XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	256		XMM6													
XMM9 304 XMM10 320 XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	272								M7	XM						
XMM10 320 XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	288								M8	XM						
XMM11 336 XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	304								M9	XM						
XMM12 352 XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	320								V110	XMI						
XMM13 368 XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	336		XMM11													
XMM14 384 XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	352								V112	XMI						
XMM15 400 Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	368								V113	XMI						
Reserved 416 Reserved 432 Reserved 448 Reserved 464 Reserved 480	384								M14	XMI						
Reserved 432 Reserved 448 Reserved 464 Reserved 480	400								M15	XMI						
Reserved 448 Reserved 464 Reserved 480	416								rved	Rese						
Reserved 464 Reserved 480	432								erved	Rese						
Reserved 480	448								erved	Rese						
	464								erved	Rese						
Reserved 496	480								erved	Rese						
	496								erved	Rese						

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment. (See the

description of the alignment check exception [#AC] below.)

#SS(0) For an illegal address in the SS segment.

#MF If there is a pending x87 FPU exception.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If CPUID feature flag FXSR is 0.

If instruction is preceded by a LOCK override prefix.

#AC If this exception is disabled a general protection exception (#GP) is signaled if the memory

operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte

misalignments).

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

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

#MF If there is a pending x87 FPU exception.

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If CPUID feature flag FXSR is 0.

If instruction is preceded by a LOCK override prefix.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#MF If there is a pending x87 FPU exception.

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

#AC For unaligned memory reference if the current privilege level is 3.

Compatibility Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#MF If there is a pending x87 FPU exception.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If CPUID feature flag FXSR is 0.

If instruction is preceded by a LOCK prefix.

#AC If this exception is disabled a general protection exception (#GP) is signaled if the memory

operand is not aligned on a 16-byte boundary, as described above. If the alignment check exception (#AC) is enabled (and the CPL is 3), signaling of #AC is not guaranteed and may vary with implementation, as follows. In all implementations where #AC is not signaled, a general protection exception is signaled in its place. In addition, the width of the alignment check may also vary with implementation. For instance, for a given implementation, an alignment check exception might be signaled for a 2-byte misalignment, whereas a general protection exception might be signaled for all other misalignments (4-, 8-, or 16-byte

misalignments).

Implementation Note

The order in which the processor signals general-protection (#GP) and page-fault (#PF) exceptions when they both occur on an instruction boundary is given in Table 5-2 in the *IA-32 Intel Architecture Software Developer's Manual, Volume 3*. This order vary for the FXSAVE instruction for different IA-32 processor implementations.

FXTRACT—Extract Exponent and Significand

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F4	FXTRACT	Valid	Valid	Separate value in ST(0) into exponent and significand, store exponent in ST(0), and push the significand onto the register stack.

IA-32e Mode Operation

Same as legacy mode

FPU Flags Affected

C1 Set to 0 if stack underflow occurred; set to 1 if stack overflow occurred.

C0, C2, C3 Undefined.

Floating-Point Exceptions

#IS Stack underflow occurred.

Stack overflow occurred.

#IA Source operand is an SNaN value or unsupported format.

#Z ST(0) operand is ± 0 .

#D Source operand is a denormal value.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



FYL2X—Compute y * log₂x

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F1	FYL2X	Valid	Valid	Replace ST(1) with (ST(1) * log ₂ ST(0)) and pop the register stack

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow$ not

roundup; $1 \leftarrow roundup$.

C0, C2, C3 Undefined.

IA-32e Mode Operation

Same as legacy mode

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Either operand is an SNaN or unsupported format.

Source operand in register ST(0) is a negative finite value (not -0).

#Z Source operand in register ST(0) is ±0.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

FYL2XP1—Compute $y * log_2(x + 1)$

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
D9 F9	FYL2XP1	Valid	Valid	Replace ST(1) with ST(1) * $\log_2(ST(0) + 1.0)$ and pop the register stack

FPU Flags Affected

C1 Set to 0 if stack underflow occurred.

Indicates rounding direction if the inexact-result exception (#P) is generated: $0 \leftarrow \text{not}$

 $roundup; \ 1 \leftarrow roundup.$

C0, C2, C3 Undefined.

IA-32e Mode Operation

Same as legacy mode

Floating-Point Exceptions

#IS Stack underflow occurred.

#IA Either operand is an SNaN value or unsupported format.

#D Source operand is a denormal value.

#U Result is too small for destination format.

#O Result is too large for destination format.

#P Value cannot be represented exactly in destination format.

Protected Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Real-Address Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Virtual-8086 Mode Exceptions

#NM EM or TS in CR0 is set.

#MF If there is a pending x87 FPU exception.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



HADDPD—Horizontal Add Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 7C /r	HADDPD xmm1, xmm2/m128	Valid	Valid	Horizontal add packed double-precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault

Compatibility Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.



HADDPS—Horizontal Add Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 7C /r	HADDPS xmm1, xmm2/m128	Valid	Valid	Horizontal add packed single-precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault

Compatibility Mode Exceptions

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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

HLT—Halt

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F4	HLT	Valid	Valid	Halt

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

#GP(0) If the current privilege level is not 0.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

HSUBPD—Horizontal Subtract Packed Double-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
66 0F 7D /r	HSUBPD xmm1, xmm2/m128	Valid	Valid	Horizontal subtract packed double- precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault

Compatibility Mode Exceptions



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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

HSUBPS—Horizontal Subtract Packed Single-Precision Floating-Point Values

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F 7D /r	HSUBPS xmm1, xmm2/m128	Valid	Valid	Horizontal subtract packed single- precision floating-point values from xmm2/m128 to xmm1.

IA-32e Mode Operation

Enables access to XMM8-XMM15 registers.

SIMD Floating-Point Exceptions

Overflow, Underflow, Invalid, Precision, Denormal.

Protected Mode Exceptions

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#SS(0) For an illegal address in the SS segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Real-Address Mode Exceptions

#GP(0) If memory operand is not aligned on a 16-byte boundary, regardless of segment.

If any part of the operand lies outside the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault

Compatibility Mode Exceptions



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

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

If memory operand is not aligned on a 16-byte boundary, regardless of segment.

#PF(fault-code) For a page fault. #NM If TS in CR0 is set.

#XM If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 1.

#UD If an unmasked SIMD floating-point exception and OSXMMEXCPT in CR4 is 0.

If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

IDIV—Signed Divide

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F6 /7	IDIV r/m8	Valid	Valid	Signed divide AX by $r/m8$, with result stored in AL \leftarrow Quotient, AH \leftarrow Remainder
REX + F6 /7	IDIV r/m8*	Valid	N.E.	Signed divide AX by r/m 8, with result stored in AL \leftarrow Quotient, AH \leftarrow Remainder
F7 /7	IDIV r/m16	Valid	Valid	Signed divide DX:AX by $r/m16$, with result stored in AX \leftarrow Quotient, DX \leftarrow Remainder
F7 /7	IDIV r/m32	Valid	Valid	Signed divide EDX:EAX by <i>r/m32</i> , with result stored in EAX ← Quotient, EDX ← Remainder
REX.W + F7 /7	IDIV r/m64	Valid	N.E.	Signed divide RDX:RAX by $r/m64$, with result stored in RAX \leftarrow Quotient, RDX \leftarrow Remainder

In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Promoted to 64-bits.

Default operation size 32-bits

RAX contains a 64-bit quotient, RDX contains a 64-bit remainder.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#DE If the source operand (divisor) is 0.

The signed result (quotient) is too large for the destination.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#DE If the source operand (divisor) is 0.

The signed result (quotient) is too large for the destination.

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.



Virtual-8086 Mode Exceptions

#DE If the source operand (divisor) is 0.

The signed result (quotient) is too large for the destination.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#DE If the source operand (divisor) is 0

If the quotient is too large for the designated register.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

IMUL—Signed Multiply

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F6 /5	IMUL r/m8*	Valid	Valid	AX← AL * <i>r/m</i> byte
F7 /5	IMUL r/m16	Valid	Valid	$DX:AX \leftarrow AX * r/m \text{ word}$
F7 /5	IMUL r/m32	Valid	Valid	EDX:EAX ← EAX * r/m32
REX.W + F7 /5	IMUL r/m64	Valid	N.E.	RDX:RAX \leftarrow RAX * $r/m64$
0F AF /r	IMUL r16,r/m16	Valid	Valid	word register ← word register * r/m16
0F AF /r	IMUL r32,r/m32	Valid	Valid	doubleword register \leftarrow doubleword register * $r/m32$
REX.W + 0F AF /r	IMUL r64,r/m64	Valid	N.E.	Quadword register ← Quadword register * r/m Quadword
6B /r ib	IMUL r16,r/m16,imm8	Valid	Valid	word register $\leftarrow r/m16 * sign-extended$ immediate byte
6B /r ib	IMUL r32,r/m32,imm8	Valid	Valid	doubleword register $\leftarrow r/m32 * sign-extended immediate byte$
REX.W + 6B /r ib	IMUL r64,r/m64,imm8	Valid	N.E.	Quadword register $\leftarrow r/m64 * sign-extended immediate byte$
6B /r ib	IMUL r16,imm8	Valid	Valid	word register \leftarrow word register $*$ signextended immediate byte
6B /r ib	IMUL r32,imm8	Valid	Valid	doubleword register \leftarrow doubleword register $*$ sign-extended immediate byte
REX.W + 6B /r ib	IMUL r64,imm8	Valid	N.E.	Quadword register ← Quadword register * sign-extended immediate byte
69 /r iw	IMUL r16,r/m16,imm16	Valid	Valid	word register ← r/m16 * immediate word
69 /r id	IMUL r32,r/m32,imm32	Valid	Valid	doubleword register ← r/m32 * immediate doubleword
REX.W + 69 /r id	IMUL r64,r/m64,imm32	Valid	N.E.	Quadword register $\leftarrow r/m64 * immediate$ doubleword
69 /r iw	IMUL r16,imm16	Valid	Valid	word register ← r/m16 * immediate word
69 /r id	IMUL r32,imm32	Valid	Valid	doubleword register $\leftarrow r/m32 *$ immediate doubleword
REX.W + 69 /r id	IMUL r64,imm32	Valid	N.E.	Quadword register $\leftarrow r/m64 * immediate$ doubleword

In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also refer to Section 1.4.2.2.

Flags Affected

For the one operand form of the instruction, the CF and OF flags are set when significant 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.

IA-32e Mode Operation

Promoted to 64-bits.

Default operation size 32-bits

Enables access to new registers R8-R15.



Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.



IN—Input from Port

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
E4 ib	IN AL,imm8	Valid	Valid	Input byte from imm8 I/O port address into AL
E5 ib	IN AX,imm8	Valid	Valid	Input word from imm8 I/O port address into AX
E5 ib	IN EAX,imm8	Valid	Valid	Input dword from imm8 I/O port address into EAX
REX.W + E5 ib	IN RAX,imm8	N.P.	N.E.	REX does not change ensuing instruction
EC	IN AL,DX	Valid	Valid	Input byte from I/O port in DX into AL
ED	IN AX,DX	Valid	Valid	Input word from I/O port in DX into AX
ED	IN EAX,DX	Valid	Valid	Input doubleword from I/O port in DX into EAX
REX.W + ED	IN RAX,DX	N.P.	N.E.	REX does not change ensuing instruction

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode.

Default operation size 32-bits

Protected Mode Exceptions

#GP(0) If the CPL is greater than (has less privilege) the I/O privilege level (IOPL) and any of the

corresponding I/O permission bits in TSS for the I/O port being accessed is 1.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

#GP(0) If any of the I/O permission bits in the TSS for the I/O port being accessed is 1.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

#GP(0) If the CPL is greater than (has less privilege) the I/O privilege level (IOPL) and any of the

corresponding I/O permission bits in TSS for the I/O port being accessed is 1.



INC—Increment by 1

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
FE /0	INC r/m8	Valid	Valid	Increment r/m byte by 1
REX + FE /0	INC r/m8*	Valid	N.E.	Increment r/m byte by 1
FF /0	INC r/m16	Valid	Valid	Increment r/m word by 1
FF /0	INC r/m32	Valid	Valid	Increment r/m doubleword by 1
REX.W + FF /0	INC r/m64	Valid	N.E.	Increment r/m quadword by 1
40+ <i>rw</i>	INC <i>r16</i>	N.E.	Valid	Increment word register by 1
40+ rd	INC <i>r32</i>	N.E.	Valid	Increment doubleword register by 1

In 64-bit mode, r/m8 can not be encoded to access the following byte registers if an REX prefix is used: AH, BH, CH, DH. Also, refer to Section 1.4.2.2.

Flags Affected

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

IA-32e Mode Operation

Promoted to 64-bits.

Default operation size 32-bits

Opcode 40H through 47H are REX prefixes in 64-bit mode.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If the destination operand is located in a nonwritable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.



INS/INSB/INSW/INSD—Input from Port to String

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
6C	INS m8, DX	Valid	Valid	Input byte from I/O port specified in DX into memory location specified in ES:(E)DI
REX.W + 6C	INS m8, DX	Valid	N.E.	Input byte from I/O port specified in DX into memory location specified in RDI
6D	INS m16, DX	Valid	Valid	Input word from I/O port specified in DX into memory location specified in ES:(E)DI
6D	INS m32, DX	Valid	Valid	Input doubleword from I/O port specified in DX into memory location specified in ES:(E)DI
REX.W + 6D	INS m32, DX	N.P.	N.E.	Input default size from I/O port specified in DX into memory location specified in RDI
6C	INSB	Valid	Valid	Input byte from I/O port specified in DX into memory location specified with ES:(E)DI
REX.W + 6C	INSB	Valid	N.E.	Input byte from I/O port specified in DX into memory location specified with RDI
6D	INSW	Valid	Valid	Input word from I/O port specified in DX into memory location specified in ES:(E)DI
6D	INSD	Valid	Valid	Input doubleword from I/O port specified in DX into memory location specified in ES:(E)DI
REX.W + 6D	INSD	N.P.	N.E.	Input default size from I/O port specified in DX into memory location specified in RDI

Flags Affected

None.

IA-32e Mode Operation

Default operand size is 32 bits and is not promoted by REX.W.

64-bit mode enables the use of RDI.

Protected Mode Exceptions

#GP(0) If the CPL is greater than (has less privilege) the I/O privilege level (IOPL) and any of the

corresponding I/O permission bits in TSS for the I/O port being accessed is 1.

If the destination is located in a nonwritable segment.

If an illegal memory operand effective address in the ES segments is given.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If any of the I/O permission bits in the TSS for the I/O port being accessed is 1.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.



Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

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

#GP(0) If the CPL is greater than (has less privilege) the I/O privilege level (IOPL) and any of the

corresponding I/O permission bits in TSS for the I/O port being accessed is 1.

If the memory address is in a non-canonical form.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

INT n/INTO/INT 3—Call to Interrupt Procedure

Opco	ode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
CC		INT 3	Valid	Valid	Interrupt 3—trap to debugger
CD	ib	INT imm8	Valid	Valid	Interrupt vector number specified by immediate byte
CE		INTO	No	Valid	Interrupt 4—if overflow flag is 1

Flags Affected

The EFLAGS register is pushed onto the stack. The IF, TF, NT, AC, RF, and VM flags may be cleared, depending on the mode of operation of the processor when the INT instruction is executed (see the "Operation" section). If the interrupt uses a task gate, any flags may be set or cleared, controlled by the EFLAGS image in the new task's TSS.

IA-32e Mode Operation

Default operation size 32-bits

Protected Mode Exceptions

#GP(0)	If the instruction pointer in the IDT or in the interrupt-, trap-, or task gate is beyond the code

segment limits.

#GP(selector) If the segment selector in the interrupt-, trap-, or task gate is null.

If a interrupt-, trap-, or task gate, code segment, or TSS segment selector index is outside

its descriptor table limits.

If the interrupt vector number is outside the IDT limits.

If an IDT descriptor is not an interrupt-, trap-, or task-descriptor.

If an interrupt is generated by the INT n, INT 3, or INTO instruction and the DPL of an

interrupt-, trap-, or task-descriptor is less than the CPL.

If the segment selector in an interrupt- or trap-gate does not point to a segment descriptor

for a code segment.

If the segment selector for a TSS has its local/global bit set for local.

If a TSS segment descriptor specifies that the TSS is busy or not available.

#SS(0) If pushing the return address, flags, or error code onto the stack exceeds the bounds of the

stack segment and no stack switch occurs.

#SS(selector) If the SS register is being loaded and the segment pointed to is marked not present.

If pushing the return address, flags, error code, or stack segment pointer exceeds the bounds

of the new stack segment when a stack switch occurs.

#NP(selector) If code segment, interrupt-, trap-, or task gate, or TSS is not present.

#TS(selector) If the RPL of the stack segment selector in the TSS is not equal to the DPL of the code

segment being accessed by the interrupt or trap gate.

If DPL of the stack segment descriptor pointed to by the stack segment selector in the TSS is not equal to the DPL of the code segment descriptor for the interrupt or trap gate.

If the stack segment selector in the TSS is null.

If the stack segment for the TSS is not a writable data segment.

If segment-selector index for stack segment is outside descriptor table limits.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the interrupt vector number is outside the IDT limits.

#SS If stack limit violation on push.

If pushing the return address, flags, or error code onto the stack exceeds the bounds of the

stack segment.

Virtual-8086 Mode Exceptions

#GP(0) (For INT n, INTO, or BOUND instruction) If the IOPL is less than 3 or the DPL of the

interrupt-, trap-, or task-gate descriptor is not equal to 3.

If the instruction pointer in the IDT or in the interrupt-, trap-, or task gate is beyond the code

segment limits.

#GP(selector) If the segment selector in the interrupt-, trap-, or task gate is null.

If a interrupt-, trap-, or task gate, code segment, or TSS segment selector index is outside

its descriptor table limits.

If the interrupt vector number is outside the IDT limits.

If an IDT descriptor is not an interrupt-, trap-, or task-descriptor.

If an interrupt is generated by the INT *n* instruction and the DPL of an interrupt-, trap-, or

task-descriptor is less than the CPL.

If the segment selector in an interrupt- or trap-gate does not point to a segment descriptor

for a code segment.

If the segment selector for a TSS has its local/global bit set for local.

#SS(selector) If the SS register is being loaded and the segment pointed to is marked not present.

If pushing the return address, flags, error code, stack segment pointer, or data segments

exceeds the bounds of the stack segment.

#NP(selector) If code segment, interrupt-, trap-, or task gate, or TSS is not present.

#TS(selector) If the RPL of the stack segment selector in the TSS is not equal to the DPL of the code

segment being accessed by the interrupt or trap gate.

If DPL of the stack segment descriptor for the TSS's stack segment is not equal to the DPL

of the code segment descriptor for the interrupt or trap gate.

If the stack segment selector in the TSS is null.

If the stack segment for the TSS is not a writable data segment.

If segment-selector index for stack segment is outside descriptor table limits.

#PF(fault-code) If a page fault occurs.

#BP If the INT 3 instruction is executed.

#OF If the INTO instruction is executed and the OF flag is set.

Compatibility Mode Exceptions

#GP(0) If the instruction pointer in the 64-bit interrupt gate or 64-bit trap gate is non-canonical.

#GP(selector) If the segment selector in the 64-bit interrupt or trap gate is null.

If the interrupt vector number is outside the IDT limits.

If the interrupt vector number points to a gate which is in non-canonical space.

If the interrupt vector number points to a descriptor which is not a 64-bit interrupt gate or

64-bit trap gate.

If the descriptor pointed to by the gate selector is outside the descriptor table limit.

If the descriptor pointed to by the gate selector is in non-canonical space.

If the descriptor pointed to by the gate selector is not a code segment.

If the descriptor pointed to by the gate selector doesn't have the L-bit set, or has both the L-

bit and D-bit set.

If the descriptor pointed to by the gate selector has DPL > CPL.

#SS(0) If a push of the old EFLAGS, CS selector, EIP, or error code is in non-canonical space with

no stack switch.

#SS(selector) If a push of the old SS selector, ESP, EFLAGS, CS selector, EIP, or error code is in non-

canonical space on a stack switch (either CPL change or no-CPL with IST).

#NP(selector) If the 64-bit interrupt-gate, 64-bit trap-gate, or code segment is not present.

#TS(selector) If an attempt to load RSP from the TSS causes an access to non-canonical space.

If the RSP from the TSS is outside descriptor table limits.

#PF(fault-code) If a page fault occurs.

INVD—Invalidate Internal Caches

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 08	INVD	Valid	Valid	Flush internal caches; initiate flushing of external caches.

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

#GP(0) The INVD instruction cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions



INVLPG—Invalidate TLB Entry

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	Ī
0F 01/7	INVLPG m	Valid	Valid	Invalidate TLB Entry for page that contains m	

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.

#UD Operand is a register.

Real-Address Mode Exceptions

#UD Operand is a register.

Virtual-8086 Mode Exceptions

#GP(0) The INVLPG instruction cannot be executed at the virtual-8086 mode.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#GP(0) If the current privilege level is not 0.

If the memory address is in a non-canonical form.

#UD Operand is a register.

IRET/IRETD—Interrupt Return

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
CF	IRET	Valid	Valid	Interrupt return (16-bit operand size)
CF	IRETD	Valid	Valid	Interrupt return (32-bit operand size)
REX.W + CF	IRETQ	Valid	N.E.	Interrupt return (64-bit operand size)

Flags Affected

All the flags and fields in the EFLAGS register are potentially modified, depending on the mode of operation of the processor. If performing a return from a nested task to a previous task, the EFLAGS register will be modified according to the EFLAGS image stored in the previous task's TSS.

IA-32e Mode Operation

Promoted to 64-bits.

Default operation size 32-bits

Protected Mode Exceptions

#GP(0) If the return code or stack segment selector is null.

If the return instruction pointer is not within the return code segment limit.

#GP(selector) If a segment selector index is outside its descriptor table limits.

If the return code segment selector RPL is greater than the CPL.

If the DPL of a conforming-code segment is greater than the return code segment selector

RPL.

If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment

selector.

If the stack segment descriptor DPL is not equal to the RPL of the return code segment

selector.

If the stack segment is not a writable data segment.

If the stack segment selector RPL is not equal to the RPL of the return code segment

selector.

If the segment descriptor for a code segment does not indicate it is a code segment.

If the segment selector for a TSS has its local/global bit set for local.

If a TSS segment descriptor specifies that the TSS is busy or not available.

#SS(0) If the top bytes of stack are not within stack limits.

#NP(selector) If the return code or stack segment is not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference occurs when the CPL is 3 and alignment checking is

enabled.

Real-Address Mode Exceptions

#GP If the return instruction pointer is not within the return code segment limit.

#SS If the top bytes of stack are not within stack limits.



Virtual-8086 Mode Exceptions

#GP(0) If the return instruction pointer is not within the return code segment limit.

IF IOPL not equal to 3

#PF(fault-code) If a page fault occurs.

#SS(0) If the top bytes of stack are not within stack limits.

#AC(0) If an unaligned memory reference occurs and alignment checking is enabled.

#GP(0) The INVLPG instruction cannot be executed at the virtual-8086 mode.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

#GP(0) If the EFLAGS.NT bit is set.

If the return code segment selector is null.

If the stack segment selector is null going back to compatibility mode.

If the stack segment selector is null going back to CPL3 64-bit mode.

If a null stack segment selector RPL is not equal to CPL going back to non-CPL3 64-bit

mode.

If the return instruction pointer is not within the return code segment limit.

If the return instruction pointer is non-canonical.

#GP(Selector) If a segment selector index is outside its descriptor table limits.

If a segment descriptor memory address is non-canonical.

If the segment descriptor for a code segment does not indicate it is a code segment.

If the proposed new code segment descriptor has both the D-bit and L-bit set.

If the DPL for a nonconforming-code segment is not equal to the RPL of the code segment

selector.

If CPL is greater than the RPL of the code segment selector.

If the DPL of a conforming-code segment is greater than the return code segment selector

RPL.

If the stack segment is not a writable data segment.

If the stack segment descriptor DPL is not equal to the RPL of the return code segment

selector.

If the stack segment selector RPL is not equal to the RPL of the return code segment

selector.

#SS(0) If an attempt to pop a value off the stack violates the SS limit.

If an attempt to pop a value off the stack causes a non-canonical address to be referenced.

#NP(selector) If the return code or stack segment is not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If an unaligned memory reference occurs when the CPL is 3 and alignment checking is

enabled.

J*cc*—Jump if Condition Is Met

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
77 cb	JA <i>rel8</i>	Valid	Valid	Jump short if above (CF=0 and ZF=0)
73 <i>cb</i>	JAE <i>rel8</i>	Valid	Valid	Jump short if above or equal (CF=0)
72 <i>cb</i>	JB <i>rel8</i>	Valid	Valid	Jump short if below (CF=1)
76 <i>cb</i>	JBE rel8	Valid	Valid	Jump short if below or equal (CF=1 or ZF=1)
72 cb	JC rel8	Valid	Valid	Jump short if carry (CF=1)
E3 cb	JCXZ rel8	Valid	Valid	Jump short if CX register is 0
E3 cb	JECXZ rel8	Valid	Valid	Jump short if ECX register is 0
74 <i>cb</i>	JE rel8	Valid	Valid	Jump short if equal (ZF=1)
7F <i>cb</i>	JG <i>rel8</i>	Valid	Valid	Jump short if greater (ZF=0 and SF=OF)
7D <i>cb</i>	JGE rel8	Valid	Valid	Jump short if greater or equal (SF=OF)
7C <i>cb</i>	JL <i>rel8</i>	Valid	Valid	Jump short if less (SF<>OF)
7E <i>cb</i>	JLE rel8	Valid	Valid	Jump short if less or equal (ZF=1 or SF<>OF)
76 <i>cb</i>	JNA rel8	Valid	Valid	Jump short if not above (CF=1 or ZF=1)
72 <i>cb</i>	JNAE <i>rel8</i>	Valid	Valid	Jump short if not above or equal (CF=1)
73 <i>cb</i>	JNB rel8	Valid	Valid	Jump short if not below (CF=0)
77 cb	JNBE <i>rel8</i>	Valid	Valid	Jump short if not below or equal (CF=0 and ZF=0)
73 <i>cb</i>	JNC rel8	Valid	Valid	Jump short if not carry (CF=0)
75 <i>cb</i>	JNE rel8	Valid	Valid	Jump short if not equal (ZF=0)
7E <i>cb</i>	JNG rel8	Valid	Valid	Jump short if not greater (ZF=1 or SF<>OF)
7C <i>cb</i>	JNGE rel8	Valid	Valid	Jump short if not greater or equal (SF<>OF)
7D <i>cb</i>	JNL rel8	Valid	Valid	Jump short if not less (SF=OF)
7F cb	JNLE rel8	Valid	Valid	Jump short if not less or equal (ZF=0 and SF=OF)
71 <i>cb</i>	JNO rel8	Valid	Valid	Jump short if not overflow (OF=0)
7B <i>cb</i>	JNP rel8	Valid	Valid	Jump short if not parity (PF=0)
79 <i>cb</i>	JNS rel8	Valid	Valid	Jump short if not sign (SF=0)
75 <i>cb</i>	JNZ rel8	Valid	Valid	Jump short if not zero (ZF=0)
70 <i>cb</i>	JO <i>rel8</i>	Valid	Valid	Jump short if overflow (OF=1)
7A <i>cb</i>	JP <i>rel8</i>	Valid	Valid	Jump short if parity (PF=1)
7A <i>cb</i>	JPE <i>rel8</i>	Valid	Valid	Jump short if parity even (PF=1)
7B <i>cb</i>	JPO <i>rel8</i>	Valid	Valid	Jump short if parity odd (PF=0)
78 <i>cb</i>	JS <i>rel8</i>	Valid	Valid	Jump short if sign (SF=1)
74 <i>cb</i>	JZ rel8	Valid	Valid	Jump short if zero (ZF ← 1)
0F 87 <i>cw</i>	JA <i>rel16</i>	N.S.	Valid	Jump near if above (CF=0 and ZF=0). Not supported in 64-bit mode.
0F 87 <i>cd</i>	JA <i>rel32</i>	Valid	Valid	Jump near if above (CF=0 and ZF=0)
0F 83 <i>cw</i>	JAE <i>rel16</i>	N.S.	Valid	Jump near if above or equal (CF=0). Not supported in 64-bit mode.
0F 83 <i>cd</i>	JAE <i>rel32</i>	Valid	Valid	Jump near if above or equal (CF=0)
0F 82 <i>cw</i>	JB rel16	N.S.	Valid	Jump near if below (CF=1). Not supported in 64-bit mode.
0F 82 <i>cd</i>	JB <i>rel32</i>	Valid	Valid	Jump near if below (CF=1)
0F 86 <i>cw</i>	JBE rel16	N.S.	Valid	Jump near if below or equal (CF=1 or ZF=1). Not supported in 64-bit mode.
0F 86 <i>cd</i>	JBE rel32	Valid	Valid	Jump near if below or equal (CF=1 or ZF=1)
0F 82 <i>cw</i>	JC rel16	N.S.	Valid	Jump near if carry (CF=1). Not supported in 64-bit mode.
0F 82 <i>cd</i>	JC rel32	Valid	Valid	Jump near if carry (CF=1)
0F 84 <i>cw</i>	JE rel16	N.S.	Valid	Jump near if equal (ZF=1). Not supported in 64-bit mode.



Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 84 <i>cd</i>	JE rel32	Valid	Valid	Jump near if equal (ZF=1)
0F 84 <i>cw</i>	JZ rel16	N.S.	Valid	Jump near if 0 (ZF=1). Not supported in 64-bit mode.
0F 84 <i>cd</i>	JZ rel32	Valid	Valid	Jump near if 0 (ZF=1)
0F 8F <i>cw</i>	JG rel16	N.S.	Valid	Jump near if greater (ZF=0 and SF=OF). Not supported in 64-bit mode.
0F 8F <i>cd</i>	JG rel32	Valid	Valid	Jump near if greater (ZF=0 and SF=OF)
0F 8D <i>cw</i>	JGE rel16	N.S.	Valid	Jump near if greater or equal (SF=OF). Not supported in 64-bit mode.
0F 8D <i>cd</i>	JGE rel32	Valid	Valid	Jump near if greater or equal (SF=OF)
0F 8C <i>cw</i>	JL <i>rel16</i>	N.S.	Valid	Jump near if less (SF<>OF). Not supported in 64-bit mode.
0F 8C <i>cd</i>	JL rel32	Valid	Valid	Jump near if less (SF<>OF)
0F 8E <i>cw</i>	JLE rel16	N.S.	Valid	Jump near if less or equal (ZF=1 or SF<>OF). Not supported in 64-bit mode.
0F 8E <i>cd</i>	JLE rel32	Valid	Valid	Jump near if less or equal (ZF=1 or SF<>OF)
0F 86 <i>cw</i>	JNA rel16	N.S.	Valid	Jump near if not above (CF=1 or ZF=1). Not supported in 64-bit mode.
0F 86 <i>cd</i>	JNA rel32	Valid	Valid	Jump near if not above (CF=1 or ZF=1)
0F 82 <i>cw</i>	JNAE <i>rel16</i>	N.S.	Valid	Jump near if not above or equal (CF=1). Not supported in 64-bit mode.
0F 82 <i>cd</i>	JNAE <i>rel32</i>	Valid	Valid	Jump near if not above or equal (CF=1)
0F 83 <i>cw</i>	JNB rel16	N.S.	Valid	Jump near if not below (CF=0). Not supported in 64-bit mode.
0F 83 <i>cd</i>	JNB rel32	Valid	Valid	Jump near if not below (CF=0)
0F 87 <i>cw</i>	JNBE rel16	N.S.	Valid	Jump near if not below or equal (CF=0 and ZF=0). Not supported in 64-bit mode.
0F 87 <i>cd</i>	JNBE rel32	Valid	Valid	Jump near if not below or equal (CF=0 and ZF=0)
0F 83 <i>cw</i>	JNC rel16	N.S.	Valid	Jump near if not carry (CF=0). Not supported in 64-bit mode.
0F 83 <i>cd</i>	JNC rel32	Valid	Valid	Jump near if not carry (CF=0)
0F 85 <i>cw</i>	JNE rel16	N.S.	Valid	Jump near if not equal (ZF=0). Not supported in 64-bit mode.
0F 85 <i>cd</i>	JNE rel32	Valid	Valid	Jump near if not equal (ZF=0)
0F 8E <i>cw</i>	JNG rel16	N.S.	Valid	Jump near if not greater (ZF=1 or SF<>OF). Not supported in 64-bit mode.
0F 8E <i>cd</i>	JNG rel32	Valid	Valid	Jump near if not greater (ZF=1 or SF<>OF)
0F 8C <i>cw</i>	JNGE rel16	N.S.	Valid	Jump near if not greater or equal (SF<>OF). Not supported in 64-bit mode.
0F 8C <i>cd</i>	JNGE rel32	Valid	Valid	Jump near if not greater or equal (SF<>OF)
0F 8D <i>cw</i>	JNL rel16	N.S.	Valid	Jump near if not less (SF=OF). Not supported in 64-bit mode.
0F 8D <i>cd</i>	JNL rel32	Valid	Valid	Jump near if not less (SF=OF)
0F 8F <i>cw</i>	JNLE rel16	N.S.	Valid	Jump near if not less or equal (ZF=0 and SF=OF). Not supported in 64-bit mode.
0F 8F <i>cd</i>	JNLE rel32	Valid	Valid	Jump near if not less or equal (ZF=0 and SF=OF)
0F 81 <i>cw</i>	JNO rel16	N.S.	Valid	Jump near if not overflow (OF=0). Not supported in 64-bit mode.
0F 81 <i>cd</i>	JNO rel32	Valid	Valid	Jump near if not overflow (OF=0)
0F 8B <i>cw</i>	JNP rel16	N.S.	Valid	Jump near if not parity (PF=0). Not supported in 64-bit mode.
0F 8B <i>cd</i>	JNP rel32	Valid	Valid	Jump near if not parity (PF=0)
0F 89 <i>cw</i>	JNS rel16	N.S.	Valid	Jump near if not sign (SF=0). Not supported in 64-bit mode.
0F 89 <i>cd</i>	JNS rel32	Valid	Valid	Jump near if not sign (SF=0)

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 85 <i>cw</i>	JNZ rel16	N.S.	Valid	Jump near if not zero (ZF=0). Not supported in 64-bit mode.
0F 85 <i>cd</i>	JNZ rel32	Valid	Valid	Jump near if not zero (ZF=0)
0F 80 <i>cw</i>	JO rel16	N.S.	Valid	Jump near if overflow (OF=1). Not supported in 64-bit mode.
0F 80 <i>cd</i>	JO <i>rel32</i>	Valid	Valid	Jump near if overflow (OF=1)
0F 8A <i>cw</i>	JP rel16	N.S.	Valid	Jump near if parity (PF=1). Not supported in 64-bit mode.
0F 8A <i>cd</i>	JP rel32	Valid	Valid	Jump near if parity (PF=1)
0F 8A <i>cw</i>	JPE rel16	N.S.	Valid	Jump near if parity even (PF=1). Not supported in 64-bit mode.
0F 8A <i>cd</i>	JPE rel32	Valid	Valid	Jump near if parity even (PF=1)
0F 8B <i>cw</i>	JPO rel16	N.S.	Valid	Jump near if parity odd (PF=0). Not supported in 64-bit mode.
0F 8B <i>cd</i>	JPO rel32	Valid	Valid	Jump near if parity odd (PF=0)
0F 88 <i>cw</i>	JS rel16	N.S.	Valid	Jump near if sign (SF=1). Not supported in 64-bit mode.
0F 88 <i>cd</i>	JS rel32	Valid	Valid	Jump near if sign (SF=1)
0F 84 <i>cw</i>	JZ rel16	N.S.	Valid	Jump near if 0 (ZF=1). Not supported in 64-bit mode.
0F 84 <i>cd</i>	JZ rel32	Valid	Valid	Jump near if 0 (ZF=1)

Flags Affected

None.

IA-32e Mode Operation

Operand size fixed at 64-bits

JMP Short is RIP = RIP + 8-bit offset sign extended to 64 bits

JMP Near is RIP = RIP + 32-bit offset sign extended to 64-bits

Protected Mode Exceptions

#GP(0) If the offset being jumped to is beyond the limits of the CS segment.

Real-Address Mode Exceptions

#GP

If the offset being jumped to is beyond the limits of the CS segment or is outside of the effective address space from 0 to FFFFH. This condition can occur if a 32-bit address size override prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

#GP(0)

If the memory address is in a non-canonical form.



JMP—Jump

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
EB cb	JMP rel8	Valid	Valid	Jump short, RIP = RIP + 8-bit displacement sign extended to 64-bits
E9 <i>cw</i>	JMP rel16	N.S.	Valid	Jump near, relative, displacement relative to next instruction. Not supported in 64-bit mode.
E9 <i>cd</i>	JMP rel32	Valid	Valid	Jump near, relative, RIP = RIP + 32-bit displacement sign extended to 64-bits
FF /4	JMP <i>r/m16</i>	N.S.	Valid	Jump near, absolute indirect, address = sign-extended r/m16. Not supported in 64-bit mode.
FF /4	JMP <i>r/m32</i>	N.S.	Valid	Jump near, absolute indirect, address = sign-extended r/m32. Not supported in 64-bit mode.
FF /4	JMP <i>r/m64</i>	Valid	N.E.	Jump near, absolute indirect, RIP = 64-Bit offset from register or memory
EA cd	JMP ptr16:16	Inv.	Valid	Jump far, absolute, address given in operand
EA cp	JMP ptr16:32	Inv.	Valid	Jump far, absolute, address given in operand
FF /5	JMP <i>m16:16</i>	Valid	Valid	Jump far, absolute indirect, address given in m16:16
FF /5	JMP <i>m16:32</i>	Valid	Valid	Jump far, absolute indirect, address given in $m16:32$ In 32-bit mode of operation If selector points to a gate then RIP = 32-bit zero extended displacement taken from gate else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.
FF /5	JMP <i>m16:64</i>	Valid	N.E.	In 64-bit mode of operation If selector points to a gate then RIP = 64-bit displacement taken from gate else RIP = zero extended 32-bit offset from far pointer referenced in the instruction.

Flags Affected

All flags are affected if a task switch occurs; no flags are affected if a task switch does not occur.

IA-32e Mode Operation

Promoted to 64-bits.

Operand size fixed at 64-bits

Protected Mode Exceptions

#GP(0) If offset in target operand, call gate, or TSS is beyond the code segment limits.

If the segment selector in the destination operand, call gate, task gate, or TSS is null.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#GP(selector) If segment selector index is outside descriptor table limits.

If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, call gate, task gate, or task state segment.

If the DPL for a nonconforming-code segment is not equal to the CPL

(When not using a call gate.) If the RPL for the segment's segment selector is greater than the CPL.

If the DPL for a conforming-code segment is greater than the CPL.

If the DPL from a call-gate, task-gate, or TSS segment descriptor is less than the CPL or than the RPL of the call-gate, task-gate, or TSS's segment selector.

If the segment descriptor for selector in a call gate does not indicate it is a code segment.

If the segment descriptor for the segment selector in a task gate does not indicate available

TSS.

If the segment selector for a TSS has its local/global bit set for local.

If a TSS segment descriptor specifies that the TSS is busy or not available.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NP (selector) If the code segment being accessed is not present.

If call gate, task gate, or TSS not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3. (Only occurs when fetching target from memory.)

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If the target operand is beyond the code segment limits.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made. (Only occurs

when fetching target from memory.)

Compatibility Mode Exceptions

Same as 64-bit mode exceptions.

64-Bit Mode Exceptions

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

If target offset in destination operand is non-canonical.

If target offset in destination operand is beyond the new code segment limit.

If the segment selector in the destination operand is null. If the code segment selector in the 64-bit gate is null.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.

#GP(selector) If the code segment or 64-bit call gate is outside descriptor table limits.

If the code segment or 64-bit call gate overlaps non-canonical space.

If the segment descriptor from a 64-bit call gate is in non-canonical space.



If the segment descriptor pointed to by the segment selector in the destination operand is not for a conforming-code segment, nonconforming-code segment, 64-bit call gate.

If the segment descriptor pointed to by the segment selector in the destination operand is a code segment, and has both the D-bit and the L-bit set.

If the DPL for a nonconforming-code segment is not equal to the CPL, or the RPL for the segment's segment selector is greater than the CPL.

If the DPL for a conforming-code segment is greater than the CPL.

If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate.

If the upper type field of a 64-bit call gate is not 0x0.

If the segment selector from a 64-bit call gate is beyond the descriptor table limits.

If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear.

If the segment descriptor for a segment selector from the 64-bit call gate does not indicate it is a code segment.

If the code segment is non-confirming and CPL != DPL.

If the code segment is confirming and CPL < DPL.

#NP(selector) If a code segment or 64-bit call gate is not present.

#UD (64-bit mode only) If a far jump is direct to an absolute address in memory.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

LAHF—Load Status Flags into AH Register

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
9F	LAHF	Inv.	Valid	Load: AH ← EFLAGS(SF:ZF:0:AF:0:PF:1:CF)

Flags Affected

None (that is, the state of the flags in the EFLAGS register is not affected).

IA-32e Mode Operation

Invalid in 64-bit mode.

Protected Mode Exceptions

None

Real-Address Mode Exceptions

None

Virtual-8086 Mode Exceptions

None

Compatibility Mode Exceptions

None

64-Bit Mode Exceptions

#UD If in 64-bit mode.



LAR—Load Access Rights Byte

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 02 /r	LAR r16,r/m16	Valid	Valid	$r16 \leftarrow r/m16$ masked by FF00H
0F 02 /r	LAR r32,r/m32	Valid	Valid	$r32 \leftarrow r/m32$ masked by 00FxFF00H
0F 02 /r	LAR <i>r64,r/m64</i>	Valid	N.E.	r64 ← zero extended r/m32 masked by 00FxFF00H

Flags Affected

The ZF flag is set to 1 if the access rights are loaded successfully; otherwise, it is cleared to 0.

IA-32e Mode Operation

Same as legacy mode.

Default operand size 32-bits.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3. (Only occurs when fetching target from memory.)

Real-Address Mode Exceptions

#UD The LAR instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The LAR instruction cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3. (Only occurs when fetching target from memory.)

LDDQU—Load Unaligned Double Quadword

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
F2 0F F0 /r	LDDQU xmm1, mem	Valid	Valid	Load unaligned data from <i>mem</i> and return double quadword in <i>xmm1</i> .

Operation

If the memory address is not aligned on 16 byte boundary, some implementation may load up to 32 bytes and return 16 bytes in the destination.

IA-32e Mode Operation

Enables access to XMM8-XMM15.

SIMD Floating-Point Exceptions

None.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NM If TS in CR0 is set.
#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode #PF(fault-code) For a page fault.

Compatibility Mode Exceptions



64-Bit Mode Exceptions

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

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

#NM If TS in CR0 is set.
#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE3 is 0.

#PF(fault-code) If a page fault occurs.

LDMXCSR—Load MXCSR Register

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description	
0F,AE,/2	LDMXCSR m32	Valid	Valid	Load MXCSR register from m32.	

IA-32e Mode Operation

Same as legacy mode.

C/C++ Compiler Intrinsic Equivalent

_mm_setcsr(unsigned int i)

Numeric Exceptions

None.

Protected Mode Exceptions

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

For an attempt to set reserved bits in MXCSR.

#SS(0) For an illegal address in the SS segment.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real Address Mode Exceptions

#GP(0) For an attempt to set reserved bits in MXCSR.

#GP(0) If any part of the operand would lie outside of the effective address space from 0 to FFFFH.

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode.

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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions



64-Bit Mode Exceptions

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

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

For an attempt to set reserved bits in MXCSR.

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

#NM If TS in CR0 is set.

#UD If EM in CR0 is set.

If OSFXSR in CR4 is 0.

If CPUID feature flag SSE is 0.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

LDS/LES/LFS/LGS/LSS—Load Far Pointer

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
C5 /r	LDS r16,m16:16	Inv.	Valid	Load DS:r16 with far pointer from memory
C5 /r	LDS r32,m16:32	Inv.	Valid	Load DS:r32 with far pointer from memory
0F B2 /r	LSS r16,m16:16	Valid	Valid	Load SS:r16 with far pointer from memory
0F B2 /r	LSS r32,m16:32	Valid	Valid	Load SS: r32 with far pointer from memory
0F B2 /r	LSS r64,m16:64	Valid	N.E.	Load SS: r64 with far pointer from memory
C4 /r	LES r16,m16:16	Inv.	Valid	Load ES: r16 with far pointer from memory
C4 /r	LES r32,m16:32	Inv.	Valid	Load ES: r32 with far pointer from memory
0F B4 /r	LFS r16,m16:16	Valid	Valid	Load FS:r16 with far pointer from memory
0F B4 /r	LFS r32,m16:32	Valid	Valid	Load FS:r32 with far pointer from memory
0F B4 /r	LFS r64,m16:64	Valid	N.E.	Load FS:r64 with far pointer from memory
0F B5 /r	LGS r16,m16:16	Valid	Valid	Load GS:r16 with far pointer from memory
0F B5 /r	LGS r32,m16:32	Valid	Valid	Load GS: r32 with far pointer from memory
0F B5 /r	LGS r64,m16:64	Valid	N.E.	Load GS: r64 with far pointer from memory

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode.

Default operand size 32-bits.

Enables access to new registers R8-R15.

Protected Mode Exceptions

#UD If source operand is not a memory location.
#GP(0) If a null selector is loaded into the SS register.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#GP(selector) If the SS register is being loaded and any of the following is true: the segment selector index

is not within the descriptor table limits, the segment selector RPL is not equal to CPL, the

segment is a nonwritable data segment, or DPL is not equal to CPL.

If the DS, ES, FS, or GS register is being loaded with a non-null segment selector and any of the following is true: the segment selector index is not within descriptor table limits, the segment is neither a data nor a readable code segment, or the segment is a data or noncon-

forming-code segment and both RPL and CPL are greater than DPL.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#SS(selector) If the SS register is being loaded and the segment is marked not present.

#NP(selector) If DS, ES, FS, or GS register is being loaded with a non-null segment selector and the

segment is marked not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.



Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

#UD If source operand is not a memory location.

Virtual-8086 Mode Exceptions

#UD If source operand is not a memory location.

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

If a null selector is attempted to be loaded into the SS register in compatibility mode.

If a null selector is attempted to be loaded into the SS register in CPL3 and 64-bit mode.

If a null selector is attempted to be loaded into the SS register in non-CPL3 and 64-bit mode

where its RPL is not equal to CPL.

#GP(Selector) If the DS, ES, FS, or GS register is being loaded with a non-null segment selector and any

of the following is true: the segment selector index is not within descriptor table limits, the memory address of the descriptor is non-canonical, the segment is neither a data nor a readable code segment, or the segment is a data or nonconforming-code segment and both RPL

and CPL are greater than DPL.

If the SS register is being loaded and any of the following is true: the segment selector index is not within the descriptor table limits, the memory address of the descriptor is non-canonical, the segment selector RPL is not equal to CPL, the segment is a nonwritable data

segment, or DPL is not equal to CPL.

#SS(0) If a memory operand effective address is outside the SS segment limit.

If a memory operand effective address is non-canonical

#SS(Selector) If the SS register is being loaded and the segment is marked not present.

#NP(selector) If DS, ES, FS, or GS register is being loaded with a non-null segment selector and the

segment is marked not present.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

#UD If source operand is not a memory location.

LEA—Load Effective Address

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
8D /r	LEA <i>r16,m</i>	Valid	Valid	Store effective address for m in register r16
8D /r	LEA <i>r32,m</i>	Valid	Valid	Store effective address for <i>m</i> in register <i>r32</i>
8D /r	LEA <i>r64,m</i>	Valid	N.E.	Store effective address for <i>m</i> in register <i>r64</i> . Zero extended 32-bit register results to 64-bits.

Flags Affected

None.

IA-32e Mode Operation

Default operand size 32-bits.

Protected Mode Exceptions

#UD If source operand is not a memory location.

Real-Address Mode Exceptions

#UD If source operand is not a memory location.

Virtual-8086 Mode Exceptions

#UD If source operand is not a memory location.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#UD If source operand is not a memory location.



LEAVE—High Level Procedure Exit

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
C9	LEAVE	Valid	Valid	Set SP to BP, then pop BP
C9	LEAVE	Valid	Valid	Set ESP to EBP, then pop EBP
C9	LEAVE	Valid	N.E.	Set RSP to RBP, then pop RBP

Flags Affected

None.

IA-32e Mode Operation

Default Operation Size is 64 bits In 64-bit mode a 32-bit operation

Protected Mode Exceptions

#SS(0) If the EBP register points to a location that is not within the limits of the current stack

segment.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If the EBP register points to a location outside of the effective address space from 0 to

FFFFH.

Virtual-8086 Mode Exceptions

#GP(0) If the EBP register points to a location outside of the effective address space from 0 to

FFFFH.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

LES—Load Full Pointer

See entry for LDS/LES/LFS/LGS/LSS—Load Far Pointer.



LFENCE—Load Fence

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F AE /5	LFENCE	Valid	Valid	Serializes load operations.

IA-32e Mode Operation

Same as legacy mode.

Intel C/C++ Compiler Intrinsic Equivalent

void_mm_lfence(void)

Exceptions (All Modes of Operation)

None.

LFS—Load Full Pointer

See entry for LDS/LES/LFS/LGS/LSS—Load Far Pointer.



LGDT/LIDT—Load Global/Interrupt Descriptor Table Register

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 01 /2	LGDT m16&32	N.E.	Valid	Load <i>m</i> into GDTR
0F 01 /3	LIDT m16&32	N.E.	Valid	Load <i>m</i> into IDTR
0F 01 /2	LGDT m16&64	Valid	N.E.	Load <i>m</i> into GDTR
0F 01 /3	LIDT <i>m16&64</i>	Valid	N.E.	Load <i>m</i> into IDTR

Flags Affected

None.

64-bit Mode Operation

Promoted to 64-bits.

Operand size fixed at 8+2 bytes.

Loads 8 byte base and 2 byte limit.

Protected Mode Exceptions

#UD If source operand is not a memory location.

#GP(0) If the current privilege level is not 0.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#UD If source operand is not a memory location.

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#GP(0) If the current privilege level is not 0.

If the memory address is in a non-canonical form.

#UD If source operand is not a memory location.

#PF(fault-code) If a page fault occurs.

LGS—Load Full Pointer

See entry for LDS/LES/LFS/LGS/LSS—Load Far Pointer.



LLDT—Load Local Descriptor Table Register

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 00 /2	LLDT r/m16	Valid	Valid	Load segment selector r/m16 into LDTR

Flags Affected

None.

IA-32e Mode Operation

Operand size fixed at 16 bits.

References 64-bit mode descriptor to load 64-bit base.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#GP(selector) If the selector operand does not point into the Global Descriptor Table or if the entry in the

GDT is not a Local Descriptor Table.

Segment selector is beyond GDT limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#NP(selector) If the LDT descriptor is not present.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#UD The LLDT instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The LLDT instruction is recognized in virtual-8086 mode.

Compatibility Mode Exceptions

64-Bit Mode Exceptions

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

#GP(0) If the current privilege level is not 0.

If the memory address is in a non-canonical form.

#GP(selector) If the selector operand does not point into the Global Descriptor Table or if the entry in the

GDT is not a Local Descriptor Table.

Segment selector is beyond GDT limit.

#NP(selector) If the LDT descriptor is not present.

#PF(fault-code) If a page fault occurs.

LIDT—Load Interrupt Descriptor Table Register

See entry for LGDT/LIDT—Load Global/Interrupt Descriptor Table Register.

LMSW—Load Machine Status Word

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 01 /6	LMSW r/m16	Valid	Valid	Loads r/m16 in machine status word of CR0

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode.

Operand size fixed at 16 bits.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If the current privilege level is not 0.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

#GP(0) If the current privilege level is not 0.

If the memory address is in a non-canonical form.

#PF(fault-code) If a page fault occurs.



LOCK—Assert LOCK# Signal Prefix

(Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
	F0	LOCK	Valid	Valid	Asserts LOCK# signal for duration of the accompanying instruction

Operation

AssertLOCK#(DurationOfAccompaningInstruction)

Flags Affected

None.

IA-32e Mode Operation

Same as legacy mode.

Protected Mode Exceptions

#UD If the LOCK prefix is used with an instruction not listed in the "Description" section above.

Other exceptions can be generated by the instruction that the LOCK prefix is being applied

to.

Real-Address Mode Exceptions

#UD If the LOCK prefix is used with an instruction not listed in the "Description" section above.

Other exceptions can be generated by the instruction that the LOCK prefix is being applied

to.

Virtual-8086 Mode Exceptions

#UD If the LOCK prefix is used with an instruction not listed in the "Description" section above.

Other exceptions can be generated by the instruction that the LOCK prefix is being applied

to.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

LODS/LODSB/LODSW/LODSD/LODSQ—Load String

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
AC	LODS m8	Valid	Valid	For legacy mode, Load byte at address DS:(E)SI into AL. For 64-bit mode load byte at address (R)SI into AL.
AD	LODS m16	Valid	Valid	For legacy mode, Load word at address DS:(E)SI into AX. For 64-bit mode load word at address (R)SI into AX.
AD	LODS m32	Valid	Valid	For legacy mode, Load dword at address DS:(E)SI into EAX. For 64-bit mode load dword at address (R)SI into EAX.
REX.W + AD	LODS m64	Valid	N.E.	Load qword at address (R)SI into RAX.
AC	LODSB	Valid	Valid	For legacy mode, Load byte at address DS:(E)SI into AL. For 64-bit mode load byte at address (R)SI into AL.
AD	LODSW	Valid	Valid	For legacy mode, Load word at address DS:(E)SI into AX. For 64-bit mode load word at address (R)SI into AX.
AD	LODSD	Valid	Valid	For legacy mode, Load dword at address DS:(E)SI into EAX. For 64-bit mode load dword at address (R)SI into EAX.
REX.W + AD	LODSQ	Valid	N.E.	Load qword at address (R)SI into RAX.

Flags Affected

None.

IA-32e Mode Operation

Promoted to 64 bits.

Default operand size 32 bits.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a null segment selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#GP If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS If a memory operand effective address is outside the SS segment limit.

Virtual-8086 Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made.

Compatibility Mode Exceptions



64-Bit Mode Exceptions

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

LOOP/LOOPcc—Loop According to ECX Counter

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
E2 cb	LOOP rel8	Valid	Valid	Decrement count; jump short if count ≠ 0
REX.W + E2 <i>cb</i>	LOOP rel8	Valid	N.E.	Decrement count; jump short if count \neq 0. JMP Short is RIP = RIP + 8-bit offset sign-extended to 64-bits.
E1 cb	LOOPE rel8	Valid	Valid	Decrement count; jump short if count ≠ 0 and ZF=1
REX.W + E1 cb	LOOPE rel8	Valid	N.E.	Decrement count; jump short if count \neq 0 and ZF=1. JMP Short is RIP = RIP + 8-bit offset sign-extended to 64-bits.
E0 cb	LOOPNE rel8	Valid	Valid	Decrement count; jump short if count ≠ 0 and ZF=0
REX.W + E0 cb	LOOPNZ rel8	Valid	N.E.	Decrement count; jump short if count \neq 0 and ZF=0. JMP Short is RIP = RIP + 8-bit offset sign-extended to 64-bits.

Flags Affected

None.

IA-32e Mode Operation

Promoted to 64 bits.

Operand size at 64 bits.

JMP Short is RIP = RIP + 8-bit offset sign extended to 64 bits.

Protected Mode Exceptions

#GP(0) If the offset being jumped to is beyond the limits of the CS segment.

Real-Address Mode Exceptions

#GP

If the offset being jumped to is beyond the limits of the CS segment or is outside of the effective address space from 0 to FFFFH. This condition can occur if a 32-bit address size override prefix is used.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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



LSL—Load Segment Limit

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 03 /r	LSL r16,r/m16	Valid	Valid	Load: r16 ← segment limit, selector r/m16
0F 03 /r	LSL r32,r/m32	Valid	Valid	Load: r32 ← segment limit, selector r/m32
REX.W + 0F 03 /r	LSL r64,r/m32	Valid	Valid	Load: $r64 \leftarrow zero\ extended$ segment limit, selector $r/m64$

Flags Affected

The ZF flag is set to 1 if the segment limit is loaded successfully; otherwise, it is cleared to 0.

IA-32e Mode Operation

Same as legacy mode

Default operand size at 32 bits.

Protected Mode Exceptions

#GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

Real-Address Mode Exceptions

#UD The LSL instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The LSL instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same as for protected mode exceptions.

64-Bit Mode Exceptions

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

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

#PF(fault-code) If a page fault occurs.

#AC(0) If alignment checking is enabled and an unaligned memory reference is made while the

current privilege level is 3.

LSS—Load Full Pointer

See entry for LDS/LES/LFS/LGS/LSS—Load Far Pointer.



LTR—Load Task Register

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
0F 00 /3	LTR r/m16	Valid	Valid	Load r/m16 into task register

Flags Affected

None.

IA-32e Mode Operation

Operand size fixed at 16 bits.

References 64-bit mode descriptor to load 64-bit base.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register is used to access memory and it contains a null segment

selector

#GP(selector) If the source selector points to a segment that is not a TSS or to one for a task that is already

busy.

If the selector points to LDT or is beyond the GDT limit.

#NP(selector) If the TSS is marked not present.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#PF(fault-code) If a page fault occurs.

Real-Address Mode Exceptions

#UD The LTR instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The LTR instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

64-Bit Mode Exceptions

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

#GP(0) If the current privilege level is not 0.

If the memory address is in a non-canonical form.

#GP(selector) If the source selector points to a segment that is not a TSS or to one for a task that is already

busy.

If the selector points to LDT or is beyond the GDT limit.

#NP(selector) If the TSS is marked not present.

#PF(fault-code) If a page fault occurs.